Simulink® Real-Time™

I/O Reference

MATLAB®&SIMULINK®

R2021**b**

MathWorks®

How to Contact MathWorks

	Latest news:	www.mathworks.com
	Sales and services:	www.mathworks.com/sales_and_services
	User community:	www.mathworks.com/matlabcentral
	Technical support:	www.mathworks.com/support/contact_us
	Phone:	508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Revision History

Contents

Async Block Library

Asynchronous Event: Blocks

3

CAN Message Blocks Library

CAN Utility Blocks

4

EtherCAT Blocks Library

Model-Based EtherCAT Communications Support

5

6

TCP Transmission Control Protocol Blocks Library

7

8

9

10

SAE J1939 Blocks Library

11

SAE J1939 Blocks

12

Logitech

Logitech Blocks

13

LIN

LIN Blocks

14

Logging Blocks Library

Logging Blocks

15

Profiling Blocks Library

Profiling Blocks

16

PTP Precision Time Protocol Blocks Library

PTP Blocks

17

Precision Time Protocol

18

Precision Time Protocol Blocks

19

RS232 Serial Blocks Library

Serial Communications Support

20

21
Serial Communications Support: Blocks

Target Management

22
Target Management Blocks

Utilities

23
Utility Blocks

XCP Universal Measurement and Calibration Protocol

24
XCP Master Mode

25
Stimulation Support

26

Speedgoat Blocks Library

27

Simulink Real-Time I/O Library

I/O Driver Blocks

In this section...
"Speedgoat I/O Modules" on page 1-2
"Speedgoat I/O Blockset" on page 1-2
"Simulink Real-Time Block Library" on page 1-2

The Simulink Real-Time environment is a solution for prototyping and testing real-time systems by using a desktop computer. To apply this solution, you add I/O blocks to your model. The blocks of the Simulink Real-Time library provide a particular function of an I/O module. By using I/O blocks in your model, you can generate executable code tuned specifically to your I/O requirements.

You add I/O driver blocks to your Simulink model to connect your model to I/O modules (I/O boards). These I/O modules then connect to the sensors and actuators in a physical system.

Speedgoat I/O Modules

Speedgoat® real-time target machines are available with various I/O modules. See "Speedgoat I/O Hardware" on page 27-2.

Speedgoat I/O Blockset

You can use the blocks in the Speedgoat I/O Blockset and the blocks in the Simulink Real-Time library in your Simulink Real-Time model. For a description of the Speedgoat blocks, see "Speedgoat I/O Hardware" on page 27-2.

Simulink Real-Time Block Library

A driver block does not represent an entire board, but an I/O section supported by a board. The Simulink Real-Time library can have more than one block for each physical board. I/O driver blocks are written as C-code S-functions (noninlined S-functions). The source code for the C-code S-functions is included with the Simulink Real-Time software.

If your model contains I/O blocks, take I/O latency values into account for the model sample time.

To find latency values for Speedgoat boards, contact Speedgoat technical support.

You can open the I/O device driver library by using the Simulink library browser or by using the MATLAB® command `slrealtimelib`. The library `slrealtimelib` contains sublibraries grouped by the type of I/O function they provide.

When you double-click one of the I/O block groups, the sublibrary opens, displaying a list grouped by manufacturer. Double-clicking one of the manufacturer groups displays the I/O device driver blocks for the specified I/O functionality.

When you double-click one of the blocks, a Block Parameters dialog box opens, where you enter system-specific parameters. Parameters typically include block sample time and other block-specific parameters.

See Also

More About

- *"Add I/O Blocks to Simulink Model" on page 1-4*
- *"Configure Block Parameters" on page 1-6*

Add I/O Blocks to Simulink Model

You can transform a Simulink model into a Simulink Real-Time model that accesses I/O drivers either by using the Simulink Real-Time block library or the Simulink Real-Time: Speedgoat I/O Blockset. In the Simulink Real-Time block library, the highest hierarchical level in the library lists I/O function groups. The second level lists board manufacturer groups. The manufacturer groups contain the driver blocks for specific boards.

This example uses the Simulink model `slrt_ex_osc` to show how to replace Simulink blocks with Simulink Real-Time I/O blocks. For examples, at the MATLAB command prompt, type:

```
open_system(fullfile(matlabroot, 'toolbox', 'slrealtime', ...
'examples', 'slrt_ex_osc'))
```

1 To browse the Simulink Real-Time block library, open the **Library: slrealtimelib** window. In the Command Window, type:

```
slrealtimelib
```

2 To browse the Simulink Real-Time: Speedgoat I/O Blockset, open the **Library: speedgoatlib** window. In the MATLAB Command Window, type:

```
speedgoatlib
```

3 In the MATLAB Command Window, type:

```
open_system(fullfile(matlabroot, 'toolbox', 'slrealtime', ...
'examples','slrt_ex_osc'))
```

The Simulink block diagram opens for the model `slrt_ex_osc`.
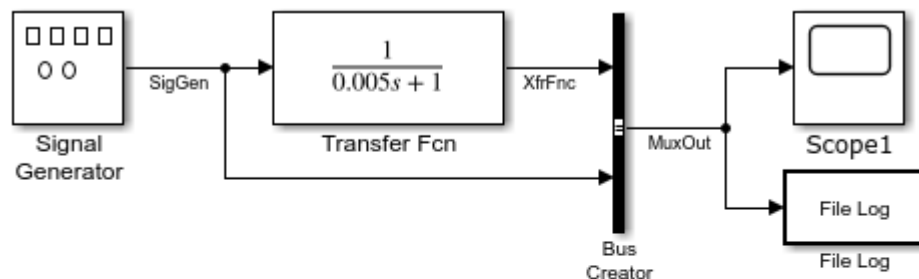
Model slrt_ex_osc
Simulink Real-Time example model

Copyright 1999-2020 The MathWorks, Inc.

4 Open the Simulink Library Browser. Select **Simulink Real-Time: Speedgoat I/O Blockset > IO101**. Drag each of these blocks to the Simulink block diagram: Speedgoat IO101 Analog input block, Speedgoat IO101 Analog output block, and Speedgoat IO101 Setup.

The Simulink Editor adds the new I/O blocks to your model.

5 Remove the Signal Generator block and add the Speedgoat IO101 Analog input block in its place. Remove the Scope block and add the Speedgoat IO101 Analog output block in its place. The block parameters select the number of outputs for the block.

6 Save the model with a new name, such as ex_slrt_iob_osc . To open the completed model, in the MATLAB Command Window, type:

```
open_system(fullfile(docroot, 'toolbox', 'slrealtime', ...
'examples', 'slrt_ex_iob_osc'))
```


Simulink Real-Time tutorial model
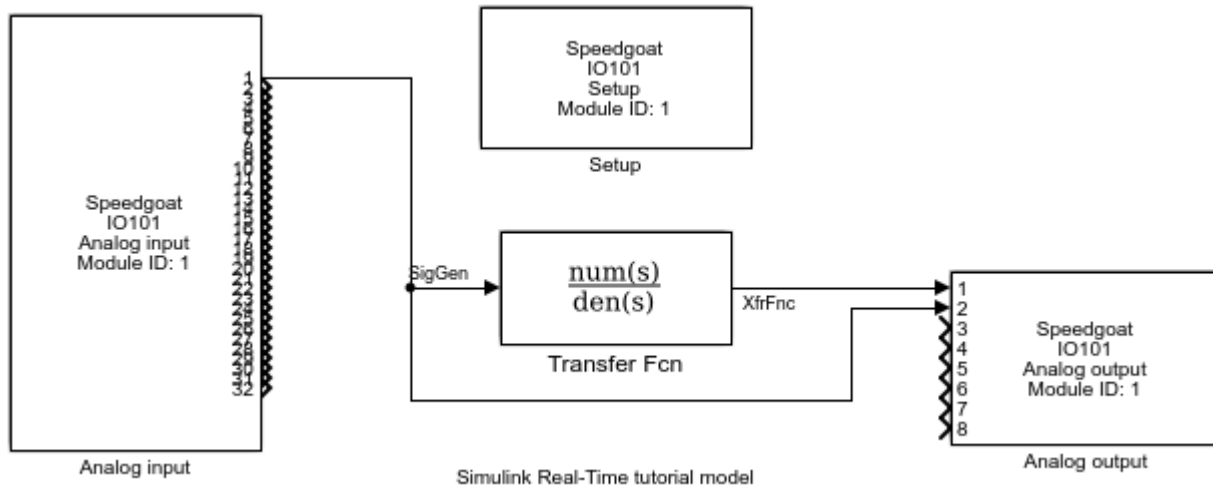
You cannot run this model unless the required I/O board is installed in your target computer. You can substitute driver blocks for another I/O board that is installed in the target computer.

After you add I/O blocks to the model, set up the I/O operation by selecting block parameter values. For more information, see "Configure Block Parameters" on page 1-6.

See Also

More About

- "I/O Driver Blocks" on page 1-2
- "Configure Block Parameters" on page 1-6

Configure Block Parameters

The block parameters define values for blocks in your model. For example, block parameters include channel numbers for multichannel boards, input and output voltage ranges, and sample time. For information about configuring block parameters for Speedgoat I/O modules, see the Speedgoat documentation at www.speedgoat.com/help.

This procedure uses the Simulink model `slrt_ex_osc`. To open this model, in the MATLAB Command Window, type:

```
open_system(fullfile(matlabroot, 'toolbox', 'slrealtime', ...
'examples', 'slrt_ex_osc'))
```

1   In the Simulink Editor, double-click the File Log block.

2   Fill in the File Log dialog box. For example, enter a decimation value to reduce file logging data collection.
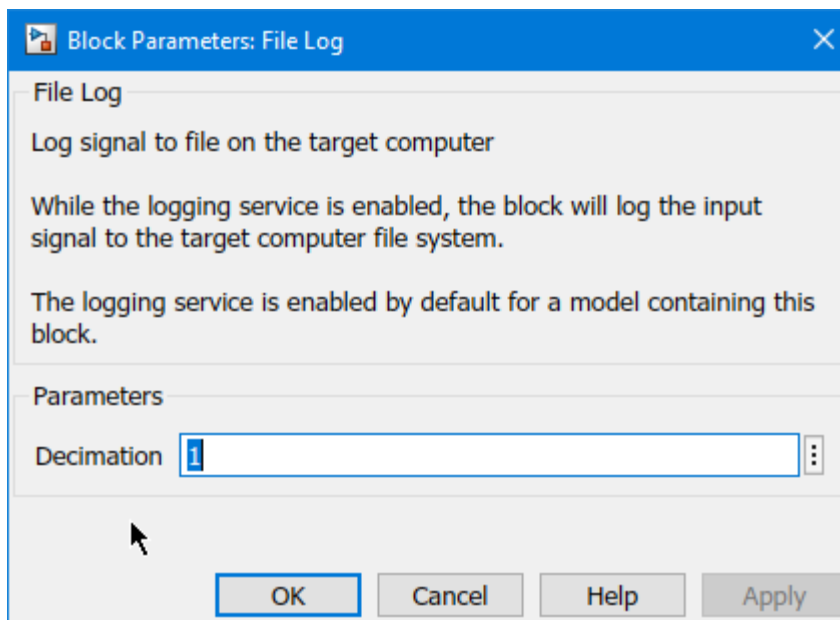
When you change block parameter values through the block parameters dialog box, the changes take effect when you build the real-time application. You can also change parameter values in a real-time application without rebuilding the application. For more information, see "Tunable Block Parameters and Tunable Global Parameters".

See Also

More About

- "I/O Driver Blocks" on page 1-2
- "Add I/O Blocks to Simulink Model" on page 1-4

Async Block Library

Periodic and Nonperiodic Events

About RTOS Tasks and Priorities

Real-time application execution tasks come from an number of model features, including:

- Each sample rate group in the model
- Function-call subsystem block connected to interrupt from an I/O module block
- Function-call subsystem block connected to a Thread Trigger block

The RTOS task scheduler automates scheduling of tasks for all sample rates in the model. You can configure your model to influence task priority of some rate-related tasks. For more information, see "Concurrent Execution on Simulink® Real-Time™".

See Also
Thread Trigger

Related Examples
- "Concurrent Execution on Simulink® Real-Time™"

More About
- "Execution Modes"

Asynchronous Event: Blocks

Thread Trigger

Call downstream function-call subsystem when selected input edge transition occurs
Library: Simulink Real-Time / Async
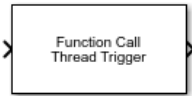
Description

When the selected input edge transition occurs, the Thread Trigger block calls the downstream Function-Call Subsystem block or Triggered Subsystem block and sets the interrupt priority of the task. The block checks for the edge transition at the block sample rate.

If the input transition occurs while the task is running, a CPU overload occurs on the target computer.

Ports

Input

T — Trigger input tested for selected transition
boolean

Detects the edge type that is selected by using the **Trigger Edge** parameter as 1 for detected and 0 for not detected..

Example: 0

Output

F — Function call output for task
function call

Outputs the call to the downstream function and provides the thread scheduling priority for the task

Parameters

Trigger Edge — Selects trigger edge type
Rising Edge (default) | Falling Edge | Both Edges

Selects the trigger edge type that is detected by the block input.

Programmatic Use
Block Parameter: edge

Function-Call Task Priority — Selects the ISR task priority for function
245 (high priority) (default) | 235 (medium priority) | 225 (low priority)

Selects the interrupt service routine task priority for the function call. You can select task priority values from 254 (highest priority) to 193 (lowest priority).

The Thread Trigger block provides means to call a Function-Call Subsystem block or Triggered Subsystem block that is a separate model thread. This thread can run with higher or lower priority than other execution threads. Each rate group in the model executes as a different execution thread. For more information, see "About RTOS Tasks and Priorities" on page 2-2.

Programmatic Use
Block Parameter: `taskpri`

Sample Time — Selects sample rate for edge detection
0 (default) | -1 (inherit)

Select the sample time for the block input to sample for edge detection.

Programmatic Use
Block Parameter: `sampletime`

See Also

Topics
"About RTOS Tasks and Priorities" on page 2-2

Introduced in R2020b

CAN Message Blocks Library

CAN Utility Blocks

CAN Pack

Pack individual signals into CAN message

Library: Vehicle Network Toolbox / CAN Communication
Embedded Coder / Embedded Targets / Host Communication
Embedded Coder Support Package for Texas Instruments
C2000 Processors / Target Communication
Simulink Real-Time / CAN / CAN MSG blocks

Description

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

To use this block, you must have a license for Simulink software.

The CAN Pack block supports:

- Simulink Accelerator™ rapid accelerator mode. You can speed up the execution of Simulink models.
- Model referencing. Your model can include other Simulink models as modular components.

For more information, see "Design Your Model for Effective Acceleration".

Tip

- This block can be used to encode the signals of J1939 parameter groups up to 8 bytes. However, to work with J1939 messages, it is preferable to use the blocks in the J1939 Communication block library instead of this block. See "J1939".

Ports

Input

Data — CAN message signal input
single | double | int8 | int16 | int32 | int64 | uint32 | uint64 | boolean

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

The block supports the following input signal data types: single, double, int8, int16, int32, int64, uint8, uint16, uint32, uint64, and boolean. The block does not support fixed-point data types.

Code generation to deploy models to targets. If your signal information consists of signed or unsigned integers greater than 32 bits long, code generation is not supported.

Output

CAN Msg — CAN message output
CAN_MESSAGE | CAN_MESSAGE_BUS

This block has one output port, CAN Msg. The CAN Pack block takes the specified input signals and packs them into a CAN message. The output data type is determined by the **Output as bus** parameter setting.

Parameters

Data input as — Select your data signal
raw data (default) | manually specified signals | CANdb specified signals

- `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.

  The conversion formula is:

  `raw_value = (physical_value - Offset) / Factor`

  where `physical_value` is the original value of the signal and `raw_value` is the packed signal value.

- `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 block inputs depends on the number of signals you specify.

- `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 block inputs depends on the number of signals specified in the CANdb file for the selected message.

Programmatic Use
Block Parameter: DataFormat
Type: string | character vector
Values: 'raw data' | 'manually specified signals' | 'CANdb specified signals'
Default: 'raw data'

CANdb file — CAN database file
character vector

This option is available if you specify that your data is input through a CANdb file in the **Data is input as** list. Click **Browse** to find the 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.

File names that contain non-alphanumeric characters such as equal signs, ampersands, and so on are not valid CAN database file names. You can use periods in your database name. Before you use the CAN database files, rename them with non-alphanumeric characters.

Programmatic Use
Block Parameter: CANdbFile
Type: string | character vector

Message list — CAN message list
array of character vectors

This option is available if you specify that your data is input through 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.

Programmatic Use
Block Parameter: MsgList
Type: string | character vector

Name — CAN message name
CAN Msg (default) | character vector

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 is not available if you choose to use signals from a CANdb file.

Programmatic Use
Block Parameter: MsgName
Type: string | character vector

Identifier type — CAN identifier type
Standard (11-bit identifier) (default) | Extended (29-bit identifier)

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.

Programmatic Use
Block Parameter: MsgIDType
Type: string | character vector
Values: 'Standard (11-bit identifier)' | 'Extended (29-bit identifier)'
Default: 'Standard (11-bit identifier)'

CAN Identifier — CAN message ID
0 (default) | 0 to 536870911

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 by using the hex2dec function. This option is available if you choose to input raw data or manually specify signals.

Programmatic Use
Block Parameter: MsgIdentifier
Type: string | character vector
Values: '0' to '536870911'

Length (bytes) — CAN message length
8 (default) | 0 to 8

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.

Programmatic Use
Block Parameter: MsgLength
Type: string | character vector
Values: '0' to '8'
Default: '8'

Remote frame — CAN message as remote frame
off (default) | on

Specify the CAN message as a remote frame.

Programmatic Use
Block Parameter: Remote
Type: string | character vector
Values: 'off' | 'on'
Default: 'off'

Output as bus — CAN message as bus
off (default) | on

Select this option for the block to output CAN messages as a Simulink bus signal. For more information on Simulink bus objects, see "Composite Signals".

Programmatic Use
Block Parameter: BusOutput
Type: string | character vector
Values: 'off' | 'on'
Default: 'off'

Add signal — Add CAN signal

Add a new signal to the signal table.

Programmatic Use

None

Delete signal — Remove CAN signal

Remove the selected signal from the signal table.

Programmatic Use

None

Signals — Signals table
table

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

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

If you have selected to specify signals manually, create your signals in this table. Each signal that you create has these 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 these options:

- LE: Where the byte order is in little-endian format (Intel®). In this format you count bits from the least significant bit, to the most significant bit. 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 Number

	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 LSB	19	18	17	16
Byte 3	31	30	29	28	27	26	25	24
Byte 4	39	38	37	36	35 MSB	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 Byte Number

Data begins at the least significant bit and starts at 20.

Data is written 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 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.

Bit Number

	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 Byte Number

MSB

Data is written up to the most significant bit and ends at 11.

LSB

Data begins at the least significant bit and starts at 20.

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 time step:

- `Standard`: The signal is packed at each time step.
- `Multiplexor`: The `Multiplexor` signal, or the mode signal is 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 these signals with the following types and values.

Signal Name	Multiplex Type	Multiplex Value
Signal-A	Standard	Not applicable
Signal-B	Multiplexed	1
Signal-C	Multiplexed	0
Signal-D	Multiplexor	Not applicable

In this example:

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

Multiplex value

This option is available only if you have selected the **Multiplex type** to be `Multiplexed`. The value you provide 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. For more information, see the **Data input as** parameter conversion formula.

Offset

Specify the **Offset** value to apply to convert the physical value (signal value) to the raw value packed in the message. For more information, see the **Data input as** parameter conversion formula.

Min, **Max**

Define a range of signal values. The default settings are `-Inf` (negative infinity) and `Inf`, respectively. For **CANdb specified signals**, these settings are read from the CAN database. For **manually specified signals**, you can specify the minimum and maximum physical value of the signal. By default, these settings do not clip signal values that exceed the settings.

Programmatic Use
Block Parameter: `SignalInfo`
Type: `string` | `character vector`

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

CAN Unpack | CAN FD Pack

Topics
"Design Your Model for Effective Acceleration"

Introduced in R2009a

CAN Unpack

Unpack individual signals from CAN messages

Library: Vehicle Network Toolbox / CAN Communication
                      Embedded Coder / Embedded Targets / Host Communication
                      Embedded Coder Support Package for Texas Instruments
                      C2000 Processors / Target Communication
                      Simulink Real-Time / CAN / CAN MSG blocks

Description

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

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

The CAN Unpack block supports:

- The use of Simulink Accelerator Rapid 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.

For more information on these features, see "Design Your Model for Effective Acceleration".

Tip

- To process every message coming through a channel, it is recommended that you use the CAN Unpack block in a function trigger subsystem. See "Using Triggered Subsystems".
- This block can be used to decode the signals of J1939 parameter groups up to 8 bytes. However, to work with J1939 messages, it is preferable to use the blocks in the J1939 Communication block library instead of this block. See "J1939".

Ports

Input

CAN Msg — CAN message input
CAN_MESSAGE | CAN_MESSAGE_BUS

This block has one input port, **CAN Msg**. The block takes the specified input CAN messages and unpacks their signal data to separate outputs.

The block supports the following signal data types: single, double, int8, int16, int32, int64, uint8, uint16, uint32, uint64, and boolean. The block does not support fixed-point data types.

Code generation to deploy models to targets. Code generation is not supported if your signal information consists of signed or unsigned integers greater than 32 bits long.

Output

Data — CAN signal output
single | double | int8 | int16 | int32 | int64 | uint32 | uint64 | boolean

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

For signals specified manually or by a CANdb, the default output data type for CAN signals is double. To specify other types, use a Signal Specification block. This allows the block to support the following output signal data types: single, double, int8, int16, int32, int64, uint8, uint16, uint32, uint64, and boolean. The block does not support fixed-point types.

Additional output ports can be added by selecting the options in the parameters **Output ports** pane. For more information, see the parameters `Output identifier`, `Output timestamp`, `Output error`, `Output remote`, `Output length`, and `Output status`.

Parameters

Data to output as — Select your data signal
raw data (default) | manually specify signals | CANdb specified signals

- `raw data`: Output data as a uint8 vector array. If you select this option, you specify only the message fields. The other signal parameter fields are unavailable. This option opens only one output port on your block.

  The conversion formula is:

  physical_value = raw_value * Factor + Offset

  where `raw_value` is the unpacked signal value and `physical_value` is the scaled signal value.
- `manually specified signals`: You can specify data signals. If you select this option, use the `Signals` table to create your signals message manually. The number of output ports on your block depends on the number of signals that you specify. For example, if you specify four signals, your block has four output ports.
- `CANdb specified signals`: You can specify a CAN database file that contains data signals. If you select this option, select a CANdb file. 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.

Programmatic Use
Block Parameter: DataFormat
Type: string | character vector
Values: 'raw data' | 'manually specified signals' | 'CANdb specified signals'
Default: 'raw data'

CANdb file — CAN database file
character vector

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 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. File names that contain non-alphanumeric characters such as equal signs, ampersands, and so forth are not valid CAN database file names. You can use periods in your database name. Rename CAN database files with non-alphanumeric characters before you use them.

Programmatic Use
Block Parameter: `CANdbFile`
Type: `string` | `character vector`

Message list — CAN message list
array of character vectors

This option is available if you specify in the **Data to be output as** list that your data is to be output as a CANdb file 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.

Programmatic Use
Block Parameter: `MsgList`
Type: `string` | `character vector`

Name — CAN message name
CAN Msg (default) | character vector

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.

Programmatic Use
Block Parameter: `MsgName`
Type: `string` | `character vector`

Identifier type — CAN identifier type
Standard (11-bit identifier) (default) | Extended (29-bit identifier)

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.

Programmatic Use
Block Parameter: `MsgIDType`
Type: `string` | `character vector`
Values: `'Standard (11-bit identifier)'` | `'Extended (29-bit identifier)'`
Default: `'Standard (11-bit identifier)'`

CAN Identifier — CAN message identifier
0 (default) | 0 to 536870911

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 the messages that match the length specified for the message. You can also specify hexadecimal values by using the `hex2dec` function. This option is available if you choose to output raw data or manually specify signals.

Programmatic Use
Block Parameter: `MsgIdentifier`

Type: string | character vector
Values: '0' to '536870911'

Length (bytes) — CAN message length
8 (default) | 0 to 8

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. Otherwise, this field defaults to 8. This option is available if you choose to output raw data or manually specify signals.

Programmatic Use
Block Parameter: MsgLength
Type: string | character vector
Values: '0' to '8'
Default: '8'

Add signal — Add CAN signal

Add a signal to the signal table.

Programmatic Use

None

Delete signal — Remove CAN signal

Remove the selected signal from the signal table.

Programmatic Use

None

Signals — Signals table
table

If you choose to specify signals manually or define signals by using a CANdb file, this table appears.

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

If you have selected to specify signals manually, create your signals manually in this table. Each signal that you create has these 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 these options:

- LE: Where the byte order is in little-endian format (Intel). In this format you count bits from the least-significant bit to the most-significant bit. 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 Number

	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

LSB

Data begins at the least significant bit and starts at 20.

MSB

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

Data Byte Number

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 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.

Bit Number

	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 Byte Number

MSB

Data is written up to the most significant bit and ends at 11.

LSB

Data begins at the least significant bit and starts at 20.

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 time step:

- `Standard`: The signal is unpacked at each time step.
- `Multiplexor`: The `Multiplexor` signal or the mode signal is 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, a message has four signals with these values.

Signal Name	Multiplex Type	Multiplex Value
Signal-A	Standard	Not applicable
Signal-B	Multiplexed	1
Signal-C	Multiplexed	0
Signal-D	Multiplexor	Not applicable

In this example:

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

Multiplex value

This option is available only if you have selected the **Multiplex type** to be `Multiplexed`. The value you provide 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). For more information, see the **Data input as** parameter conversion formula.

Offset

Specify the **Offset** value applied to convert the physical value (signal value) to the unpacked raw value. For more information, see the **Data input as** parameter conversion formula.

Min, **Max**

Define a range of raw signal values. The default settings are `-Inf` (negative infinity) and `Inf`, respectively. For **CANdb specified signals**, these settings are read from the CAN database. For **manually specified signals**, you can specify the minimum and maximum physical value of the signal. By default, these settings do not clip signal values that exceed them.

Programmatic Use
Block Parameter: `SignalInfo`
Type: `string | character vector`

Output identifier — Add CAN ID output port
`off` (default)

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

Programmatic Use
Block Parameter: `IDPort`
Type: `string | character vector`
Values: `'off' | 'on'`
Default: `'off'`

`Output timestamp` — Add Timestamp output port
off (default) | on

Select this option to output the message timestamp. This value indicates when the message was received, measured as the number of seconds elapsed since the model simulation began. This option adds a new output port to the block. The data type of this port is `double`.

Programmatic Use
Block Parameter: `TimestampPort`
Type: `string | character vector`
Values: `'off' | 'on'`
Default: `'off'`

`Output error` — Add Error output port
off (default) | on

Select this option to output the message error status. This option adds a new output port to the block. An output value of `1` on this port indicates that the incoming message is an error frame. If the output value is `0`, there is no error. The data type of this port is `uint8`.

Programmatic Use
Block Parameter: `ErrorPort`
Type: `string | character vector`
Values: `'off' | 'on'`
Default: `'off'`

`Output remote` — Add Remote output port
off (default) | on

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`.

Programmatic Use
Block Parameter: `RemotePort`
Type: `string | character vector`
Values: `'off' | 'on'`
Default: `'off'`

`Output length` — Add Length output port
off (default) | on

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`.

Programmatic Use
Block Parameter: `LengthPort`
Type: `string | character vector`
Values: `'off' | 'on'`
Default: `'off'`

Output status — Add Status output port
off (default) | on

Select this option to output the message received status. The status is 1 if the block receives a 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.

Programmatic Use
Block Parameter: StatusPort
Type: string | character vector
Values: 'off' | 'on'
Default: 'off'

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also
CAN Pack | CAN FD Unpack

Topics
"Design Your Model for Effective Acceleration"

Introduced in R2009a

CAN FD Pack

Pack individual signals into message for CAN FD bus
Library: Vehicle Network Toolbox / CAN FD Communication
                            Embedded Coder Support Package for Texas Instruments
                            C2000 Processors / Target Communication
                            Simulink Real-Time / CAN / CAN-FD MSG blocks

Description

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

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

The CAN FD Pack block supports:

- The use of Simulink Accelerator mode. Using this feature, you can speed up the execution of Simulink models. For more information, see "Design Your Model for Effective Acceleration".

Tip

- To work with J1939 messages, use the blocks in the J1939 Communication block library instead of this block. See "J1939".

Ports

Input

`Data` — CAN FD message signal input
single | double | int8 | int16 | int32 | int64 | uint32 | uint64 | boolean

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

Code generation to deploy models to targets. Code generation is not supported if your signal information consists of signed or unsigned integers greater than 32 bits long.

Output

`Msg` — CAN FD message output
CAN_FD_MESSAGE_BUS

This block has one output port, Msg. The CAN FD Pack block takes the specified input signals and packs them into a CAN FD message, output as a Simulink CAN_FD_MESSAGE_BUS signal. For more information on Simulink bus objects, see "Composite Signals".

Parameters

`Data input as` — Select your data signal
`raw data` (default) | `manually specified signals` | `CANdb specified signals`

- `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.

  The conversion formula is:

  `raw_value = (physical_value - Offset) / Factor`

  where `physical_value` is the original value of the signal and `raw_value` is the packed signal value.

- `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 block inputs depends on the number of signals you specify.

- `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 block inputs depends on the number of signals specified in the CANdb file for the selected message.

Programmatic Use
Block Parameter: `DataFormat`
Type: `string` | `character vector`
Values: `'raw data'` | `'manually specified signals'` | `'CANdb specified signals'`
Default: `'raw data'`

`CANdb file` — CAN database file
character vector

This option is available if you specify that your data is input through a CANdb file in the **Data is input as** list. Click **Browse** to find the 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. File names that contain non-alphanumeric characters such as equal signs, ampersands, and so on are not valid CAN database file names. You can use periods in your database name. Before you use the CAN database files, rename them with non-alphanumeric characters.

Programmatic Use
Block Parameter: `CANdbFile`
Type: `string` | `character vector`

`Message list` — CAN message list
array of character vectors

This option is available if you specify that your data is input through 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.

Programmatic Use
Block Parameter: `MsgList`
Type: `string` | `character vector`

Name — CAN FD message name
CAN Msg (default) | character vector

Specify a name for your CAN FD message. The default is CAN Msg. This option is available if you choose to input raw data or manually specify signals. This option is not available if you choose to use signals from a CANdb file.

Programmatic Use
Block Parameter: MsgName
Type: string | character vector

Protocol mode — CAN FD message protocol
CAN FD (default) | CAN

Specify the message protocol mode.

Programmatic Use
Block Parameter: ProtocolMode
Type: string | character vector
Values: 'CAN FD' | 'CAN'
Default: 'CAN FD'

Identifier type — CAN identifier type
Standard (11-bit identifier) (default) | Extended (29-bit identifier)

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.

Programmatic Use
Block Parameter: MsgIDType
Type: string | character vector
Values: 'Standard (11-bit identifier)' | 'Extended (29-bit identifier)'
Default: 'Standard (11-bit identifier)'

Identifier — Message identifier
0 (default) | 0 .. 536870911

Specify your 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 by using the hex2dec function. This option is available if you choose to input raw data or manually specify signals.

Programmatic Use
Block Parameter: MsgIdentifier
Type: string | character vector
Values: '0' to '536870911'

Length (bytes) — CAN FD message length
8 (default) | 0 to 64

Specify the length of your message. For CAN messages the value can be 0 to 8 bytes; for CAN FD the value can be 0 to 8, 12, 16, 20, 24, 32, 48, or 64 bytes. If you are using CANdb specified signals for your data input, the CANdb file defines the length of your message. This option is available if you choose to input raw data or manually specify signals.

Programmatic Use
Block Parameter: `MsgLength`
Type: `string | character vector`
Values: `'0'` to `'8'`, `'12'`, `'16'`, `'20'`, `'24'`, `'32'`, `'48'`, `'64'`
Default: `'8'`

`Remote frame` — CAN message as remote frame
`off` (default) | `on`

(Disabled for CAN FD protocol mode.) Specify the CAN message as a remote frame.

Programmatic Use
Block Parameter: `Remote`
Type: `string | character vector`
Values: `'off'` | `'on'`
Default: `'off'`

`Bit Rate Switch (BRS)` — Enable bit rate switch
`off` (default) | `on`

(Disabled for CAN protocol mode.) Enable bit rate switch.

Programmatic Use
Block Parameter: `BRSSwitch`
Type: `string | character vector`
Values: `'off'` | `'on'`
Default: `'off'`

`Add signal` — Add CAN FD signal

Add a signal to the signal table.

Programmatic Use

None

`Delete signal` — Remove CAN FD signal

Remove the selected signal from the signal table.

Programmatic Use

None

`Signals` — Signals table
table

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

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

If you have selected to specify signals manually, create your signals in this table. Each signal that you create has these 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. For CAN the start bit must be an integer from 0 through 63, for CAN FD 0 through 511, within the number of bits in the message. (Note that message length is specified in bytes.)

Length (bits)

Specify the number of bits the signal occupies in the message. The length must be an integer from 1 through 64. The sum of all the signal lengths in a message is limited to the number of bits in the message length; that is, all signals must cumulatively fit within the length of the message. (Note that message length is specified in bytes and signal length in bits.)

Byte order

Select either of these options:

*   LE: Where the byte order is in little-endian format (Intel). In this format you count bits from the least significant bit, to the most significant bit. 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 Number

	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 Byte Number

LSB

Data begins at the least significant bit and starts at 20.

MSB

Data is written 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 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.

Bit Number

	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
					MSB			
Byte 2	23	22	21	20	19	18	17	16
				LSB	Data is written up to the most significant bit and ends at 11.			
Byte 3	31	30	29	28	27	26	25	24
	Data begins at the least significant bit and starts at 20.							
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 Byte Number

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`

Note: If you have a `double` signal that does not align exactly to the message byte boundaries, to generate code with Embedded Coder® you must check **Support long long** under **Device Details** in the **Hardware Implementation** pane of the Configuration Parameters dialog.

Multiplex type

Specify how the block packs the signals into the message at each time step:

- `Standard`: The signal is packed at each time step.
- `Multiplexor`: The `Multiplexor` signal, or the mode signal is 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 these types and values.

Signal Name	Multiplex Type	Multiplex Value
Signal-A	Standard	Not applicable
Signal-B	Multiplexed	1
Signal-C	Multiplexed	0
Signal-D	Multiplexor	Not applicable

In this example:

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

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 the **Data input as** parameter conversion formula 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 the **Data input as** parameter conversion formula to understand how physical values are converted to raw values packed into a message.

Min, **Max**

Define a range of signal values. The default settings are `-Inf` (negative infinity) and `Inf`, respectively. For **CANdb specified signals**, these settings are read from the CAN database. For

manually specified signals, you can specify the minimum and maximum physical value of the signal. By default, these settings do not clip signal values that exceed them.

Programmatic Use
Block Parameter: `SignalInfo`
Type: `string` | `character vector`

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

Blocks
CAN FD Unpack | CAN Pack

Topics
"Design Your Model for Effective Acceleration"
"Composite Signals"

Introduced in R2018a

CAN FD Unpack

Unpack individual signals from CAN FD messages

Library: Vehicle Network Toolbox / CAN FD Communication

                              Embedded Coder Support Package for Texas Instruments
C2000 Processors / Target Communication

                              Simulink Real-Time / CAN / CAN-FD MSG blocks

Description

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

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

The CAN FD Unpack block supports:

- Simulink Accelerator mode. You can speed up the execution of Simulink models. For more information, see "Design Your Model for Effective Acceleration".

Tip

- To process every message coming through a channel, it is recommended that you use the CAN FD Unpack block in a function trigger subsystem. See "Using Triggered Subsystems".

- To work with J1939 messages, use the blocks in the J1939 Communication block library instead of this block. See "J1939".

Ports

Input

`Msg` — CAN FD message input
`CAN_FD_MESSAGE_BUS`

This block has one input port, Msg. The CAN FD Unpack block takes the specified input CAN messages and unpacks their signal data to separate outputs.

The block supports the following input signal data types: single, double, int8, int16, int32, int64, uint8, uint16, uint32, uint64, and boolean. The block does not support fixed-point data types.

Code generation to deploy models to targets. Code generation is not supported if your signal information consists of signed or unsigned integers greater than 32 bits long.

Output

Data — CAN message output
single | double | int8 | int16 | int32 | int64 | uint32 | uint64 | boolean

The CAN FD Unpack block has one output port by default. The number of data 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, labeled by signal name.

For manually or CANdb specified signals, the default output signal data type is double. To specify other types, use a Signal Specification block. This allows the block to support the following output signal data types: single, double, int8, int16, int32, int64, uint8, uint16, uint32, uint64, and boolean. The block does not support fixed-point types.

Additional output ports can be added by the options in the parameters **Output ports** pane.

Parameters

Data to output as — Select your data signal
raw data (default) | manually specify signals | CANdb specified signals

- `raw data`: Output data as a uint8 vector array. If you select this option, you specify only the message fields. The other signal parameter fields are unavailable. This option opens only one output port on your block.

  The conversion formula is:

  `physical_value = raw_value * Factor + Offset`

  where `raw_value` is the unpacked signal value and `physical_value` is the scaled signal value.
- `manually specified signals`: You can specify data signals. If you select this option, use the `Signals` table to create your signals message manually. The number of output ports on your block depends on the number of signals that you specify. For example, if you specify four signals, your block has four output ports.
- `CANdb specified signals`: You can specify a CAN database file that contains data signals. If you select this option, select a CANdb file. 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.

Programmatic Use
Block Parameter: DataFormat
Type: string | character vector
Values: 'raw data' | 'manually specified signals' | 'CANdb specified signals'
Default: 'raw data'

CANdb file — CAN database file
character vector

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 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 the **Signals** table. File names that contain non-alphanumeric characters such as equal signs, ampersands, and so forth, are not valid CAN database file names. You

can use periods in your database name. Rename CAN database files with non-alphanumeric characters before you use them.

Programmatic Use
Block Parameter: CANdbFile
Type: string | character vector

Message list — Message list
array of character vectors

This option is available if you specify in the **Data to be output as** list that your data is to be output as a CANdb file 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.

Programmatic Use
Block Parameter: MsgList
Type: string | character vector

Name — Message name
CAN Msg (default) | character vector

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

Programmatic Use
Block Parameter: MsgName
Type: string | character vector

Identifier type — Identifier type
Standard (11-bit identifier) (default) | Extended (29-bit identifier)

Specify whether your 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.

Programmatic Use
Block Parameter: MsgIDType
Type: string | character vector
Values: 'Standard (11-bit identifier)' | 'Extended (29-bit identifier)'
Default: 'Standard (11-bit identifier)'

Identifier — Message identifier
0 (default) | 0 .. 536870911

Specify your 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 the 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.

Programmatic Use
Block Parameter: MsgIdentifier
Type: string | character vector
Values: '0' to '536870911'

Length (bytes) — CAN message length
8 (default) | 0 .. 8

Specify the length of your message. For CAN messages the value can be 0-8 bytes; for CAN FD the value can be 0-8, 12, 16, 20, 24, 32, 48, or 64 bytes. If you are using `CANdb specified signals` for your output data, the CANdb file defines the length of your message. This option is available if you choose to output raw data or manually specify signals.

Programmatic Use
Block Parameter: MsgLength
Type: string | character vector
Values: '0' to '8', '12', '16', '20', '24', '32', '48', '64'
Default: '8'

Add signal — Add CAN signal

Add a signal to the signal table.

Programmatic Use

None

Delete signal — Remove CAN signal

Remove the selected signal from the signal table.

Programmatic Use

None

Signals — Signals table
table

If you choose to specify signals manually or define signals by using a CANdb file, this table appears.

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

If you have selected to specify signals manually, create your signals manually in this table. Each signal that you create has these 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. For CAN the start bit must be an integer from 0 through 63, for CAN FD 0 through 511, within the number of bits in the message. (Note that message length is specified in bytes.)

Length (bits)

Specify the number of bits the signal occupies in the message. The length must be an integer from 1 through 64. The sum of all the signal lengths in a message is limited to the number of bits in the message length; that is, all signals must cumulatively fit within the length of the message. (Note that message length is specified in bytes and signal length in bits.)

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 least-significant bit to the most-significant bit and proceeding to the next higher byte as you cross a byte boundary. 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 Number

	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 LSB	19	18	17	16
Byte 3	31	30	29	28	27	26	25	24 MSB
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 Byte Number

Data begins at the least significant bit and starts at 20.

Data is written 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 the byte order is in big-endian format (Motorola). In this format you count bits from the least-significant bit to the most-significant bit and proceeding to the next lower byte as you

cross a byte boundary. 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.

Bit Number

	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 Byte Number

MSB

Data is written up to the most significant bit and ends at 11.

LSB

Data begins at the least significant bit and starts at 20.

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`

Note: If you have a `double` signal that does not align exactly to the message byte boundaries, to generate code with Embedded Coder you must check **Support long long** under **Device Details** in the **Hardware Implementation** pane of the Configuration Parameters dialog.

Multiplex type

Specify how the block unpacks the signals from the message at each time step:

- `Standard`: The signal is unpacked at each time step.
- `Multiplexor`: The `Multiplexor` signal, or the mode signal is 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, a message has four signals with these values.

Signal Name	Multiplex Type	Multiplex Value
Signal-A	Standard	Not applicable
Signal-B	Multiplexed	1
Signal-C	Multiplexed	0
Signal-D	Multiplexor	Not applicable

In this example:

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

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). For more information, see the **Data input as** parameter conversion formula.

Offset

Specify the **Offset** value applied to convert the physical value (signal value) to the unpacked raw value. For more information, see the **Data input as** parameter conversion formula.

Min, **Max**

Define a range of raw signal values. The default settings are `-Inf` (negative infinity) and `Inf`, respectively. For **CANdb specified signals**, these settings are read from the CAN database. For **manually specified signals**, you can specify the minimum and maximum physical value of the signal. By default, these settings do not clip signal values that exceed them.

Programmatic Use
Block Parameter: `SignalInfo`
Type: `string | character vector`

`Output identifier` — Add CAN ID output port

`off` (default) | `on`

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

Programmatic Use
Block Parameter: `IDPort`
Type: `string | character vector`
Values: `'off' | 'on'`
Default: `'off'`

`Output timestamp` — Add Timestamp output port

`off` (default) | `on`

Select this option to output the message timestamp. This value indicates when the message was received, measured as the number of seconds elapsed since the model simulation began. This option adds a new output port to the block. The data type of this port is `double`.

Programmatic Use
Block Parameter: `TimestampPort`
Type: `string | character vector`
Values: `'off' | 'on'`
Default: `'off'`

`Output error` — Add Error output port

`off` (default) | `on`

Select this option to output the message error status. This option adds a new output port to the block. An output value of `1` on this port indicates that the incoming message is an error frame. If the output value is `0`, there is no error. The data type of this port is `uint8`.

Programmatic Use
Block Parameter: `ErrorPort`
Type: `string | character vector`
Values: `'off' | 'on'`
Default: `'off'`

`Output remote` — Add Remote output port

`off` (default) | `on`

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`.

Programmatic Use
Block Parameter: `RemotePort`
Type: `string | character vector`
Values: `'off' | 'on'`
Default: `'off'`

`Output length` — Add Length output port

`off` (default) | `on`

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`.

Programmatic Use
Block Parameter: `LengthPort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Output status — Add Status output port
`off` (default) | `on`

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`.

Programmatic Use
Block Parameter: `StatusPort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Output Bit Rate Switch (BRS) — Add BRS output port
`off` (default) | `on`

(Disabled for CAN protocol.) Select this option to output the message bit rate switch. This option adds a new output port to the block. The data type of this port is `boolean`.

Programmatic Use
Block Parameter: `BRSPort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Output Error Status Indicator (ESI) — Add ESI output port
`off` (default) | `on`

(Disabled for CAN protocol.) 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 `boolean`.

Programmatic Use
Block Parameter: `ESIPort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Output Data Length Code (DLC) — Add DLC output port
`off` (default) | `on`

(Disabled for CAN protocol.) Select this option to output the message data length. This option adds a new output port to the block. The data type of this port is `double`.

Programmatic Use
Block Parameter: `DLCPort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`

Default: `'off'`

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also

Blocks
CAN FD Pack

Topics
"Design Your Model for Effective Acceleration"
"Composite Signals"

Introduced in R2018a

EtherCAT Blocks Library

5

Model-Based EtherCAT Communications Support

Modeling EtherCAT Networks

Ethernet for Control Automation (EtherCAT) is an open Ethernet network protocol for real-time distributed control, for example in automotive and industrial systems. The EtherCAT protocol provides:

- Deterministic and fast cycle times
- Inexpensive I/O module cost

EtherCAT networks consist of one master node and several slave nodes. The Simulink Real-Time EtherCAT sublibrary supports only the master node of an EtherCAT network. You cannot emulate slave nodes by using the blocks in the EtherCAT sublibrary. You can use these blocks to prototype multiple EtherCAT networks by using multiple Ethernet cards.

You model an EtherCAT network by using one of the third-party EtherCAT configurators: TwinCAT® 3 from Beckhoff® or EC-Engineer from Acontis.

To map the network model into a Simulink Real-Time model, become familiar with these mappings:

- "EtherCAT Protocol Mapping" on page 5-21
- "EtherCAT Configurator Component Mapping" on page 5-22

Blocks and Tasks

At a minimum, each EtherCAT model must contain an EtherCAT Init block. The EtherCAT Init block contains a reference to an EtherCAT Network Information (ENI) file. The ENI file describes the network, including the device variables of the network.

If you generate the configuration file by using TwinCAT 3, use the software to create at least one cyclic input/output task. Link this task to at least one input channel and one output channel on each slave device. If you generate the file by using Acontis EC-Engineer, the software creates one default task linked to all slave device input/output channels. The task rate equals the sample time of the EtherCAT blocks.

When you know the input/output cycle ticks, in the Model Configuration Parameters dialog box, set the **Fixed-step size** to a value that is consistent with these constraints:

- The cycle tick of all EtherCAT slave devices.
- The sample times of all other blocks in the Simulink model.

For more information, see "Fixed-Step Size Derivation" on page 5-20.

When you know the device variables that you are using in your model, add an EtherCAT PDO Receive or EtherCAT PDO Transmit block for every EtherCAT device variable. When you add these blocks to the model, the block obtains the list of device variables from the configuration file in the EtherCAT Init block. When you specify a device variable in the block dialog box, the software updates the block information with device variable information from the configuration file.

To transmit CANopen over EtherCAT (CoE) information through your network, add SDO/CoE blocks to your model. To transmit by using the SERCOS (SErial Real time COmmunication Specification) over EtherCAT (SoE) interface through your network, add SSC/SoE blocks to your model.

The SDO/CoE blocks and SSC/SoE block s come in two types, synchronous and asynchronous. From the EtherCAT perspective, there is little difference in the behavior of these types. The difference

occurs during the execution of the real-time application. The synchronous blocks halt execution while the blocks wait for a response. The asynchronous blocks continue executing and poll the I/O module for a response.

To avoid a CPU overload, set the sample time for the synchronous blocks to a value at least three times of that for the PDO blocks.

To track the state of the network or force the network into a particular state, add an EtherCAT Get State or EtherCAT Set State block.

Order of Network Events

The EtherCAT Init block schedules network events in *Phase 1* and *Phase 2*.

Phase 1

1 Read data from EtherCAT variables from the last received frame into EtherCAT PDO Receive blocks.

2 Use either of these blocks in any order:

- EtherCAT PDO Receive — Processes data read from the last frame received from a slave device.
- EtherCAT PDO Transmit — Buffers data to send in the next frame to a slave device.

3 Use each of these blocks in any order. Synchronous upload and download take at least three ticks of the fastest PDO cycle tick to complete processing.

- EtherCAT Sync SDO Upload — Queues an SDO frame with new value, waits for response.
- EtherCAT Sync SDO Download — Queues an SDO frame with request for data, waits for response.
- EtherCAT Async SDO Upload — Queues an SDO frame with new value, checks for response, continues execution.
- EtherCAT Async SDO Download — Queues an SDO frame with request for data, checks for response, continues execution.
- EtherCAT Sync SSC/SoE Upload — Queues an SSC/SoE frame with new value, waits for response.
- EtherCAT Sync SSC/SoE Download — Queues an SSC/SoE frame with request for data, waits for response.
- EtherCAT Async SSC/SoE Upload — Queues an SSC/SoE frame with new value, checks for response, continues execution.
- EtherCAT Async SSC/SoE Download — Queues an SSC/SoE frame with request for data, checks for response, continues execution.
- EtherCAT Get State — Reads current state of EtherCAT network.
- EtherCAT Set State — Queues request to change current state of EtherCAT network.

Phase 2

1 Send the PDO frames followed by the next available queued SDO frames.

See Also

EtherCAT Init | EtherCAT PDO Receive | EtherCAT PDO Transmit | EtherCAT Sync SDO Upload | EtherCAT Sync SDO Download | EtherCAT Async SDO Upload | EtherCAT Async SDO Download | EtherCAT Sync SSC/SoE Upload | EtherCAT Sync SSC/SoE Download | EtherCAT Async SSC/SoE Upload | EtherCAT Async SSC/SoE Download | EtherCAT Get State | EtherCAT Set State

More About

- "Fixed-Step Size Derivation" on page 5-20
- "EtherCAT Protocol Mapping" on page 5-21
- "EtherCAT Configurator Component Mapping" on page 5-22
- "EtherCAT Data Types" on page 5-23
- "EtherCAT Init Block DC Error Values" on page 5-24

Install EtherCAT Network Tools TwinCAT or EC-Engineer

To configure and diagnose your EtherCAT network, install third-party EtherCAT software. Example vendors and products are:

- acontis technologies GmbH, EC-Engineer
- Beckhoff Automation GmbH, TwinCAT 3

For TwinCAT 3 requirements, see "Hardware Setup Requirements for TwinCAT 3" on page 5-6. To install the TwinCAT 3 network and configuration software:

1 Install a dedicated, EtherCAT compatible Ethernet card on the development computer.

2 Download or buy the Beckhoff TwinCAT 3 configurator (www.beckhoff.com).

3 Install Microsoft® Visual Studio® on your development computer.

   TwinCAT 3 uses the Microsoft Visual Studio IDE desktop as its user interface. For the required version, see the TwinCAT 3 documentation.

4 Install the TwinCAT 3 software on your development computer.

The next task is "Configure EtherCAT Network by Using TwinCAT 3" on page 5-7.

See Also

External Websites

- www.beckhoff.com
- www.acontis.com/en/

Hardware Setup Requirements for TwinCAT 3

For the development and target computers, the EtherCAT I/O module has these requirements:

- Each Ethernet card must be compatible with EtherCAT communication.
- On the development computer, as a best practice, install two Ethernet cards in addition to your local area network card. Dedicate one card to linking the development and target computers. Dedicate the other card to EtherCAT network configuration.

  Assign each Ethernet card a static IP address and a nonroutable subnet and netmask. You assign an IP address on the development computer extra Ethernet port that connects to the target EtherCAT slave network by using a switch. For information on setting up the dedicated Ethernet card, see your network administrator.

  Configure the development computer Ethernet card that you are using for EtherCAT to enable only the Internet Protocol Version 4 (TCP/IPv4) driver. For information on creating an EtherCAT configuration file, see the TwinCAT 3 documentation.

- On the target computer, consult with Speedgoat support for configuration of the two Ethernet cards in the Speedgoat target machine. A typical configuration dedicates one card to linking the development and target computers and dedicates the other card to model-based EtherCAT communication. See "Install EtherCAT Network for Execution" on page 5-11.

With only one card on the development computer, before configuring the EtherCAT network, unplug the Ethernet link cable and plug in the EtherCAT network cable. Before building and downloading the model, unplug the EtherCAT network cable, plug in the Ethernet link cable, disable the EtherCAT filter, and restart your development computer.

Configure EtherCAT Network by Using TwinCAT 3

Before you start this procedure, familiarize yourself with TwinCAT 3 and its documentation.

Before configuring the network, perform the steps in "Install EtherCAT Network Tools TwinCAT or EC-Engineer" on page 5-5.

Scan EtherCAT Network

This example uses an EtherCAT network that consists of Beckhoff EK1100, EL3062, and EL4002 modules connected in that order.

To scan an EtherCAT network by using TwinCAT 3:

1   Connect your EtherCAT network to the development computer Ethernet port dedicated to EtherCAT. Turn on the network.

2   Start Microsoft Visual Studio and create a TwinCAT 3 project.

3   In the TwinCAT menu, start the device scanner.

The scanner reports that new I/O devices have been found.

4   In the list of Ethernet devices that the scanner detects on the development computer, select the Ethernet device into which you plugged your EtherCAT network.

If you do not see an Ethernet device identified as an EtherCAT device, check your EtherCAT network configuration and power supply.

5   Scan for EtherCAT boxes on your network.

The scanner reports the EtherCAT devices on your network.

6   Disable free run mode.

7   In your TwinCAT project, make sure that the scanner downloaded the required information about your EtherCAT devices. If not a Beckhoff device, you could need the EtherCAT Slave Information (ESI) file from the device vendor.

Configure EtherCAT Master Node Data

Before configuring the master node of an EtherCAT network, scan the network by using TwinCAT.

Create EtherCAT Task

To create and configure an EtherCAT task:

1   In TwinCAT 3, add an item to your system task list.

In the Solution Explorer display tree, open the SYSTEM subtree and use a right-click on the Tasks entry and choose **Add New Item**. In the new dialog, select **TwinCAT Task With Image**. This allows you to add variables to the task PDO.

Provide a name for the task, for example `Task 1`. Configure `Task 1` as a task with an image.

2   In the task list, select `Task 1` and set its cycle ticks value to a value in milliseconds, such as `1` for 1 millisecond.

3 Record the cycle tick in milliseconds.

In the Model Configuration Parameters dialog box, use the cycle tick to calculate a value for the **Fixed-step size (fundamental sample time)** box. To enable Simulink to calculate the sample time, select `Auto`.

Configure EtherCAT Task Inputs

To configure the task inputs:

1 In TwinCAT 3, under `Term 1`, access the nodes `Term 2` and `AI Standard Channel 1`.
2 Drag the `Value` node of `AI Standard Channel 1` to the `Task 1` inputs.
3 Configure the `Term 1` inputs as variables.
4 Link the `AI Standard Channel 1` variable to `Term 2`.

Adding a variable to the task you created requires that you:

1 Add an empty variable to the task with the same type as the PDO variable you want to add.
2 Link it to the PDO variable that you want to add to the task from that task entry using the **Linked to...** button.

Adding any one variable from a specific slave device adds all PDO variables from that slave to the task. For instance, with the EL3102 Analog Input module:

1 Left-click on your task Inputs entry and select **Add New Item**. The **Insert Variable** dialog opens.
2 Select the data type. For the EL3102 the AD values is an INT in the list. That is a 2 byte signed integer.
3 Change the name if needed, but this is not necessary.
4 Click on OK
5 A new dialog opens, click on **Linked to...**.
6 In the new dialog, find the EL3102. Only variables with the data type selected above are visible. Both INT and UINT appear. Select any one of the EL3102 variables. You may need to change with check boxes are active under the **Show Variables** or **Show Variable Types** lists.
7 All of the EL3102 transmit (Input to master stack) variables are now included in the task just by selecting one of them.
8 Repeat for one receive (Output to slave) variable for that slave.

Configure EtherCAT Task Outputs

To configure the task outputs:

1 In TwinCAT 3, under `Term 1`, access the nodes `Term 3` and `AO Outputs Channel 1`.
2 Drag the `Analog output` node of `AO Outputs Channel 1` to the `Task 1` outputs.
3 Configure the `Term 1` analog outputs as variables.
4 Link the `Analog output` variable to `Term 3`.

Configure EtherCAT Distributed Clocks

To configure the `Term 3` distributed clock:

1   In TwinCAT 3, under `Term 3`, access the DC tab.

2   Change the DC operation mode to `DC Synchron`.

There are two main steps to configure distributed clocks:

1   Select the synchronization mode, either Master shift or bus shift. TwinCAT refers to these with TwinCAT centric names, not generic master stack names.

2   In the Solution Explorer, select **I/O > Devices > Device 1 (EtherCAT)**

3   In the dialog in the right side, select the **EtherCAT** tab. Select the **Advanced Settings** button. On the left of the new dialog, select **Distributed Clocks**.

4   By default TwinCAT3 has **Automatic DC Mode Selection** chosen. Deselect that and choose **DC in use**. Choose the mode you want.

5   **Independent DC Time (Master Mode)** causes the target machine clock to be adjusted to synchronize with the first DC enabled EtherCAT slave device. This mode is also known as Master Shift DC mode.

6   **DC Time controlled by TwinCAT Time (Slave Mode)** uses the target computer execution time as the reference clock and to adjust the first DC enabled slave to match the target computer. This is also known as Bus Shift mode.

For each DC enabled slave device, you need to ensure that it is configured correctly to participate in DC synchronization. For each slave:

1   Select the slave in Solution Explorer.

2   In the dialog, select the **DC** tab if it is available.

3   In the Operation Mode drop down menu, there could be several different names given. For instance `DC Latch` or `DC Synchron` are common and mean that the device synchronizes and uses DC timing. `SM synchron` is a common listing to mean that IO is not DC synchronized, but occurs on packet arrival (SM), not on DC time.

4   Click the **Advanced Settings** button.

5   Make sure the **Enable** checkbox is selected. There are additional settings that can be modified, but these are generally advanced options.

Export and Save EtherCAT Configuration by Using TwinCAT 3

The EtherCAT Network Information (ENI) file represents the master node of an EtherCAT network. To create the ENI file, scan and configure the network by using TwinCAT 3.

To export the ENI file from TwinCAT 3:

1   Under the **Device 1 (EtherCAT)** node, in the **EtherCAT** tab, execute the command to export the configuration file.

2   In the file save dialog box, enter an XML file name, such as `BeckhoffAIOconfig.xml`.

Caution The ENI file is formatted as an XML file with the `.xml` file extension. Building the real-time application produces an XML file with the same name as your model. To avoid a conflict, use an ENI file name that is different from the name of your model.

3   When you close Microsoft Visual Studio TwinCAT the project file is saved.

To review or modify your configuration, open the project SLN file by using Microsoft Visual Studio. If you modify the configuration, save both the XML and SLN files.

The next task is "Install EtherCAT Network for Execution" on page 5-11.

Install EtherCAT Network for Execution

For TwinCAT 3 requirements, see "Hardware Setup Requirements for TwinCAT 3" on page 5-6. To install the EtherCAT Network for execution by using the target computer as master node:

1   Run the Ethernet configuration tool. In the MATLAB Command Window, type:

    ```
    speedgoat.configureEthernet
    ```

    This tool lets you select either an IP address or reserve for EtherCAT each of the Ethernet ports that is not the host-target link port. In the EtherCAT Init block, you need an Ethernet port number. There, the number 1 refers to the first port that is reserved for EtherCAT using the Ethernet configuration tool. For more information, see the Speedgoat documentation.

2   Connect your EtherCAT network to the target computer Ethernet port dedicated to EtherCAT. Turn on the network.

The next task is "Configure EtherCAT Master Node Model" on page 5-12.

Configure EtherCAT Master Node Model

Before configuring the model, complete the procedure in "Configure EtherCAT Network by Using TwinCAT 3" on page 5-7.
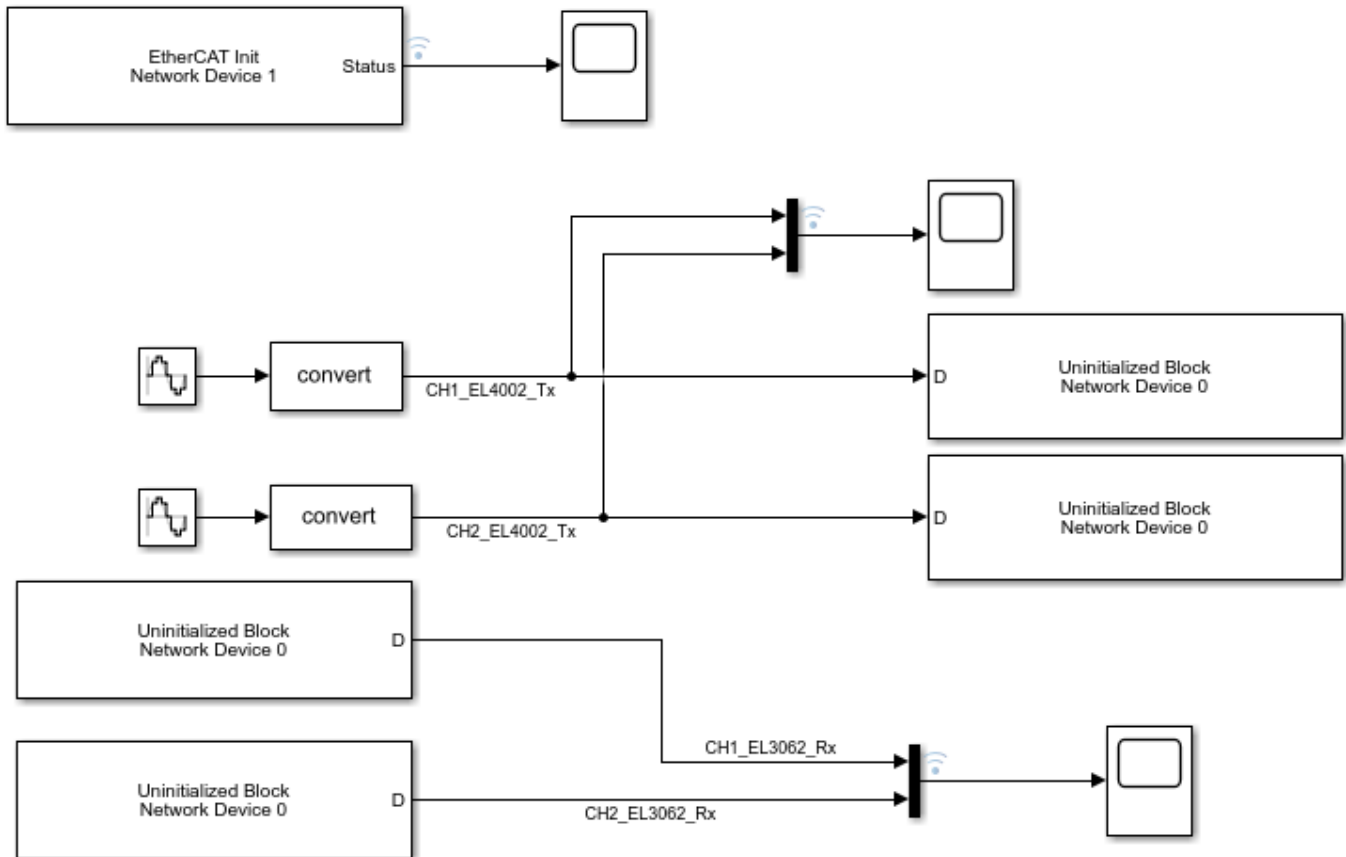
To configure model `slrt_ex_ethercat_beckhoff_aio` for execution by using the target computer as master node, complete the procedure in "Configure EtherCAT Init Block" on page 5-12.

Configure EtherCAT Init Block

Before you use the EtherCAT Init block, configure the EtherCAT network with TwinCAT 3.

Before you start this procedure, familiarize yourself with TwinCAT 3 and its documentation.

As part of the configuration process, create and save an EtherCAT Network Information (ENI) file. See "Configure EtherCAT Network by Using TwinCAT 3" on page 5-7.

To include EtherCAT distributed clocks when PTP is enabled for the model, use EtherCAT bus shift mode.

To configure the EtherCAT Init block of model `slrt_ex_ethercat_beckhoff_aio`:

1   Open model `slrt_ex_ethercat_beckhoff_aio`. In the MATLAB Command Window, type:

```
open_system(fullfile(matlabroot, 'toolbox', 'slrealtime',...
'examples', 'slrt_ex_ethercat_beckhoff_aio'))
```

2   Double-click the EtherCAT Init block.

3   In the **Config file (ENI)** text box, browse to the EtherCAT Network Information (ENI) file that you created when you configured the network (here, `'BeckhoffAIOconfig.xml'`). You can enter the file name with or without single quotes.

4   Use the default value `0` for parameter **Device index**.

   If the model includes more than one EtherCAT network, enter a unique **Device index** for each network. Enter the same value for all blocks in each network.

5   Enter the **Ethernet Port Number** for the EtherCAT port that you are connecting to your EtherCAT network. See "Install EtherCAT Network for Execution" on page 5-11.

6   Take the default value `Large model` for parameter **DC Tuning**.

7   To update the data in the EtherCAT Init block and propagate it to the other EtherCAT blocks, click **Refresh Data**.

8   Click **OK**.

Configure EtherCAT PDO Receive Blocks

Before beginning this procedure, you must have selected a valid ENI file in the EtherCAT Init block.

Before you start this procedure, familiarize yourself with TwinCAT 3 and its documentation.

To configure the EtherCAT PDO Receive blocks of model `slrt_ex_ethercat_beckhoff_aio`:

1 Double-click the EtherCAT PDO Receive block labeled `EtherCAT PDO Receive`.

2 Set parameter **Device Index** to the value set in the EtherCAT Init block.

3 From the **Signal Name** list, select the variable, here `Term 2 (EL3062).AI Standard Channel 1.Value`.

4 Observe the value in seconds of parameter **Sample Time**.

5 Click **OK**.

Execute steps 1–5 for the EtherCAT PDO Receive block labeled `EtherCAT PDO Receive 1`.

Configure EtherCAT PDO Transmit Blocks

Before beginning this procedure, you must have selected a valid ENI file in the EtherCAT Init block.

Before you start this procedure, familiarize yourself with TwinCAT 3 and its documentation.

To configure the EtherCAT PDO Transmit blocks of model `slrt_ex_ethercat_beckhoff_aio`:

1 Open model `slrt_ex_ethercat_beckhoff_aio`.

2 Double-click the EtherCAT PDO Transmit block labeled `EtherCAT PDO Transmit`.

3 Set parameter **Device Index** to the value set in the EtherCAT Init block.

4 Select a **Signal Name** variable, here `Term 3 (EL4002).AO Outputs Channel 1.Analog output`.

5 Observe the value in seconds of parameter **Sample Time**.

Block Parameters: EtherCAT PDO Transmit

Block Information

EtherCAT transmit variable

Device Index: 0

Signal Name: Term 3 (EL4002).AO Outputs Channel 1.Analog output

Signal Information

Signal Offset :	208
Signal Type :	int16
Type Size (bits) :	16
Signal Dimension :	1
Sample Time :	0.01

OK Cancel Help Apply
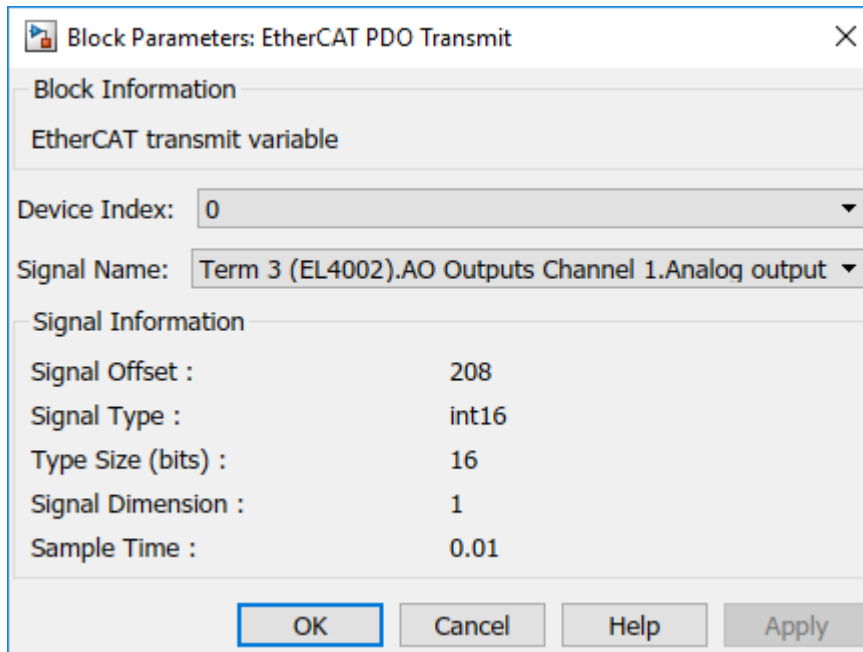
6 Click **OK**.

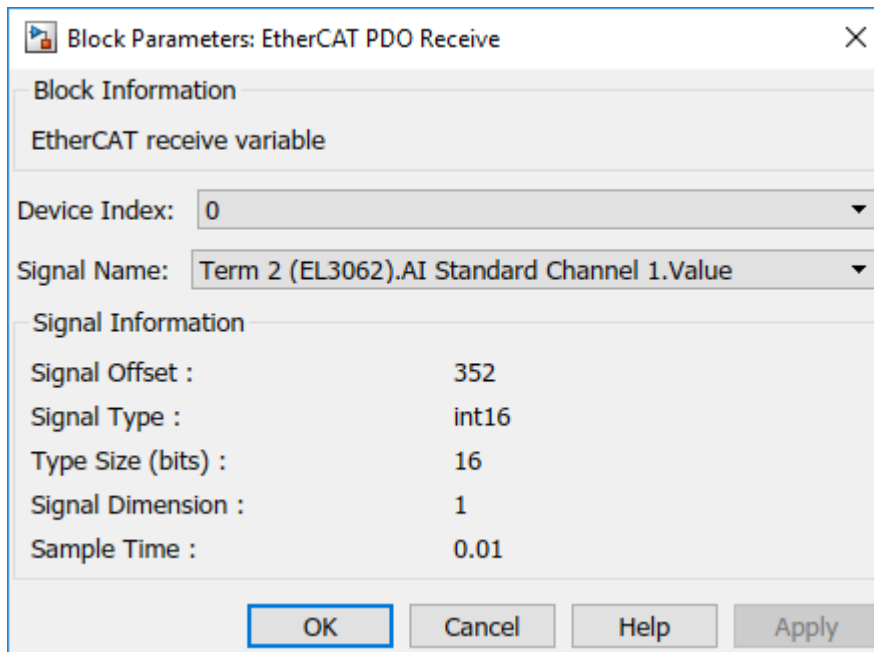Execute steps 2–6 for the EtherCAT PDO Transmit block labeled `EtherCAT PDO Transmit 1`.

Configure EtherCAT Model Configuration Parameters

Before beginning this procedure, you must have selected a valid ENI file in the EtherCAT Init block. For more information, see "Fixed-Step Size Derivation" on page 5-20.

To configure the configuration parameters for model `slrt_ex_ethercat_beckhoff_aio`:

1 Open model `slrt_ex_ethercat_beckhoff_aio`.

2 Calculate the greatest common divisor (GCD) of the **Sample Time** values for the EtherCAT tasks and for all source blocks in the model. In this case, the GCD is `0.010`.

3 In the Simulink Editor, on the **Real-Time** tab, from the **Prepare** section, click **Hardware Settings**. Select **Configuration Parameters > Solver**.

4 Set the **Type** parameter to `Fixed-step` and **Fixed-step size (fundamental sample time)** to one of the following:

- An integral divisor of the GCD value, in seconds.
- `auto`, if all other source blocks in the model have defined sample times.

In this case, set the parameter to `0.010`.
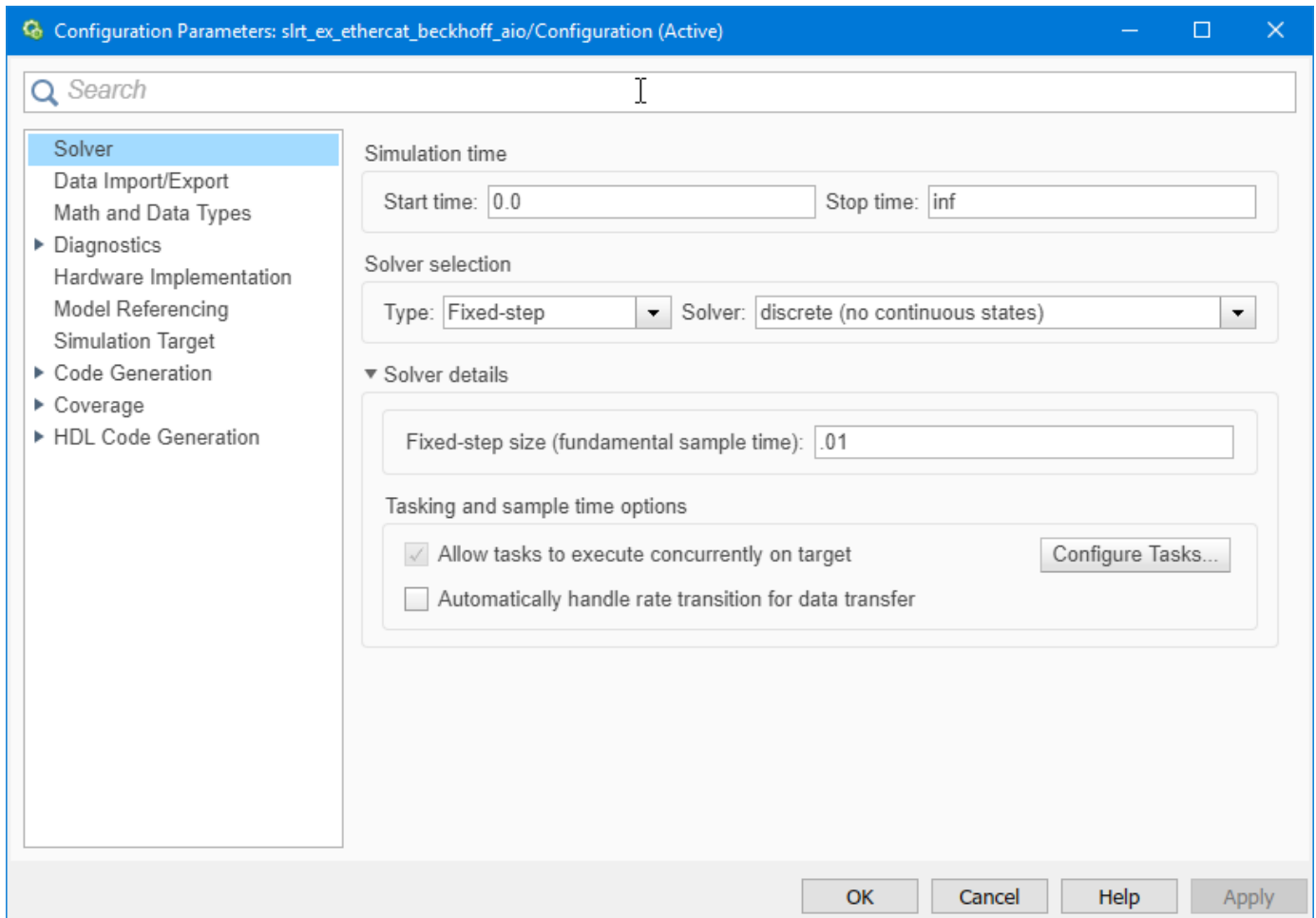
5 Click **OK**.

The next tasks are building, downloading, and executing the EtherCAT master node model.

EtherCAT Distributed Clock Algorithm

In this section...
"Master Shift Mode" on page 5-17
"Bus Shift Mode" on page 5-18
"Limitations" on page 5-19

An EtherCAT network consists of a master node (the target computer) connected to an arbitrary number of slave nodes (devices). Each node contains a clock that controls its internal operation. When you enable distributed clocks in the ENI file by using the configurator program, EtherCAT designates one clock in the network as the reference clock. The EtherCAT distributed clock (DC) algorithm then synchronizes the operation of multiple network nodes to the reference clock.

The DC algorithm operates in two phases. In phase 1, the algorithm aligns the clocks of DC-enabled network nodes other than the master node with the clock of the first DC-enabled slave node. In phase 2, the algorithm aligns the remaining unaligned clock with the reference clock.

Master Shift Mode

In master shift mode, the reference clock is the clock of the first DC-enabled slave in the network.

In phase 1, the algorithm shifts the sample time of the network nodes to align with the clock of the first slave node. In that process, the EtherCAT Init block output value `NetworkToSlaveClkDiff` decreases to near zero.
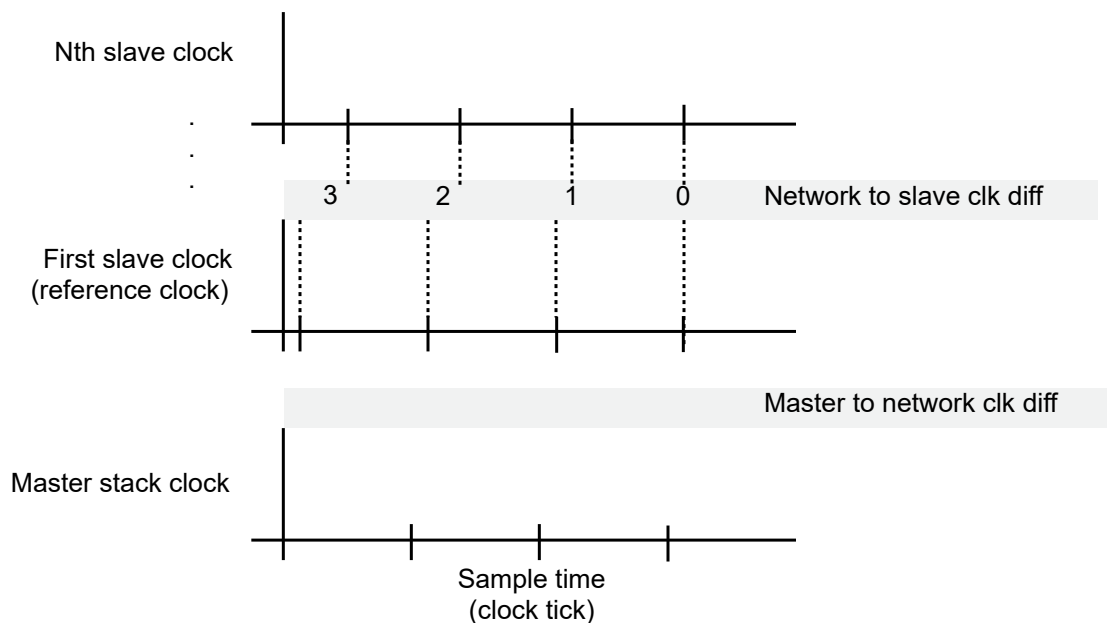
In phase 2, the algorithm shifts the sample time of the master stack running on the target computer to align with the first slave node clock. In that process, the EtherCAT Init block output value `MasterToNetworkClkDiff` decreases to near zero. If there are no DC enabled devices, both values are zero.

Nth slave clock

First slave clock
(reference clock)

Network to slave clk diff

0   0   0   0

Master to network clk diff

1   0   0

Master stack clock

Sample time
(clock tick)
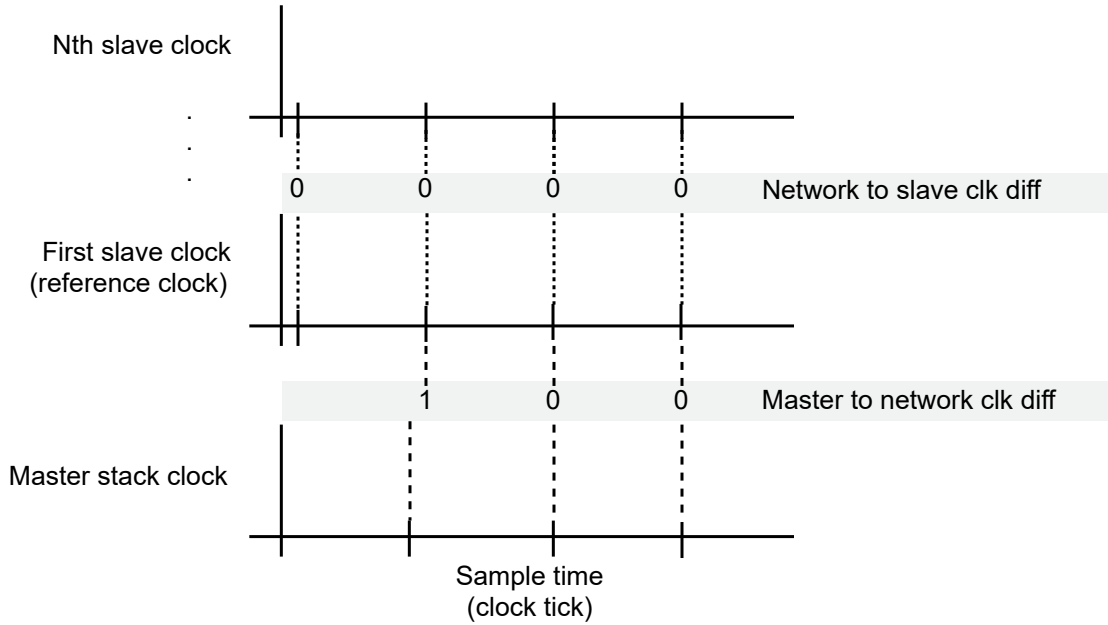
Bus Shift Mode

In bus shift mode, the reference clock is the clock of the master stack running on the target computer.

In phase 1, the algorithm shifts the sample time of the DC-enabled network nodes to align with the clock of the first DC-enabled slave node. In that process, the value `NetworkToSlaveClkDiff` decreases to near zero.

Nth slave clock

First slave clock

Network to slave clk diff

3   2   1   0

Master to network clk diff

Master stack clock
(reference clock)

Sample time
(clock tick)
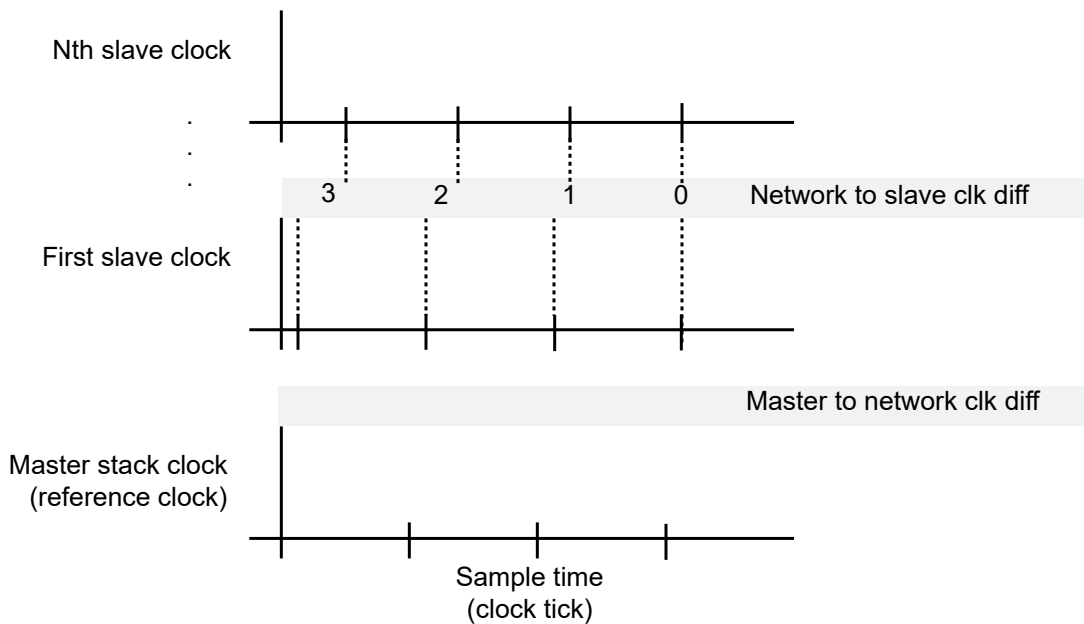
In phase 2, the algorithm shifts the sample time of the first DC-enabled slave node to align with the clock of the master stack. In that process, the value `MasterToNetworkClkDiff` decreases to near zero. The algorithm shifts the sample time of the other network nodes to stay aligned with the first slave node clock. In that process, the value of `NetworkToSlaveClkDiff` can first increase, then decrease to near zero.
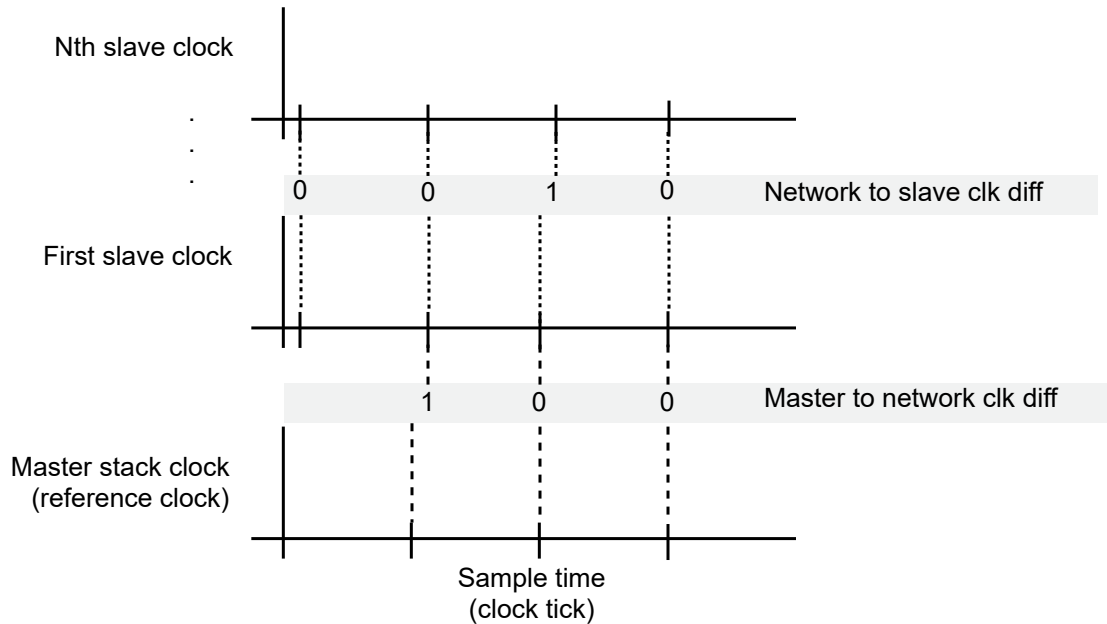
Limitations

To include EtherCAT distributed clocks when PTP is enabled for the model, use EtherCAT bus shift mode.

See Also
EtherCAT Init

More About
• "EtherCAT Init Block DC Error Values" on page 5-24

Fixed-Step Size Derivation

To configure the sample time for an EtherCAT model, set the **Fixed-step size (fundamental sample time)** for the entire model in the model Configuration Parameters **Solver** pane. You can also specify the sample times for key blocks. Sample times for blocks are integer multiples or divisors of the fundamental sample time.

During execution, the fixed-step size determines the cycle tick of the EtherCAT tasks and the sample times of the other source blocks in the model. Subject to the fixed step size value, the block type determines the sample time groups: a comparatively long sample time for the synchronous SDO blocks and another, shorter sample time for the rest of the blocks. As a best practice, set the sample time for the synchronous SDO blocks to a value at least three times of that for the PDO blocks.

Using an EtherCAT network configurator, specify the EtherCAT task cycle tick based on the requirements of the EtherCAT network. Specify the fixed-step size so that the GCD of the task cycle tick and the block sample times is an integer multiple of the fixed-step size.

For example, assume that the fastest EtherCAT task rate is 50 Hz for a corresponding cycle tick of 20 ms. The model block sample times, scaled to ms, are [20, 30, 40 50]. The FSS is:

```
FSS = min(gcd(20, [20, 30, 40, 50]))

FSS =

    10
```

The software sends all PDO data updates at the fastest EtherCAT task cycle tick (20 ms), even if you created multiple EtherCAT tasks running at different cycle ticks. The PDO Read and Write blocks run at the cycle tick for the tasks containing the given EtherCAT variable.

If you know that the other source blocks have defined sample times, you can set **Fixed-step size** to `auto`. If one or more block sample times are incompatible with the fixed sample time, there is an error during system update. If you do not encounter an error, in the Simulink Editor, on the **Debug** tab, from **Information Overlays**, click **Sample Time Colors** to see the block sample times.

EtherCAT Protocol Mapping

EtherCAT supports several overlay protocols. Simulink Real-Time supports some of the protocols directly, provides others with minimal support, and does not support some other protocols.

Overlay Protocol	Protocol Description	Support Type	Action
CANopen over EtherCAT (CoE)	Implements CANopen functionality by using EtherCAT	Direct	Model CoE by using SDO upload and download blocks
Ethernet over EtherCAT (EoE)	Provides EtherCAT wrapper around Ethernet packets. EtherCAT acts as network switch	Minimal	Send wrapped EoE messages between separate slave devices
File Access over EtherCAT (FoE)	Updates the EtherCAT board ROM	Not supported	Update the EtherCAT slave ROM with TwinCAT 3.
Functional Safety over EtherCAT (FSoE)	Sends asynchronous safety messages over the network	Not supported	Not applicable
Servcos over EtherCAT (SoE)	Wraps vendor-specific servo commands in a common protocol	Direct	SoE by using SSC up and down

CoE and SoE are two addressing schemes for configuration parameters in slave devices. SoE is used for some motor drives. For more information, see the device documentation.

EtherCAT Configurator Component Mapping

This table summarizes the mapping between third-party EtherCAT configurator components and Simulink Real-Time blocks and block attributes. For more information, see the TwinCAT 3 or Acontis EC-Engineer documentation.

EtherCAT Configurator Component		Simulink Real-Time Component
TwinCAT	**Acontis EC-Engineer**	
Cycle ticks (task step)	Cycle time	Sample time
Scalars and vectors	Dimension	Dimension
BitSize	Byte size of type	Type Size
Data Type, BitSize	Data type	Signal Type
EtherCAT device variable names linked to variables in the task PDO	All PDO variable names included in default task as defined in the ENI file	EtherCAT PDO signal names for the Receive and Transmit PDO blocks

See Also

EtherCAT PDO Receive | EtherCAT PDO Transmit

More About

*   "EtherCAT Data Types" on page 5-23

EtherCAT Data Types

The Simulink Real-Time EtherCAT blocks directly support the following EtherCAT data types. The software maps other EtherCAT data types to a byte array. The byte array requires explicit data type conversion by using Byte Pack, Byte Unpack, or S-function blocks.

EtherCAT Data Type	Data Type Size (bits)	Converted Simulink Data Type
bit	1	uint8
bit8	8	uint8
bitarr	8 (bit array)	uint8
bitarr16	16 (bit array)	uint16
bitarr32	32 (bit array)	uint32
BOOL	1	Boolean
int8	8	int8
int16	16	int16
int32	32	int32
int64	64	int64
uint8	8	uint8
uint16	16	uint16
uint32	32	uint32

EtherCAT Init Block DC Error Values

The Simulink Real-Time EtherCAT Init block returns the following EtherCAT distributed clock (DC) error values related to the master shift controller and the bus shift controller.

Error Value	Description
1 (0x1)	Initialization function not called or not successful
2 (0x2)	Controller error — synchronization out of limit Master shift: error 2, Maximum controller error exceeded Bus shift: error 2, Maximum controller error exceeded
3 (0x3)	Not enough memory
4 (0x4)	Hardware layer — (BSP) invalid
5 (0x5)	Hardware layer — error modifying the timer
6 (0x6)	Hardware layer — timer is not running
7 (0x7)	Hardware layer — function is called on wrong CPU
8 (0x8)	Invalid DC synchronization period length
9 (0x9)	Error DCM Controller SetVal is too small
10 (0xA)	Error DCM Controller — Drift between local timer and ref clock too high
11 (0xB)	ERROR: Error DCM Controller - Bus cycle time (dwBusCycleTimeUsec) doesn't match real cycle
27 (0x1B)	DC controller not ready Master shift: error 27, DC controller not ready Bus shift: error 27, DC controller not ready
28 (0x1C)	DC controller busy

EtherCAT Error Codes

The **Error** output for the EtherCAT blocks returns an EtherCAT error code. These blocks include:

- EtherCAT Sync SSC/SoE Upload
- EtherCAT Sync SSC/SoE Download
- EtherCAT Async SSC/SoE Upload
- EtherCAT Async SSC/SoE Download
- EtherCAT Sync SDO Upload
- EtherCAT Sync SDO Download
- EtherCAT Async SDO Upload
- EtherCAT Async SDO Download
- EtherCAT Set State

These EtherCAT error codes are prepended onto the small number error codes. These prepended codes must not appear without the small number added. These prepended codes appear in the upper 16 bits of the unsigned 32-bit error code. You can masked the codes to display the small number error code. In the table, the Decimal column shows the base 10 value after the upper 16 bits are masked.

Prepended Codes	Hexadecimal	Error Text
EC_E_NOERROR	0x00000000	No Error
EC_E_ERROR	0x98110000	Unspecific Error
EMRAS_E_ERROR	0x98110180	Unspecific RAS Error The RAS (Remote Access Server) is not yet implemented
DCM_E_ERROR	0x981201C0	Unspecific DCM Error This class of error comes from the master shift DC driver
EC_TEXTBASE	0x0200	Unknown Text (Base)
EC_ALSTATEBASE	0x0300	AL Status No Error

These codes are the small-number EtherCAT error codes.

Hexadecimal	Decimal	Error Text
EC_E_ERROR+0x01	1	ERROR: Feature not supported
EC_E_ERROR+0x02	2	ERROR: Invalid index
EC_E_ERROR+0x03	3	ERROR: Invalid offset
EC_E_ERROR+0x04	4	ERROR: Cancel
EC_E_ERROR+0x05	5	ERROR: Invalid size
EC_E_ERROR+0x06	6	ERROR: Invalid data
EC_E_ERROR+0x07	7	ERROR: Not ready
EC_E_ERROR+0x08	8	ERROR: Busy

Hexadecimal	Decimal	Error Text
EC_E_ERROR+0x09	9	ERROR: Cannot queue acyclic EtherCAT command (MasterConfig.dwMaxQueuedEthFrames)
EC_E_ERROR+0x0A	10	ERROR: No memory left
EC_E_ERROR+0x0B	11	ERROR: Invalid parameter
EC_E_ERROR+0x0C	12	ERROR: Not found
EC_E_ERROR+0x0D	13	ERROR: Duplicate
EC_E_ERROR+0x0E	14	ERROR: Invalid state
EC_E_ERROR+0x0F	15	ERROR: Cannot add slave to timer list
EC_E_ERROR+0x10	16	ERROR: Time-out
EC_E_ERROR+0x11	17	ERROR: Open failed
EC_E_ERROR+0x12	18	ERROR: Send failed
EC_E_ERROR+0x13	19	ERROR: Insert mailbox error
EC_E_ERROR+0x14	20	ERROR: Invalid mailbox command
EC_E_ERROR+0x15	21	ERROR: Unknown mailbox protocol command
EC_E_ERROR+0x16	22	ERROR: Access denied
EC_E_ERROR+0x17	23	ERROR: Identification failed
EC_E_ERROR+0x1A	26	ERROR: Invalid product key
EC_E_ERROR+0x1B	27	ERROR: Wrong format of master XML file
EC_E_ERROR+0x1C	28	ERROR: Feature disabled
EC_E_ERROR+0x1D	29	ERROR: Shadow memory requested in wrong mode
EC_E_ERROR+0x1E	30	Bus configuration mismatch
EC_E_ERROR+0x1F	31	ERROR: Error in reading config file
EC_E_ERROR+0x20	32	ERROR: Configuration doesn't support SAFEOP and OP requested state
EC_E_ERROR+0x21	33	ERROR: Cyclic commands are missing
EC_E_ERROR+0x22	34	ERROR: AL_STATUS register read missing in XML file for at least one state
EC_E_ERROR+0x23	35	ERROR: Fatal internal McSm
EC_E_ERROR+0x24	36	ERROR: Slave error
EC_E_ERROR+0x25	37	ERROR: Frame lost, IDX mismatch
EC_E_ERROR+0x26	38	ERROR: At least one EtherCAT command is missing in the received frame
EC_E_ERROR+0x28	40	ERROR: IOCTL EC_IOCTL_DC_LATCH_REQ_LTIMVALS not possible in DC Latching auto read mode
EC_E_ERROR+0x29	41	ERROR: Auto increment address - increment mismatch (slave missing)
EC_E_ERROR+0x2A	42	ERROR: Slave in invalid state, e.g. not in OP (API not callable in this state

Hexadecimal	Decimal	Error Text
EC_E_ERROR+0x2B	43	ERROR: Station address lost or slave missing - FPRD to AL_STATUS failed
EC_E_ERROR+0x2C	44	ERROR: Too many cyclic commands in XML configuration file. (Check EC_T_MASTER_CONFIG.dwMaxQueuedEthFrames
EC_E_ERROR+0x2D	45	ERROR: Ethernet link cable disconnected
EC_E_ERROR+0x2E	46	ERROR: Master core not accessible
EC_E_ERROR+0x2F	47	ERROR CoE: Mailbox send: working counter
EC_E_ERROR+0x31	49	ERROR: No mailbox support
EC_E_ERROR+0x32	50	ERROR CoE: Protocol not supported
EC_E_ERROR+0x33	51	ERROR EoE: Protocol not supported
EC_E_ERROR+0x34	52	ERROR FoE: Protocol not supported
EC_E_ERROR+0x35	53	ERROR SoE: Protocol not supported
EC_E_ERROR+0x36	54	ERROR VoE: Protocol not supported
CoE SDO command errors can be returned by the 4 SDO/CoE blocks		
EC_E_ERROR+0x40	64	ERROR SDO: Toggle bit not alternated
EC_E_ERROR+0x41	65	ERROR SDO: SDO protocol time-out
EC_E_ERROR+0x42	66	ERROR SDO: Client/server command specifier not valid or unknown
EC_E_ERROR+0x43	67	ERROR SDO: Invalid block size (block mode only
EC_E_ERROR+0x44	68	ERROR SDO: Invalid sequence number (block mode only
EC_E_ERROR+0x45	69	ERROR SDO: CRC error (block mode only
EC_E_ERROR+0x46	70	ERROR SDO: Out of memory
EC_E_ERROR+0x47	71	ERROR SDO: Unsupported access to an object
EC_E_ERROR+0x48	72	ERROR SDO: Attempt to read a write only object
EC_E_ERROR+0x49	73	ERROR SDO: Attempt to write a read only object
EC_E_ERROR+0x4A	74	ERROR SDO: Object does not exist in the object dictionary
EC_E_ERROR+0x4B	75	ERROR SDO: Object cannot be mapped to the PDO
EC_E_ERROR+0x4C	76	ERROR SDO: Number and length of objects to be mapped exceed PDO length
EC_E_ERROR+0x4D	77	ERROR SDO: General parameter incompatibility
EC_E_ERROR+0x4E	78	ERROR SDO: General internal incompatibility in the device.
EC_E_ERROR+0x4F	79	ERROR SDO: Access failed due to an hardware error

Hexadecimal	Decimal	Error Text
EC_E_ERROR+0x50	80	ERROR SDO: Data type does not match, length of service parameter does not match
EC_E_ERROR+0x51	81	ERROR SDO: Data type does not match, service parameter too long
EC_E_ERROR+0x52	82	ERROR SDO: Data type does not match, service parameter too short
EC_E_ERROR+0x53	83	ERROR SDO: Sub-index does not exist
EC_E_ERROR+0x54	84	ERROR SDO: Write access - Parameter value out of range
EC_E_ERROR+0x55	85	ERROR SDO: Write access - Parameter value out of high limit
EC_E_ERROR+0x56	86	ERROR SDO: Write access - Parameter value out of low limit
EC_E_ERROR+0x57	87	ERROR SDO: Maximum value is less than minimum value
EC_E_ERROR+0x58	88	ERROR SDO: General error
EC_E_ERROR+0x59	89	ERROR SDO: Unable to transfer or store data to the application
EC_E_ERROR+0x5A	90	ERROR SDO: Unable to transfer or store data to the application because of local control
EC_E_ERROR+0x5B	91	ERROR SDO: Unable to transfer or store data to the application because of the present device state
EC_E_ERROR+0x5C	92	ERROR SDO: Dynamic generation of object dictionary failed or missing object dictionary
EC_E_ERROR+0x5D	93	ERROR SDO: Unknown code
FoE commands (not reachable with SLRT implementation)		
EC_E_ERROR+0x60	96	ERROR FoE: Vendor specific FoE error
EC_E_ERROR+0x61	97	ERROR FoE: Not found
EC_E_ERROR+0x62	98	ERROR FoE: Access denied
EC_E_ERROR+0x63	99	ERROR FoE: Disk full
EC_E_ERROR+0x64	100	ERROR FoE: Illegal
EC_E_ERROR+0x65	101	ERROR FoE: Wrong packet number
EC_E_ERROR+0x66	102	ERROR FoE: Already exists
EC_E_ERROR+0x67	103	ERROR FoE: User missing
EC_E_ERROR+0x68	104	ERROR FoE: Bootstrap only
EC_E_ERROR+0x69	105	ERROR FoE: Not bootstrap
EC_E_ERROR+0x6A	106	ERROR FoE: No rights

Hexadecimal	Decimal	Error Text
EC_E_ERROR+0x6B	107	ERROR FoE: Program error
End of FoE specific errors		
General errors again		
EC_E_ERROR+0x70	112	ERROR: Master configuration not found
EC_E_ERROR+0x71	113	ERROR: Command error while EEPROM upload
EC_E_ERROR+0x72	114	ERROR: Command error while EEPROM download
EC_E_ERROR+0x73	115	ERROR: Cyclic command wrong size (too long)
EC_E_ERROR+0x74	116	ERROR: Invalid input offset in cyc cmd, please check InputOffs
EC_E_ERROR+0x75	117	ERROR: Invalid output offset in cyc cmd, please check OutputOffs
EC_E_ERROR+0x76	118	ERROR: Port Close failed
EC_E_ERROR+0x77	119	ERROR: Port Open failed
SoE command errors, can be returned by the 4 SSC/SoE blocks		
EC_E_ERROR+0x78	120	ERROR SoE: Invalid access to element 0
EC_E_ERROR+0x79	121	ERROR SoE: Does not exist
EC_E_ERROR+0x7a	122	ERROR SoE: Invalid access to element 1
EC_E_ERROR+0x7b	123	ERROR SoE: Name does not exist
EC_E_ERROR+0x7c	124	ERROR SoE: Name undersize in transmission
EC_E_ERROR+0x7d	125	ERROR SoE: Name oversize in transmission
EC_E_ERROR+0x7e	126	ERROR SoE: Name unchangeable
EC_E_ERROR+0x7f	127	ERROR SoE: Name currently write-protected
EC_E_ERROR+0x80	128	ERROR SoE: Attribute undersize in transmission
EC_E_ERROR+0x81	129	ERROR SoE: Attribute oversize in transmission
EC_E_ERROR+0x82	130	ERROR SoE: Attribute unchangeable
EC_E_ERROR+0x83	131	ERROR SoE: Attribute currently write-protected
EC_E_ERROR+0x84	132	ERROR SoE: Unit does not exist
EC_E_ERROR+0x85	133	ERROR SoE: Unit undersize in transmission
EC_E_ERROR+0x86	134	ERROR SoE: Unit oversize in transmission

Hexadecimal	Decimal	Error Text
EC_E_ERROR+0x87	135	ERROR SoE: Unit unchangeable
EC_E_ERROR+0x88	136	ERROR SoE: Unit currently write-protected
EC_E_ERROR+0x89	137	ERROR SoE: Minimum input value does not exist
EC_E_ERROR+0x8a	138	ERROR SoE: Minimum input value undersize in transmission
EC_E_ERROR+0x8b	139	ERROR SoE: Minimum input value oversize in transmission
EC_E_ERROR+0x8c	140	ERROR SoE: Minimum input value unchangeable
EC_E_ERROR+0x8d	141	ERROR SoE: Minimum input value currently write-protected
EC_E_ERROR+0x8e	142	ERROR SoE: Maximum input value does not exist
EC_E_ERROR+0x8f	143	ERROR SoE: Maximum input value undersize in transmission
EC_E_ERROR+0x90	144	ERROR SoE: Maximum input value oversize in transmission
EC_E_ERROR+0x91	145	ERROR SoE: Maximum input value unchangeable
EC_E_ERROR+0x92	146	ERROR SoE: Maximum input value currently write-protected
EC_E_ERROR+0x93	147	ERROR SoE: Data item does not exist
EC_E_ERROR+0x94	148	ERROR SoE: Data item undersize in transmission
EC_E_ERROR+0x95	149	ERROR SoE: Data item oversize in transmission
EC_E_ERROR+0x96	150	ERROR SoE: Data item unchangeable
EC_E_ERROR+0x97	151	ERROR SoE: Data item currently write-protected
EC_E_ERROR+0x98	152	ERROR SoE: Data item less than minimum input value limit
EC_E_ERROR+0x99	153	ERROR SoE: Data item exceeds maximum input value limit
EC_E_ERROR+0x9a	154	ERROR SoE: Data item is incorrect
EC_E_ERROR+0x9b	155	ERROR SoE: Data item is protected by password
EC_E_ERROR+0x9c	156	ERROR SoE: Data item temporary unchangeable (in AT or MDT)
EC_E_ERROR+0x9d	157	ERROR SoE: Invalid indirect
EC_E_ERROR+0x9e	158	ERROR SoE: Data item temporary unchangeable (parameter or opmode...)
EC_E_ERROR+0x9f	159	ERROR SoE: Command already active
EC_E_ERROR+0x100	256	ERROR SoE: Command not interruptible
EC_E_ERROR+0x101	257	ERROR SoE: Command not available (in this phase)
EC_E_ERROR+0x102	258	ERROR SoE: Command not available (invalid parameter...)

Hexadecimal	Decimal	Error Text
EC_E_ERROR+0x103	259	ERROR SoE: Response drive number not identical with the requested drive number
EC_E_ERROR+0x104	260	ERROR SoE: Response IDN not identical with the requested IDN
EC_E_ERROR+0x105	261	ERROR SoE: At least one fragment lost
EC_E_ERROR+0x106	262	ERROR SoE: RX buffer is full (ecat call with to small data-buffer)
EC_E_ERROR+0x107	263	ERROR SoE: No data state.
EC_E_ERROR+0x108	264	ERROR SoE: No default value.
EC_E_ERROR+0x109	265	ERROR SoE: Default value transmission too long.
EC_E_ERROR+0x10a	266	ERROR SoE: Default value cannot be changed, read only.
EC_E_ERROR+0x10b	267	ERROR SoE: Invalid drive number.
EC_E_ERROR+0x10c	268	ERROR SoE: General error
EC_E_ERROR+0x10d	269	ERROR SoE: No element addressed.
End of SoE specific error codes		
EC_E_ERROR+0x10e	270	Command not executed. Slave is not present on Bus
EC_E_ERROR+0x10f	271	ERROR FoE: Protocol not supported in boot strap
EC_E_ERROR+0x110	272	ERROR: command error while EEPROM reload
EC_E_ERROR+0x111	273	ERROR: command error while Reset Slave Controller
EC_E_ERROR+0x11E	286	Bus configuration not detected, Topology changed
EC_E_ERROR+0x11F	287	ERROR EoE: Mailbox receive: working counter
EC_E_ERROR+0x120	288	ERROR FoE: Mailbox receive: working counter
EC_E_ERROR+0x121	289	ERROR SoE: mailbox receive: working counter
EC_E_ERROR+0x122	290	ERROR AoE: Mailbox receive: working counter
EC_E_ERROR+0x123	291	ERROR VoE: Mailbox receive: working counter
EC_E_ERROR+0x124	292	ERROR: EEPROM assignment failed
EC_E_ERROR+0x125	293	ERROR: Error mailbox received
EC_E_ERROR+0x126	294	ERROR: Redundancy line break
EC_E_ERROR+0x127	295	ERROR: Invalid EtherCAT cmd in cyclic frame with redundancy
EC_E_ERROR+0x128	296	ERROR: <PreviousPort>-tag is missing!
EC_E_ERROR+0x129	297	ERROR: DC is enabled and DC cyclic commands are missing (e.g. access to 0x900)!
EC_E_ERROR+0x130	304	ERROR: DL Status Interrupt because of changed Topology

Hexadecimal	Decimal	Error Text
EC_E_ERROR+0x131	305	ERROR: The Pass Through Server is not running!
EC_E_ERROR+0x132	306	ERROR: The ADS adapter (Pass Through Server) is running
EC_E_ERROR+0x133	307	ERROR: Could not start the Pass Through Server
EC_E_ERROR+0x134	308	ERROR: The Pass Through Server could not bind the IP address with a socket
EC_E_ERROR+0x135	309	The Pass Through Server is running but not enabled
EC_E_ERROR+0x136	310	ERROR: This LinkLayer mode is not supported by the Pass Through Server
EC_E_ERROR+0x137	311	ERROR VoE: No VoE mailbox received
EC_E_ERROR+0x138	312	ERROR: SYNC out unit of reference clock is disabled
EC_E_ERROR+0x139	313	ERROR: Reference clock not found
EC_E_ERROR+0x13B	315	ERROR: Mailbox command working counter error
AoE is not supported by any SLRT blocks. These should never be returned.		
EC_E_ERROR+0x13C	316	ERROR AoE: Protocol not supported
EC_E_ERROR+0x13D	317	ERROR AoE: Invalid AoE response received
EC_E_ERROR+0x13E	318	ERROR AoE: Common AoE device error
EC_E_ERROR+0x13F	319	ERROR AoE: Service is not supported by server
EC_E_ERROR+0x140	320	ERROR AoE: Invalid index group
EC_E_ERROR+0x141	321	ERROR AoE: Invalid index offset
EC_E_ERROR+0x142	322	ERROR AoE: Reading/writing not permitted
EC_E_ERROR+0x143	323	ERROR AoE: Parameter size not correct
EC_E_ERROR+0x144	324	ERROR AoE: Invalid parameter value(s)
EC_E_ERROR+0x145	325	ERROR AoE: Device is not in a ready state
EC_E_ERROR+0x146	326	ERROR AoE: Device is busy
EC_E_ERROR+0x147	327	ERROR AoE: Invalid context
EC_E_ERROR+0x148	328	ERROR AoE: Out of memory
EC_E_ERROR+0x149	329	ERROR AoE: Invalid parameter value(s)
EC_E_ERROR+0x14A	330	ERROR AoE: Not found
EC_E_ERROR+0x14B	331	ERROR AoE: Syntax error in command or file
EC_E_ERROR+0x14C	332	ERROR AoE: Objects do not match
EC_E_ERROR+0x14D	333	ERROR AoE: Object already exists
EC_E_ERROR+0x14E	334	ERROR AoE: Symbol not found
EC_E_ERROR+0x14F	335	ERROR AoE: Symbol version invalid
EC_E_ERROR+0x150	336	ERROR AoE: Server is in invalid state

Hexadecimal	Decimal	Error Text
EC_E_ERROR+0x151	337	ERROR AoE: AdsTransMode not supported
EC_E_ERROR+0x152	338	ERROR AoE: Notification handle is invalid
EC_E_ERROR+0x153	339	ERROR AoE: Notification client not registered
EC_E_ERROR+0x154	340	ERROR AoE: No more notification handles
EC_E_ERROR+0x155	341	ERROR AoE: Size for watch to big
EC_E_ERROR+0x156	342	ERROR AoE: Device not initialized
EC_E_ERROR+0x157	343	ERROR AoE: Device has a timeout
EC_E_ERROR+0x158	344	ERROR AoE: Query interface failed
EC_E_ERROR+0x159	345	ERROR AoE: Wrong interface required
EC_E_ERROR+0x15A	346	ERROR AoE: Class ID is invalid
EC_E_ERROR+0x15B	347	ERROR AoE: Object ID is invalid
EC_E_ERROR+0x15C	348	ERROR AoE: Request is pending
EC_E_ERROR+0x15D	349	ERROR AoE: Request is aborted
EC_E_ERROR+0x15E	350	ERROR AoE: Signal warning
EC_E_ERROR+0x15F	351	ERROR AoE: Invalid array index
EC_E_ERROR+0x160	352	ERROR AoE: Symbol not active -> release handle and try again
EC_E_ERROR+0x161	353	ERROR AoE: Access denied
EC_E_ERROR+0x162	354	ERROR AoE: Internal error
EC_E_ERROR+0x163	355	ERROR AoE: Target port not found
EC_E_ERROR+0x164	356	ERROR AoE: Target machine not found
EC_E_ERROR+0x165	357	ERROR AoE: Unknown command ID
EC_E_ERROR+0x166	358	ERROR AoE: Port not connected
EC_E_ERROR+0x167	359	ERROR AoE: Invalid AMS length
EC_E_ERROR+0x168	360	ERROR AoE: invalid AMS Net ID
EC_E_ERROR+0x169	361	ERROR AoE: Port disabled
EC_E_ERROR+0x16A	362	ERROR AoE: Port already connected
EC_E_ERROR+0x16B	363	ERROR AoE: Invalid AMS port!
EC_E_ERROR+0x16C	364	ERROR AoE: No memory!
EC_E_ERROR+0x16D	365	ERROR AoE: Vendor specific AoE device error
EC_E_ERROR+0x16E	366	ERROR: Invalid AoE NetID!

End of AoE specific errors

me

Hexadecimal	Decimal	Error Text
Generic errors that indicate configuration problems, should never happen.		
EC_E_ERROR+0x16F	367	ERROR: Maximum number of bus slave has been exceeded
EC_E_ERROR+0x170	368	ERROR Mailbox: Syntax of 6 octet Mailbox header is wrong!
EC_E_ERROR+0x171	369	ERROR Mailbox: The Mailbox protocol is not supported
EC_E_ERROR+0x172	370	ERROR Mailbox: Field contains wrong value
EC_E_ERROR+0x173	371	ERROR Mailbox: The service in the Mailbox protocol is not supported
EC_E_ERROR+0x174	372	ERROR Mailbox: The mailbox protocol header of the mailbox protocol is wrong
EC_E_ERROR+0x175	373	ERROR Mailbox: Length of received mailbox data is too short
EC_E_ERROR+0x176	374	ERROR Mailbox: Mailbox protocol can not be processed because of limited resources
EC_E_ERROR+0x177	375	ERROR Mailbox: The length of data is inconsistent
EC_E_ERROR+0x178	376	ERROR: Slaves with DC configured are present on bus before the reference clock
EC_E_ERROR+0x179	377	ERROR: Data type conversion failed
EC_E_ERROR+0x17A	378	ERROR FoE: File is bigger than max file size
EC_E_ERROR+0x17B	379	ERROR: Line crossed
EC_E_ERROR+0x17C	380	ERROR: Line crossed at slave \%s\", EtherCAT auto-increment address=%d, station address=%d. Error at port %d."
EC_E_ERROR+0x17D	381	ERROR: Socket disconnected

See Also

EtherCAT Sync SSC/SoE Upload | EtherCAT Sync SSC/SoE Download | EtherCAT Async SSC/SoE Upload | EtherCAT Async SSC/SoE Download | EtherCAT Sync SDO Upload | EtherCAT Sync SDO Download | EtherCAT Async SDO Upload | EtherCAT Async SDO Download | EtherCAT Set State

EtherCAT Blocks

EtherCAT Init

Initialize EtherCAT Master node with data in the EtherCAT Network Information (ENI) file
Library: Simulink Real-Time / EtherCAT

```
EtherCAT Init
Network Device 0        Status
Master Shift DC
```

Description

The EtherCAT Init block initializes the EtherCAT master stack. The block specifies the Ethernet interface cards in the network.

Before you use this block, create and save an EtherCAT Network Information (ENI) file. You export the ENI file from the Beckhoff TwinCAT or the acontis EC-Engineer. See "Configure EtherCAT Network by Using TwinCAT 3" on page 5-7.

To find the ENI file, click **Browse**. To read the ENI file and store the data in the EtherCAT Init block, click **Refresh Data**.

The Simulink Real-Time software supports multiple EtherCAT networks. To use multiple networks:

- Use a different Ethernet card interface for each EtherCAT network.
- In the model, use one EtherCAT Init block for each network.

To include EtherCAT distributed clocks when PTP is enabled for the model, use EtherCAT bus shift mode.

Ports

Output

`Status` — Status information about the EtherCAT network
vector

The `Status` vector contains these values: `ErrVal`, `MasterState`, `DCErrVal`, `MasterToNetworkClkDiff`, `DCInitState`, and `NetworkToSlaveClkDiff`.

- `ErrVal` — Error status:

  - No error: `0`
  - Error: Value less than `0`.

  Because `ErrVal` shows the latest error status, the propagation of errors can hide the original error. To find the original error, add an EtherCAT Get Notifications block and use the `slrealtime.EtherCAT.filterNotifications` command to print the status codes that the EtherCAT stack transmits.

- `MasterState` — Operating state of the EtherCAT network.

State	Value	Description
INIT	1	Initialization — The system finds terminal devices and initializes the communication controller.
PREOP	2	Preoperational — The system uses the communication controller to exchange system-specific initialization data. In this state, the network cannot transmit or receive signal data.
SAFEOP	4	Safe operational — The network is running and ready for full operation. The supervisor sends input data to the terminal device. The terminal device output remains in a safe state.
OP	8	Operational — The network is in full operation. The supervisor sends input data to the terminal device. The terminal device responds with output data.

- `DCErrVal` — DC error status for the master shift controller:

  - No DC Error for master shift controller: `0`

  - DC error for master shift controller: Value from "EtherCAT Init Block DC Error Values" on page 5-24.

  When you select master shift controller mode, the value `0` indicates successful clock distribution. The `DCErrVal` does not apply when the distributed clock is disabled.

- `MasterToNetworkClkDiff` — Time difference, in nanoseconds, between the master stack clock and the clock on the first slave device that has enabled DC.

- `DCInitState` — Operating state of the distributed clock:

  - DC not enabled, not initialized, or single EtherCAT DC slave: `0`

  - DC has been started and the EtherCAT DC slaves are in sync with each other: `1`

- `NetworkToSlaveClkDiff` — Time difference, in nanoseconds, between the clock on the first EtherCAT slave device and the least closely locked clock on the remaining slave devices.

  This value applies only to slave devices that have enabled DC. If only one device on the network has enabled DC, this value is `0`.

Data Types: `int32`

Parameters

`Config file (ENI)` — ENI file from the EtherCAT configurator
character vector

Specify the ENI file that you exported from the EtherCAT configurator.

You can specify the absolute path name or a relative path name from the current folder. If you specify only the file name, the software searches for the file in the current folder and on the MATLAB path. If more than one file with that name exists on the path, MATLAB displays a message box that indicates a clearer file specification is needed.

Clicking **Browse** inserts a full, editable path name.

Programmatic Use
Block Parameter: config_file

Device index — EtherCAT network identifier
0-15

A unique integer in the range 0–15 that identifies the Ethernet card for an EtherCAT network.

For each EtherCAT network, the software generates a unique device index. The software inserts that device index as **Device index** into the EtherCAT Init block that represents the network. For more information about Speedgoat Target Machine settings, see "Install EtherCAT Network for Execution" on page 5-11.

Programmatic Use
Block Parameter: device_id

Ethernet Port Number — Port number in target computer
int

The first port reserved for EtherCAT by the speedgoat.configureEthernet function is port 1 here. For more information, see the description of the speedgoat.configureEthernet function in Speedgoat documentation. For more information about Speedgoat Target Machine settings, see "Install EtherCAT Network for Execution" on page 5-11.

Programmatic Use
Block Parameter: portnum

DC Tuning — Distributed clock initialization parameter
Large model (default) | Medium model | Small model

Enter the distributed clock initialization parameter from one of these values:

- Large model (default) — Sends 16,000 timing initialization packets and allows 1 second of settling time. Provides best initial synchronization between multiple slaves that have DC enabled.

- Medium model — Sends 8,000 timing initialization packets and allows 0.3 seconds of settling time. The model reaches operational state about a second earlier than it does with the Large model setting.

- Small model — Sends 2,000 timing initialization packets and allows 0.2 seconds of settling time. The model reaches operational state earlier than it does with the other settings.

Monitor device synchronization at the moment that the model enters the operational state. If the ENI file enables DC, make sure that the devices are synchronized closely enough for your application.

Programmatic Use
Block Parameter: dctuning

Enable EtherCAT Log for Debugging — Access to debugging and logging block parameters
Off (default) | Warning | Info | Verbose | All

The selections choose:

- Off — Skip all except fatal errors in log.

- Warning — Include warnings and fatal errors in log.

- `Info` — Include information displays, warnings, and fatal errors in log.
- `Verbose` — Include sequencing information from EtherCAT stack, information displays, warnings, and fatal errors in log.
- `All` — include low level debugging information and all other categories in log.

The `Verbose` and `All` logging levels can produce so much data that it can cause overloads at fast task rates.

The target log file name is `E_Master`%d, where %d is the device id value.

Programmatic Use
Block Parameter: enaDebug

See Also
EtherCAT Get Notifications | `slrealtime.EtherCAT.filterNotifications` | "EtherCAT Init Block DC Error Values" on page 5-24

Topics
"Configure EtherCAT Network by Using TwinCAT 3" on page 5-7
"Configure EtherCAT Master Node Model" on page 5-12

External Websites
www.ethercat.org
www.beckhoff.com
www.acontis.com/en/

Introduced in R2020b

EtherCAT Get Notifications

Collect notifications from the EtherCAT bus
Library: Simulink Real-Time / EtherCAT

```
EtherCAT Get Notifications    Values
    Network Device 0
```

Description

The EtherCAT Get Notifications block collects notifications from the EtherCAT stack and presents them to the output as a 21-element vector of `int32`. At each time step, the block outputs what it has accumulated and clears itself for the next time step.

The vector contains the number of notifications in element 1, followed by up to 20 notification codes. The maximum number of notifications is 20. If the bus presents more than 20 notifications to the output, the block discards the newest notifications and presents the first 20 that were received.

Ports

Output

Values — Self-descriptive 21-element vector containing EtherCAT notification codes
[Length 20 * Notification]

- `Length` (0 – 20) — the number of notifications in the vector.
- `Notification` — a composite of a notification type and a specific value. The types are:
- • `EC_NOTIFY_GENERIC` [0x00000000 (0)] — Represents state changes, such as:0x00000001 (1) — `EtherCAT operational state change`.
  - `EC_NOTIFY_ERROR` [0x00010000 (65536)] — Represents error states, such as 0x00010001 (65537):`cyclic command: working counter error`. Some describe changes in error state.
  - `EC_NOTIFY_SCANBUS` [0x00030000 (3*65536)] — Represents ScanBus error states, such as 0x00030002 (196610):`ScanBus mismatch`.
  - `EC_NOTIFY_HOTCONNECT` [0x00040000 (4*65536)] — Represents hot connect states, such as 0x00040005 (262149):`Slave disappears`.

To print the valid notification values and descriptions, call `slrealtime.etherCAT.filterNotifications` without an argument.

Data Types: `int32`

Parameters

Device index — EtherCAT network identifier
0 (default) | 0-11

To associate a block with an EtherCAT network, enter the **Device index** value from the EtherCAT Init block representing that network into the **Device index** for the block.

Programmatic Use
Block Parameter: device_id

Sample time — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time. Use the EtherCAT task sample time.

Programmatic Use
Block Parameter: sample_time

Tips

To collect notifications:

1   Add the EtherCAT Get Notifications block to your model.
2   Connect the EtherCAT Get Notifications block to a File Log block.
3   Use exported data log data from signal data displayed in the Simulation Data Inspector. See example Get Time and Data Log from EtherCAT Get Notifications Block for the slrealtime.EtherCAT.filterNotifications function.

See Also
slrealtime.EtherCAT.filterNotifications | EtherCAT Init

External Websites
www.ethercat.org
www.beckhoff.com
www.acontis.com/en/

Introduced in R2020b

EtherCAT PDO Receive

Receive data from slave device represented by process data object
Library: Simulink Real-Time / EtherCAT

Description

The EtherCAT PDO Receive block receives data from the EtherCAT slave device.

The block parameter dialog box has two sections, parameters and signal information. When you specify an EtherCAT network and device variable name:

- The EtherCAT PDO Receive block mask is updated with the selected signal name.
- The signal information in the block parameter dialog box is updated to reflect the device variable.

Note If an error occurs while the software parses the configuration file specified in the EtherCAT Init block, this block displays an error message.

Ports

Output

D — Data received from slave device
[double]

Vector of data received from the EtherCAT slave device.

Parameters

Device index — EtherCAT network identifier
0 (default) | 0-15

To associate a block with an EtherCAT network, copy the **Device index** value from the EtherCAT Init block representing that network into the **Device index** for the block.

Programmatic Use
Block Parameter: device_id

Signal name — EtherCAT device variable name
character vector

From the list, select the EtherCAT device variable name.

The block parameter dialog box updates the read-only signal information to reflect the device variable that you selected.

For a mapping of EtherCAT configurator components to Simulink Real-Time blocks and block attributes, see "EtherCAT Configurator Component Mapping" on page 5-22.

For a mapping of Simulink data types to EtherCAT data types, see "EtherCAT Data Types" on page 5-23.

Programmatic Use
Block Parameter: `sig_name`

Signal Offset — Location in the process of signal data
integer

This property is read-only.

After the execution of the EtherCAT Init block, location in the process image from which the data is available.

Programmatic Use
Block Parameter: `sig_offset`

Signal Type — Data type for EtherCAT data
character vector

This property is read-only.

Simulink data type for the EtherCAT data.

Programmatic Use
Block Parameter: `sig_type`

Type Size (bits) — Size of EtherCAT data type
integer

This property is read-only.

Size in bits of the EtherCAT data type.

Programmatic Use
Block Parameter: `type_size`

Signal Dimension — Dimension of the signal
integer

This property is read-only.

The EtherCAT blocks support vectors and scalars (vectors of dimension 1).

Programmatic Use
Block Parameter: `sig_dim`

Sample Time — Rate at which this block is executed
numeric

This property is read-only.

This rate is the execution rate of the EtherCAT task, as specified in the ENI file.

Programmatic Use
Block Parameter: `sample_time`

See Also

EtherCAT Init | EtherCAT PDO Transmit

Topics
"EtherCAT Configurator Component Mapping" on page 5-22
"EtherCAT Data Types" on page 5-23

External Websites
www.ethercat.org
www.beckhoff.com
www.acontis.com/en/

Introduced in R2020b

EtherCAT PDO Transmit

Send data to slave device represented by process data object
Library: Simulink Real-Time / EtherCAT

Description

The EtherCAT PDO Transmit block transmits computed data to a particular variable in the EtherCAT slave device.

The block parameter dialog box has two sections, parameters and signal information. When you specify an EtherCAT network and device variable name:

- The EtherCAT PDO Receive block mask is updated with the selected signal name.

- The signal information in the block parameter dialog box is updated to reflect the device variable.

Note If an error occurs while the software parses the configuration file specified in the EtherCAT Init block, this block displays an error message.

Ports

Input

D — Data to transmit to slave device
[double]

Vector of data to transmit to the EtherCAT slave device.

Parameters

`Device index` — EtherCAT network identifier
0 (default) | 0-15

To associate a block with an EtherCAT network, copy the **Device index** value from the EtherCAT Init block representing that network into the **Device index** for the block.

Programmatic Use
Block Parameter: device_id

`Signal name` — EtherCAT device variable name
character vector

From the list, select the EtherCAT device variable name.

The block parameter dialog box updates the read-only signal information to reflect the device variable that you selected.

For a mapping of EtherCAT configurator components to Simulink Real-Time blocks and block attributes, see "EtherCAT Configurator Component Mapping" on page 5-22.

For a mapping of Simulink data types to EtherCAT data types, see "EtherCAT Data Types" on page 5-23.

Programmatic Use
Block Parameter: `sig_name`

Signal Offset — Location in the process of signal data
integer

This property is read-only.

After the execution of the EtherCAT Init block, location in the process image from which the data is available.

Programmatic Use
Block Parameter: `sig_offset`

Signal Type — Data type for EtherCAT data
character vector

This property is read-only.

Simulink data type for the EtherCAT data.

Programmatic Use
Block Parameter: `sig_type`

Type Size (bits) — Size of EtherCAT data type
integer

This property is read-only.

Size in bits of the EtherCAT data type.

Programmatic Use
Block Parameter: `type_size`

Signal Dimension — Dimension of the signal
integer

This property is read-only.

The EtherCAT blocks support vectors and scalars (vectors of dimension 1).

Programmatic Use
Block Parameter: `sig_dim`

Sample Time — Rate at which this block is executed
numeric

This property is read-only.

This rate is the execution rate of the EtherCAT task, as specified in the ENI file.

Programmatic Use
Block Parameter: `sample_time`

See Also
EtherCAT Init | EtherCAT PDO Receive

Topics
"EtherCAT Configurator Component Mapping" on page 5-22
"EtherCAT Data Types" on page 5-23

External Websites
www.ethercat.org
www.beckhoff.com
www.acontis.com/en/

Introduced in R2020b

EtherCAT Get State

Get state of EtherCAT network
Library: Simulink Real-Time / EtherCAT

Description

The EtherCAT Get State block returns the state of the EtherCAT network.

Ports

Output

State — State received from the EtherCAT network
1 | 2 | 4 | 8

State	Value	Description
INIT	1	Initialization — The system finds terminal devices and initializes the communication controller.
PREOP	2	Preoperational — The system uses the communication controller to exchange system-specific initialization data. In this state, the network cannot transmit or receive signal data.
SAFEOP	4	Safe operational — The network is running and ready for full operation. The supervisor sends input data to the terminal device. The terminal device output remains in a safe state.
OP	8	Operational — The network is in full operation. The supervisor sends input data to the terminal device. The terminal device responds with output data.

Parameters

Device index — EtherCAT network identifier
0 (default) | 0-11

To associate a block with an EtherCAT network, enter the **Device index** value from the EtherCAT Init block representing that network into the **Device index** for the block.

Programmatic Use
Block Parameter: device_id

Sample time — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time. -1 means that the sample time is inherited.

Programmatic Use
Block Parameter: sample_time

See Also

EtherCAT Init | EtherCAT Set State

Topics
"EtherCAT Configurator Component Mapping" on page 5-22
"EtherCAT Data Types" on page 5-23

External Websites

www.ethercat.org
www.beckhoff.com
www.acontis.com/en/

Introduced in R2020b

EtherCAT Set State

Set state of EtherCAT network
Library: Simulink Real-Time / EtherCAT

Description

The EtherCAT Set State block sets the state of the EtherCAT network to the value passed in through the `New State` port.

Ports

Input

`New State` — State transmitted to the EtherCAT network
1 | 2 | 4 | 8

State	Value	Description
INIT	1	Initialization — The system finds terminal devices and initializes the communication controller.
PREOP	2	Preoperational — The system uses the communication controller to exchange system-specific initialization data. In this state, the network cannot transmit or receive signal data.
SAFEOP	4	Safe operational — The network is running and ready for full operation. The supervisor sends input data to the terminal device. The terminal device output remains in a safe state.
OP	8	Operational — The network is in full operation. The supervisor sends input data to the terminal device. The terminal device responds with output data.

Output

`Prev State` — Previous state of the network
1 | 2 | 4 | 8

This port returns the value of the previous setting of the `New State` port.

`Error` — Report an EtherCAT state error
0 | integer

If no error occurs, this port returns 0. Otherwise, it returns a nonzero value.

Parameters

`Device index` — EtherCAT network identifier
0 (default) | 0-11

To associate a block with an EtherCAT network, enter the **Device index** value from the EtherCAT Init block representing that network into the **Device index** for the block.

Programmatic Use
Block Parameter: `device_id`

Timeout — Time to wait for the network to change state
integer

Enter the number of seconds to wait for the EtherCAT network state to transition.

Set the timeout to `0` to return immediately. If you specify a nonzero **Timeout** value, in the Configuration Parameters **Solver** pane, set the **Fixed-step size** parameter to a value larger than the **Timeout** value.

Programmatic Use
Block Parameter: `timeout`

Sample time — Sample time of block
`-1` (default) | numeric

Enter the base sample time or a multiple of the base sample time. `-1` means that the sample time is inherited.

Programmatic Use
Block Parameter: `sample_time`

See Also

EtherCAT Init | EtherCAT Get State

Topics
"EtherCAT Configurator Component Mapping" on page 5-22
"EtherCAT Data Types" on page 5-23

External Websites
www.ethercat.org
www.beckhoff.com
www.acontis.com/en/

Introduced in R2020b

EtherCAT Get Device State

Get state of EtherCAT network
Library: Simulink Real-Time / EtherCAT

EtherCAT Get Device State
Network Device 0   State

Description

The EtherCAT Get Device State block returns the state of the EtherCAT network.

Ports

Output

State — State received from the selected device
1 | 2 | 4 | 8

State	Value	Description
INIT	1	Initialization — The system finds terminal devices and initializes the communication controller.
PREOP	2	Preoperational — The system uses the communication controller to exchange system-specific initialization data. In this state, the network cannot transmit or receive signal data.
SAFEOP	4	Safe operational — The network is running and ready for full operation. The supervisor sends input data to the terminal device. The terminal device output remains in a safe state.
OP	8	Operational — The network is in full operation. The supervisor sends input data to the terminal device. The terminal device responds with output data.

Parameters

Device index — EtherCAT network identifier
0 (default) | 0-11

To associate a block with an EtherCAT network, enter the **Device index** value from the EtherCAT Init block representing that network into the **Device index** for the block.

Programmatic Use
Block Parameter: device_id

Device Name — Name of EtherCAT device
character vector

Select an EtherCAT device from the list of available devices provided by the ENI configuration file. The configuration file is selected by the EtherCAT Init block.

Programmatic Use
Block Parameter: device_name

Sample time — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time. -1 means that the sample time is inherited.

Programmatic Use
Block Parameter: sample_time

See Also

EtherCAT Init | EtherCAT Set State | EtherCAT Get State | EtherCAT Set Device State

Topics
"EtherCAT Configurator Component Mapping" on page 5-22
"EtherCAT Data Types" on page 5-23

External Websites
www.ethercat.org
www.beckhoff.com
www.acontis.com/en/

Introduced in R2021b

EtherCAT Set Device State

Set state of EtherCAT network
Library: Simulink Real-Time / EtherCAT

Description

The EtherCAT Set Device State block sets the state of the EtherCAT network to the value passed through the `New State` port.

Ports

Input

`New State` — State transmitted to the EtherCAT network
1 | 2 | 4 | 8

State	Value	Description
INIT	1	Initialization — The system finds terminal devices and initializes the communication controller.
PREOP	2	Preoperational — The system uses the communication controller to exchange system-specific initialization data. In this state, the network cannot transmit or receive signal data.
SAFEOP	4	Safe operational — The network is running and ready for full operation. The supervisor sends input data to the terminal device. The terminal device output remains in a safe state.
OP	8	Operational — The network is in full operation. The supervisor sends input data to the terminal device. The terminal device responds with output data.

`Enable` — Enables block to send state request
0 | 1

The port takes a Boolean value. When true (1), the block sends a new state request. When false (0), the block does nothing.

Output

`Prev State` — Previous state of the selected device
1 | 2 | 4 | 8

This port returns the value of the previous state of the device. That value might not be the same as the previous state of this block.

`Error` — Report an EtherCAT state error
0 | integer

If no error occurs, this port returns 0. Otherwise, it returns a nonzero value. For more information, see "EtherCAT Error Codes" on page 5-25.

Parameters

`Device index` — EtherCAT network identifier
0 (default) | 0-11

To associate a block with an EtherCAT network, enter the **Device index** value from the EtherCAT Init block representing that network into the **Device index** for the block.

Programmatic Use
Block Parameter: `device_id`

`Device Name` — Name of EtherCAT device
character vector

Select an EtherCAT device from the list of available devices provided by the ENI configuration file. The configuration file is selected by the EtherCAT Init block.

Programmatic Use
Block Parameter: `device_name`

`Sample time` — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time. -1 means that the sample time is inherited.

Programmatic Use
Block Parameter: `sample_time`

See Also
EtherCAT Init | EtherCAT Set State | EtherCAT Get State | EtherCAT Get Device State

Topics
"EtherCAT Configurator Component Mapping" on page 5-22
"EtherCAT Data Types" on page 5-23

External Websites
www.ethercat.org
www.beckhoff.com
www.acontis.com/en/

Introduced in R2021b

EtherCAT Sync SDO Upload

Read data synchronously from slave device represented by service data object
Library: Simulink Real-Time / EtherCAT

Description

The EtherCAT Sync SDO Upload block selects a CANopen register by **Index** value in the specified EtherCAT slave and sends a read request. The block then waits until it receives a response or until the timeout period is over.

The response to an operation takes several ticks of the main task sample time. Assign the synchronous blocks a sample time slower than the main task sample time.

Ports

Output

Data — Data received from slave device
numeric

Returns data received from the EtherCAT slave device.

Error — Report an EtherCAT network error
0 | integer

If no error occurs, this port transmits 0. Otherwise, it transmits a nonzero value. For a list of error codes, see "EtherCAT Error Codes" on page 5-25.

Parameters

Index — Index of CANopen register
integer

Specify the hexadecimal (for example, 0x7) or decimal index of the CANopen register.

If you specify an invalid index, the block returns a nonzero value through the Error output.

Programmatic Use
Block Parameter: index

Subindex — Subindex of CANopen register
integer

Specify the hexadecimal (for example, 0x7) or decimal subindex of the CANopen register.

If you specify an invalid subindex, the block returns a nonzero value through the Error output.

Programmatic Use
Block Parameter: `subIndex`

`Data Type` — Data type of CANopen register
`double` (default) | `single` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `boolean`

From the list, select the data type of the CANopen register.

If you select a data type that does not match the type of the entry, the block returns a nonzero value through the `Error` output.

Programmatic Use
Block Parameter: `sig_type`

`Dimension` — Dimension of CANopen register
1 (default)

Specify the row and column dimension of the CANopen register.

Enter the vector length as found in the CoE description for the slave in its manual.

Programmatic Use
Block Parameter: `sig_dim`

`Device index` — EtherCAT network identifier
0 (default) | 0-11

To associate a block with an EtherCAT network, enter the **Device index** value from the EtherCAT Init block representing that network into the **Device index** for the block.

Programmatic Use
Block Parameter: `device_id`

`Slave Name` — Name of slave that contains CANopen register
character vector

From the list, select the name of the slave that contains the CANopen register.

The block populates this drop-down list with the contents of the configuration file.

Programmatic Use
Block Parameter: `selected_slave`

`Sample time` — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time. `-1` means that the sample time is inherited.

Programmatic Use
Block Parameter: `sample_time`

`Timeout` — Time to wait for response from slave
numeric

The maximum number of milliseconds to wait for a response before returning a timeout error.

Programmatic Use
Block Parameter: `timeout`

See Also

EtherCAT Init | EtherCAT Sync SDO Download

Topics
"EtherCAT Configurator Component Mapping" on page 5-22
"EtherCAT Data Types" on page 5-23

External Websites
www.ethercat.org
www.beckhoff.com
www.acontis.com/en/

Introduced in R2020b

EtherCAT Sync SDO Download

Transmit data synchronously to slave device represented by service data object
Library: Simulink Real-Time / EtherCAT

Description

The EtherCAT Sync SDO Download block selects a CANopen register by **Index** value in the specified EtherCAT slave and sends a write request. The block then waits until it receives a response or until the timeout period is over.

The response to an operation takes several ticks of the main task sample time. Assign the synchronous blocks a sample time slower than the main task sample time.

Ports

Input

Data — Data to write to slave device
numeric

Input data for writing to the EtherCAT slave device.

Output

Error — Report an EtherCAT network error
0 | integer

If no error occurs, this port transmits 0. Otherwise, it transmits a nonzero value. For a list of error codes, see "EtherCAT Error Codes" on page 5-25.

Parameters

Index — Index of CANopen register
integer

Specify the hexadecimal (for example, 0x7) or decimal index of the CANopen register.

If you specify an invalid index, the block returns a nonzero value through the Error output.

Programmatic Use
Block Parameter: index

Subindex — Subindex of CANopen register
integer

Specify the hexadecimal (for example, 0x7) or decimal subindex of the CANopen register.

If you specify an invalid subindex, the block returns a nonzero value through the Error output.

Programmatic Use
Block Parameter: subIndex

Data Type — Data type of CANopen register
double (default) | single | int8 | uint8 | int16 | uint16 | int32 | uint32 | boolean

From the list, select the data type of the CANopen register.

If you select a data type that does not match the type of the entry, the block returns a nonzero value through the Error output.

Programmatic Use
Block Parameter: sig_type

Dimension — Dimension of CANopen register
1 (default)

Specify the row and column dimension of the CANopen register.

Enter the vector length as found in the CoE description for the slave in its manual.

Programmatic Use
Block Parameter: sig_dim

Device index — EtherCAT network identifier
0 (default) | 0-11

To associate a block with an EtherCAT network, enter the **Device index** value from the EtherCAT Init block representing that network into the **Device index** for the block.

Programmatic Use
Block Parameter: device_id

Slave Name — Name of slave that contains CANopen register
character vector

From the list, select the name of the slave that contains the CANopen register.

The block populates this drop-down list with the contents of the configuration file.

Programmatic Use
Block Parameter: selected_slave

Sample time — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time. -1 means that the sample time is inherited.

Programmatic Use
Block Parameter: sample_time

Timeout — Time to wait for response from slave
numeric

The maximum number of milliseconds to wait for a response before returning a timeout error.

Programmatic Use
Block Parameter: `timeout`

See Also

EtherCAT Init | EtherCAT Sync SDO Upload

Topics
"EtherCAT Configurator Component Mapping" on page 5-22
"EtherCAT Data Types" on page 5-23

External Websites

www.ethercat.org
www.beckhoff.com
www.acontis.com/en/

Introduced in R2020b

EtherCAT Async SDO Upload

Read data asynchronously from slave device represented by service data object
Library: Simulink Real-Time / EtherCAT

Description

The EtherCAT Async SDO Upload block selects a CANopen register by **Index** value in the specified EtherCAT slave and sends a read request. It then immediately returns whatever value was returned from the device on an earlier call to the block.

Ports

Input

`Enable` — Enables block to upload data
int32

A value of `0` disables uploads. A constant value of 1 will send a new request each time the status returns to the idle state.

Output

`Data` — Data received from slave device
numeric

Returns data received from the EtherCAT slave device.

`Status` — Status of data transfer
0 | 1 | 2 | 3

Status of asynchronous data transfer:

- `0` — Mailbox transfer object idle, transfer not running
- `1` — Mailbox transfer object running, transfer not complete
- `2` — Transfer successfully executed
- `3` — Error occurred during transfer request

`Error` — Report an EtherCAT network error
0 | integer

If no error occurs, this port transmits `0`. Otherwise, it transmits a nonzero value. For a list of error codes, see "EtherCAT Error Codes" on page 5-25.

Parameters

Index — Index of CANopen register
integer

Specify the hexadecimal (for example, 0x7) or decimal index of the CANopen register.

If you specify an invalid index, the block returns a 3 through the `Status` output and a nonzero value through the `Error` output.

Programmatic Use
Block Parameter: `index`

Subindex — Subindex of CANopen register
integer

Specify the hexadecimal (for example, 0x7) or decimal subindex of the CANopen register.

If you specify an invalid subindex, the block returns a nonzerio value through the `Error` output.

Programmatic Use
Block Parameter: `subIndex`

Data Type — Data type of CANopen register
double (default) | single | int8 | uint8 | int16 | uint16 | int32 | uint32 | boolean

From the list, select the data type of the CANopen register.

If you select a data type that does not match the type of the entry, the block returns a nonzero value through the `Error` output.

Programmatic Use
Block Parameter: `sig_type`

Dimension — Dimension of CANopen register
1 (default)

Specify the row and column dimension of the CANopen register.

Enter the vector length as found in the CoE description for the slave in its manual.

Programmatic Use
Block Parameter: `sig_dim`

Device index — EtherCAT network identifier
0 (default) | 0-11

To associate a block with an EtherCAT network, enter the **Device index** value from the EtherCAT Init block representing that network into the **Device index** for the block.

Programmatic Use
Block Parameter: `device_id`

Slave Name — Name of slave that contains CANopen register
character vector

From the list, select the name of the slave that contains the CANopen register.

The block populates this drop-down list with the contents of the configuration file.

Programmatic Use
Block Parameter: `selected_slave`

Sample time — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time. -1 means that the sample time is inherited.

Programmatic Use
Block Parameter: sample_time

See Also
EtherCAT Init | EtherCAT Async SDO Download

Topics
"EtherCAT Configurator Component Mapping" on page 5-22
"EtherCAT Data Types" on page 5-23

External Websites
www.ethercat.org
www.beckhoff.com
www.acontis.com/en/

Introduced in R2020b

EtherCAT Async SDO Download

Transmit data asynchronously to slave device represented by service data object

Library: Simulink Real-Time / EtherCAT

Description

The EtherCAT Async SDO Download block selects a CANopen register by **Index** value in the specified EtherCAT slave and sends a write request. The block then immediately continues processing its input data.

Ports

Input

Data — Data to write to slave device
numeric

Input data for writing to the EtherCAT slave device.

Enable — Enables block to download data
int32

A value of 0 disables downloads. A value greater than or equal to 1 enables the block to download data.

Output

Status — Status of data transfer
0 | 1 | 2 | 3

Status of asynchronous data transfer:

- 0 — Mailbox transfer object idle, transfer not running
- 1 — Mailbox transfer object running, transfer not complete
- 2 — Transfer successfully executed
- 3 — Error occurred during transfer request

Error — Report an EtherCAT network error
0 | integer

If no error occurs, this port transmits 0. Otherwise, it transmits a nonzero value. For a list of error codes, see "EtherCAT Error Codes" on page 5-25.

Parameters

Index — Index of CANopen register
integer

Specify the hexadecimal (for example, 0x7) or decimal index of the CANopen register.

If you specify an invalid index, the block returns a `3` through the `Status` output and a nonzero value through the `Error` output.

Programmatic Use
Block Parameter: `index`

Subindex — Subindex of CANopen register
integer

Specify the hexadecimal (for example, 0x7) or decimal subindex of the CANopen register.

If you specify an invalid subindex, the block returns a nonzerio value through the `Error` output.

Programmatic Use
Block Parameter: `subIndex`

Data Type — Data type of CANopen register
double (default) | single | int8 | uint8 | int16 | uint16 | int32 | uint32 | boolean

From the list, select the data type of the CANopen register.

If you select a data type that does not match the type of the entry, the block returns a nonzero value through the `Error` output.

Programmatic Use
Block Parameter: `sig_type`

Dimension — Dimension of CANopen register
1 (default)

Specify the row and column dimension of the CANopen register.

Enter the vector length as found in the CoE description for the slave in its manual.

Programmatic Use
Block Parameter: `sig_dim`

Device index — EtherCAT network identifier
0 (default) | 0-11

To associate a block with an EtherCAT network, enter the **Device index** value from the EtherCAT Init block representing that network into the **Device index** for the block.

Programmatic Use
Block Parameter: `device_id`

Slave Name — Name of slave that contains CANopen register
character vector

From the list, select the name of the slave that contains the CANopen register.

The block populates this drop-down list with the contents of the configuration file.

Programmatic Use
Block Parameter: `selected_slave`

Sample time — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time. -1 means that the sample time is inherited.

Programmatic Use
Block Parameter: sample_time

See Also
EtherCAT Init | EtherCAT Async SDO Upload

Topics
"EtherCAT Configurator Component Mapping" on page 5-22
"EtherCAT Data Types" on page 5-23

External Websites
www.ethercat.org
www.beckhoff.com
www.acontis.com/en/

Introduced in R2020b

EtherCAT Sync SSC/SoE Upload

Read data synchronously from slave device represented by service data object
Library: Simulink Real-Time / EtherCAT

```
EtherCAT Sync SSC/SoE Upload   Data
        Network Device 0
         S-0-0000             Error
```

Description

The EtherCAT Sync SSC/SoE Upload block provides synchronous SERCOS interface (SErial Real time COmmunication Specification) over EtherCAT (SoE) upload. The block selects an **IDN** in the specified slave and sends an upload (read) request. The block then waits until it receives a response to the request or until the timeout period expires.

The response to an operation takes several ticks of the main task sample time. Assign the synchronous blocks a sample time slower than the main task sample time.

Ports

Output

`Data` — Data received from slave device
numeric

Returns data received from the EtherCAT slave device. The data signal has the type specified in Data Type and a vector dimension given by Dimension.

`Error` — Report an EtherCAT network error
0 | integer

If no error occurs, this port transmits 0. Otherwise, it transmits a nonzero value. For list of error codes, see "EtherCAT Error Codes" on page 5-25.

Parameters

`IDN` — Identification Number
character vector

Identify parameters.

The documentation for your EtherCAT device specifies the **IDN** values. You can select the **IDN** as a character vector that represents a 16-bit integer (according to IEC 61800 -7 -204), such as `S-0-0150` or `P-0-0150` with:

- First field (bit 15): `S` for Standard data, `P` for product-specific data
- Second field (bit 14 - 12): `0..7` for parameter set
- Third field (bit 11 - 0): `0..4095` for data-block number

Programmatic Use
Block Parameter: `idn`

`Drive Number` — Index number of the drive
integer

Specify the decimal index of the drive.

SoE blocks apply to only motor controllers. A single slave can support one or more drive or motor channels. The drive number is the zero-based index of the drive or motor channel on this slave at which this block is aimed. In SoE terminology, the drive is the logic that sends control signals to the motor. Typically, this logic is a small processor inside the slave.

Programmatic Use
Block Parameter: `drive`

`Data Type` — Data type of the IDN
double (default) | single | int8 | uint8 | int16 | uint16 | int32 | uint32 | boolean

To identify the data type for the IDN, refer to the slave documentation for the description of the IDN and the data type it uses. From the list, select the data type of the **IDN**.

If you select a data type that does not match the type of the entry, the block returns a nonzero value through the `Error` output.

Programmatic Use
Block Parameter: `sig_type`

`Dimension` — Dimension of data for this IDN
1 (default)

Specify the row dimension of data for this IDN.

To identify the dimension of data (vector size) for the IDN, refer to the slave documentation for the description of the IDN and the number of data type values (the dimension) it uses. Enter the vector length as found in the SoE description for the slave in its manual.

Programmatic Use
Block Parameter: `sig_dim`

`Device index` — EtherCAT network identifier
0 (default) | 0-11

To associate a block with an EtherCAT network, enter the **Device index** value from the EtherCAT Init block representing that network into the **Device index** for the block.

Programmatic Use
Block Parameter: `device_id`

`Slave Name` — Name of slave that contains the IDN
character vector

From the list, select the name of the slave that contains the **IDN**.

The block populates this drop-down list with the contents of the configuration file.

6-35

Programmatic Use
Block Parameter: `slave_name`

`Sample time` — Sample time of block
`-1` (default) | numeric

Enter the base sample time or a multiple of the base sample time. `-1` means that the sample time is inherited. The EtherCAT Sync SSC/SoE Download block and the EtherCAT Sync SSC/SoE Uploadblock require at least three steps of the main EtherCAT processing task. Select a sample time that is three times that of the main task sample time or the model can overload and stop.

Programmatic Use
Block Parameter: `sample_time`

`Timeout` — Time to wait for response from slave
numeric

Enter the maximum number of milliseconds to wait for a response from the EtherCAT slave.

Programmatic Use
Block Parameter: `timeout`

See Also
EtherCAT Init | EtherCAT Sync SSC/SoE Download

Topics
"EtherCAT Configurator Component Mapping" on page 5-22
"EtherCAT Data Types" on page 5-23

External Websites
www.ethercat.org
www.beckhoff.com
www.acontis.com/en/

Introduced in R2020b

EtherCAT Sync SSC/SoE Download

Transmit data synchronously to slave device represented by service data object
Library: Simulink Real-Time / EtherCAT

Description

The EtherCAT Sync SSC/SoE Download block provides synchronous SERCOS interface (SErial Real time COmmunication Specification) over EtherCAT (SoE) download. The block selects an **IDN** in the specified slave and sends a download (write) request. The block then waits until it receives a response to the request or until the timeout period expires.

The response to an operation takes several ticks of the main task sample time. Assign the synchronous blocks a sample time slower than the main task sample time.

Ports

Input

Data — Data to write to slave device
numeric

Input data for writing to the EtherCAT slave device. The data signal has the type specified in **Data Type** and vector dimension given by **Dimension**.

Output

Error — Report an EtherCAT network error
0 | integer

If no error occurs, this port transmits 0. Otherwise, it transmits a nonzero value. For list of error codes, see "EtherCAT Error Codes" on page 5-25.

Parameters

IDN — Identification Number
character vector

Identify parameters.

The documentation for your EtherCAT device specifies the **IDN** values. You can select the **IDN** as a character vector that represents a 16-bit integer (according to IEC 61800 -7 -204), such as S-0-0150 or P-0-0150 with:

- First field (bit 15): S for Standard data, P for product-specific data
- Second field (bit 14 - 12): 0..7 for parameter set
- Third field (bit 11 - 0): 0..4095 for data-block number

Programmatic Use
Block Parameter: `idn`

Drive Number — Index number of the drive
integer

Specify the decimal index of the drive.

SoE blocks apply to only motor controllers. A single slave can support one or more drive or motor channels. The drive number is the zero-based index of the drive or motor channel on this slave at which this block is aimed. In SoE terminology, the drive is the logic that sends control signals to the motor. Typically, this logic is a small processor inside the slave.

Programmatic Use
Block Parameter: `drive`

Data Type — Data type of the IDN
double (default) | single | int8 | uint8 | int16 | uint16 | int32 | uint32 | boolean

To identify the data type for the IDN, refer to the slave documentation for the description of the IDN and the data type it uses. From the list, select the data type of the **IDN**.

If you select a data type that does not match the type of the entry, the block returns a nonzero value through the `Error` output.

Programmatic Use
Block Parameter: `sig_type`

Dimension — Dimension of data for this IDN
1 (default)

Specify the row dimension of data for this IDN.

To identify the dimension of data (vector size) for the IDN, refer to the slave documentation for the description of the IDN and the number of data type values (the dimension) it uses. Enter the vector length as found in the SoE description for the slave in its manual.

Programmatic Use
Block Parameter: `sig_dim`

Device index — EtherCAT network identifier
0 (default) | 0-11

To associate a block with an EtherCAT network, enter the **Device index** value from the EtherCAT Init block representing that network into the **Device index** for the block.

Programmatic Use
Block Parameter: `device_id`

Slave Name — Name of slave that contains the IDN
character vector

From the list, select the name of the slave that contains the **IDN**.

The block populates this drop-down list with the contents of the configuration file.

Programmatic Use
Block Parameter: `slave_name`

`Sample time` — Sample time of block
`-1` (default) | numeric

Enter the base sample time or a multiple of the base sample time. `-1` means that the sample time is inherited. The EtherCAT Sync SSC/SoE Download block and the EtherCAT Sync SSC/SoE Uploadblock require at least three steps of the main EtherCAT processing task. Select a sample time that is three times that of the main task sample time or the model can overload and stop.

Programmatic Use
Block Parameter: `sample_time`

`Timeout` — Time to wait for response from slave
numeric

Enter the maximum number of milliseconds to wait for a response from the EtherCAT slave.

Programmatic Use
Block Parameter: `timeout`

See Also
EtherCAT Init | EtherCAT Sync SSC/SoE Upload

Topics
"EtherCAT Configurator Component Mapping" on page 5-22
"EtherCAT Data Types" on page 5-23

External Websites
www.ethercat.org
www.beckhoff.com
www.acontis.com/en/

Introduced in R2020b

EtherCAT Async SSC/SoE Upload

Read data asynchronously from slave device represented by service data object
Library: Simulink Real-Time / EtherCAT

Description

The EtherCAT Async SSC/SoE Upload block provides asynchronous SERCOS interface (SErial Real time COmmunication Specification) over EtherCAT (SoE) upload. The block selects an **IDN** in the specified slave and sends an upload (read) request. After sending the request, the block immediately returns whatever value was returned from the device on an earlier call to the block.

Ports

Input

`Enable` — Enables block to upload data
int32

A value 0 disables uploads. A value greater than or equal to 1 enables the block to upload data.

Output

`Data` — Data received from slave device
numeric

Returns data received from the EtherCAT slave device. The data signal has the type specified in Data Type and a vector dimension given by Dimension.

`Status` — Status of data transfer
0 | 1 | 2 | 3

Status of asynchronous data transfer:

- 0 — Mailbox transfer object idle, transfer not running
- 1 — Mailbox transfer object running, transfer not complete
- 2 — Transfer successfully executed
- 3 — Error occurred during transfer request

`Error` — Report an EtherCAT network error
0 | integer

If no error occurs, this port transmits 0. Otherwise, it transmits a nonzero value. For list of error codes, see "EtherCAT Error Codes" on page 5-25.

Parameters

IDN — Identification Number
character vector

Identify parameters.

The documentation for your EtherCAT device specifies the **IDN** values. You can select the **IDN** as a character vector that represents a 16-bit integer (according to IEC 61800 -7 -204), such as `S-0-0150` or `P-0-0150` with:

- First field (bit 15): `S` for Standard data, `P` for product-specific data
- Second field (bit 14 - 12): `0..7` for parameter set
- Third field (bit 11 - 0): `0..4095` for data-block number

Programmatic Use
Block Parameter: `idn`

Drive Number — Index number of the drive
integer

Specify the decimal index of the drive.

SoE blocks apply to only motor controllers. A single slave can support one or more drive or motor channels. The drive number is the zero-based index of the drive or motor channel on this slave at which this block is aimed. In SoE terminology, the drive is the logic that sends control signals to the motor. Typically, this logic is a small processor inside the slave.

Programmatic Use
Block Parameter: `drive`

Data Type — Data type of the IDN
double (default) | single | int8 | uint8 | int16 | uint16 | int32 | uint32 | boolean

To identify the data type for the IDN, refer to the slave documentation for the description of the IDN and the data type it uses. From the list, select the data type of the **IDN**.

If you select a data type that does not match the type of the entry, the block returns a nonzero value through the `Error` output.

Programmatic Use
Block Parameter: `sig_type`

Dimension — Dimension of data for this IDN
1 (default)

Specify the row dimension of data for this IDN.

To identify the dimension of data (vector size) for the IDN, refer to the slave documentation for the description of the IDN and the number of data type values (the dimension) it uses. Enter the vector length as found in the SoE description for the slave in its manual.

Programmatic Use
Block Parameter: `sig_dim`

Device index — EtherCAT network identifier
0 (default) | 0-11

To associate a block with an EtherCAT network, enter the **Device index** value from the EtherCAT Init block representing that network into the **Device index** for the block.

Programmatic Use
Block Parameter: device_id

Slave Name — Name of slave that contains the IDN
character vector

From the list, select the name of the slave that contains the **IDN**.

The block populates this drop-down list with the contents of the configuration file.

Programmatic Use
Block Parameter: slave_name

Sample time — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time. -1 means that sample time is inherited.

Programmatic Use
Block Parameter: sample_time

Timeout — Time to wait for response from slave
numeric

Enter the maximum number of milliseconds to wait for a response from the EtherCAT slave.

Programmatic Use
Block Parameter: timeout

See Also
EtherCAT Init | EtherCAT Async SSC/SoE Download

Topics
"EtherCAT Configurator Component Mapping" on page 5-22
"EtherCAT Data Types" on page 5-23

External Websites
www.ethercat.org
www.beckhoff.com
www.acontis.com/en/

Introduced in R2020b

EtherCAT Async SSC/SoE Download

Transmit data asynchronously to slave device represented by service data object
Library: Simulink Real-Time / EtherCAT

Description

The EtherCAT Async SSC/SoE Download block provides asynchronous SERCOS interface (SErial Real time COmmunication Specification) over EtherCAT (SoE) download. The block selects an **IDN** in the specified slave and sends a download (write) request. After sending the request, the block immediately continues processing its input data.

Ports

Input

Data — Data to write to slave device
numeric

Input data for writing to the EtherCAT slave device. The data signal has the type specified in **Data Type** and vector dimension given by **Dimension**.

Enable — Enables block to download data
int32

The Enable input is level sensitive and the block remains enabled while the input is non-zero. To send a value just once, you can enable the block with a single sample time pulse. There is a lag of approximately three cycles after the pulse for the data to send.

A value `0` disables downloads. A value greater than or equal to `1` enables the block to download data.

Output

Status — Status of data transfer
0 | 1 | 2 | 3

Status of asynchronous data transfer:

*   `0` — Mailbox transfer object idle, transfer not running
*   `1` — Mailbox transfer object running, transfer not complete
*   `2` — Transfer successfully executed
*   `3` — Error occurred during transfer request

Error — Report an EtherCAT network error
0 | integer

If no error occurs, this port transmits `0`. Otherwise, it transmits a nonzero value. For list of error codes, see "EtherCAT Error Codes" on page 5-25.

Parameters

IDN — Identification Number
character vector

Identify parameters.

The documentation for your EtherCAT device specifies the **IDN** values. You can select the **IDN** as a character vector that represents a 16-bit integer (according to IEC 61800 -7 -204), such as `S-0-0150` or `P-0-0150` with:

- First field (bit 15): `S` for Standard data, `P` for product-specific data
- Second field (bit 14 - 12): `0..7` for parameter set
- Third field (bit 11 - 0): `0..4095` for data-block number

Programmatic Use
Block Parameter: `idn`

Drive Number — Index number of the drive
integer

Specify the decimal index of the drive.

SoE blocks apply to only motor controllers. A single slave can support one or more drive or motor channels. The drive number is the zero-based index of the drive or motor channel on this slave at which this block is aimed. In SoE terminology, the drive is the logic that sends control signals to the motor. Typically, this logic is a small processor inside the slave.

Programmatic Use
Block Parameter: `drive`

Data Type — Data type of the IDN
double (default) | single | int8 | uint8 | int16 | uint16 | int32 | uint32 | boolean

To identify the data type for the IDN, refer to the slave documentation for the description of the IDN and the data type it uses. From the list, select the data type of the **IDN**.

If you select a data type that does not match the type of the entry, the block returns a nonzero value through the `Error` output.

Programmatic Use
Block Parameter: `sig_type`

Dimension — Dimension of data for this IDN
1 (default)

Specify the row dimension of data for this IDN.

To identify the dimension of data (vector size) for the IDN, refer to the slave documentation for the description of the IDN and the number of data type values (the dimension) it uses. Enter the vector length as found in the SoE description for the slave in its manual.

Programmatic Use
Block Parameter: `sig_dim`

Device index — EtherCAT network identifier
0 (default) | 0-11

To associate a block with an EtherCAT network, enter the **Device index** value from the EtherCAT Init block representing that network into the **Device index** for the block.

Programmatic Use
Block Parameter: device_id

Slave Name — Name of slave that contains the IDN
character vector

From the list, select the name of the slave that contains the **IDN**.

The block populates this drop-down list with the contents of the configuration file.

Programmatic Use
Block Parameter: slave_name

Sample time — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time. -1 means that sample time is inherited.

Programmatic Use
Block Parameter: sample_time

Timeout — Time to wait for response from slave
numeric

Enter the maximum number of milliseconds to wait for a response from the EtherCAT slave.

Programmatic Use
Block Parameter: timeout

See Also
EtherCAT Init | EtherCAT Async SSC/SoE Upload

Topics
"EtherCAT Configurator Component Mapping" on page 5-22
"EtherCAT Data Types" on page 5-23

External Websites
www.ethercat.org
www.beckhoff.com
www.acontis.com/en/

Introduced in R2020b

TCP Transmission Control Protocol Blocks Library

Real-Time TCP Communication Support

TCP Transport Protocol

The Simulink Real-Time software supports communication from the target computer to other systems or devices by using Transmission Control Protocol (TCP). TCP provides ordered and error-checked packet transport.

TCP is a transport protocol layered on top of the Internet Protocol (IP). It is commonly known as TCP/IP.

- Stream — TCP is a *stream-oriented* protocol.

  TCP is a stream of data that flows from one end of the network to the other. Another stream of data flows in the other direction. The TCP stack at the transmitting end is responsible for breaking the stream of data into packets and sending those packets. The stack at the receiving end is responsible for reassembling the packets into a data stream by using information in the packet headers.

- Connection — TCP is a *connection-based* protocol.

  In TCP, the two ends of the communication link must be connected throughout the communication.

- Error Detection — TCP detects errors.

  TCP packets contain a unique sequence number. The starting sequence number is communicated from the transmitter to the receiver at the beginning of communication. The receiver acknowledges each packet. That acknowledgment contains the sequence number so that the sender knows which packet was acknowledged. Lost packets can be retransmitted. The sender knows that they did not reach their destination because the sender did not receive an acknowledgment. The receiver can reassemble in order packets that arrive out of sequence. Timeouts can be established because the sender knows from the first few packets how long it takes to transmit a packet and receive its acknowledgment.

TCP communication requires a continuous connection and two-way streaming data is exchanged.

When describing TCP, the words *reliable* and *unreliable* have a specific meaning.

- *Reliable* means that if a packet is not acknowledged, it is retransmitted. It does not mean that the protocol always succeeds.
- *Unreliable* means that if too many packets are not acknowledged, the protocol can time out. It does not mean that the protocol packets usually fail to arrive.

You can construct a packet from Simulink data types such as `double`, `int8`, `int32`, `uint8`, or a combination of these data types. The Simulink Real-Time block library provides blocks for combining various signals into one packet (packing), and then transmitting it. It also provides blocks for splitting a packet (unpacking) into its component signals that you can then use in a Simulink model.

This information applies to both communication with a shared Ethernet board and communication with a dedicated Ethernet board. Consider adding a dedicated Ethernet board for enhanced performance over communication that uses a shared Ethernet board. Shared TCP communication shares bandwidth with the link between the development and target computers.

See Also
TCP Receive | TCP Send | TCP Client | TCP Server
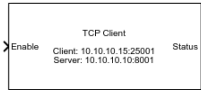
External Websites

- www.ietf.org/rfc/rfc793.txt

TCP Blocks

TCP Client

Configure TCP client
Library: Simulink Real-Time / IP

Description

The TCP Client block configures a TCP client.

The parameters **Client IP address** and **Client port** are optional. The **Client IP address** applies when the block executes in a real-time application on a target computer or in a model simulation on a development computer. If your model is running in Simulink on the development computer, you can use this block to transmit data to a remote device. In this case, the Windows® operating system determines the network connection.

Ports

Input

Enable — Connect block to remote Ethernet device
integer

If Enable is greater than zero, the block connects to the Ethernet device. Otherwise, the block does not connect to the Ethernet device.

Output

Status — Device returns a status of not connected or connected
0 | 1

The status value is one of:

- 0 — Not connected
- 1 — Connected

As a best practice, connect the Status output of a TCP configure block to the Enable input of the associated TCP Send and TCP Receive blocks.

Parameters

Use host-target connection — Use Ethernet connection between development and target computers
'off' (default) | 'on'

Dependency

When you select this parameter, it deactivates the **Client IP address** parameter and excludes the ports 1 through 1023 and 5500 through 5560 from use by TCP.

Programmatic Use
Block Parameter: `useHostTargetConn`

Remote server IP address — IP address of the server device
`x.x.x.x`

Enter the IP address of the server to which you want to connect the client.

Programmatic Use
Block Parameter: `remoteAddress`

Remote server port — Port number of the server device
1—65535

Enter the port number of the server to which you want to connect the client.

Ports 1 through 1023 and 5500 through 5560 are reserved for Simulink Real-Time communications.

Programmatic Use
Block Parameter: `remotePort`

Advanced Parameters

Client IP address — IP address of the client device that is being configured
no value (default) | `x.x.x.x`

If you are using a dedicated Ethernet card, this value must match the **Local IP Address** parameter in the IP Config block for the network interface. The default value for the Client IP address field is no value. This empty field means that the operating system chooses the Client IP address for TCP transmission.

Programmatic Use
Block Parameter: `clientAddress`

Client local port — IP port of the client device that is being configured
no value (default) | 0—65535

The combination of **Client IP address** and **Client local port** must be unique.

Ports 1 through 1023 and 5500 through 5560 are reserved for Simulink Real-Time communications. Values '-1', '0', or 'empty' mean that the block transmits data by using any available port.

Programmatic Use
Block Parameter: `clientPort`

See Also
TCP Server

External Websites
www.ietf.org/rfc/rfc793.txt

Introduced in R2020b

TCP Receive

Receive data over TCP network from a remote device
Library: Simulink Real-Time / IP
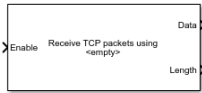
Description

The TCP Receive block receives data sent from a remote client device to a server application on a target computer.

Ports

Input

Enable — Allow data reception
integer

When `Enable > 0`, the block attempts to receive data sent to the remote device.

As a best practice, connect the `Status` output of a TCP configure block to the `Enable` input of the associated TCP Send and TCP Receive blocks.

Output

Data — Data that is received from the remote client
vector

The parameter **Receive width** determines the maximum size of the data vector.

Data Types: `uint8`

Length — Actual size of data vector
double

To test whether the number of data items exceeds the width of the data output port, use this value.

Parameters

Receive using — List of IP address and port pairs
`x.x.x.x:y`

This property is read-only.

The block receives the list of IP address and port pairs from the TCP configuration blocks in the model.

Ports `1` through `1023` and `5500` through `5560` are reserved for Simulink Real-Time communications.

Programmatic Use
Block Parameter: `socketAddPort`

Receive width — Maximum expected length of data vector
1–65504

Maximum number of `uint8` values that the block expects to receive from the client device.

Programmatic Use
Block Parameter: `rcvWidth`

Sample time — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time. -1 means that the sample time is inherited.

Programmatic Use
Block Parameter: `sampleTime`

See Also
TCP Client | TCP Server

External Websites
www.ietf.org/rfc/rfc793.txt

Introduced in R2020b

TCP Send

Send data over TCP network to a remote device
Library: Simulink Real-Time / IP
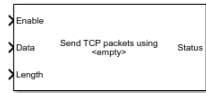
Description

The TCP Send block sends data from a server application on a target computer to a remote client device.

Ports

Input

Enable — Allow data transmission
integer

When Enable > 0, the block attempts to transmit data to the remote device.

As a best practice, connect the Status output of a TCP configure block to the Enable input of the associated TCP Send and TCP Receive blocks.

Data — Data to transmit over the TCP network
vector

Vector of Length to transmit to the client device.

Data Types: uint8

Length — Length of data vector
double

Number of uint8 values to transmit to the client device.

Output

Status — Number of bytes sent
double

Returns the number of uint8 values transmitted to the client device.

Parameters

Send using — List of IP address and port pairs
x.x.x.x:y

This property is read-only.

The block receives the list of IP address and port pairs from the TCP configuration blocks in the model.

Ports 1 through 1023 and 5500 through 5560 are reserved for Simulink Real-Time communications.

Programmatic Use
Block Parameter: `socketAddPort`

Sample time — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time. -1 means that the sample time is inherited.

Programmatic Use
Block Parameter: `sampleTime`

See Also
TCP Client | TCP Server

External Websites
www.ietf.org/rfc/rfc793.txt

Introduced in R2020b

TCP Server

Configure TCP server
Library: Simulink Real-Time / IP
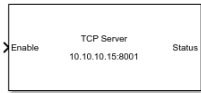
Description

The TCP Server block configures a TCP server.

Ports

Input

Enable — Connect block to remote Ethernet device
integer

If Enable is greater than zero, the block connects to the Ethernet device. Otherwise, the block does not connect to the Ethernet device.

Output

Status — Device returns a status of not connected or connected
0 | 1

The status value is one of:

- 0 — Not connected
- 1 — Connected

As a best practice, connect the Status output of a TCP configure block to the Enable input of the associated TCP Send and TCP Receive blocks.

Parameters

Use host-target connection — Use Ethernet connection between development and target computers
'off' (default) | 'on'

Dependency

When you select this parameter, it deactivates the **Server IP address** parameter and excludes the ports 1 through 1023 and 5500 through 5560 from use by TCP.

Programmatic Use
Block Parameter: useHostTargetConn

Server IP address — IP address of the server device that is being configured
x.x.x.x

If you are using a dedicated Ethernet card, this value must match the IP address configured for the Ethernet card on the target computer.

Programmatic Use
Block Parameter: `serverAddress`

`Server port` — IP port of the server device that is being configured
1–65535

The combination of **Server IP address** and **Server port** must be unique.

Ports 1 through 1023 and 5500 through 5560 are reserved for Simulink Real-Time communications.

Programmatic Use
Block Parameter: `serverPort`

See Also

TCP Client

External Websites
www.ietf.org/rfc/rfc793.txt

Introduced in R2020b

Real-Time UDP Communication Support

UDP Transport Protocol

The Simulink Real-Time software supports communication from the target computer to other systems or devices by using User Datagram Protocol (UDP) packets. UDP is a transport protocol that provides a direct method to send and receive packets over an IP network. UDP uses this direct method at the expense of reliability by limiting error checking and recovery.

UDP is a transport protocol layered on top of the Internet Protocol (IP). It is commonly known as UDP/IP.

- Packet — UDP is a *packet-oriented* protocol. You divide the data into packets and the protocol sends them to the receiver.
- Connectionless — UDP is a *connectionless* protocol. The protocol sends a packet to the receiver without checking to see if the receiver is ready to receive a packet. If the receiver is not ready, the packet is lost.
- No Error Detection— UDP does not support error detection. The protocol sends packets and does not track them. If packets arrive out of sequence, or are lost in transmission, the receiving end or the sending end does not know.

UDP communication requires that the sender identifies the receiver. If the receiver is not found or the communication is lost in transit, the packet is discarded.

When describing UDP, the words *reliable* and *unreliable* have a specific meaning.

- *Reliable* means that the protocol is not guaranteed to succeed. It does not mean that the protocol always succeeds.
- *Unreliable* means that protocol packets can fail to arrive without the system detecting that the packets did not arrive. It does not mean that the protocol packets usually fail to arrive.

UDP continues to receive packets while the receiver is active and processes data as quickly as it arrives.

UDP is a commonly used transport layer because of its lightweight nature. When used from Simulink Real-Time, UDP gives the real-time application a good chance of succeeding in real-time execution. Also, the datagram nature of UDP is optimal for sending samples of data from the real-time application generated by the Simulink Coder™ software. If the real-time application cannot process the data as quickly as it arrives, only the most recent packet is used. The earlier packets are ignored.

You can construct a packet from Simulink data types such as `double`, `int8`, `int32`, `uint8`, or a combination of these data types. The Simulink Real-Time block library provides blocks for combining various signals into one packet (packing), and then transmitting it. It also provides blocks for splitting a packet (unpacking) into its component signals that you can then use in a Simulink model.

The preceding information applies to communication with a shared Ethernet board and communication with a dedicated Ethernet board. Consider adding a dedicated Ethernet board for enhanced performance over communication by using a shared Ethernet board. Shared UDP communication shares bandwidth with the link between the development and target computers.

See Also

UDP Send | UDP Receive

Related Examples

- "Target to Host Transmission by Using UDP"

More About

- "UDP Communication Setup" on page 9-9
- "UDP and Variable-Size Signals" on page 9-10

UDP Data Exchange by Using Shared Ethernet Board

In this section...
"UDP Data Transfer" on page 9-4
"Set Up slrt_ex_udpsendreceiveA" on page 9-5
"Set Up slrt_ex_udpsendreceiveB" on page 9-6

This example shows how to set up two-way data exchange by using an Ethernet board that is shared with the connection between the development and target computers. Using this configuration, you can communicate between two Simulink Real-Time systems, between the Simulink Real-Time and Simulink products, or between two Simulink models. When one or both of the systems are running as a non-real-time Simulink model, be sure to set the sample time.

This example does not require configuring a dedicated Ethernet card because the example uses the connection between the development and target computers.

The example models are named `slrt_ex_udpsendreceiveA` and `slrt_ex_udpsendreceiveB`. Replace the port and IP address examples with ports and addresses as required by your network. This example uses a target computer located at IP address `192.168.7.5` and uses a development computer located at IP address `192.168.7.2`.

UDP Data Transfer

The models transfer two different data sets between them, one data set from `slrt_ex_udpsendreceiveA` to `slrt_ex_udpsendreceiveB` and another data set in the opposite direction.

For this example, the inputs are generated by using Simulink Constant blocks that use the MATLAB random number function (`rand`). The Simulink Coder software uses this function during code generation to generate random numbers. To generate the vector of `uint8` (3x3), use the MATLAB function:

```
uint8(255 * rand(3,3))
```

because `255` is the maximum value for an unsigned 8-bit integer. The other values are generated similarly.

slrt_ex_udpsendreceiveA to slrt_ex_udpsendreceiveB

The UDP data to send is 75 bytes wide. The data to transfer is in these formats:

- [3 3] of `uint8` (9 bytes)
- [1 1] of `uint16` (2 bytes)
- [2 4] of `double` (64 bytes)

When packed, the data is aligned on 1-byte boundaries.

slrt_ex_udpsendreceiveB to slrt_ex_udpsendreceiveA

The UDP data to be sent is 79 bytes wide. The data to transfer is in these formats:

- [4 1] of `single` (16 bytes)
- [2 2] of `double` (32 bytes)
- [2 2] of `uint32` (16 bytes)
- [5 3] of `int8` (15 bytes)

When packed, the data is aligned on 2-byte boundaries. A zero-valued pad byte is added during packing.

Set Up slrt_ex_udpsendreceiveA

The final `slrt_ex_udpsendreceiveA` is shown in the figure.

The tables list the parameters for the send and receive sides of the model.
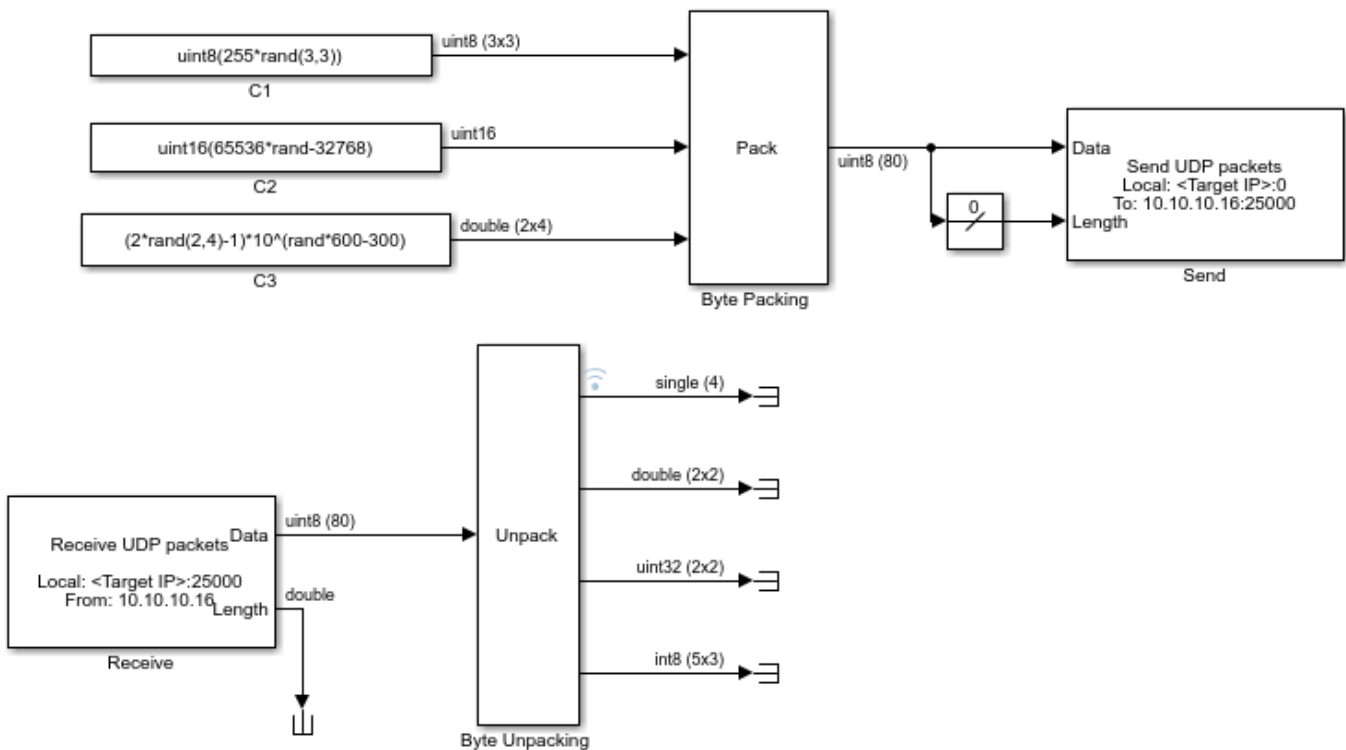
slrt_ex_udpsendreceiveA Send Side

Block	Parameter	Value
Byte Packing	**Output port (packed) data type**	`'uint8'`
	Input port (unpacked) data types (cell array)	`{'uint8', 'uint16', 'double'}`
	Byte alignment	1
UDP Send	**Local IP address**	`Use host-target connection`

Block	Parameter	Value
	Local port	`-1` (autoselect)
	To IP address	`192.168.7.5`
	To port	`25000`
	Sample time (-1 for inherited)	`0.01`

- The `Length` input port receives the output of a Width block that calculates the width of the signal connected to the `Data` port.
- The Byte Packing block settings match the Byte Unpacking block of `slrt_ex_udpsendreceiveB`.

slrt_ex_udpsendreceiveA Receive Side

Block	Parameter	Value
UDP Receive	**Local IP address**	`Use host-target connection`
	Local port	`25000`
	Receive width	`80`
	Receive from any source	`off`
	From IP address	`192.168.7.2`
	Sample time (-1 for inherited)	`0.01`
Byte Unpacking	**Output port (unpacked) data types (cell array)**	`{'single', 'double', 'uint32', 'int8'}`
	Output port (unpacked) dimensions (cell array)	`{4, [2 2], [2 2], [5 3]}`
	Byte alignment	`2`

- The second output port of the UDP Receive block is sent into a terminator. You can use this output to determine when a packet has arrived. The same is true for the outputs of the Byte Unpack block.
- The **Receive width** of the UDP Receive block matches the output port width of the Byte Packing block in `slrt_ex_udpsendreceiveB`.
- The Byte Unpacking block settings match the settings of the Byte Packing block of `slrt_ex_udpsendreceiveB`.
- The number of unpacked bytes is 79. The byte alignment is 2. The Byte Unpacking block assumes that the input vector includes a pad 0 to align the vector on an even-numbered boundary.

Set Up slrt_ex_udpsendreceiveB

The final `slrt_ex_udpsendreceiveB` model is shown in the figure.

The tables list the parameters for the receive side and the send side of the model.
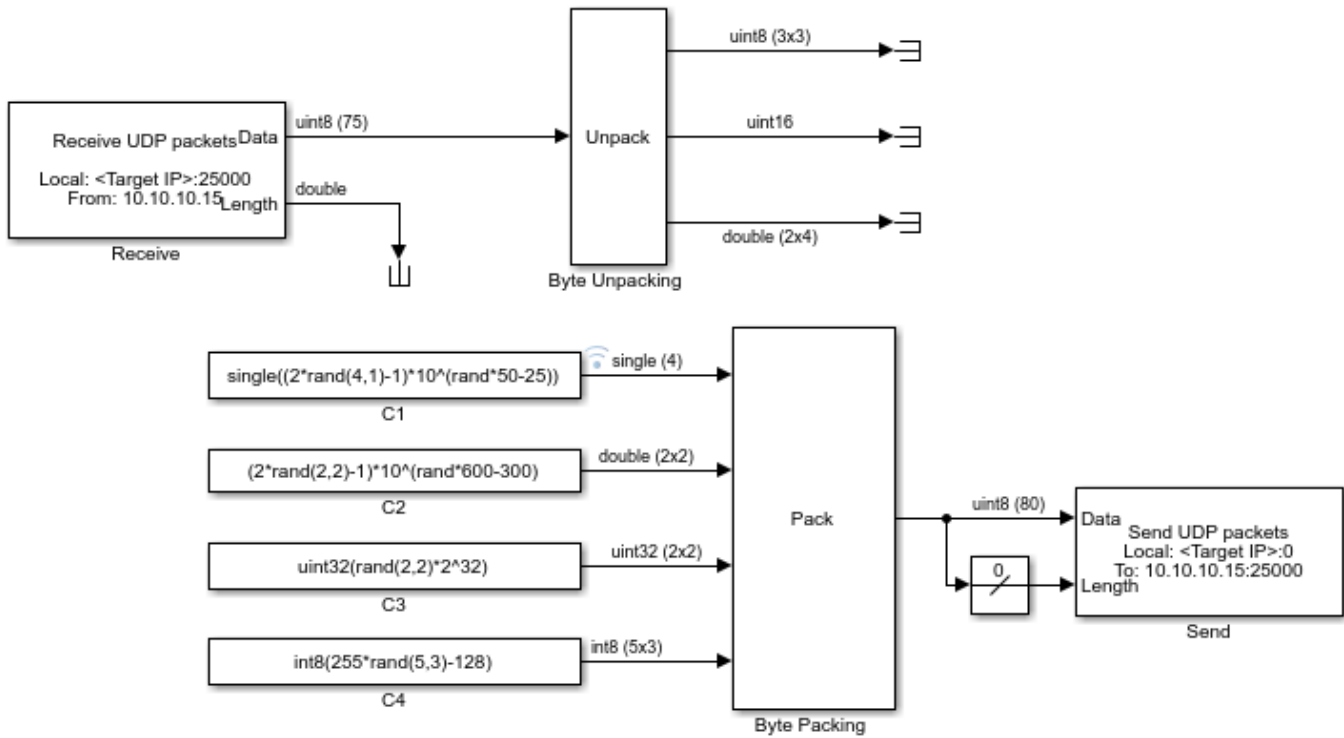
slrt_ex_udpsendreceiveB Receive Side

Block	Parameter	Value
UDP Receive	**Local IP address**	`Use host-target connection`
	Local port	`25000`
	Receive width	`75`
	Receive from any source	`off`
	From IP address	`192.168.7.5`
	Sample time (-1 for inherited)	`0.01`
Byte Unpacking	**Output port (unpacked) data types (cell array)**	`{'uint8', 'uint16', 'double'}`
	Output port (unpacked) dimensions (cell array)	`{[3 3], 1, [2 4]}`
	Byte alignment	`1`

- The second output port of the UDP Receive block is sent into a terminator. You can use this output to determine when a packet has arrived. The same is true for the outputs of the Byte Unpack block.

- The **Receive width** of the UDP Receive block matches the output port width of the Byte Packing block in `slrt_ex_udpsendreceiveA`.

- The Byte Unpacking block settings match the Byte Packing block in `slrt_ex_udpsendreceiveA`.

slrt_ex_udpsendreceiveB Send Side

Block	Parameter	Value
Byte Packing	**Output port (packed) data type**	`'uint8'`
	Input port (unpacked) data types (cell array)	`{'single', 'double', 'uint32', 'int8'}`
	Byte alignment	`2`
UDP Send	**Local IP address**	`Use host-target connection`
	Local port	`-1` (autoselect)
	To IP address	`192.168.7.2`
	To port	`25000`
	Sample time (-1 for inherited)	`0.01`

- The `Length` input port receives the output of a Width block that calculates the width of the signal connected to the `Data` port.
- The Byte Packing block settings match the settings of the Byte Unpacking block of `slrt_ex_udpsendreceiveA`.
- The number of unpacked bytes is `79`. The byte alignment is `2`. The Byte Packing block pads the output vector with `0` to align on an even-numbered boundary.

See Also
UDP Send | UDP Receive

Related Examples
- "Target to Host Transmission by Using UDP"

More About

UDP Communication Setup

The infrastructure provided in the Simulink Real-Time Library for UDP communication consists of two blocks: a UDP Send block and a UDP Receive block. These blocks are in the Simulink Real-Time Library, available from the Simulink Library under **Simulink Real-Time**. You can also access them from the MATLAB command line by typing:

```
slrealtimelib
```

The blocks are located under the IP heading in the library. The UDP Send block takes as input a vector of type uint8. The UDP Receive block outputs a vector of uint8. To convert arbitrary Simulink data types into this vector of uint8, use a Byte Packing block. To convert a vector of uint8s back into arbitrary Simulink data types, use a Byte Unpacking block.

If you are using a dedicated Ethernet port for UDP communication, use the Speedgoat Ethernet Configuration utility to configure the dedicated Ethernet board. For more information, see "Troubleshoot Model Upgrade for R2020b".

To communicate with *big-endian* architecture systems, use the Byte Reversal/Change Endianess block. Your model does not need this block for communicating between 80x86-based computer systems running either the Simulink Real-Time RTOS or the Microsoft Windows operating system.

The blocks work from within the Simulink environment and from a real-time application running under the Simulink Real-Time system. Be careful about transmitting data between a Simulink simulation and a real-time application or using two Simulink models. A Simulink model is not a real-time model and can run several times faster or slower than a real-time application. Set the sample time of the UDP Send and UDP Receive blocks and the sample time of the Simulink model so that the blocks can communicate.

- You cannot configure two UDP Receive blocks with the same local port. For example, two UDP Receive blocks cannot have the same local port and different IP addresses.
- You cannot configure two UDP Send blocks with the same local port. For example, two UDP Send blocks cannot have the same local port and different IP addresses.

See Also
UDP Send | UDP Receive

Related Examples
- "Target to Host Transmission by Using UDP"

More About
- "UDP Transport Protocol" on page 9-2

UDP and Variable-Size Signals

The Simulink Real-Time UDP sublibrary does not directly support variable-size signals. The UDP Send block input port accepts only fixed-size signals.

To send variable-size signals though UDP, determine the maximum number of elements of a fixed-size input signal that you expect to connect to the block. Use the second input, Length, to specify the number of elements of this input signal to send through UDP.

This example configures the MATLAB Function block to accept a variable-size signal and maps that signal to a fixed-size output signal. It outputs the number of relevant elements. You can output the fixed-size output signal and number of elements to the inputs of the UDP Send block.

1 To accept a variable-size input signal, create a MATLAB Function block.

2 In the MATLAB Function block, enter code like the following code. In this code, the maximum size of the variable-size input signal is 9.

```
function [y,y_length] = fcn(u)
%#codegen
y = uint8(zeros(9,1));
y_length = length(u);
for a = 1:y_length
    y(a) = u(a);
end
```

3 In the MATLAB Function Editor, click the **Edit Data** button. In **Ports and Data Manager**, select the data u, and then select the corresponding **Variable size** check box.

4 Select the data y and enter the size of the variable-size data input signal in the corresponding **Size** parameter. For this example, the size value is 9.

5 Provide a variable-size signal source for the MATLAB Function block.
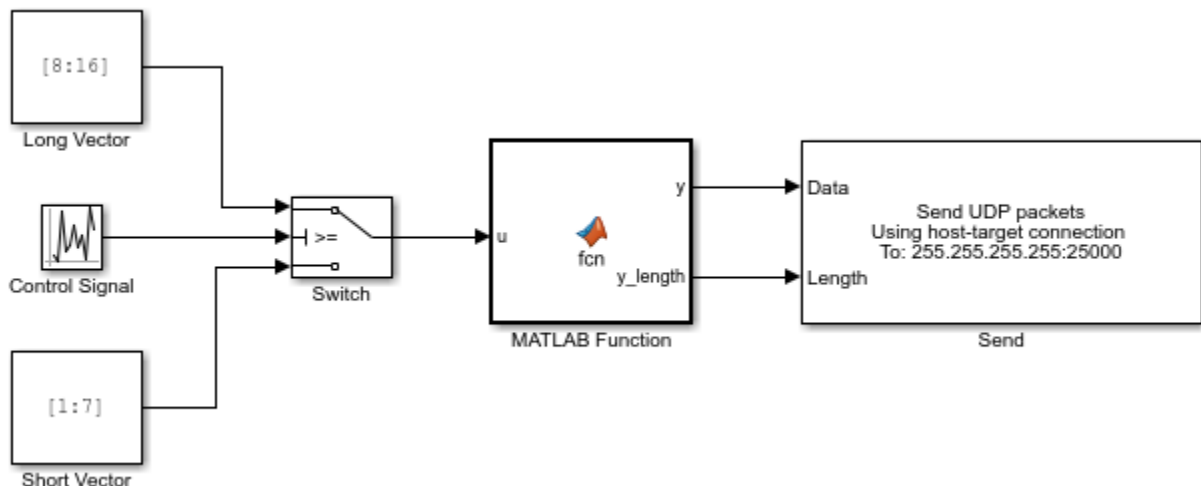
See Also
UDP Send | MATLAB Function

Real-Time UDP Blocks

UDP Multicast Receive

Receive data over UDP network from a remote device
Library: Simulink Real-Time / IP
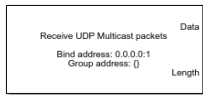
Description

The UDP Multicast Receive block receives multicast data over a UDP network from a remote device. It can receive data by using the connection between the development and target computers or by using a dedicated Ethernet card. If you use a dedicated Ethernet card, use the Speedgoat configuration utility to configure the dedicated Ethernet board.

The UDP Multicast Receive block operates in a real-time application running on a target computer. The block does not operate in a model simulation on a development computer.

A maximum of 20 multicast groups can be joined. This number is determined as a product of the group address and group interface fields on the block mask.

Ports

Output

Data — Data received
vector

Vector of `uint8` containing data received over the UDP network. If no new packet is received, the data values are held. To determine whether a new packet has been received, use the **Length** output port.

Data Types: `uint8`

Length — Number of bytes received
double

Number of bytes in the new packet received, otherwise 0. If more bytes are received than can be output through the receive port with width defined by **Receive width**, the excess bytes are discarded.

Parameters

General Parameters

Bind address — Bind IP address for receiving multicast data
`0.0.0.0` (default)

The **Bind address** can be either `0.0.0.0` or a multicast address. When **Bind address** is set to `0.0.0.0`, the block binds to INADDR_ANY, which enables the socket to receive datagrams on all

interfaces. This specification enables the **Group address** field on the block mask. When **Bind address** is set to a multicast IP address, the **Group address** field is hidden on the block mask.

If the **Bind address** is set to `0.0.0.0`, the block also can receive unicast messages if the block is configured on a group interface over which unicast messages are sent. This unicast message receive occurs because address `0.0.0.0` allows listening to any messages on all interfaces of the target computer. To avoid this unicast message receive operation, use a multicast address as a bind address. Using that setup, the block only receives multicast messages and does not receive unicast messages.

Programmatic Use
Block Parameter: `bindAddress`

`Local port` — Destination UDP port to receive data
1–65535

Specifies UDP port to receive data.

Ports `1` through `1023` and `5500` through `5560` are reserved for Simulink Real-Time communications.

Programmatic Use
Block Parameter: `localPort`

`Receive width` — Width of `Data` output vector
16 (default) | 1–65504

Determines the width of the `Data` output vector. If this value is less than the number of bytes in the received packet, the excess bytes are discarded.

Programmatic Use
Block Parameter: `rcvWidth`

`Group address` — Multicast group to join
{} (default) | {'x.x.x.x'} | dotted decimal character vector

This field is hidden when the bind address is a multicast address. Enter a valid IP address as a dotted decimal character vector, for example, `{'224.0.0.0'}`. One or more group addresses can be specified.

The UDP Multicast Receive block issues an error at model update if the group IP address is not a valid multicast address in the range `224.0.0.0` through `239.255.255.255`.

Example: `{'224.100.1.1'}`

Example: `{'224.100.1.1', '224.100.1.2'}`

Programmatic Use
Block Parameter: `multicastAddress`

`Group interface` — Local IP interface from which to receive data
{} (default) | {'x.x.x.x'} | dotted decimal character vector

The **Group interface** IP address specifies the interfaces over which incoming multicast messages should be received. When the bind address is `0.0.0.0`, the multicast groups specified in the field **Group address** receive messages over the specified group interface or interfaces. When the bind address is a multicast address, that address receives messages over the specified group interface or interfaces. Enter a valid interface IP address as a dotted decimal character vector, for example, `{'192.168.7.5'}`. You can also use a MATLAB expression that returns a valid IP address as a character vector. One or more group interfaces can be specified.

Example: {'224.100.1.1'}

Example: {'224.100.1.1', '224.100.1.2'}

Programmatic Use
Block Parameter: multicastInterface

Sample time (-1 for inherited) — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time.

Programmatic Use
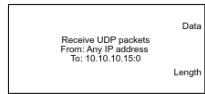**Block Parameter:** sampleTime

See Also
UDP Receive | UDP Send

Topics
"UDP Transport Protocol" on page 9-2

Introduced in R2020b

UDP Receive

Receive data over UDP network from a remote device
Library: Simulink Real-Time / IP

Description

The UDP Receive block receives data over a UDP network from a remote device. It can receive data by using the connection between the development and target computers or by using a dedicated Ethernet card. If you use a dedicated Ethernet card, use the Speedgoat configuration utility to configure the dedicated Ethernet board.

The parameter **Local IP address** applies when the block executes in a real-time application on a target computer or in a model simulation on a development computer. If your model is running in Simulink on the development computer, you can use this block to receive data to a remote device. In this case, the Windows operating system determines the network connection.

Ports

Output

Data — Data received
vector

Vector of `uint8` containing data received over the UDP network. If no new packet is received, the data values are held. To determine whether a new packet has been received, use the **Length** output port.

Data Types: `uint8`

Length — Number of bytes received
double

Number of bytes in the new packet received, otherwise `0`. If more bytes are received than can be output through the receive port with width defined by **Receive width**, the excess bytes are discarded.

Parameters

General Parameters

Use host-target connection — Use Ethernet connection between development and target computers
`'off'` (default) | `'on'`

Dependency

When you select this parameter, it deactivates the **Local IP address** parameter and excludes the ports 1 through 1023 and 5500 through 5560 from use by UDP.

Programmatic Use
Block Parameter: useHostTargetConn

Local IP address — Destination IP address for receiving data
Use host-target connection (default)

When **Local IP address** is set to Use host-target connection, the block uses the connection between the development and target computers. Use 0.0.0.0 to bind to INADDR_ANY, which enables the socket to receive broadcast datagrams.

Programmatic Use
Block Parameter: ipAddress

Local port — Destination UDP port to receive data
1–65535

Specifies UDP port to receive data.

Ports 1 through 1023 and 5500 through 5560 are reserved for Simulink Real-Time communications.

Programmatic Use
Block Parameter: localPort

Receive width — Width of Data output vector
1–65504

Determines the width of the Data output vector. If this value is less than the number of bytes in the received packet, the excess bytes are discarded.

Programmatic Use
Block Parameter: rcvWidth

Receive from any source — Causes receiver to accept data from any IP address
on (default) | off

When **Receive from any source** is on, the block receives data from any accessible IP address. When it is off, the block receives data from only the address that you specify in **From IP address**.

To make the **From IP address** parameter visible, clear the **Receive from any source** check box.

Programmatic Use
Block Parameter: rcvFmAny

From IP address — Source from which to receive data
x.x.x.x

Enter a valid IP address as a dotted decimal character vector, for example, 192.168.7.2. You can also use a MATLAB expression that returns a valid IP address as a character vector.

The address 255.255.255.255 is an invalid IP address.

To make this parameter visible, clear the **Receive from any source** check box.

Programmatic Use
Block Parameter: fmAddress

Sample time (-1 for inherited) — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time.

Programmatic Use
Block Parameter: sampleTime

See Also
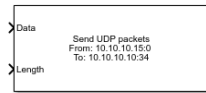UDP Multicast Receive | UDP Send

Topics
"UDP Transport Protocol" on page 9-2

Introduced in R2020b

UDP Send

Send data over UDP network to a remote device
Library: Simulink Real-Time / IP

Description

The UDP Send block sends data over a UDP network to a remote device. The block can send data by using the connection between the development and target computers or by using a dedicated Ethernet card. If you use a dedicated Ethernet card, use the Speedgoat configuration utility to configure the dedicated Ethernet board. One of the settings provided by this utility is the **Default Gateway** setting. When using the UDP Send block to broadcast to 255.255.255.255, the IP interface for broadcast is based on this **Default Gateway** setting.

To broadcast to all devices on the local subnetwork, set **To IP address** to 255.255.255.255. Otherwise, set **To IP address** to a valid IP address.

For unicasting, broadcasting and multicasting, the parameters **Local IP address** and **Local port** are optional. The **Local IP address** applies when the block executes in a real-time application on a target computer or in a model simulation on a development computer. If your model is running in Simulink on the development computer, you can use this block to transmit data to a remote device. In this case, the Windows operating system determines the network connection.

Ports

Input

Data — Data to transmit
vector

Vector of `uint8` containing data to transmit over the UDP network. To determine how many bytes of data to transmit, use the **Length** input port.

Data Types: `uint8`

Length — Number of bytes of data to transmit
double

Determines the number of bytes of data to transmit. Specify the width of the `Data` vector as the maximum number of bytes that you expect to transmit.

Parameters

Use host-target connection — Use Ethernet connection between development and target computers
`'off'` (default) | `'on'`

Dependency

When you select this parameter, it deactivates the **Local IP address** parameter and excludes the ports 1 through 1023 and 5500 through 5560 from use by UDP.

Programmatic Use
Block Parameter: useHostTargetConn

To IP address — IP address of target device
10.10.10.10 (default) | *x.x.x.x*

Specifies IP address of target device. To broadcast to all devices on the local subnetwork, send to 255.255.255.255.

Programmatic Use
Block Parameter: toAddress

To port — UDP port of target device
34 (default) | 1–65535

Specify the UDP port of target device. With **To IP address**, this parameter defines the destination of the data transmission.

Programmatic Use
Block Parameter: toPort

Sample time (-1 for inherited) — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time.

Programmatic Use
Block Parameter: sampleTime

Advanced Parameters

Local IP address — Source IP address for sending data
no value (default) | x.x.x.x

When **Local IP address** is set to **Use host-target connection**, the block uses the connection between the development and target computers. When **To IP address** is specified as a multicast address, the local IP address is used to determine the outbound interface over which multicast datagrams are sent. The default value for the Local IP address field is no value. This empty field means that the operating system chooses the Local IP address for UDP transmission.

Programmatic Use
Block Parameter: ipAddress

Local port — Source UDP port to transmit data
no value (default) | 0–65535 | -1

Specifies local UDP port to transmit data.

Ports 1 through 1023 and 5500 through 5560 are reserved for Simulink Real-Time communications.

The value 0, −1, or no value mean that the block transmits data by using any available port.

Programmatic Use
Block Parameter: `localPort`

See Also
UDP Multicast Receive | UDP Receive

Topics
"UDP Transport Protocol" on page 9-2

Introduced in R2020b

SAE J1939 Blocks Library

SAE J1939

SAE J1939 Blocks

The Simulink Real-Time J1939 blocks enable you to send and receive messages over a FIFO-mode CAN network by using the SAE J1939 message protocol. See "CAN".

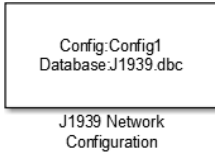Before you start, provide a J1939 database in `.dbc` format.

See Also

More About

- "CAN"

SAE J1939 Blocks

J1939 Network Configuration

Define J1939 network configuration name and database file
Library: Simulink Real-Time / J1939 Communication
 Vehicle Network Toolbox / J1939 Communication

Config:Config1
Database:J1939.dbc

J1939 Network
Configuration

Description

The J1939 Network Configuration block is where you define a configuration name and specify the associated user-supplied J1939 database. You can include more than one block per model, each corresponding to a unique configuration on the CAN bus.

To use this block, you must have a license for both Vehicle Network Toolbox™ and Simulink software.

The J1939 communication blocks support the use of Simulink accelerator and rapid accelerator modes. You can speed up the execution of Simulink models by using these modes. For more information on these modes, see the Simulink documentation.

The J1939 communication blocks also support code generation that have limited deployment capabilities. Code generation requires a C++ compiler that is compatible with the code generation target. For the current list of supported compilers, see Supported and Compatible Compilers.

Parameters

Configuration name — Define a name for this J1939 network configuration
ConfigX (default) | character vector

The default value is ConfigX, where the number X increases from 1 based on the number of existing blocks.

Database File — Specify the J1939 database file name relative to the current folder
not set (default) | character vector

An example file name, enter J1939.dbc if the file is in the current folder; otherwise enter the full path with the file name, such as C:\work\J1939.dbc.

The database file defines the J1939 parameter groups and nodes. This file must be in the DBC file format defined by Vector Informatik GmbH.
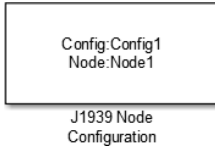
See Also

J1939 CAN Transport Layer | J1939 Receive | J1939 Transmit | J1939 Node Configuration

Introduced in R2015b

J1939 Node Configuration

Configure J1939 node with address and network management attributes

Library: Simulink Real-Time / J1939 Communication
                     Vehicle Network Toolbox / J1939 Communication

```
Config:Config1
  Node:Node1

  J1939 Node
  Configuration
```

Description

The J1939 Node Configuration block is where you define a node and associate it with a specific network configuration. Its Message information is read from the database for that configuration, unless you are creating and configuring a custom node.

To use this block, you must have a license for both Vehicle Network Toolbox and Simulink software.

The J1939 communication blocks support the use of Simulink accelerator and rapid accelerator modes. You can speed up the execution of Simulink models by using these modes. For more information on these modes, see "Design Your Model for Effective Acceleration".

The J1939 communication blocks also support code generation that have limited deployment capabilities. Code generation requires a C++ compiler that is compatible with the code generation target. For the current list of supported compilers, see Supported and Compatible Compilers.

Ports

Output

`Address` — Returns the effective address of the node
int8

This optional output port exists when you select the **Output current node address** check box in the dialog box.

`AC Status` — Indicates the success (1) or failure (0) of the node's address claim
0 | 1

This optional output port exists when you select the **Output address claim status** check box in the dialog box.

Parameters

`Config name` — ID of the J1939 network configuration to associate with this node
ConfigX (default) | character vector

To access the corresponding J1939 database, use this ID.

Node name — name of this J1939 node
NodeX (default) | character vector

The available list shows none if no J1939 network configuration is found or no node is defined in the associated database. If you are creating a custom node, the node name must be unique within its J1939 network configuration.

Message — Nine network attributes as defined by the database file consistent with the J1939 protocol
vector array

Unless you are defining a custom node, these parameters are read-only:

- **Allow arbitrary address** — Allow/disallow the node to switch to an arbitrary address if the station address is not available. If this option is off and the node loses its address claim, the node goes silent.

  Node Address — Station address, decimal, 8-bit.
- **Industry Group** — Decimal, 3-bit.
- **Vehicle System** — Decimal, 7-bit.
- **Vehicle System Instance** — Identifies one particular occurrence of a given vehicle system in a given network. If only one instance of a certain vehicle system exists in a network, then this field must be set to 0 to define it as the first instance. Decimal, 4-bit.
- **Function ID** — Decimal, 8-bit.
- **Function Instance** — Identifies the particular occurrence of a given function in a vehicle system and given network. If only one instance of a certain function exists in a network, then this field must be set to 0 to define it as the first instance. Decimal, 5-bit.
- **ECU Instance** — This 3-bit field is used when multiple electronic control units (ECU) are involved in performing a single function. If only one ECU is used for a particular controller application (CA), then this field must be set to 0 to define it as the first instance.
- **Manufacturer Code** — Decimal, 11-bit.
- **Identity Number** — Decimal, 21-bit.

Sample time — Simulation refresh rate
0.01 (default) | double

Specify the sampling time of the block during simulation. This value defines the frequency at which the J1939 Node Configuration updates its optional output ports. If the block is inside a triggered subsystem or inherits a sample time, specify a value of -1. You can also specify a MATLAB variable for sample time. The default value is 0.01 simulation seconds. For information about simulation sample timing, see "What Is Sample Time?".

Output current node address — Enable or disable the Address port display
off (default) | on

Enable or disable the **Address** output port to show the effective address. The effective address is different from the predefined station address. If **Allow arbitrary address** is selected, a name conflict occurs, and the current node has lower priority. The output signal is a double value from 0 to 253. This port is disabled by default.

Output address claim status — Enable or disable the address claim AC Status display
off (default) | on

Enable or disable the address claim **AC Status** output port to show the success of an address claim. The output value is binary, 1 for success or 0 for failure. This port is disabled by default.
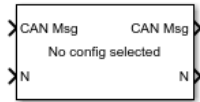
See Also

J1939 CAN Transport Layer | J1939 Receive | J1939 Transmit | J1939 Network Configuration

Introduced in R2015b

J1939 CAN Transport Layer

Generate and consume J1939 messages that are transported by CAN hardware
Library: Simulink Real-Time / J1939

Description

The J1939 CAN Transport Layer block handles CAN messages that your model transmits or receives by using Simulink Real-Time CAN library blocks.

Connect the input side of the block to a block that receives CAN messages. Connect the output side of the block to a block that transmits the J1939 messages over CAN. Set up the transmitting block so that a CAN message is sent only when an J1939 message is available. Otherwise, the block sends 0 byte data when J1939 messages are not available, causing undefined behavior.

Ports

Input

CAN Msg — CAN MESSAGE structures being consumed
vector

Vector of CAN MESSAGE structures being consumed.

N — Number of messages
integer

Number of messages in the vector.

Output

CAN Msg — CAN MESSAGE structures being generated
vector

Vector of CAN MESSAGE structures being generated.

N — Number of messages
integer

Number of messages in the vector.

See Also

J1939 Receive | J1939 Transmit | J1939 Node Configuration | J1939 Network Configuration

Introduced in R2020b

J1939 Receive

Receive J1939 parameter group messages
Library: Simulink Real-Time / J1939 Communication
                         Vehicle Network Toolbox / J1939 Communication

```
        ┌──────────────────────┐
        │           Data ▷     │
        │  Config:             │
        │  Node:               │
        │  PG:                 │
        │                      │
        │      Msg Status ▷    │
        └──────────────────────┘
           J1939 Receive
```
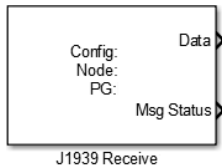
Description

The J1939 Receive block receives a J1939 message from the configured CAN device. The J1939 database file defines the nodes and parameter groups. You specify the J1939 database by using the J1939 Network Configuration block.

To use this block, you must have a license for both Vehicle Network Toolbox and Simulink software.

The J1939 communication blocks support the use of Simulink accelerator and rapid accelerator modes. You can speed up the execution of Simulink models by using these modes. For more information on these modes, see "Design Your Model for Effective Acceleration".

The J1939 communication blocks also support code generation that have limited deployment capabilities. Code generation requires a C++ compiler that is compatible with the code generation target. For the current list of supported compilers, see Supported and Compatible Compilers.

Ports

Output

Data — Data output
double

Depending on the J1939 parameter group defined in the J1939 database file, the block can have multiple data output signal ports. The block output data type is double.

Msg Status — Message received status
0 | 1

When you select the **Output New Message Received status** check box in the parameters dialog, this port outputs 1 when a new message is received from the CAN bus. Otherwise, this port outputs 0.

Parameters

Config name — Name of the J1939 network configuration to associate
ConfigX (default) | character vector

The name of the J1939 network configuration to associate. This value is used to access the corresponding J1939 database. Only the nodes defined in the model and associated with the specified

J1939 network configuration appear in the Node name list. The option shows `none` if no J1939 network configuration is found.

Node name — Name of the J1939 node
`NodeX` (default) | character vector

The name of the J1939 node. The drop-down list includes all the nodes in the model, both custom nodes and nodes from the database.

Parameter Group — Parameter group number (PGN) and name from database
character vector

The parameter group number (PGN) and name from the database. The contents of this list vary depending on the parameter groups that the J1939 database file specifies. The default is the first parameter group for the selected node.

If you change any parameter group settings within your J1939 database file, open the J1939 Receive block dialog box and select the same **Parameter Group** and click **OK** or **Apply**.

Signals — Signals defined in the parameter group
array of character vectors

Signals that are defined in the parameter group. The **Min** and **Max** settings are read from the database, but by default the block does not clip signal values that exceed this range.

Source Address Filter — Filter messages based on source address
`Allow all` (default) | `Allow only`

Filter messages based on source address are:

- `Allow only` — Specify a single source address.
- `Allow all` — Accepts messages from any source address. This option is the default.

Destination Address Filter — Filter out message based on destination address
`global and node specific` (default) | `global only` | `node specific only`

Filter out a message based on the destination address:

- `global only` — Receive only broadcast messages.
- `node specific only` — Receive only messages addressed to this node.
- `global and node specific` — Receive all broadcast and node-addressed messages. This option is the default.

Sample time — Simulation refresh rate
`0.01` (default) | double

The simulation refresh rate. Specify the sampling time of the block during simulation. This value defines the frequency at which the J1939 Receive block updates its output ports. If the block is inside a triggered subsystem or inherits a sample time, specify a value of `-1`. You can also specify a MATLAB variable for sample time. The default value is `0.01` simulation seconds. For information about simulation sample timing, see "What Is Sample Time?".

Output New Message Received status — Create a Msg Status output
`0` (default) | `1`

Select this check box to create a **Msg Status** output port. Its output signal indicates a new incoming message, showing 1 for a new message received, or 0 when there is no new message.
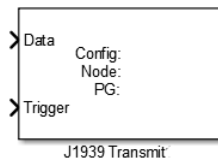
See Also

J1939 CAN Transport Layer | J1939 Transmit | J1939 Network Configuration | J1939 Node Configuration

Introduced in R2015b

J1939 Transmit

Transmit J1939 message
Library: Simulink Real-Time / J1939 Communication
Vehicle Network Toolbox / J1939 Communication

Description

The J1939 Transmit block transmits a J1939 message. The J1939 database file defines the nodes and parameter groups. You specify the J1939 database by using the J1939 Network Configuration block.

To use this block, you must have a license for both Vehicle Network Toolbox and Simulink software.

The J1939 communication blocks support the use of Simulink accelerator and rapid accelerator modes. You can speed up the execution of Simulink models by using these modes. For more information on these modes, see "Design Your Model for Effective Acceleration".

The J1939 communication blocks also support code generation that have limited deployment capabilities. Code generation requires a C++ compiler that is compatible with the code generation target. For the current list of supported compilers, see Supported and Compatible Compilers.

Ports

Input

`Data` — Input data
signal

Depending on the J1939 parameter group and signals defined in the J1939 database file, the block can have multiple data input ports.

`Trigger` — Enables the transmission of message
0 | 1

Enables the transmission of the message for that sample. A value of 1 specifies to send, a value of 0 specifies not to send.

Parameters

`Config name` — Name of the J1939 network configuration to associate
ConfigX (default) | character vector

The name of the J1939 network configuration to associate with. This is used to access the corresponding J1939 database. Only the nodes defined in the model and associated with the specified J1939 network configuration appear in the Node name list. The option shows none if no J1939 network configuration is found.

`Node name` — Name of the J1939 node
`NodeX` (default) | character vector

The name of the J1939 node. The drop-down list includes all the nodes in the model, both custom nodes and nodes from the database.

`Parameter Group` — Group number (PGN) and name
int8

The parameter group number (PGN) and name from the database. The contents of this list vary depending on the parameter groups that the J1939 database file specifies. The default is the first parameter group for the selected node.

If you change any parameter group settings within your J1939 database file, you must then open the J1939 Transmit block dialog box and select the same **Parameter Group**, then click **OK** or **Apply** to update the parameter group information in the block.

`Signals` — Signals defined in parameter group
array of character vectors

Signals defined in the parameter group. The **Min** and **Max** settings are read from the database, but by default the block does not clip signal values that exceed this range.

`PG Priority` — Priority of the parameter group
int8

Priority of the parameter group, read from the database. This priority setting resolves clashes of multiple parameter groups transmitting on the same bus at the same time. If a conflict occurs, the priority group with lower priority (higher value) will refrain from transmitting. The value can range from 0 (highest priority) to 7 (lowest).

`Destination Address` — Name of the destination node
int8

The name of the destination node. The default is the first node defined in the database, otherwise `Custom`.

For a custom destination address, you can specify 0–253 for the address of the destination node. For broadcasting to all nodes, use the `Custom` **Destination Address** setting with an address of 255.

See Also
J1939 CAN Transport Layer | J1939 Receive | J1939 Network Configuration | J1939 Node Configuration

Introduced in R2015b

Logitech

Logitech Blocks

The Simulink Real-Time Logitech blocks support Logitech G29 Steering Wheel functions.

Logitech G29 Steering Wheel

Receive Logitech G29 Steering Wheel read data
Library: Simulink Real-Time / Logitech G29

Description

The Logitech G29 Steering Wheel block reads data from a Logitech G29 Steering Wheel (PS3 only).
The block does not support a Stick Shift module.

Ports

Output

`Buttons` — Status of steering wheel buttons
0 (= unpressed) | 1 (= pressed)

The `Buttons` output is a vector of Boolean values that indicate the status of buttons on the steering
wheel. The order of the button values in the vector is:

1   Square

2   X

3   Circle

4   Triangle

5   LPaddle

6   RPaddle

7   L2

8   R2

9   L3

10 R3

11 Share

12 Option

13 PS

Data Types: `Boolean`

`Steering` — Value of steering wheel position
0 (= left most) | 65535 (= right most)

The `Steering` output value indicates the position of the steering wheel.

Data Types: `uint16`

Pedals — Status of throttle, brake, and clutch pedals
0 (= not engaged) | 255 (= fully engaged)

The `Pedals` output is a vector of values that indicate the status of the throttle, brake, and clutch pedals. The order of the pedal values in the vector is:

1   Throttle

2   Brake

3   Clutch

Data Types: `uint8`

Direction — Status of direction pad buttons on steering wheel
0 (= UP) | 2 (= RIGHT) | 4 (= DOWN) | 6 (= LEFT) | 8 (= UNPRESSED)

The `Direction` output value indicates a button press on the direction pad. When you press the direction pad in between the pad buttons, intermediate values occur.

Data Types: `uint8`

Status — Status of communications with steering wheel
0 (default) | negative value

The `Status` output value indicates successful communications with the steering wheel (0) or unsuccessful communications (negative value).

Data Types: `int32`

Parameters

sampleTime — Sample time for steering wheel data
-1 (inherited) (default)

Select the sample time for steering wheel data. The recommended `sampleTime` is 10ms.

Programmatic Use
Block Parameter: `sampleTime`

See Also

External Websites
Logitech G29 Driving Force Racing Wheel

Introduced in R2018b

LIN

LIN Blocks

The Simulink Real-Time LIN blocks support packing signals into frames and unpacking signals from frames by using information in an LIN description file (LDF).

LIN Pack

Pack signals into data frame by using configuration in LIN description file
Library: Simulink Real-Time / LIN

Description

The LIN Pack block packs signals into the data frame by using the configuration in the LIN description file (LDF). A local interconnect network (LIN) bus can use the packed data. The LIN Pack block supports LDF version 2.2 and previous.

Ports

Input

`in` — input data to pack
data

Receives data to pack for the LIN. The number of input ports is allocated based on the selected frame. The port name is assigned to each port.

Output

`lin_data` — output LIN data
LIN data

Sends packed LIN data.

Parameters

`LDF File Name` — File path of LDF
empty (default) | character vector | string scalar

Provides the file path of the LDF. If you do not set this parameter, you cannot use the block.

Programmatic Use
Block Parameter: `LDFFileName`

`Select Frame` — LIN frame
None (default) | character vector | string scalar

Selects the LIN frame from selections available in LDF. If you do not select an LDF, the block uses None.

Programmatic Use
Block Parameter: `selectedFrameName`

Frame ID — LIN frame ID
1 (default) | uint8

Selects the LIN frame ID from selections available in LDF. If you do not select an LDF, the block uses
1.

Programmatic Use
Block Parameter: frameId

See Also
LIN Unpack

External Websites
Introduction to the Local Interconnect Network (LIN) Bus

Introduced in R2021b

LIN Unpack

Unpack signals from data frame by using configuration in LIN description file
Library: Simulink Real-Time / LIN
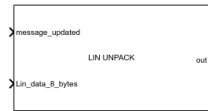
Description

The LIN Unpack block unpacks the data frame by using the configuration in the LIN description file (LDF) and outputs signal data. You can use the unpacked local interconnect network (LIN) bus data as signal data in the model. The LIN Unpack block supports LDF version 2.2 and previous.

Ports

Input

`message_updated` — indicates message update
0 (logical false) | 1 (logical true)

When the `message_updated` value is 1 (logical true), there is a message coming in from the LIN bus. This block outputs the unpacked signals. When the `message_updated` value is 0 (logical false), there is no message updated on the LIN bus, and this block outputs the default value for each signal defined in the LDF file.

`Lin_data_8_bytes` — input LIN data to unpack
LIN data

Receives LIN data to unpack.

Output

`out` — output data
data

Sends unpacked data. The number of the output ports is allocated based on the number of signals in the selected frame. When a frame is selected, the port name is assigned to each port.

Parameters

`LDF File Name` — File path of LDF
empty (default) | character vector | string scalar

Provides the file path of the LDF. If you do not set this parameter, you cannot use the block.

Programmatic Use
Block Parameter: LDFFileName

`Select Frame` — LIN frame
None (default) | character vector | string scalar

Selects the LIN frame from selections available in LDF. If you do not select an LDF, the block uses None.

Programmatic Use
Block Parameter: selectedFrameName

Frame ID — LIN frame ID
1 (default) | uint8

Selects LIN frame ID from selections available in LDF. If you do not select an LDF, the block uses 1.

Programmatic Use
Block Parameter: frameId

See Also
LIN Pack

External Websites
Introduction to the Local Interconnect Network (LIN) Bus

Introduced in R2021b

Logging Blocks Library

Logging Blocks

The Simulink Real-Time file logging blocks support logging run data for viewing in the Simulation Data Inspector.

File Log

Write signal data file log on target computer
Library: Simulink Real-Time / Logging /

Description

When you enable logging service, the File Log block logs the signal at its input port to the target computer file system.

Ports

Input

S — Signal data for file log
scalar | vector | matrix | array | bus | nonvirtual bus

Provides the signal data input for the file log.

Parameters

Decimation — Applies decimation to file log data
1 (default) | integer

The decimation parameter value selects decimation for the samples captured in the file log.

Programmatic Use

decimation

See Also

Enable File Log | enable | disable | list | import | discard

Introduced in R2020b

Enable File Log

Enable or disable file logging of signals on target computer
Library: Simulink Real-Time / Logging /

Description

When the input port value is true, the Enable File Log block enables the file logging service on the target computer so that File Log blocks in the model start recording data. When the input port value is false, the block disables file logging on the target computer, and the File Log blocks do not record data.

Ports

Input

E — Enable or disable file logging
boolean

Provides enable or disable control over File Log block operation.

See Also
File Log | enable | disable | list | import | discard

Introduced in R2020b

Profiling Blocks Library

Profiling Blocks

The Simulink Real-Time profiling blocks support real-time application execution profiling functions.

Enable Profiler

Start and stop execution profiler on target computer
Library: Simulink Real-Time / Logging

```
> Start
       Start/Stop profiler
> Stop
```

Description

A rising edge on the Enable Profiler block `Start` input starts the execution profiler. A rising edge on the `Stop` input stops the execution profiler. A rising edge on both ports has no effect.

Ports

Input

Start — Starts the profiler
`0 | 1`

When the `Start` input changes from `0` to `1`, the block starts the profiler.

After the resources required to collect the data become available in the background, the profiler starts collecting data. Profiler preparation can span several time steps.

Data Types: `Boolean`

Stop — Stops the profiler
`0 | 1`

When the `Stop` input changes from `0` to `1`, the block stops the profiler.

If the profiler is still running when the application stops, the profiler stops by itself. You do not have to trigger the `Stop` input.

The amount of data collected is limited to `1GB`. The profiler stops by itself when it reaches this limit.

Data Types: `Boolean`

See Also
`startProfiler | stopProfiler | getProfilerData | resetProfiler | ProfilerData`

Introduced in R2020b

Log Event

Log an execution profiling event
Library: Simulink Real-Time / Profiling

Description

When triggered, the Log Event block inserts a user-defined event into the execution profiling event stream. The user-defined event includes a channel ID, a code number, and a value number. The channel ID value ranges from 0 to 1023.

Ports

Input

T — Trigger for log event
`1..9`

When the T (trigger input) changes from 0 to 1 (Rising) or from 1 to 9 (Falling), the block inserts a user-defined simple event defined by the parameters **Channel**, **Code**, and **Value** into the execution profiling event stream.

Example: 1

Parameters

Channel — Event channel ID
0 (default) | int

Select the event change ID, for example 500.

Programmatic Use

`channel`

Code — Event code number
0 (default) | uint32

Select the event code number, for example 200.

Programmatic Use

`code`

Value — Event value number
0 (default) | uint32

Select the event value number, for example 200.

Programmatic Use

value

Trigger type — Selects trigger signal edge
Rising (default) | Falling | Either

Select the type of signal change that triggers the block. When the T (trigger input) changes from 0 to 1 (Rising) or from 1 to 9 (Falling), the block inserts a user-defined simple event defined by the parameters **Channel**, **Code**, and **Value** into the execution profiling event stream.

Programmatic Use

trigger

See Also
Enable Profiler

Introduced in R2020b

PTP Precision Time Protocol Blocks Library

PTP Blocks

The Simulink Real-Time PTP blocks support PTP protocol-related functions.

Precision Time Protocol

Precision Time Protocol

Measurement and control systems increasingly use distributed system technologies. To distribute measurement or control tasks over interconnected computing devices, such systems maintain a system-wide sense of time. Simulink Real-Time uses the Precision Time Protocol (PTP) to synchronize the system clock of each target computer to a reference time.

PTP (IEEE® 1588) is a protocol that synchronizes PTP clocks throughout a computer network. The current version of PTP (IEEE 1588-2008) describes a hierarchical master-slave architecture for clock distribution.

By design, this protocol is more accurate for local systems than the Network Time Protocol (NTP) and more robust than the Global Positioning System (GPS). On a local area network, the protocol achieves PTP clock accuracy in the submicrosecond range, making it suitable for distributed measurement. When you use this protocol to synchronize Simulink Real-Time applications across multiple target computers, it can synchronize execution to under 10 µs.

Suppose that you are designing a control system for a wind power plant. To determine the plant parameters, you attach sensors that acquire the data shown in the diagram.
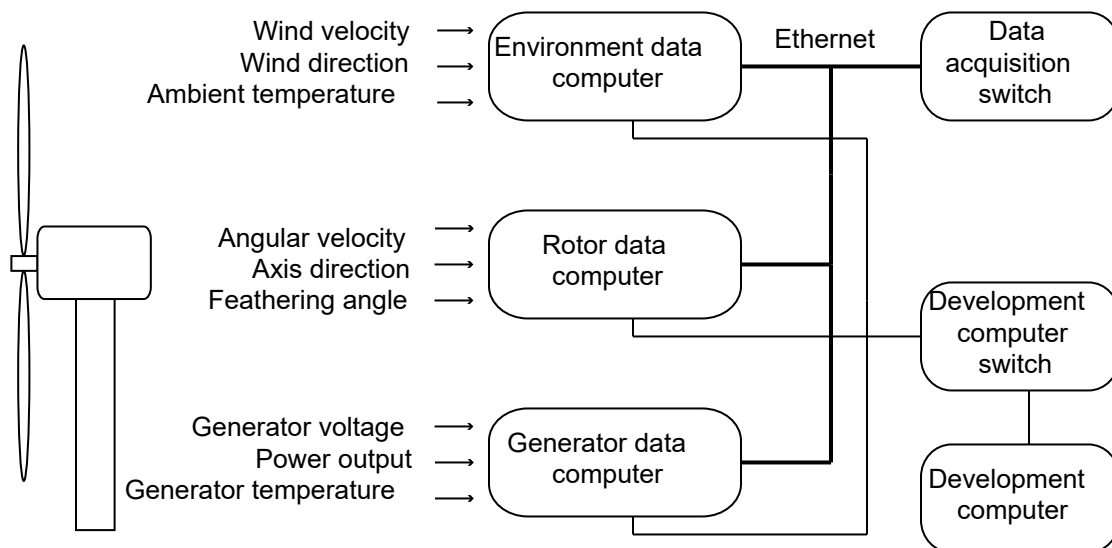
To record the data and timestamps, connect the sensors to a set of data acquisition target computers. Interconnect the data acquisition computers through an Ethernet network and a switch that supports the PTP protocol (a PTP transparent clock or boundary clock). To access the data and timestamps, connect the target computers to a development computer through another Ethernet network and switch. On the development computer, run MATLAB to do the data analysis, including:

- Sorting by time the data recorded on the different computers to analyze the event sequence over time.
- Filtering sensor data that has invalid (unsynchronized) timestamps.
- Integrating values of measured data collected at the same time from sensors connected to different computers.

Finally, you build and download the real-time applications to each target computer, run the applications, and collect and analyze the results at each valid timestamp. You use the results to design a control system for the wind power generator.

Simulink Real-Time supports the PTP protocol by using the RTOS PTP daemon. The ptpd daemon implements the Precision Time Protocol (PTP) Version 2 as defined by the IEEE 1588-2008 standard. For more information about the daemon, see the QNX Neutrino documentation.

To synchronize the target computer PTP clocks:

1 Connect the target computers by using their Intel i210 Ethernet cards.
2 On the master target computer, start the PTP daemon by using the `start(target_object.ptpd)` command or the `target_object.ptpd.AutoStart` property.
3 On the slave target computer, start the PTP daemon by using the `start(target_object.ptpd)` command or the `target_object.ptpd.AutoStart` property.

The PTP slave synchronizes its clock to the PTP master clock. The hardware timestamp from the Intel i210 Ethernet cards provides IEEE 1588-2008 standard compliant clock synchronization.

To get information about clock synchronization, add an IEEE 1588 Read Parameter block to the model. This block outputs a Precision Time Protocol parameter value that you select. The parameters are:

- **System time**
- **Calendar time**
- **Offset from Master**
- **Master to Slave Delay**
- **Slave to Master Delay**
- **One Way Delay**

See Also

Subsystem, Atomic Subsystem, CodeReuse Subsystem | IEEE 1588 Read Parameter

More About

- "PTP Prerequisites" on page 18-4

External Websites

- standards.ieee.org

PTP Prerequisites

The Simulink Real-Time implementation of PTP enforces specific requirements:

- PTP functionality is available only with a Speedgoat target computer. If you have not installed the Speedgoat I/O Blockset, attempting to build a real-time application with PTP causes a build error.
- The support for PTP IEEE-1588 is part of the on-board or installed I/O module connectivity of real-time target machines. After installing the Speedgoat I/O Blockset, you can confirm PTP support by using the function:

```
speedgoat.showPtpInterfaces('TargetObject', tg)
```

```
>> speedgoat.showPtpInterfaces('TargetObject', tg)
     Label          Index      PTPSupport              Configuration

    {'Host Link'}   {'wm0'}     {'No'}      {'IP: 192.168.7.5 Subnet: 255.255.255.0'}
    {'ETH1'     }   {'wm1'}     {'No'}      {'Not configured'                        }


To use PTP, the ethernet interface needs to be configured with an IP stack.
Use the Ethernet configuration tool: speedgoat.configureEthernet
```

See Also

Subsystem, Atomic Subsystem, CodeReuse Subsystem | IEEE 1588 Read Parameter

More About

- "Precision Time Protocol" on page 18-2

External Websites

- standards.ieee.org

Precision Time Protocol Blocks

IEEE 1588 Read Parameter

Output Precision Time Protocol status parameter values such as target computer system time, calendar time, and delays
Library: Simulink Real-Time / IEEE 1588

```
             PTP
Read System time (nanosecond)
```
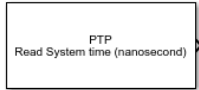
Description

The IEEE 1588 Read Parameter block reads the parameter that you select and send its value to the block output. The block label changes based on the parameter that you select.

You can use the IEEE 1588 Read Parameter block to read the target computer system time even without a PTPd process running.

Ports

Output Arguments

Output — Output selected by Parameter to read parameter
double | [uint32]

The current parameter to read.

Parameters

Parameter to read — Parameter to display at output
System time (nanosecond) (default) | Calendar time (time_t) | Offset from Master | Master to Slave Delay | One Way Delay

Specify the parameter to read. Select one of:

- System time (nanosecond) — Current SLRT system time number of nanoseconds, counting from the current epoch. The output is a double.
- Calendar time (time_t) — Current SLRT calendar time in time_t int32 vector.
- Offset from Master — Time offset in nanoseconds. The output is a double.
- Master to Slave Delay — Time delay in nanoseconds. The output is a double.
- One Way Delay — Time delay in nanoseconds. The output is a double.

Programmatic Use
Block Parameter: param

Sample time (-1 for inherited) — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time.

Programmatic Use
Block Parameter: `sample_time`

See Also

Subsystem, Atomic Subsystem, CodeReuse Subsystem | IEEE 1588 Read Parameter

Topics
"PTP Prerequisites" on page 18-4

External Websites
standards.ieee.org

Introduced in R2020b

RS232 Serial Blocks Library

Serial Communications Support

RS-232 Serial Communication

The Simulink Real-Time software supports RS-232 serial communication by using the serial ports on the target computer mainboard as the RS-232 I/O devices. You can initiate RS-232 communication with these serial ports and the accompanying Simulink Real-Time drivers.

The Simulink Real-Time block library supplies legacy drivers to support RS-232 communication (see "RS-232 Legacy Drivers" on page 20-3). The composite drivers support RS-232 communication in asynchronous binary mode. They provide a simple ASCII encode/decode for the send and receive RS-232 blocks.

These drivers are described as legacy because the library supports legacy RS-232 communications by using the RTOS resource manager. To use the Legacy Serial Read block and Legacy Serial Write block, set up the serial ports by using the Legacy Serial Setup block.

Serial Connections for RS-232

The Simulink Real-Time software supports serial communication by using the COM1 and COM2 ports on the target computer.
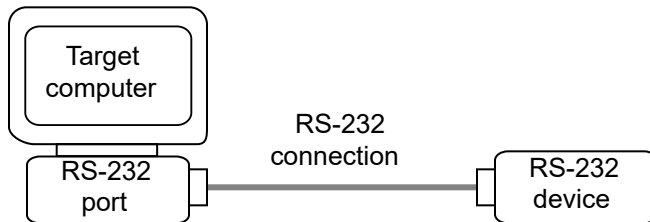
Your real-time applications can use these RS-232 ports as I/O devices. With the typical DTE/DCE configuration of the RS-232 device, the target computer is connected to the device with a null modem cable.

See Also
ASCII Encode | ASCII Decode | FIFO Read | FIFO Write | FIFO Read HDRS | FIFO Read Binary | Modem Control | Modem Status | ASCII Decode V2

RS-232 Legacy Drivers

There are components that make up the RS-232 legacy drivers. You can create a model by using these drivers. These drivers perform RS-232 asynchronous communications.

The Simulink Real-Time software provides legacy drivers that support the target computer (main board) serial ports.

These drivers encapsulate the functionality of the device by using the Legacy Serial Port block and the Legacy Serial Port F blocks. For most RS-232 requirements, you can use these RS-232 drivers. For modem requirements, use the Modem Control block and Modem Status block.

Add RS-232 Blocks

When you want to use the serial ports on the target computer for serial I/O, add RS-232 Legacy Serial Port or Legacy Serial Port F subsystem blocks to your Simulink model .

Before you start, decide what legacy serial ports you want to use. The example has you configure the Legacy Serial Port block. To configure this block, first select serial ports.

The following procedure shows how to use the serial ports on the target computer for I/O with the legacy drivers. It shows a model that uses legacy serial port 1 and legacy serial port 2.

1 Open the Simulink Real-Time block library. You can access the library from the Simulink Library Browser. In the Simulink Editor, on the **Real-Time** tab, from the **Prepare** section, click **Library Browser**. In the left pane, double-click **Simulink Real-Time**, and then click **RS232**. To open the library from the MATLAB Command Window, type:

    slrealtimelib

2 In the Simulink Real-Time driver block library, double-click the **RS232** group block.

3 In the window with blocks for RS-232 legacy drivers, drag an ASCII Encode block to your Simulink model. This block encodes input for the Legacy Serial Port block XMT port.

4 Configure this block.

5 Drag an ASCII Decode block to your Simulink model. This block decodes output from the Legacy Serial Port block RCV port.

6 Configure this block.

7 Double-click the **Mainboard** group block.

8 Drag two Legacy Serial Port blocks to your Simulink model.

9 Double-click the first Legacy Serial Port block.

10 Configure this block for Legacy Serial Port 1

11 Double-click the second Legacy Serial Port block.

12 Configure this block for Legacy Serial Port 2

13 Add a Pulse Generator block and a target Scope block.

14 Configure the Pulse Generator block so that its **Pulse type** is Sample based.

    The dialog box changes to display a **Sample time** parameter. Enter a **Sample time** that is slower than the one you set for Receive Setup.

15 From the Simulink Library Browser, select **Sinks**. Depending on your configuration, drag one or more Terminator blocks to your model.

From the Simulink Library Browser, select **Sources**. Depending on your configuration, drag the Ground block to your model.

A pre-constructed example model is available. The `slrt_ex_serialbaseboardasciitest` model uses two legacy serial ports. To open this model, in the MATLAB Command Window, type:

```
open_system(fullfile(matlabroot, 'toolbox', 'slrealtime',...
'examples', 'slrt_ex_serialbaseboardasciitest'))
```
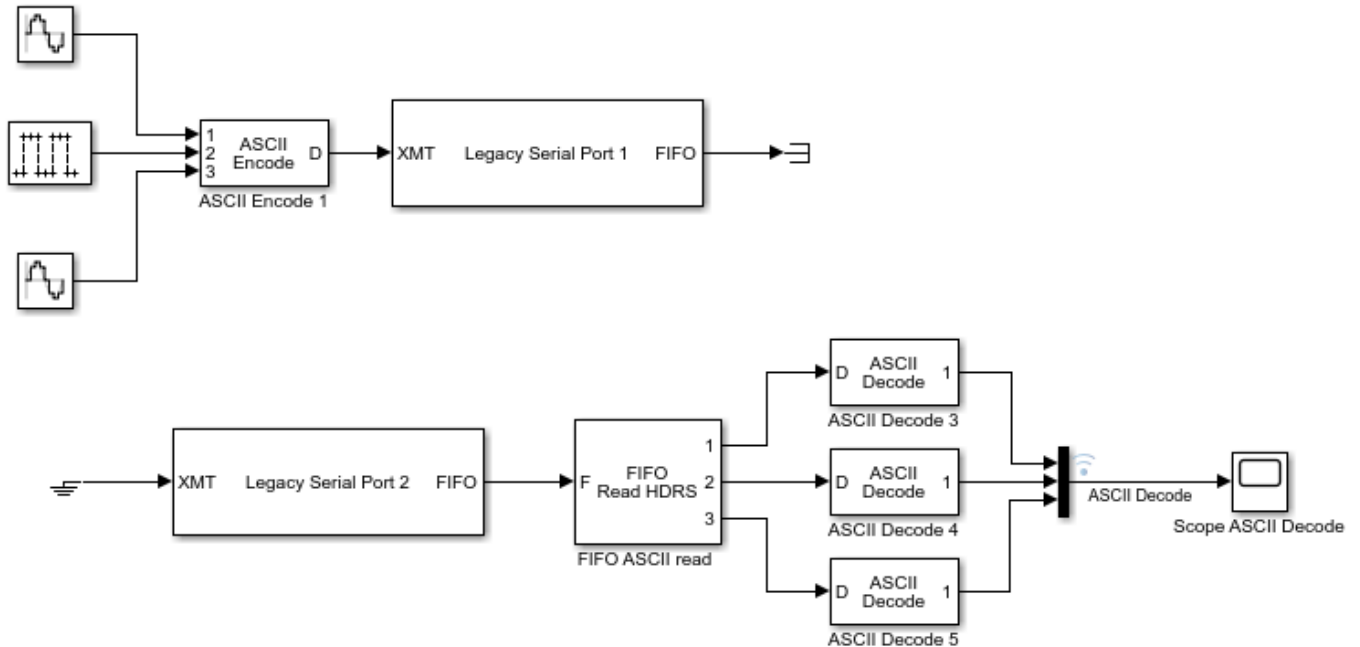

Your next task is to build and run the real-time application.

Building and Running the Real-Time Application

The Simulink Real-Time and Simulink Coder software create C code from your Simulink model. You can then use a C compiler to create executable code that runs on the target computer. You must know how to configure your model to create a real-time application. See "Build and Download Real-Time Application by Using Run on Target".

After you add the RS-232 blocks for the main board to your Simulink model and configure your model, you can build your real-time application.

In the Simulink Editor, on the **Real-Time** tab, click **Run on Target**.

Simulink Real-Time RS-232 Reference

- "RS-232 FIFO Read Blocks" on page 20-5
- "RS-232 Signal Data Types" on page 20-6
- "RS-232 Zero Length Messages" on page 20-6

- "RS-232 Control When You Send a Message" on page 20-7

The Simulink Real-Time software supports RS-232 communication by using driver blocks in your Simulink model.

RS-232 FIFO Read Blocks

There are three kinds of FIFO Read blocks: FIFO Read, FIFO Read HDRS, and FIFO Read Binary. To develop your model, use the following guidelines:

- Simple data streams — Use the FIFO Read block to read simple data streams. An example of a simple data stream is one that has numbers separated by spaces and ends with a new-line character. The FIFO Read block is a simple block that can easily extract these numbers.

- Complicated data streams — Use the FIFO Read HDRS and FIFO Read Binary blocks for more complicated data streams. A more complicated data stream can be one that contains headers, messages of varying lengths, or messages without specific terminators. A message header consists of one or more character identifiers at the beginning of a message that specify what data follows. ASCII messages normally have a variable length and a terminator. Typically, the messages of a particular device use the same predefined terminator. Binary messages are normally of fixed length without a specific terminator.

  You can also use the FIFO Read HDRS and FIFO Read Binary blocks to work with devices that can send different messages at different times.

The input to these FIFO read blocks must be of type `serialfifoptr`, which is output from F type Send Receive subsystems.

These examples show instances when you can use the FIFO Read block.

- For an instrument that sends a character vector like this one:

  ```
  <number> <number> ... <CR><LF>
  ```

  use the simple FIFO Read block to read the message. Configure the FIFO Read block **Delimiter** parameter for a line feed (value of 10). Connect the output to an ASCII Decode block with a format that separates the numbers and feeds them to the output ports.

- For an instrument that can send one of several different messages, each beginning with a different fixed character vector, use the FIFO Read HDRS block. For example, a digital multimeter connected through an RS-232 port sends a voltage and amp reading with messages in this format:

  ```
  volts <number> <CR><LF>
  amps <number> <CR><LF>
  ```

  Configure the FIFO Read HDRS block **Header** parameter for the `volts` and `amps` headers in a cell array: `{'volts', 'amps'}`. Configure the **Terminating string** parameter for carriage return (13) and line feed (10): `[13 10]`.

  Connect the output to multiple ASCII Decode blocks, one for each header and message. For examples of how to use this block in a model, see the `slrt_ex_serialasciitest` and `slrt_ex_serialasciisplit` models in `matlab/toolbox/slrealtime/examples`.

- For an instrument that sends a binary message, you could know the length of each full message, including the header. Configure the FIFO Read Binary block **Header** parameter for the headers of the message in a cell array and the **Message Lengths** parameter for the message lengths. For further examples of how to use this block in a model, see the `slrt_ex_serialbinarytest` and `slrt_ex_serialbinarysplit` models in `matlabroot/toolbox/slrealtime/examples`.

RS-232 Signal Data Types

Signals between blocks in composite drivers can be one of several basic data types: 8-bit, 16-bit, and 32-bit. These types are structures.

The 8-bit data types are NULL-terminated character vectors that are represented as Simulink vectors. The width is the maximum number of characters that can be stored. In the figure, `M` is the actual set of stored characters and `N` is the maximum number of characters that can be stored. This figure illustrates 8-bit `int` NULL-terminated and 8–bit `uint` NULL-terminated data types.
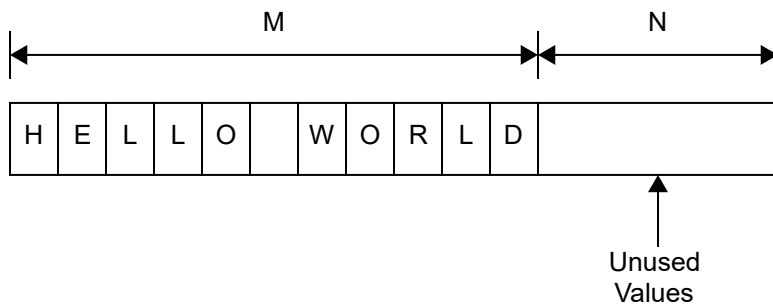
The character vector has 11 characters terminated with a NULL byte (`0`). The data type cannot contain a NULL byte as part of the real data.

The 16-bit and 32-bit data types use the first element of the vector as a count of the valid data. In the figure of a 16-bit data type, `C` is the count of the valid data and `N` is the width of the vector. This figure illustrates count + 16-bit `int` and count + 16-bit `uint` data types. This arrangement also applies to count + 32-bit `int` and count + 32-bit `uint` data types.
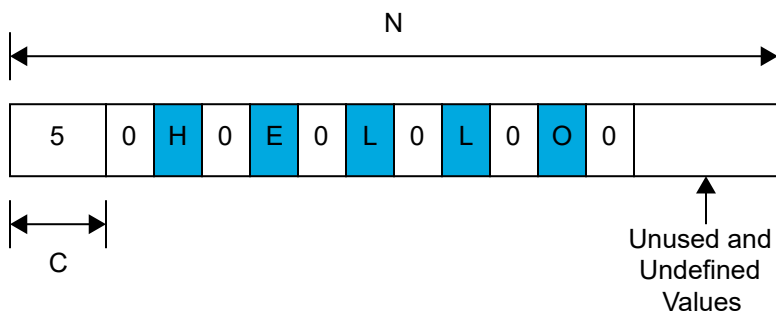
The serial blocks interpret each entry in the vector as a single character. The low-level Send block writes the low-order byte of each entry to the UART. The 16-bit and 32-bit data types allow the embedding of 8-bit data values, including `0`. The 8-bit data type is most useful with the ASCII Encode and Decode blocks. The 16-bit and 32-bit data types are most useful for binary data streams.

RS-232 Zero Length Messages

Usually, you configure a FIFO read block of your model serial I/O to execute faster than the model receives data. Doing so prevents the receive FIFO buffer from overflowing. You must also configure your model to deal with the possibility that a FIFO Read block does not have a message on its output.

Receive FIFOs can have too few characters for a FIFO read operation. A model that receives serial I/O can have a FIFO Read block that executes in this situation. Depending on how you configure the behavior, this condition causes a FIFO Read block to perform one of these operations:

- Return the last message it received.

- Return a zero-length message.

The Simulink Real-Time library of composite serial drivers has three FIFO Read blocks: FIFO Read HDRS, FIFO Read Binary, and FIFO Read. For the FIFO Read HDRS or FIFO Read Binary blocks, you configure this behavior by using the **Output behavior** parameter. The FIFO Read block returns either a new message or a zero-length message.

To execute model code only if a new message arrives, check the first element of the returned vector, depending on the character vector data type:

- In the 8-bit data type, the returned character vector is NULL-terminated. If the first element is 0, the character vector has zero length and the FIFO read did not detect a new message.

- In the 16-bit and 32-bit data types, the first element is the number of characters in the character vector. This value is 0 if the FIFO read did not detect a new message.

If the message has nonzero length, enable a subsystem to process the new character vector. Otherwise, do not process it.

RS-232 Control When You Send a Message

You can use the structure of both serial data types to control when you send a message. For more information, see "RS-232 Signal Data Types" on page 20-6. In both cases, a 0 in the first position indicates an empty character vector.

- 8-bit data types — A value of 0 in the first position is the NULL terminator for the character vector.

- 16-bit and 32-bit data types — The first position is the number of characters that follow.

If you connect an empty character vector to the XMT port on one of the send/receive subsystems, no characters are pushed onto the transmit FIFO. You can get this empty character vector by using one of these methods:

- To send a specific character vector occasionally, use the Product block to multiply the entire character vector by either 0 or 1. In this case, the 0 or 1 value becomes a transmit enable. To optimize this operation, use a Demux block to extract the first element. Multiply just that element by 0 or 1, then use the Mux block to combine it again.

- Use a Manual Switch, Multiport Switch, or Switch block. Configure the blocks for two ports to choose between different messages, with one of the choices being a vector of 0 values. The Switch block chooses only between vectors of the same width. Because the character vector length does not use the whole vector, you can pad your data to the same width with 0 values.
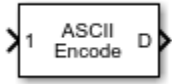
See Also
ASCII Encode | ASCII Decode | FIFO Read | FIFO Write | FIFO Read HDRS | FIFO Read Binary | Modem Control | Modem Status | ASCII Decode V2

Serial Communications Support: Blocks

ASCII Encode

Convert Simulink values into `uint8` character vector
Library: Simulink Real-Time / RS232

Description

The ASCII Encode block generates a `uint8` output vector that contains a NULL-terminated character vector based on a `printf` like format string. The data comes from the input ports.

Ports

Input

1 — Numbered ports that receive values to encode
numeric

Values that the block encodes as a null-terminated character vector.

Data Types: `double` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32`

Output

D — Null-terminated character vector
character vector

Generated `uint8` output vector that contains a NULL-terminated character vector.

Parameters

`Format string` — Format specifiers for converting values to ASCII
%d\r (default) | %c | %i | %o | %u | %x | %e | %f | %g

Enter a `printf` like format string. For each format specifier such as `%d`, the block replaces the format specifier by the converted value in the corresponding input variable. The format specifiers follow the normal description for `printf`.

Programmatic Use
Block Parameter: `format`

`Number of variables` — Number of block inputs
1 (default) | integer

The value on each port is inserted into the output character vector with the format specified in **Format string**.

Programmatic Use
Block Parameter: `nvars`

Max output string length — Maximum length of converted character vector, in bytes
128 (default) | integer

The block allocates enough memory to support this length for the output port. When specifying this length, include the NULL termination on the character vector.

If the converted character vector exceeds this length, the block returns an error and does not write that character vector to the output port.

Programmatic Use
Block Parameter: `maxlength`

Variable types — Simulink data types allowed for input ports
`{'double'}` (default) | `{'int8'}` | `{'uint8'}` | `{'int16}` | `{'uint16'}` | `{'int32'}` | `{'uint32'}`

A cell vector with the same number of elements as specified in **Number of variables** can specify a different data type for each input port. A single element is replicated. For example:

`nvars=3`

`{ }` — The three inputs are doubles.

`{'uint8'}` — The three inputs are `uint8`.

`{'uint16', 'double', 'uint8'}` — The first input is a `uint16`, the second input is a `double`, and the third input is a `uint8`.

Programmatic Use
Block Parameter: `vartypes`

See Also
ASCII Decode

Topics
"RS-232 Serial Communication" on page 20-2
"RS-232 Legacy Drivers" on page 20-3

Introduced in R2020b

ASCII Decode

Parse ASCII character vector into Simulink values
Library:　　　　　Simulink Real-Time / RS232

Description

The ASCII Decode block parses an input character vector according to a format specifier similar to `scanf` and makes converted values available to the real-time application.

Ports

Input

D — Input vector to parse
`character vector`

The input vector can be either 8-bit or 16-bit and signed or unsigned. If the data format is 16-bit, the block ignores the upper 8 bits of each entry.

Data Types: `int8` | `uint8` | `int16` | `uint16`

Output

1 — Numbered ports that send Simulink values
`numeric`

Output ports corresponding to items in **Format string**.

Dependency

Number of variables determines the number of output ports.

Data Types: `double` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32`

Parameters

`Format string` — Format specifier for parsing input vector
`%d\r` (default) | `%c` | `%i` | `%o` | `%u` | `%x` | `%e` | `%f` | `%g`

Enter a `scanf` like format string. Each format specifier such as `%d` must match a corresponding part of the input vector. Literal strings in the format must match the first character plus the number of characters. The format specifiers follow the normal description for `scanf`.

An example format string is:

`'alpha %d bravo %f\n'`

Programmatic Use
Block Parameter: format

Number of variables — Number of output ports for this block
1 (default) | integer

Enter the number of output ports for this block. For example:

If **Format string** has the value of %xmore text%x and the input vector for the block has cdmabcdefgh90, you must specify the value of the **Number of variables** parameter as 2.

The first variable is assigned the value 0xcd. Next, the character vector mabcdefgh is considered a match to more text because:

• The first character for both character vectors is m.

• Both character vectors have the same number of characters.

The second variable is then assigned the value 0x90. The character vector mabcdefgh does not have to match exactly the value of **Format string**. This behavior is different from the behavior for scanf, which requires an exact match.

Programmatic Use
Block Parameter: nvars

Variable types — Simulink data types allowed for output ports
{'double'} (default) | {'int8'} | {'uint8'} | {'int16} | {'uint16'} | {'int32'} | {'uint32'}

A cell vector with the same number of elements as specified in **Number of variables** can specify a different data type for each output port. A single element is replicated. For example:

nvars=3

{ } — The three outputs are doubles.

{'uint8'} — The three outputs are uint8.

{'uint16', 'double', 'uint8'} — The first output is a uint16, the second output is a double, and the third output is a uint8.

Programmatic Use
Block Parameter: varids

See Also
ASCII Encode

Topics
"RS-232 Serial Communication" on page 20-2
"RS-232 Legacy Drivers" on page 20-3

Introduced in R2020b

ASCII Decode V2

Parse ASCII character vector into Simulink values
Library: Simulink Real-Time / RS232

Description

The ASCII Decode block parses an input vector produced by one of the following:

- Serial port Receive block
- Serial port FIFO Read block
- ASCII Encode block

This block makes the converted values available to a real-time application. It assumes that the input vector was prepared using an output format specifier similar to `printf` and uses an input format specifier similar to `scanf`.

This block generates inline code for the target computer. You cannot use it for Simulink simulation.

Ports

Input

`Data` — **Input vector to parse**
character vector

The input vector can be either 8-bit or 16-bit and signed or unsigned. If the data format is 16-bit, the block ignores the upper 8 bits of each entry.

Data Types: `int8` | `uint8` | `int16` | `uint16`

Output

`cnt` — **Number of format specifiers satisfied by input**
integer

`cnt` receives the number of format specifiers satisfied by the input character vector.

`Value` — **Inlined ports that send Simulink values**
numeric

Output ports corresponding to items in **Format**.

This block generates inline code for the target computer. You cannot use it for Simulink simulation.

Data Types: `single` | `double` | `int8` | `uint8` | `uint16` | `int16` | `int32` | `uint32`

Parameters

Format — Format specifier for parsing input vector
'%f\n' (default) | %c | %d | %i | %o | %u | %x | %e | %g

Enter a scanf like format string. Each format specifier such as %d must match a corresponding part of the input vector. Literal strings in the format must match the characters in the input vector. The format specifiers follow the normal description for scanf. The specifiers must be enclosed in single quotes. Failure to include these quotes causes simulation failures.

An example format string is:

'alpha %d bravo %f\n'

In this example, the data from the FIFO read is 'alpha 5'. In this case, cnt is 1 and the second output is unchanged from the last time both were found in a character vector. If the model expects 2 values, and cnt is less than 2, the model detects an error in the data.

Programmatic Use
Block Parameter: format

See Also

Topics
"RS-232 Serial Communication" on page 20-2
"RS-232 Legacy Drivers" on page 20-3

Introduced in R2020b

FIFO Read

Read simple data streams
Library: Simulink Real-Time / RS232

Description

The FIFO Read block is the read side of a FIFO read/write pair. You use this block to parse simple data streams. The block functions in two modes that you set by using the **Read to delimiter** check box.

- If you select the **Read to delimiter** check box, the block reads only elements if the specified delimiter has been written to the FIFO Write block. If the delimiter is found, the block returns elements up to and including the delimiter in the output vector. If the delimiter is not found, the block returns a zero-length vector, as determined by the data type. (If you have a zero-length vector, you can have your model perform a particular operation or ignore the case.)

- If you clear the **Read to delimiter** check box, the block returns elements between **Minimum read size** and the smaller of the number of elements currently in the FIFO and **Maximum read size**.

  When performing ASCII reads, select the **Read to delimiter** check box. When performing binary reads, clear this check box.

Here are some examples of how you can set up the FIFO Read block:

- **Transmit side of the interrupt service routine** — If the interrupt reason is not an empty hardware FIFO on the UART, the maximum input port receives a value of 0. If the hardware FIFO is empty, it receives the size of the hardware FIFO. The minimum input port receives the constant value of 1.

- **Receive side of the interrupt service routine** — The typical case with ASCII data has the minimum and maximum input ports disabled. The **Read to delimiter parameter** check box is selected and the **Delimiter** parameter has the value of carriage return or line feed. The value of the **Maximum read size** parameter is large (along the order of the FIFO size) and the value of **Minimum read size** parameter is 1. In this form, the driver acts like a nonblocking read line.

  An alternate receive-side configuration for fixed-length binary blocks of data has the value of the **Maximum read size** and **Minimum read size** parameters set to the fixed length of the block. The **Read to delimiter** parameter is not selected.

For complex data streams, consider using the FIFO Read HDRS and FIFO Read Binary blocks. For guidelines on when to use these blocks, see "RS-232 FIFO Read Blocks" on page 20-5.

Ports

Input

F — FIFO from which to read data
serialfifoptr

Connects to the software FIFO containing data read from the serial port.

MAX — Maximum number of bytes to read from FIFO
integer

The maximum number of bytes to return from the block.

Dependency

To make this port visible, set parameter **Max and Min read size ports**.

MIN — Minimum number of bytes to read from FIFO
integer

The minimum number of bytes to return from the block.

Dependency

To make this port visible, set parameter **Max and Min read size ports**.

Output

D — Parsed data read from FIFO
vector

Vector containing the parsed data read from the FIFO.

Dependency

To determine the data type of this vector, set the parameter **Output vector type**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32`

ENA — Pass value of MAX through
integer

Passes the value of port MAX through to the block that reads the ENA port.

Dependency

To make this port visible, set parameters **Max and Min read size ports** and **Enable passthrough**.

Parameters

Maximum read size — Maximum number of characters returned by block
1024 (default) | integer

Specify the maximum number of characters for this block to return. The resulting vector size is one more than this maximum number of characters. This block indicates the number of characters being returned by using the extra element as:

- A null terminator for the 8-bit data types
- The character count for the 16-bit and 32-bit data types

Enter a large enough number. If this number is too small, the block cannot return anything. For example, if you enter the value 10, but on execution the FIFO contains 11 characters plus the null terminator, the block does not return any characters. If the FIFO contains 5, the block returns 5 characters plus the null terminator.

If you select the parameter **Max and Min read size ports**, the block interprets the value input on port MAX as the maximum number of characters to return. The actual maximum number of characters to return is the smaller of the value on port MAX and the maximum read size in the block parameters. Use this value in binary mode when you have not selected the **Read to delimiter** check box.

Programmatic Use
Block Parameter: maxsize

Minimum read size — Minimum number of characters returned by block
1 (default) | integer

Enter the smallest read size in bytes. The FIFO must contain at least this number of elements before elements are returned.

If you select the parameter **Max and Min read size ports**, the value of port MIN supersedes this value.

Programmatic Use
Block Parameter: minsize

Read to delimiter — Return delimited element sets
on (default) | off

Select this check box to enable the return of element sets that terminate with the **Delimiter** value. Use this parameter when working with character-based elements.

Programmatic Use
Block Parameter: usedelimiter

Delimiter — Terminator value for delimited element sets
13 (default) | uint

Enter the decimal value for an 8-bit input terminator. This parameter specifies the value on which a FIFO read operation terminates. This value works with the **Read to delimiter** parameter. By default, this block looks for a carriage return. It returns characters only when one is found. For reference, the decimal value of a carriage return is 13 and a line feed is 10.

Programmatic Use
Block Parameter: delimiter

Output vector type — Specify output data type
8 bit uint null terminated (default) | count+32 bit int | count+32 bit uint | count +16 bit int | count+16 bit uint | 8 bit int null terminated

The 8-bit data types produce a NULL-terminated character vector in the output vector. For 16-bit and 32-bit data types, the first element contains the number of elements to expect in the rest of the output vector.

Programmatic Use
Block Parameter: `outputtype`

`Max and Min read size ports` — Enable maximum and minimum input ports
`off` (default) | on

When you select this check box:

- The value from input port MAX is the maximum number of characters to be removed from the FIFO. If this number exceeds the value of **Maximum read size**, the block disregards the value from the maximum input port. It takes the value of **Maximum read size** as the maximum number of characters to be removed from the FIFO.

- The value from the input port MIN is the minimum number of characters that the FIFO must contain before elements can be returned. This value supersedes the value set with the **Minimum read size** parameter.

This setting makes the input ports MAX and MIN visible.

Programmatic Use
Block Parameter: `enable`

`Enable passthrough` — Enable passthrough of MAX value
`off` (default) | on

Select this check box to pass the value of input port MAX through to output port ENA.

Dependency

Causes output port ENA to become visible.

Programmatic Use
Block Parameter: `enableout`

`Sample time` — Sample time of block
`-1` (default) | numeric

Enter the base sample time or a multiple of the base sample time. `-1` means that sample time is inherited.

Programmatic Use

`sampletime`

See Also
FIFO Write

Topics
"RS-232 Serial Communication" on page 20-2
"RS-232 Legacy Drivers" on page 20-3

Introduced in R2020b

FIFO Write

Write simple data streams
Library: Simulink Real-Time / RS232

Description

The FIFO Write block is the write side of a FIFO read/write pair. Use this block to generate simple data streams.

Ports

Input

D — Data to write to FIFO
vector

Vector containing the data to write to the FIFO.

Dependency

To determine the data type of this vector, set the parameter **Input vector type**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32`

Output

F — FIFO vector
serialfifoptr

Connects to the FIFO that writes data to the serial port.

DP — True if new data is present in the FIFO
true | false

If data is present in the FIFO, returns `true`.

Dependency

To make this port visible, set parameters **Max and Min read size ports** and **Enable passthrough**.

Parameters

Size — Size of FIFO, in bytes
1024 (default) | integer

Enter the number of elements that can be held in the FIFO at one time. If a write operation to the FIFO causes the number of elements to exceed **Size**, an error occurs.

Programmatic Use
Block Parameter: `size`

Input vector type — Specify input data type
8 bit uint null terminated (default) | count+32 bit int | count+32 bit uint | count
+16 bit int | count+16 bit uint | 8 bit int null terminated

For the 16-bit and 32-bit data types, include as first element the number of elements to expect in the rest of the input vector. The count controls how many bytes that the block copies into the FIFO. The block does not copy the count itself into the FIFO.

For the 8-bit data types, provide a NULL-terminated character vector in the output vector. The block copies data into the FIFO up to, but not including, the NULL terminator.

For more information, see "RS-232 Legacy Drivers" on page 20-3.

Programmatic Use
Block Parameter: `inputtype`

Data present output — Enables output DP
off (default) | on

Select this check box to create the Boolean output DP. If data is present in the FIFO, DP becomes `true`. The transmit side of the send/receive subsystem uses this output. This output is given to the Enable TX block, which enables the transmitter buffer empty interrupt.

This setting makes the output port DP visible.

Programmatic Use
Block Parameter: `present`

Sample time — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time. `-1` means that sample time is inherited.

Programmatic Use

`sampletime`

ID — Identifier for overflow messages
character vector

Enter a user-defined identifier for FIFO overflow messages.

Programmatic Use
Block Parameter: `id`

See Also
FIFO Read

Topics
"RS-232 Serial Communication" on page 20-2
"RS-232 Legacy Drivers" on page 20-3

Introduced in R2020b

FIFO Read HDRS

Read multiple ASCII data streams according to header information
Library: Simulink Real-Time / RS232

Description

The FIFO Read HDRS block identifies and separates ASCII data streams that have embedded identifiers.

The data following a particular header can have varying lengths, but has a common termination marker such as <CR><LF>. Although you can attain this same functionality with the FIFO Read block, it requires a complicated state machine with this behavior:

- If the same header arrives in the FIFO more than once after the block was last executed, the block returns the latest instance of the header. The block catches up with data that arrives faster than the block executes.

- If a header arrives in the FIFO that does not match an item in the headers list, the block discards the message.

- If bytes arrive in the FIFO that do not match a header, the block interprets the message as having an unspecified header. The block skips these bytes.

The `matlab/toolbox/slrealtime/examples` folder contains examples that show how to use the FIFO Read HDRS block: `slrt_ex_serialbaseboardasciitest` and `slrt_ex_serialbaseboardasciisplit`.

Ports

Input

F — FIFO from which to read data
serialfifoptr

Connects to the software FIFO containing data read from the serial port.

E — Enable read from FIFO
true | false

If `true`, read from FIFO.

Dependency

To make this port visible, set parameter **Enable input**.

Output

1 — Numbered output streams, one per header
vector

Vectors containing the parsed data read from the FIFO. Each output corresponds to one of the headers.

Dependency

To determine the data type of this vector, set the parameter **Output vector type**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32`

Parameters

Header — Search targets in ASCII data stream
cell array of character vector

Enter the headers that you want the block to look for in a block of data from the FIFO. Enter each header in single quotes as an element in a cell array.

Programmatic Use
Block Parameter: `hdr`

`Terminating string` — Characters that end data stream
[13 10] (default) | [integer]

Enter the terminating character vector for the data. Enter the characters defining the end of character vector, typically one or two characters.

Programmatic Use
Block Parameter: `nterm`

`Output behavior` — Behavior when no new data
`Zero output if no new data` (default) | `Hold last output if no new data`

From the list, select the behavior of the block if the FIFO has not received new data:

- `Hold last output if no new data` — Block keeps the output from the last FIFO message.
- `Zero output if no new data` — Block overwrites the first element of the output with `0`.

Programmatic Use
Block Parameter: `hold`

`Enable input` — Enable read from FIFO
`off` (default) | `on`

To create an input port that enables or disables the read operation, select this check box. The input port takes a Boolean signal.

Dependency

Causes input port E to become visible.

Programmatic Use
Block Parameter: `enable`

`Maximum read size` — Maximum number of characters returned by block
1024 (default) | integer

Specify the maximum number of characters for this block to return. The resulting vector size is one more than this maximum number of characters. This block indicates the number of characters being returned by using the extra element as:

• A NULL terminator for the 8-bit data types

• The character count for the 16-bit and 32-bit data types

Enter a large enough number. If this number is too small, the block cannot return anything. For example, if you enter the value 10, but on execution the FIFO contains 11 characters plus the null terminator, the block does not return any characters. If it contains 5, the block returns 5 characters plus the NULL terminator.

Programmatic Use
Block Parameter: `maxsize`

`Output vector type` — Specify output data type
8 bit uint null terminated (default) | count+32 bit int | count+32 bit uint | count +16 bit int | count+16 bit uint | 8 bit int null terminated

The 8-bit data types produce a NULL-terminated character vector in the output vector. For 16-bit and 32-bit data types, the first element contains the number of elements to expect in the rest of the output vector.

Programmatic Use
Block Parameter: `outputtype`

`Sample time` — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time. -1 means that sample time is inherited.

Programmatic Use

`sampletime`

See Also
FIFO Read | FIFO Write | FIFO Read Binary

Topics
"RS-232 Serial Communication" on page 20-2
"RS-232 Legacy Drivers" on page 20-3

Introduced in R2020b

FIFO Read Binary

Read multiple binary data streams according to header information
Library: Simulink Real-Time / RS232

Description

The FIFO Read Binary block reads multiple binary headers from a FIFO.

This block identifies and separates data by finding unique byte sequences (headers) that mark the data. Each header indicates the start of a fixed-length binary message. If the same header arrived in the FIFO more than once since the block was last executed, the block discards the older data. It then returns the latest instance of the header. The block catches up with data that arrives faster than the block executes.

The `matlab/toolbox/slrealtime/examples` folder contains examples that show how to use the FIFO Read HDRS block: `slrt_ex_serialbaseboardbinarytest` and `slrt_ex_serialbaseboardbinarysplit`.

Ports

Input

F — FIFO from which to read data
serialfifoptr

Connects to the software FIFO containing data read from the serial port.

E — Enable read from FIFO
true | false

If `true`, read from FIFO.

Dependency

To make this port visible, set parameter **Enable input**.

Output

1 — Numbered output streams, one per header
vector

Vectors containing the parsed data read from the FIFO. Each output corresponds to one of the headers.

Dependency

To determine the data type of this vector, set the parameter **Output vector type**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32`

Parameters

Header — Search targets in binary data stream
cell array of binary data

Enter the headers that you want the block to look for in a block of data from the FIFO. Enter each header as an element in a cell array either as a quoted character vector or a concatenation with `char(val)` for non-printable byte patterns.

Programmatic Use
Block Parameter: `hdr`

Message Lengths — Message lengths, in bytes
1024 (default) | integer

Enter the message length, in bytes. Include the header in the length.

Programmatic Use
Block Parameter: `lengths`

Output behavior — Behavior when no new data
Zero output if no new data (default) | Hold last output if no new data

From the list, select the behavior of the block if the FIFO has not received new data:

- `Hold last output if no new data` — Block keeps the output from the last FIFO message.

- `Zero output if no new data` — Block overwrites the first element of the output with 0.

Programmatic Use
Block Parameter: `hold`

Enable input — Enable read from FIFO
off (default) | on

To create an input port that enables or disables the read operation, select this check box. The input port takes a Boolean signal.

Dependency

Causes input port E to become visible.

Programmatic Use
Block Parameter: `enable`

Maximum read size — Maximum number of characters returned by block
1024 (default) | integer

Specify the maximum number of characters for this block to return. The resulting vector size is one more than this maximum number of characters. This block indicates the number of characters being returned by using the extra element as:

- A NULL terminator for the 8-bit data types

- The character count for the 16-bit and 32-bit data types

Enter a large enough number. If this number is too small, the block cannot return anything. For example, if you enter the value 10, but on execution the FIFO contains 11 characters plus the null terminator, the block does not return any characters. If it contains 5, the block returns 5 characters plus the NULL terminator.

Programmatic Use
Block Parameter: `maxsize`

Output vector type — Specify output data type
8 bit uint null terminated (default) | count+32 bit int | count+32 bit uint | count+16 bit int | count+16 bit uint | 8 bit int null terminated

The 8-bit data types produce a NULL-terminated character vector in the output vector. For 16-bit and 32-bit data types, the first element contains the number of elements to expect in the rest of the output vector.

Programmatic Use
Block Parameter: `outputtype`

Sample time — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time. -1 means that sample time is inherited.

Programmatic Use

`sampletime`

See Also

Topics
"RS-232 Serial Communication" on page 20-2
"RS-232 Legacy Drivers" on page 20-3

Introduced in R2020b

Modem Control

Control state of RTS and DTR output lines on serial port
Library:　　　　　Simulink Real-Time / RS232 / Mainboard
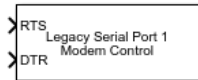
Description

The Modem Control block controls the state of either or both of the Request To Send (RTS) and Data Terminal Ready (DTR) output lines on the serial port. To choose which output lines to control, select the **RTS** and **DTR** parameters.

Ports

Input

RTS — Level-sensitive signal for setting ready-to-send line
double

The behavior of the block is:

- RTS > 0.5 — The block asserts the RTS control bit to `true`. The output goes to a positive voltage.
- RTS ≤ 0.5 — The block asserts the RTS control bit to `false`. The output goes to a negative voltage.

Dependency

If the **RTS** parameter is `off`, this input has no effect.

DTR — Level-sensitive signal for setting data-terminal-ready line
double

The behavior of the block is:

- DTR > 0.5 — The block asserts the DTR control bit to `true`. The output goes to a positive voltage.
- DTR ≤ 0.5 — The block asserts the DTR control bit to `false`. The output goes to a negative voltage.

Dependency

If the **DTR** parameter is `off`, this input has no effect.

Parameters

RTS — Enable control of RTS line for serial device
on (default) | off

Select this check box to control the RTS line for this board.

Programmatic Use

rts

DTR — Enable control of DTR line for serial device
on (default) | off

Select this check box to control the DTR line for this board.

Programmatic Use

dtr

Serial port — Select serial port
Legacy Serial Port 1 (default) | .. | Legacy Serial Port 8 | USB Serial Port 1 | .. | USB Serial Port 8

Selects the RS232 serial port for communications. If using USB-to-serial adapters, the target computer detects these adapters as serusb1, serusb2, and so on in the order that the adapters are connected to the serial devices. The order of port assignment is retained through the power cycle of the target computer or serial device if there is no change in the connections to the ports.

Programmatic Use

port

See Also
Legacy Serial Read | Legacy Serial Setup | Legacy Serial Write | Modem Status

Topics
"RS-232 Serial Communication" on page 20-2
"RS-232 Legacy Drivers" on page 20-3

Introduced in R2020b

Modem Status

Return state of modem control lines
Library: Simulink Real-Time / RS232 / Mainboard
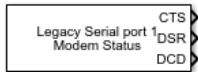
Description

The Modem Status block reads the state of the input modem control lines.

This block has outputs of type `Boolean`. If the input voltage is positive, the output is true. If the input voltage is negative, the output is false.

Ports

Output

CTS — Status of clear to send line
`true` | `false`

If `true`, the modem is ready to receive data.

Dependency

To make this output visible, select the **CTS** parameter.

DSR — Status of data set ready line
`true` | `false`

If `true`, the modem is ready to send and receive data.

Dependency

To make this output visible, select the **DSR** parameter.

DCD — Status of data carrier detect line
`true` | `false`

If `true`, the modem is receiving a carrier from a remote device.

Dependency

To make this output visible, select the **DCD** parameter.

Parameters

CTS — Enables clear to send status output
`on` (default) | `off`

Select this check box to monitor the CTS line of the modem.

Selecting this parameter makes the CTS port visible.

Programmatic Use

cts

DSR — Enables data set ready status output
on (default) | off

Select this check box to monitor the DSR line of the modem.

Selecting this parameter makes the DSR port visible.

Programmatic Use

dsr

DCD — Enables data carrier detect status output
on (default) | off

Select this check box to monitor the DCD line of the modem.

Selecting this parameter makes the DCD port visible.

Programmatic Use

dcd

Sample time — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time. -1 means that sample time is inherited.

Programmatic Use

sampletime

Serial port — Select serial port
Legacy Serial Port 1 (default) | .. | Legacy Serial Port 8 | USB Serial Port 1 | .. | USB Serial Port 8

Selects the RS232 serial port for communications. If using USB-to-serial adapters, the target computer detects these adapters as serusb1, serusb2, and so on in the order that the adapters are connected to the serial devices. The order of port assignment is retained through the power cycle of the target computer or serial device if there is no change in the connections to the ports.

Programmatic Use

port

See Also
Legacy Serial Read | Legacy Serial Setup | Legacy Serial Write | Modem Control

Topics
"RS-232 Serial Communication" on page 20-2
"RS-232 Legacy Drivers" on page 20-3

Introduced in R2020b

Legacy Serial Read

Read input data for baseboard serial communications
Library: Simulink Real-Time / RS232 / Mainboard
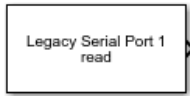
Description

The Legacy Serial Read block reads input data for baseboard serial communications.

Ports

Output

`Output` — Data from serial port read
serial data

The Output port provides the data from the serial port read.

Parameters

`Serial port` — Select serial port
`Legacy Serial Port 1` (default) | .. | `Legacy Serial Port 8` | `USB Serial Port 1` | .. | `USB Serial Port 8`

Select the RS232 serial port for communications. If using USB-to-serial adapters, the target computer detects these adapters as `serusb1`, `serusb2`, and so on in the order that the adapters are connected to the serial devices. The order of port assignment is retained through the power cycle of the target computer or serial device if there is no change in the connections to the ports.

Programmatic Use

`port`

`Max Read Count` — Select maximum word read count
`51` (default) | int

Use this value to select the maximum word read count.

Programmatic Use

`count`

`Output Datatype` — Select output data type
`8 bit uint null terminated` (default) | `8 bit int null terminated` | `count+16 bit uint` | `count+16 bit int` | `count+32 bit uint` | `count+32 bit int`

Use this value to select the output data type.

Programmatic Use

dtype

SampleTime — Select sample time in seconds
.01 (default) | double

Use this value to select the sample time in seconds.

Programmatic Use

sampletime

See Also
Legacy Serial Setup | Legacy Serial Write | Modem Control | Modem Status

Topics
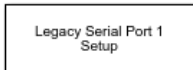"RS-232 Serial Communication" on page 20-2
"RS-232 Legacy Drivers" on page 20-3

Introduced in R2020b

Legacy Serial Setup

Set up baseboard serial communications
Library: Simulink Real-Time / RS232 / Mainboard

Description

The Legacy Serial Setup block sets up baseboard serial communications.

Parameters

`Serial port` — Select serial port
Legacy Serial Port 1 (default) | .. | Legacy Serial Port 8 | USB Serial Port 1 | .. | USB Serial Port 8

Select the RS232 serial port for communications. If using USB-to-serial adapters, the target computer detects these adapters as `serusb1`, `serusb2`, and so on in the order that the adapters are connected to the serial devices. The order of port assignment is retained through the power cycle of the target computer or serial device if there is no change in the connections to the ports.

Programmatic Use

`port`

`Baud rate` — Select serial port baud rate
115200 (default) | 57600 | 38400 | 19200 | 9600 | 4800 | 2400 | 1200 | 600 | 300 | 110

Use this value to select the serial port baud rate.

Programmatic Use

`baud`

`Number of data bits` — Select serial port data bits
8 (default) | 7 | 6 | 5

Use this value to select the number of serial port data bits.

Programmatic Use

`width`

`Number of stop bits` — Select serial port stop bits
1 (default) | 2

Use this value to select the number of serial port stop bits.

Programmatic Use

`nstop`

Parity — Select serial port parity
None (default) | Even | Odd | Mark | Space

Use this value to select serial port parity.

Programmatic Use

parity

Enable auto RTX/CTS — Select serial port RTX/CTS mode
off (default) | on

Use this value to select the serial port RTX/CTS mode.

Programmatic Use

ctsmode

See Also
Legacy Serial Read | Legacy Serial Write | Modem Control | Modem Status

Topics
"RS-232 Serial Communication" on page 20-2
"RS-232 Legacy Drivers" on page 20-3

Introduced in R2020b

Legacy Serial Write

Write output data for baseboard serial communications
Library: Simulink Real-Time / RS232 / Mainboard

> Legacy Serial Port 1
> write
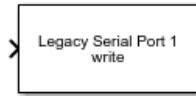
Description

The Legacy Serial Write block writes output data for baseboard serial communications.

Ports

Input

Input — Data for serial port write
serial data

The Input port contains the data to write for the serial port.

Parameters

Serial port — Select serial port
Legacy Serial Port 1 (default) | .. | Legacy Serial Port 8 | USB Serial Port 1 | .. | USB Serial Port 8

Select the RS232 serial port for communications. If using USB-to-serial adapters, the target computer detects these adapters as serusb1, serusb2, and so on in the order that the adapters are connected to the serial devices. The order of port assignment is retained through the power cycle of the target computer or serial device if there is no change in the connections to the ports.

Programmatic Use

port

Output Datatype — Select output data type
8 bit uint null terminated (default) | 8 bit int null terminated | count+16 bit uint | count+16 bit int | count+32 bit uint | count+32 bit int

Use this value to select the output data type.

Programmatic Use

dtype

SampleTime — Select sample time in seconds
.01 (default) | double

Use this value to select the sample time in seconds.

Programmatic Use

sampletime

See Also

Legacy Serial Read | Legacy Serial Setup | Modem Control | Modem Status

Topics
"RS-232 Serial Communication" on page 20-2
"RS-232 Legacy Drivers" on page 20-3

Introduced in R2020b

Legacy Serial Port

Send and receive data over mainboard baseboard serial port
Library: Simulink Real-Time / RS232 / Mainboard

> XMT   Legacy Serial Port 1   RCV >

Description

The Send/Receive block sets up the serial interface to send and receive basic character streams. This block has basic First In, First Out (FIFO) Read blocks inside the subsystem. It generates output as an array of packed integers (settable at 8 bits, 16 bits, or 32 bits). Characters appear in the lower byte and received status information appears in the upper byte.

Ports

Input

XMT — Vector of data to transmit
vector

Vector of the data used to transmit over the port.

Data Types: int8 | int16 | int32 | uint8 | uint16 | uint32

Output

RCV — Vector of data received over serial port
vector

Vector containing data that has been received from serial port.

Data Types: int8 | int16 | int32 | uint8 | uint16 | uint32

Parameters

Serial port — Port that is being accessed
Legacy Serial Port 1 (default) | .. | Legacy Serial Port 8 | USB Serial Port 1 | .. | USB Serial Port 8

This parameter specifies the port for which you want to view or modify parameters. If using USB-to-serial adapters, the target computer detects these adapters as `serusb1`, `serusb2`, and so on in the order that the adapters are connected to the serial devices. The order of port assignment is retained through the power cycle of the target computer or serial device if there is no change in the connections to the ports.

Programmatic Use
Block Parameter: `port`

Baud rate — Baud for transferring data
115200 (default) | 57600 | 38400 | 19200 | 9600 | 4800 | 2400 | 1200 | 600 | 300 | 110

Select a baud for transmitting and receiving data through the modem.

Programmatic Use
Block Parameter: baud

Data bits — Number of bits per character
8 (default) | 7 | 6 | 5

Select the number of bits that encode a character.

Programmatic Use
Block Parameter: width

Stop bits — Number of stop bits for port
1 (default) | 2

Select the number of stop bits for the character stream.

Programmatic Use
Block Parameter: nstop

Parity — Parity for checking data transfer
None (default) | Even | Odd | Mark | Space

Select a parity for checking data integrity.

Programmatic Use
Block Parameter: parity

Enable auto RTS/CTS — Enable RTS/CTS handshake
off (default) | on

To enable the Request To Send/Clear To Send (RTS/CTS) handshake of the Universal Asynchronous Receiver-Transmitter (UART) for flow control, select this check box. Serial controllers use the RTS/CTS handshake to prevent data loss due to hardware FIFO overflow on the device that you are sending to.

Usually, the interrupt service routine executes quickly enough to empty the FIFO. If your model gets FIFO overruns, select this check box.

Programmatic Use
Block Parameter: ctsmode

Max Output count — Maximum number of elements for block to return
1024 (default) | integer

Enter the maximum number of elements that you want returned by a single call to this block. The block uses this parameter to set the output vector width.

If you select the **Read to delimiter** check box and if the block does not find the delimiter before it reads **Receive maximum read** characters, the output vector is empty.

Programmatic Use
Block Parameter: maxread

Min Output count — Minimum number of elements for block to return
1 (default) | integer

Enter the minimum number of characters to read. If the FIFO does not contain at least this number of characters, the output vector is empty.

Programmatic Use
Block Parameter: minread

Read to delimiter — Return characters including message delimiter
on (default) | off

For the block to return all characters in the FIFO, up to and including the specified delimiter, select this check box.

If the buffer has errors, such as framing errors, the modem returns characters regardless of the presence of the delimiter. These returned characters help diagnose errors such as mismatched baud rates.

If the block does not find the delimiter before it reads **Receive maximum read** characters, the output vector is empty.

Programmatic Use
Block Parameter: usedelimiter

Delimiter — Numeric value of message delimiter
13 (default) | integer

Enter the numeric value of the character that is the message delimiter. Any value from 0 to 255 is valid. The common case looks for 10 (line feed) or 13 (carriage return).

Programmatic Use
Block Parameter: delimiter

Receive data type — Data type of receiver
8 bit int null terminated (default) | 8 bit uint null terminated | count+16 bit int | count+16 bit uint | count+32 bit int | count+32 bit uint

This parameter specifies the data type of the receiver. The 8-bit data types produce a NULL-terminated character vector in the output vector.

For 16-bit and 32-bit data types, the first element contains the number of valid elements in the rest of the output vector.

For 8-bit data types, only the character data is in the output vector, and a NULL terminator is appended. The 16-bit or 32-bit wide data types cause the error status from the UART to be placed in the second byte of each data element. (The error status contains the parity, overrun, framing, and break bits.) The character data is in the bottom 8 bits of each element. The first element of the vector contains the number of data elements that follow.

Programmatic Use
Block Parameter: odtype

Receive FIFO size — FIFO size in bytes
1024 (default)

This parameter specifies the receive FIFO size in bytes.

Example: 1024

Programmatic Use
Block Parameter: `fifosize`

Data Types: `int32`

Transmit data type — Data type of transmitter
count+32 bit int (default) | count+32 bit uint | count+16 bit int | count+16 bit uint | 8 bit int null terminated | 8 bit uint null terminated

This parameter specifies the data type of the transmitter. The 8-bit data types require a NULL-terminated character vector in the input vector.

The 16-bit and 32-bit data types reserve the first full element to contain the number of elements to expect in the rest of the input vector. Only the low-order byte of each data element is sent. Setting this data type enables a wider data type to hold the bytes.

If the data stream requires a NULL byte, select one of the 16-bit or 32-bit data types. Because the 8-bit data types are NULL-terminated character vectors, the NULL byte terminates the character vector.

Programmatic Use
Block Parameter: `idtype`

Sample Time — Sample time
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time. `-1` means that sample time is inherited.

Programmatic Use
Block Parameter: `sampletime`

See Also

Legacy Serial Port F

Topics
"RS-232 Serial Communication" on page 20-2
"RS-232 Legacy Drivers" on page 20-3

Introduced in R2008a

Legacy Serial Port F

Send and receive data over mainboard baseboard serial port with FIFO
Library: Simulink Real-Time / RS232 / Mainboard

[XMT   Legacy Serial Port 1   FIFO]
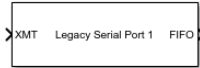
Description

The Send/Receive FIFO block sets up the serial interface to send and receive character and binary streams. It transmits input data as does the Send/Receive block, but it propagates received data through First In, First Out (FIFO) outputs.

A model that contains a Send/Receive FIFO block and the FIFO Read block provides the same capability as the Send/Receive block. A model that contains a Send/Receive FIFO block and a FIFO Read HDRS or FIFO Read Binary block provides greater capability than the Send/Receive block.

Ports

Input

XMT — Vector of data to transmit
vector

Vector of the data used to transmit over the port.

Data Types: `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output

FIF0 — FIFO of data received over serial port
serialfifoptr

First In, First Out (FIFO) containing data that has been received from the serial port.

Parameters

`Serial port` — Port that is being accessed
`Legacy Serial Port 1` (default) | .. | `Legacy Serial Port 8` | `USB Serial Port 1` | .. | `USB Serial Port 8`

This parameter specifies the port for which you want to view or modify parameters. If using USB-to-serial adapters, the target computer detects these adapters as `serusb1`, `serusb2`, and so on in the order that the adapters are connected to the serial devices. The order of port assignment is retained through the power cycle of the target computer or serial device if there is no change in the connections to the ports.

Programmatic Use
Block Parameter: `port`

Baud rate — Baud for transferring data
115200 (default) | 57600 | 38400 | 19200 | 9600 | 4800 | 2400 | 1200 | 600 | 300 | 110

Select a baud for transmitting and receiving data through the modem.

Programmatic Use
Block Parameter: baud

Data bits — Number of bits per character
8 (default) | 7 | 6 | 5

Select the number of bits that encode a character.

Programmatic Use
Block Parameter: width

Stop bits — Number of stop bits for port
1 (default) | 2

Select the number of stop bits for the character stream.

Programmatic Use
Block Parameter: nstop

Parity — Parity for checking data transfer
None (default) | Even | Odd | Mark | Space

Select a parity for checking data integrity.

Programmatic Use
Block Parameter: parity

Enable auto RTS/CTS — Enable RTS/CTS handshake
off (default) | on

To enable the Request To Send/Clear To Send (RTS/CTS) handshake of the Universal Asynchronous Receiver-Transmitter (UART) for flow control, select this check box. Serial controllers use the RTS/CTS handshake to prevent data loss due to hardware FIFO overflow on the device that you are sending to.

Usually, the interrupt service routine executes quickly enough to empty the FIFO. If your model gets FIFO overruns, select this check box.

Programmatic Use
Block Parameter: ctsmode

Port to modify — Port that is being accessed
1 (default) | 2

This parameter specifies the port for which you want to view or modify parameters. On the Simulink block, the upper port is port 1 and the lower port is port 2.

Programmatic Use
Block Parameter: port

Receive FIFO size — FIFO size in bytes
1024 (default)

This parameter specifies the receive FIFO size in bytes.

Example: `1024`

Programmatic Use
Block Parameter: `fifosize`

Data Types: `int32`

Transmit data type — Data type of transmitter
`count+32 bit int` (default) | `count+32 bit uint` | `count+16 bit int` | `count+16 bit uint` | `8 bit int null terminated` | `8 bit uint null terminated`

This parameter specifies the data type of the transmitter. The 8-bit data types require a NULL-terminated character vector in the input vector.

The 16-bit and 32-bit data types reserve the first full element to contain the number of elements to expect in the rest of the input vector. Only the low-order byte of each data element is sent. Setting this data type enables a wider data type to hold the bytes.

If the data stream requires a NULL byte, select one of the 16-bit or 32-bit data types. Because the 8-bit data types are NULL-terminated character vectors, the NULL byte terminates the character vector.

Programmatic Use
Block Parameter: `idtype`

Sample Time — Sample time
`-1` (default) | numeric

Enter the base sample time or a multiple of the base sample time. `-1` means that the sample time is inherited.

Programmatic Use
Block Parameter: `sampletime`

See Also

FIFO Read | FIFO Read HDRS | FIFO Read Binary | Legacy Serial Port

Topics
"RS-232 Serial Communication" on page 20-2
"RS-232 Legacy Drivers" on page 20-3

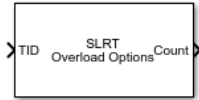**Introduced in R2008a**

Target Management

Target Management Blocks

The Simulink Real-Time target management blocks support target computer management functions.
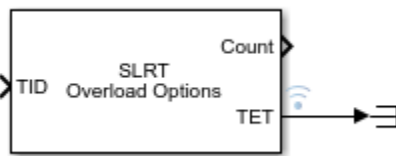
SLRT Overload Options

Select CPU overload options
Library: Simulink Real-Time / Target Management

Description

The SLRT Overload Options block configures CPU overload options for the model rate identified through the TID port. This block outputs the current CPU overload count for the identified rate. If the **Enable TET Output** parameter is on, the block outputs TET data that you can mark for data logging in the Simulation Data Inspector.

Ports

Input

TID — Task ID
int

Connect any signal from the model to the TID (Task ID) port and the block applies the overload options to the model task in which the signal executes.

Example: 1

Output

Count — Overload count
int

The CPU overload count for the selected model rate.

Parameters

`Startup Duration` — Select startup steps for ignoring CPU overloads
1 (default) | int

The number of startup execution steps for which overloads are ignored.

Programmatic Use

`startupDur`

`Source` — Selects CPU overload input during startup
Tunable Parameter (default) | Input Port

The source of CPU overloads for the model rate during startup can be a tunable parameter or a block input port.

Programmatic Use

`startupDurSrc`

`Max Overloads` — Select maximum CPU overloads
0 (default) | int

The number of accepted overloads before execution is halted. There is a separate overload counter for each sub-rate. Each counter is compared to **Max Overloads** separately, not compared to the sum of the counters.

Programmatic Use

`maxOverload`

`Source` — Selects CPU overload input after startup
Tunable Parameter (default) | Input Port

The source of CPU overloads for the model rate after startup can come from a tunable parameter or from the block input port.

Programmatic Use

`maxOverloadSrc`

`Enable TET Output` — Enable output TET data
off (default) | on

When enabled, the CPU overload data is output for display in the TET monitor and the Simulation Data Inspector.

Programmatic Use

`TETFlag`

See Also
Thread Trigger

Topics
"CPU Overload"
"Troubleshoot Overloaded CPU from Executing Real-Time Application"

Introduced in R2020b

Utilities

Utility Blocks

The Simulink Real-Time utility blocks support utility functions. Some of these blocks exist in the Utilities library, available at the top level of the Simulink Real-Time Block Library. Other blocks are available as sublibraries of the I/O function that they support.

Bit Packing

Construct data frames
Library: Simulink Real-Time / Utilities

Description

The Bit Packing block constructs data frames. Its output port is typically connected to an input port of a Send block or Digital Output block. The block has one output port. This port can be a vector of arbitrary size. It represents the data frame entity constructed by the signals entering the block at its input ports. The number of input ports depends on the setting in the block dialog box.

Parameters

Bit Patterns — Packed data bit pattern
{[0:31]} (default)

Specify bit patterns. The data type entered in the control must be a MATLAB cell array vector. The number of elements in the cell array define the number of input ports shown by this block instance. The cell array elements must be of type double array and define the position of each bit of the incoming value (data typed input port) in the outgoing double value (data frame). From a data type perspective (input ports), the block behaves like a Simulink Sink block. The data types of the input ports are inherited from the driving blocks.

Programmatic Use
Block Parameter: BitPatterns

Output port (packed) data type — Packed data type
uint32 (default) | double | single | int8 | uint8 | int16 | uint16 | int32 | boolean

From the list, select an output port (packed) data type.

Programmatic Use
Block Parameter: PackDataType

Output port (packed) dimensions — Packed data dimensions
[1] (default)

Specify the dimensions the output port (packed). Enter this value as a vector. Specify the size of the port by using a format compatible with the MATLAB size command.

Programmatic Use
Block Parameter: PackDataSize

See Also
Bit Unpacking

Introduced in R2006a

Bit Unpacking

Deconstruct data frames
**Library:**   Simulink Real-Time / Utilities

Description

The Bit Unpacking block extracts data frames. Its input port is typically connected to an output port of a Receive block or Digital Input block.

The block has one input port, which represents the data frame entity from which the signals are extracted and leaving the block at its output ports. The number of output ports and the data type of each output port depend on the settings in the block dialog box.

Bit Unpack Four Bytes

This example shows how to configure a Bit Upacking block to:

- Receive a 32-bit word as input by using input port data type `uint32`.
- Unpack four 8-bit words (bytes) from the input data by using a bit pattern.
- Send four 8-bit words as output by using the output data type `uint8`.
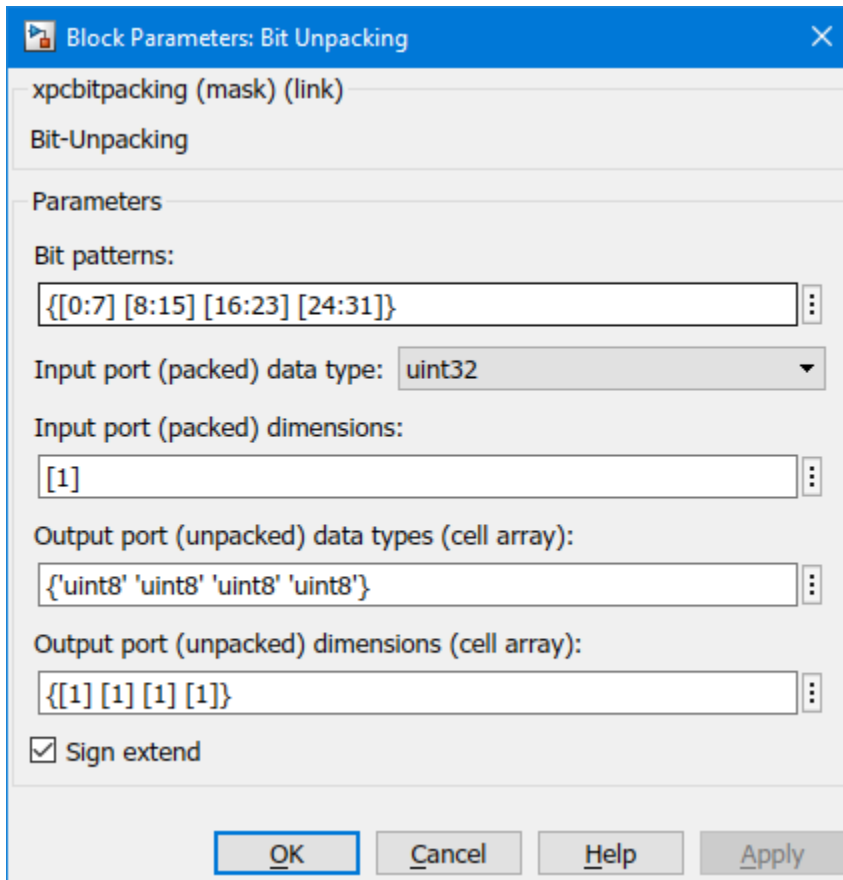
After configuring the block parameters, the Bit Upacking block appears as shown.
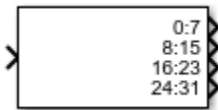
Parameters

Bit Patterns — Select bit pattern
{[0:31]} (default)

Specify bit patterns. The data type must be a MATLAB cell array vector. The number of elements in the cell array define the number of input ports shown by this block instance. The cell array elements must be of type double array and define the position of each bit of the incoming value (data typed input port) in the outgoing double value (data frame). From a data type perspective, the block behaves like a Sink block. The **Input port (packed) data types** specify the data type of the input port.

Programmatic Use
Block Parameter: BitPatterns

Input port (packed) data types — Packed data type
uint32 (default) | double | single | int8 | uint8 | int16 | uint16 | int32 | boolean

From the list, select an input port (packed) data type.

Programmatic Use
Block Parameter: PackDataType

Input port (packed) dimension — Packed data dimension
[1] (default)

Specify the dimensions of the input port (packed). Enter this value as a vector. Specify the size of the port by using a format compatible with the MATLAB size command.

Programmatic Use
Block Parameter: PackDataSize

Output port (unpacked) data types (cell array) — Unpacked data type
{'uint32'} (default)

The output ports (packed) can be of an arbitrary data type. The number of elements in the cell array define the number of output ports shown by this block instance. The data types are:

- double
- single
- int8
- uint8
- int16
- uint16
- int32
- uint32
- boolean

Programmatic Use
Block Parameter: UnpackDataTypes

Output port (unpacked) dimension (cell array) — Unpacked data dimension
{[1]} (default)

Specify the dimensions of each output port (unpacked). Enter this value as a cell array of vector sizes.

Programmatic Use
Block Parameter: UnpackDataSizes

Sign extend — Enable sign extension
on (default) | off

Select this check box to enable sign extension. If you select this check box and unpack the data frame into a signed type (int8, int16, or int32), the block performs sign extension. For example, if the bit pattern is [0:4], and the data type is int8, you are extracting 5 bits into an 8-bit wide signed type. In this case, bits 5, 6, and 7 are the same as bit 4, resulting in sign extension. This functionality enables you to pack and unpack negative numbers without losing precision.

Programmatic Use
Block Parameter: SignExtend

See Also

Bit Packing

Introduced in R2006a

Byte Packing

Construct data frames
Library: Simulink Real-Time / Utilities

Description

The Byte Packing block converts one or more signals of user-selectable data types to a single vector of varying data types. The output of this block typically connects to an input port of a Send block.

The Byte Packing block and the Byte Unpacking block support the `slrealtime.tlc` code generation target and generate code that runs on Speedgoat target machines. Due to considerations such as endianness and addressable word size, these blocks can generate incorrect results for other code generation targets or target computers.

For example, suppose that you are packing three signals into a vector of `uint8`. The signals have the following attributes:

Dimension	Size	Type
Scalar	1	single
Vector	3	uint8
Vector	3	uint8

1 Set the packed output port data type to `uint8`.

2 Set the input port data type to a cell array encoding the data types:

   `{'single', ['uint8'], ['uint8']}`

   Use square brackets to represent vectors.

3 Set the byte alignment value to `1`.

4 Connect the signals to the Byte Packing block.

Input/Output Ports

Input

Port_1 — First of *N* input ports
scalar | vector

The block has from `1` to *N* input ports. Specify the number of input ports and their types by entering them as a cell array in the parameter **Input port (unpacked) data types (cell array)**.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean`

Output

Port_1 — Output port containing packed data
vector

The block displays one output port that transmits a vector of packed data. You determine the data type of the packed data by setting **Output port (packed) data type**.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean`

Parameters

Output port (packed) data type — Data type for the packed output signal
`uint8` (default) | `double` | `single` | `int8` | `int16` | `uint16` | `int32` | `uint32` | `boolean`

From the list, select a data type for the output port.

Programmatic Use
Block Parameter: `MaskPackedDataType`

Input port (unpacked) data types (cell array) — Data types for the unpacked input signals
`{'uint8'}` (default) | `double` | `single` | `int8` | `int16` | `uint16` | `int32` | `uint32` | `boolean`

Specify as a cell array the data types of the input ports (unpacked) for the different input signals. The number of elements in the cell array determines the number of input ports shown by this block instance. To represent vector elements, use square brackets in the cell array.

Programmatic Use
Block Parameter: `MaskUnpackedDataTypes`

Byte Alignment — Alignment of the input signal data types after packing
1 (default) | 2 | 4 | 8

Each element in the input signals list starts at a multiple of the alignment value, specified from the start of the vector. If the alignment value is larger than the size of the data type in bytes, the block fills the space with pad bytes of value 0.

For example, if the alignment value is 4:

- `uint32` receives no padding
- `uint16` receives 2 bytes of padding
- `uint8` receives 3 bytes of padding

If the model accesses the data items frequently, consider selecting an alignment value equal to the largest data type that you want to access. If the model transfers data items frequently as a group, consider an alignment value of 1, which packs the data into as small a space as possible.

Programmatic Use
Block Parameter: `MaskAlignment`

See Also
Byte Unpacking

Introduced in R2006a

Byte Reversal/Change Endianess

Reverse little-endian data for big-endian processor
Library: Simulink Real-Time / Utilities

Description

You use the Byte Reversal/Change Endianess block for communication between a Simulink Real-Time system and a system running with a processor that is *big-endian*. Processors compatible with the Intel 80x86 family are *little-endian*. For this situation, insert a Byte Reversal/Change Endianess block before the Pack block and another just after the Unpack block. The following is the Change Endianess block.

Parameters

Block Parameters for Change Endianess

Number of input ports — Number of ports
1 (default)

The number of input ports adjusts to follow this parameter and the number of outputs is equal to the number of inputs.

Programmatic Use
Block Parameter: numInp

Machine word length — Word length for conversion
Byte (default) | Word | Double Word

Select a machine word length from the list to which to convert the data.

Programmatic Use
Block Parameter: />

Byte Reversal Block Parameters

Number of inputs — Number of ports
1 (default)

The number of input ports adjusts to follow this parameter and the number of outputs is equal to the number of inputs.

Programmatic Use
Block Parameter: numInp

See Also

Introduced in R2006a

Byte Unpacking

Deconstruct data frames
Library: Simulink Real-Time / Utilities

Description

The Byte Unpacking block converts a vector of varying data types into one or more signals of user-selectable data types. The input of this block typically connects to an output port of a Receive block.

The Byte Packing block and the Byte Unpacking block support the `slrealtime.tlc` code generation target and generate code that runs on Speedgoat target machines. Due to considerations such as endianness and addressable word size, these blocks can generate incorrect results for other code generation targets or target computers.

For example, suppose that you are unpacking a `uint8` vector signal into three signals. The signals have these attributes:

Dimension	Size	Type
Scalar	1	single
Vector	3	uint8
Vector	3	uint8

1   Set the output port data type to:

   `{'single', ['uint8'], ['uint8']}`

   Use square brackets to represent vectors.
2   Set the output port dimension to:

   `{[1],[3],[3]}`
3   Set the alignment value to `1`.
4   Connect the output signals to the Byte Unpacking block.

Input/Output Ports

Input

Port_1 — Input port containing packed data
vector

The block displays one input port that receives a vector of packed data. The source of the packed data determines by inheritance the data type of the packed data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean`

Output

Port_1 — First of *N* output ports
scalar | vector

The block displays from 1 to *N* output ports, as specified by elements of the cell array in the parameter **Output port (unpacked) data types (cell array)**.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean

Parameters

Output port (unpacked) data types (cell array) — Data types for the unpacked output signals
uint8 (default) | double | single | int8 | int16 | uint16 | int32 | uint32 | boolean

Specify as a cell array the data types of the output ports (unpacked) for the different output signals. The number of elements in the cell array determines the number of output ports shown by this block instance. To represent vector elements, use square brackets in the cell array.

Programmatic Use
Block Parameter: MaskUnpackedDataTypes

Output port (unpacked) dimensions (cell array) — Dimensions of each output port (unpacked)
{[1]} (default) | {[*N*], [*M*], ...}

Specify the dimensions of the output ports as a cell array of vectors.

Programmatic Use
Block Parameter: MaskUnpackedDataSizes

Byte Alignment — Alignment of the output signal data types before unpacking
1 (default) | 2 | 4 | 8

Each element in the output signals list starts at a multiple of the alignment value, specified from the start of the input vector. If the alignment value is larger than the size of the data type in bytes, the vector contains pad bytes of value 0.

For example, if the alignment value is 4:

- uint32 receives no padding
- uint16 receives 2 bytes of padding
- uint8 receives 3 bytes of padding

Programmatic Use
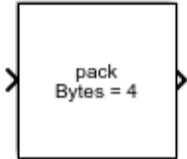**Block Parameter:** MaskAlignment

See Also
Byte Packing

Introduced in R2006a

Shared Memory Pack

Shared memory pack
Library: Simulink Real-Time / Shared Memory

Description

This block packs the specified partition structure into an unstructured double word array vector. It converts one or more Simulink signals of varying data types into the vector. Typically, the input to a pack block is the output from a write block. The Simulink interface is not aware of structures; pass the output of each structure segment as input to the Shared Memory Pack block.

Memory partitions consist of groups of Simulink signals, which are combined into blocks (packets) of 32-bit words. Before you begin to configure this block, be sure that you have a predefined shared memory partition structure as required by the shared memory manufacturer.

This block ignores the **Address** field of the partition structure.

Parameters

Partition struct — Name of structure
[] (default)

Enter the name of the predefined shared memory partition structure.

Programmatic Use
Block Parameter: partition

See Also
Shared Memory Unpack

Introduced in R2006a

Shared Memory Unpack

Shared memory unpacking
Library: Simulink Real-Time / Shared Memory
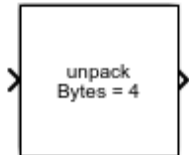
Description

This block unpacks an unstructured double word array vector (from the Shared Memory Pack block) into the specified partition structure.

Before you begin to configure this block, be sure that you have a predefined shared memory partition structure as required by the shared memory manufacturer.

This block ignores the **Address** field of the partition structure.

Parameters

`Partition struct` — Name of structure
`[]` (default)

Enter the name of the predefined shared memory partition structure. The block unpacks the double word array vector into this structure.

Programmatic Use
Block Parameter: `partition`

See Also

Shared Memory Pack

Introduced in R2006a

XCP Universal Measurement and Calibration Protocol

XCP Master Mode

XCP Master Mode

The Universal Measurement and Calibration Protocol (XCP) is a network protocol that you can use to connect calibration systems to electronic control units (ECUs).

A node in the network can run in either master mode or slave mode. Simulink Real-Time supports XCP in master mode to replace (bypass) a subsystem of the ECU controller. The bypass model acquires input signals from the ECU system, computes the output, and stimulates the result.

To support XCP master mode, the Simulink Real-Time software provides the XCP sublibrary. You can:

- Acquire real-time measurement data by using the XCP CAN Data Acquisition block or XCP UDP Data Acquisition block.
- Attach incoming data to software interrupts by using the XCP UDP Bypass block.
- Stimulate real-time measurement data by using the XCP CAN Data Stimulation or XCP UDP Data Stimulation block.

To create an XCP master model:

- Provide an A2L (ASAP2) format file that contains signal, parameter, and XCP-specific network elements for the slave ECU.
- Provide an XCP Configuration block to load the A2L data into the XCP database.
- Provide one XCP CAN Transport Layer for each XCP CAN Configuration block.

  Simulink Real-Time supports XCP implemented by using FIFO mode CAN or real-time UDP as transport protocols.
- Apply stimulus data to the slave device by using the XCP Data Stimulation block.
- Acquire measurement data from the slave device by using the XCP Data Acquisition block.

See Also

XCP CAN Configuration | XCP CAN Data Acquisition | XCP CAN Data Stimulation | XCP CAN Transport Layer | XCP UDP Data Acquisition | XCP UDP Data Stimulation | XCP UDP Configuration

More About

- "CAN"
- "Real-Time UDP"
- "Third-Party Calibration Support"

Stimulation Support

Control and Update Stimulation of Inports to Real-Time Application

You can stream data to the inports of a real-time application on a target computer by using the `Target.Stimulation` object and functions. The functions enables you to control inport stimulation for individual or all inports and monitor stimulation status.

To load data to the inports, create a time series object by using, for example the timeseries function. Load the object into the inport. Do not pause or stop the stimulation of inports before or during the stimulation. A pause or stop generates a stimulation error.

To control or monitor stimulation of inports in a real-time application:

- Start the stimulation of a specific inport or all inports by using the `start` function.
- Pause the stimulation of a specific inport or all inports by using the `pause` function.
- Stop the stimulation of a specific inport or all inports by using the `stop` function.
- Get the status of stimulation of inports by using the `getStatus` function.
- Load data to specific inports by using the `reloadData` function.

Stimulate Root Inport by Using MATLAB Language

This example shows how to stimulate root inports in a model by using the Stimulation object and related functions:

- start
- stop
- getStatus
- reloadData
- pause

Open Model and Map Inport to Wave Data

Open model `slrt_ex_osc_inport`. Save the model to a working folder. Map the inport to use square wave data. For inport `In1`, interpolated is off.

```
model = ('slrt_ex_osc_inport');
open_system(model);
load(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_inport_square.mat'));
waveform = square;
set_param(model,'ExternalInput','waveform');
set_param(model,'LoadExternalInput','on');
set_param(model,'StopTime','Inf');
```

Model slrt_ex_osc_inport
Simulink Real-Time example model

Copyright 2020 The MathWorks, Inc.

Build Model and Download Real-Time Application

Build, download, and execute the real-time application.

```
evalc('slbuild(model)');
tg = slrealtime('TargetPC1');
load(tg,model);
```

Stimulate Root Inport Data

Start root inport stimulation of inports 1. Open Scope block and observe results.

```
start(tg.Stimulation,[1]);
start(tg);
```

Pause root inport stimulation of inport 1.

```
pause(tg.Stimulation,[1]);
```

Stop and start the stimulation of inport 1.

```
stop(tg.Stimulation,[1]);
start(tg.Stimulation,[1]);
```

Check the status of stimulation of the inports.

```
getStatus(tg.Stimulation,'all');
```

Create a time-series object to load data to an inport.

```
sampleTime = 0.1;
endTime = 10;
numberOfSamples = endTime * 1/sampleTime + 1;
timeVector = (0:numberOfSamples) * sampleTime;
u = timeseries(timeVector*10,timeVector);
```

Object u is created for 10 seconds. Load it to the inport 1. Stimulation of an inport should be stopped before loading data.

```
stop(tg.Stimulation,[1]);
reloadData(tg.Stimulation,[1],u);
```

Stop real-time application and close all.

```
stop(tg);
bdclose('all');
```

See Also

`timeseries` | `Target.Stimulation`

XCP Blocks

XCP CAN Transport Layer

Generate and consume XCP messages that are transported by CAN hardware
Library: Simulink Real-Time / XCP

Description

The XCP CAN Transport Layer block handles CAN messages that your model transmits or receives by using Simulink Real-Time CAN library blocks.

Connect the input side of the block to a block that receives CAN messages. Connect the output side of the block to a block that transmits the XCP messages over CAN. Set up the transmitting block so that a CAN message is sent only when an XCP message is available. Otherwise, the block sends 0 byte data when XCP messages are not available, causing undefined behavior.

Ports

Input

CAN Msg — CAN MESSAGE structures being consumed
vector

Vector of CAN MESSAGE structures being consumed.

N — Number of messages
integer

Number of messages in the vector.

Output

CAN Msg — CAN MESSAGE structures being generated
vector

Vector of CAN MESSAGE structures being generated.

N — Number of messages
integer

Number of messages in the vector.

See Also
XCP CAN Configuration | XCP CAN Data Acquisition | XCP CAN Data Stimulation | XCP CAN Transport Layer

External Websites
www.asam.net

Introduced in R2020b

XCP CAN Configuration

Configure XCP server connection

Library: Vehicle Network Toolbox / XCP Communication / CAN
                      Simulink Real-Time / XCP / CAN
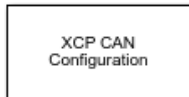
Description

The XCP CAN Configuration block uses the parameters specified in the A2L file and the ASAP2 database to establish an XCP server connection.

Before you acquire or stimulate data, specify the A2L file to use in your XCP CAN Configuration. Use one XCP CAN Configuration to configure one server connection for data acquisition or stimulation. If you add XCP CAN Data Acquisition and XCP CAN Data Stimulation blocks, your model checks to see if there is a corresponding XCP CAN Configuration block. If there is no corresponding XCP CAN Configuration block, the model prompts you to add one.

The XCP CAN communication blocks support Simulink accelerator mode and rapid accelerator mode. You can speed up the execution of Simulink models by using these modes. For more information about these simulation modes, see "Design Your Model for Effective Acceleration".

Parameters

Config name — Specify XCP CAN session name
'CAN_Config1' (default)

Specify a unique name for your XCP CAN session.

A2L File — Select an A2L file
file name

Click **Browse** to select an A2L file for your XCP CAN session..

Enable seed/key security — Select that key required to establish connection
'off'

Select this option if your server requires a secure key to establish connection. Use the **File (*DLL)** parameter to specify the DLL file that contains the seed/key definition.

File (*.DLL) — Select file for seed and key security
file name

If you select **Enable seed/key security** (EnableSecurity), this field is enabled. Click **Browse** to select the file that contains the seed and key security algorithm that unlocks an XCP server module. This parameter is available in Windows Desktop Simulation for Vehicle Network Toolbox.

The **File (*.DLL)** parameter specifies the name of the DLL-file that contains the seed and key security algorithm used to unlock an XCP server module. The file defines the algorithm for generating the

access key from a given seed according to ASAM standard definitions. For information on the file format and API, see the Vector web page Steps to Use Seed&Key Option in CANape or "Seed and Key Algorithm" in National Instruments® CAN ECU Measurement and Calibration Toolkit User Manual. **Note:** The DLL must be the same bitness as MATLAB (64-bit).

Output connection status — Display connection status
`'off'`

Select this option to display the status of the connection to the server module. Selecting this option adds a new output port.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

The XCP communication blocks support code generation with limited deployment portability that runs only on the host computer or Simulink Real-Time targets.

Code generation requires a C++ compiler that is compatible with the code generation target. For more information, see Supported and Compatible Compilers.

See Also

Blocks
XCP CAN Data Acquisition | XCP CAN Data Stimulation | XCP CAN Transport Layer

Introduced in R2013a

XCP CAN Data Acquisition

Acquire selected measurements from configured server connection
Library: Vehicle Network Toolbox / XCP Communication / CAN
Simulink Real-Time / XCP / CAN

XCP CAN
Data Acquisition

Description

The XCP CAN Data Acquisition block acquires data from the configured server connection based on measurements that you select. The block uses the XCP CAN transport layer to obtain raw data for the selected measurements at the specified simulation time step. Configure your XCP connection and use the XCP CAN Data Acquisition block to select your event and measurements for the configured server connection. The block displays the selected measurements as output ports.

The XCP communication blocks support the use of Simulink accelerator mode and rapid accelerator mode. You can speed up the execution of Simulink models by using these modes. For more information on these simulation modes, see "Design Your Model for Effective Acceleration".

Parameters

`Config name` — Specify XCP CAN session name
select from list

Select the name of the XCP configuration that you want to use. This list displays all available names specified in the XCP CAN Configuration blocks in the model. Selecting a configuration displays events and measurements available in the A2L file of this configuration.

Note You can acquire measurements for only one event by using an XCP CAN Data Acquisition block. Use one block for each event whose measurements you want to acquire.

`Event name` — Select an event
select from list

Select an event from the available list of events. The XCP CAN Configuration block uses the specified A2L file to populate the events list.

`All Measurements` — List all measurements available for event
measurements list

This list displays all measurements available for the selected event. Select the measurement that you want to use and click the add button, to add it to the selected measurements. On your keyboard, press the **Ctrl** key to select multiple measurements.

In the Block Parameters dialog box, type the name of the measurement you want to use in the **Search** box. The **All Measurements** list displays a list of all matching names. Click the x to clear your search.

Selected Measurements — List selected measurements
measurement names

This list displays selected measurements. To remove a measurement from this list, select the

measurement and click the remove button, .

In the Block Parameters dialog box, use the toggle buttons to reorder the selected measurements.

Block Output Settings — Set the port output as Compu method conversion values or raw values
Raw values as double (no Compu method conversion) (default) | Raw values (no Compu method conversion) | Physical values (apply Compu method conversion)

This parameter enables support for XCP data types and dimensions as defined in the ASAP2 standard. The Block Output Settings parameter selects whether the port outputs Compu method conversion values or raw values. The options provide:

- `Physical values (apply Compu method conversion)` enables the raw-to-physical conversion of ECU measurement values. For this option, the block port settings are set either to `'double'` or `'string'`, based on the underlying Compu method conversion. For example, Compu method `IDENTICAL`, `LINEAR`, `RAT_FUNC`, `TAB_INTP`, and `TAB_NOINTP` port settings is `'double'` while Compu method `TAB_VERB` port settings is `'string'`. The maximum string length supported for Compu method conversion is 1024 as specified in the ASAM XIL specification.

  The FORM Compu method conversion is not supported. Simulink throws a warning for such a conversion and IDENTICAL conversion is applied to the underlying measurement. Also, only scalar measurement signals are supported for TAB_VERB conversion as Simulink only supports scalar strings.

  Selecting this option shows the physical units (if any) in front of the measurement name on the block mask. This physical unit is acquired from the A2L description of the measurement and Compu method. If the physical unit is not specified, only the measurement name is displayed.
- `Raw values (no Compu method conversion)` sets the port data type according to the type definition in the A2L file and supports up to three-dimensional XCP measurements in Simulink.
- `Raw values as double (no Compu method conversion)` sets the port data type as `double`, converting all internal measurement values. This selection supports up to three-dimensional XCP measurements in Simulink.

These ASAP2 data types are supported by corresponding Simulink port data types:

- SBYTE
- UWORD
- SWORD
- ULONG

- SLONG
- A_UINT64
- A_INT64
- FLOAT32_IEEE
- FLOAT64_IEEE

The dimension support in the block accommodates the different treatment of matrices by MATLAB and the ECU. The MATLAB default operation treats matrices as row-major matrices. An XCP measurement can have a LAYOUT as COLUMN_DIR or ROW_DIR . If a matrix measurement is COLUMN_DIR, the blocks rearrange the measurement in memory and ensure that the matrix (row X, col Y) in MATLAB refers to the same entry as (row X, col Y) on the ECU. The rearrangement causes matrix entries that are contiguous on the ECU to be noncontiguous in MATLAB and Simulink.

DAQ List Priority — Specify a priority value for server device driver
priority value

Specify a priority value as an integer from 0 to 255 for the server device driver to prioritize transmission of data packets. The server can accumulate XCP packets for lower priority DAQ lists before transmission to the client. A value of 255 has the highest priority. The SET_DAQ_LIST_MODE command communicates the **DAQ List Priority** value from client to server. This communication method differs from the specification of the Event Channel Priority property, which comes from the A2L file.

Sample time — Specify sampling time of block
0.01 (default)

Specify the sampling time of the block during simulation, which is the simulation time. This value defines the frequency at which the XCP CAN Data Acquisition block runs during simulation. If the block is inside a triggered subsystem or is to inherit sample time, you can specify −1 as your sample time. You can also specify a MATLAB variable for sample time. The default value is 0.01 simulation seconds. For more information, see "What Is Sample Time?".

Enable Timestamp — Enable reading timestamp from incoming DTO packets
off (default) | on

When the Timestamp is enabled, the block reads the timestamp from incoming DTO packets and outputs the timestamp to Simulink. The **Enable Timestamp** check box appears in the block parameters dialog box when the parameter is supported in the A2L file.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

The XCP communication blocks support code generation with limited deployment portability that runs only on the host computer or Simulink Real-Time targets.

Code generation requires a C++ compiler that is compatible with the code generation target. For more information, see Supported and Compatible Compilers.
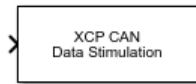
See Also

Blocks
XCP CAN Configuration | XCP CAN Data Stimulation | XCP CAN Transport Layer

Introduced in R2013a

XCP CAN Data Stimulation

Perform data stimulation on selected measurements
Library: Vehicle Network Toolbox / XCP Communication / CAN
Simulink Real-Time / XCP / CAN

```
      XCP CAN
  Data Stimulation
```

Description

The XCP CAN Data Stimulation block sends data to the selected server connection for the selected event measurements. The block uses the XCP CAN transport layer to output raw data for the selected measurements at the specified stimulation time step. Configure your XCP session and use the XCP CAN Data Stimulation block to select your event and measurements on the configured server connection. The block displays the selected measurements as input ports.

The XCP communication blocks support Simulink accelerator mode and rapid accelerator mode. You can speed up the execution of Simulink models by using these modes. For more information about these simulation modes, see "Design Your Model for Effective Acceleration".

Parameters

`Config name` — Specify XCP CAN session name
select from list

Select the name of XCP configuration that you want to use. This list displays all available names specified in the available XCP CAN Configuration blocks in the model. Selecting a configuration displays events and measurements available in the A2L file of this configuration. You can stimulate measurements for only one event by using an XCP CAN Data Stimulation block. Use one block for each event whose measurements you want to stimulate.

`Event name` — Select an event
select from list

Select an event from the event list. The XCP CAN Configuration block uses the specified A2L file to populate the events list. The block is configured with the corresponding event number from the A2L.

The event time cycle does not control transmission of stimulation packets. The block stimulates each time it executes. For use in Simulink simulation, consider enabling simulation pacing to avoid free-running stimulation.

`All Measurements` — List all measurements available for event
measurements list

This list displays all measurements available for the selected event. Select the measurement that you want to use and click the add button, to move it to the selected measurements. Hold the `Ctrl` key on your keyboard to select multiple measurements.

In the block parameters dialog box, type the name of the measurement you want to use in the **Search** box. The **All Measurements** lists displays a list of all matching names. Click the x to clear your search.

Selected Measurements — List selected measurements
measurement names

This list displays your selected measurements. To remove a measurement from this list, select the

measurement and click the remove button, .

In the **Block Parameters** dialog box, use the toggle buttons to reorder the selected measurements.

Block Input Settings — Set the port input as Compu method conversion values or raw values
Raw values as double (no Compu method conversion) (default) | Raw values (no Compu method conversion) | Physical values (apply Compu method conversion)

This parameter enables support for XCP data types and dimensions as defined in the ASAP2 standard. The Block Input Settings parameter selects whether the port outputs Compu method conversion values or raw values. The options provide:

- `Physical values (apply Compu method conversion)` enables the physical-to-raw conversion of ECU measurement values. For this option, the block port settings are set either to `'double'` or `'string'`, based on the underlying Compu method conversion. For example, Compu method IDENTICAL, LINEAR, RAT_FUNC, TAB_INTP, and TAB_NOINTP port settings is `'double'` while Compu method TAB_VERB port settings is `'string'`. The maximum string length supported for Compu method conversion is 1024 as specified in the ASAM XIL specification.

  The FORM Compu method conversion is not supported. Simulink throws a warning for such a conversion and IDENTICAL conversion is applied to the underlying measurement. Also, only scalar measurement signals are supported for TAB_VERB conversion as Simulink only supports scalar strings.

  Selecting this option shows the physical units (if any) in front of the measurement name on the block mask. This physical unit is acquired from the A2L description of the measurement and Compu method. If the physical unit is not specified, only the measurement name is displayed.

- `Raw values (no Compu method conversion)` sets the port data type according to the type definition in the A2L file and supports up to three-dimensional XCP measurements in Simulink.

- `Raw values as double (no Compu method conversion)` sets the port data type as `double`, converting all internal measurement values. This selection supports up to three-dimensional XCP measurements in Simulink.

These ASAP2 data types are supported by corresponding Simulink port data types:

- SBYTE
- UWORD
- SWORD
- ULONG

- SLONG
- A_UINT64
- A_INT64
- FLOAT32_IEEE
- FLOAT64_IEEE

The dimension support in the block accommodates the different treatment of matrices by MATLAB and the ECU. The MATLAB default operation treats matrices as row-major matrices. An XCP measurement can have a LAYOUT as COLUMN_DIR or ROW_DIR . If a matrix measurement is COLUMN_DIR, the blocks rearrange the measurement in memory and ensure that the matrix (row X, col Y) in MATLAB refers to the same entry as (row X, col Y) on the ECU. The rearrangement causes matrix entries that are contiguous on the ECU to be noncontiguous in MATLAB and Simulink.

Enable Timestamp — Enable sending Simulink timestamp in STIM DTO packets
off (default) | on

When the Timestamp is enabled, the block inputs a timestamp from Simulink and sends the timestamp in the STIM DTO packets. The **Enable Timestamp** check box appears in the block parameters dialog box when the parameter is supported in the A2L file.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

The XCP communication blocks support code generation with limited deployment portability that runs only on the host computer or Simulink Real-Time targets.

Code generation requires a C++ compiler that is compatible with the code generation target. For more information, see Supported and Compatible Compilers.
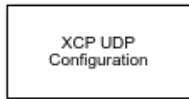
See Also

Blocks
XCP CAN Configuration | XCP CAN Data Acquisition | XCP CAN Transport Layer

Introduced in R2013a

XCP UDP Configuration

Configure XCP UDP server connection

Library: Vehicle Network Toolbox / XCP Communication / UDP
Simulink Real-Time / XCP / UDP

Description

The XCP UDP Configuration block uses the parameters specified in the A2L file and the ASAP2 database to establish an XCP server connection.

Before you acquire or stimulate data, specify the A2L file to use in your XCP UDP Configuration. Use one XCP UDP Configuration to configure one server connection for data acquisition or stimulation. If you add XCP UDP Data Acquisition and XCP UDP Data Stimulation blocks, your model checks to see if there is a corresponding XCP UDP Configuration block. If there is no corresponding XCP UDP Configuration block, the model prompts you to add one.

The XCP UDP communication blocks support Simulink accelerator mode and rapid accelerator mode. You can speed up the execution of Simulink models by using these modes. For more information about these simulation modes, see "Design Your Model for Effective Acceleration".

Parameters

Config name — Specify XCP UDP session name
'UDP_Config1' (default)

Specify a unique name for your XCP session.

A2L File — Select an A2L file
file name

Click **Browse** to select an A2L file for your XCP session.

Enable seed/key security — Select that key required to establish connection
'off'

Select this option if your server requires a secure key to establish connection. Select a file that contains the seed/key definition to enable security.

File (*.DLL) — Select file for seed and key security
file name

If you select **Enable seed/key security**, this field is enabled. Click **Browse** to select the file that contains the seed and key security algorithm that unlocks an XCP server module. This parameter is available in Windows Desktop Simulation for Vehicle Network Toolbox.

Output connection status — Display connection status
'off'

Select this option to display the status of the connection to the server module. Selecting this option adds a new output port.

Disable CTR error detection — Disable CTR error detection scheme
'on' (default) | 'off'

To detect missing packets, the block can check the counter value in each XCP packet header. When 'on', counter error detection for packet headers is disabled. When 'off', the counter **Error detection scheme** is enabled.

Error detection scheme — Select CTR error detection scheme
One counter for all CTOs and DTOs (default) | Separate counters for (RES,ERR,EV,SERV) and (DAQ) | Separate counters for (RES,ERR), (EV,SERV) and (DAQ)

To detect missing packets, the block can check the counter value in each XCP packet header and apply an error-detection scheme.

Sample time — Sample time of block
-1 (default) | numeric

Enter the base sample time or a multiple of the base sample time. -1 means that sample time is inherited. For information about simulation sample timing, see "What Is Sample Time?".

Local IP Address — Maser IP address
x.x.x.x

Enter the IP address to which you want to connect.

Local Port — Client IP port
1–65535

The combination of **Local IP address** and **Local port** must be unique.

Ports in the range 5500 through 5560 are reserved for Simulink Real-Time communications.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

The XCP communication blocks support code generation with limited deployment portability that runs only on the host computer or Simulink Real-Time targets.

Code generation requires a C++ compiler that is compatible with the code generation target. For more information, see Supported and Compatible Compilers.

See Also

Blocks
XCP UDP Data Acquisition | XCP UDP Data Stimulation | XCP UDP Bypass

Introduced in R2019a

XCP UDP Data Acquisition

Acquire selected measurements from configured server connection
Library: Vehicle Network Toolbox / XCP Communication / UDP
                      Simulink Real-Time / XCP / UDP

```
XCP UDP
Data Acquisition
```

Description

The XCP UDP Data Acquisition block acquires data from the configured server connection based on the measurements that you select. The block uses the XCP UDP transport layer to obtain raw data for the selected measurements at the specified simulation time step. Configure your XCP connection and use the XCP UDP Data Acquisition block to select your event and measurements for the configured server connection. The block displays the selected measurements as output ports.

The XCP communication blocks support the use of Simulink accelerator mode and rapid accelerator mode. You can speed up the execution of Simulink models by using these modes. For more information on these simulation modes, see "Design Your Model for Effective Acceleration".

Parameters

`Config name` — Specify XCP UDP session name
select from list

Select the name of XCP configuration that you want to use. This list displays all available names specified in the XCP UDP Configuration blocks in the model. Selecting a configuration displays events and measurements available in the A2L file of this configuration. You can acquire measurements for only one event by using an XCP UDP Data Acquisition block. Use one block for each event whose measurements you want to acquire.

`Event name` — Select an event
select from list

Select an event from the available list of events. The XCP UDP Configuration block uses the specified A2L file to populate the events list.

`All Measurements` — List all measurements available for event
measurements list

This list displays all measurements available for the selected event. Select the measurement that you want to use and click the add button, ![add button] to add it to the selected measurements. Hold the `Ctrl` key on your keyboard to select multiple measurements.

In the **Block Parameters** dialog box, type the name of the measurement you want to use in the **Search** box. The **All Measurements** lists displays a list of all matching names. Click the x to clear your search.

`Selected Measurements` — List selected measurements
measurement names

This list displays selected measurements. To remove a measurement from this list, select the

measurement and click the remove button, .

In the **Block Parameters** dialog box, use the toggle buttons to reorder the selected measurements.

`Block Output Settings` — Set the port output as Compu method conversion values or raw values
Raw values as double (no Compu method conversion) (default) | Raw values (no Compu method conversion) | Physical values (apply Compu method conversion)

This parameter enables support for XCP data types and dimensions as defined in the ASAP2 standard. The Block Output Settings parameter selects whether the port outputs Compu method conversion values or raw values. The options provide:

- `Physical values (apply Compu method conversion)` enables the raw-to-physical conversion of ECU measurement values. For this option, the block port settings are set either to `'double'` or `'string'`, based on the underlying Compu method conversion. For example, Compu method `IDENTICAL`, `LINEAR`, `RAT_FUNC`, `TAB_INTP`, and `TAB_NOINTP` port settings is `'double'` while Compu method `TAB_VERB` port settings is `'string'`. The maximum string length supported for Compu method conversion is 1024 as specified in the ASAM XIL specification.

  The FORM Compu method conversion is not supported. Simulink throws a warning for such a conversion and IDENTICAL conversion is applied to the underlying measurement. Also, only scalar measurement signals are supported for TAB_VERB conversion as Simulink only supports scalar strings.

  Selecting this option shows the physical units (if any) in front of the measurement name on the block mask. This physical unit is acquired from the A2L description of the measurement and Compu method. If the physical unit is not specified, only the measurement name is displayed.

- `Raw values (no Compu method conversion)` sets the port data type according to the type definition in the A2L file and supports up to three-dimensional XCP measurements in Simulink.

- `Raw values as double (no Compu method conversion)` sets the port data type as `double`, converting all internal measurement values. This selection supports up to three-dimensional XCP measurements in Simulink.

These ASAP2 data types are supported by corresponding Simulink port data types:

- `SBYTE`
- `UWORD`
- `SWORD`
- `ULONG`
- `SLONG`
- `A_UINT64`
- `A_INT64`

- `FLOAT32_IEEE`
- `FLOAT64_IEEE`

The dimension support in the block accommodates the different treatment of matrices by MATLAB and the ECU. The MATLAB default operation treats matrices as row-major matrices. An XCP measurement can have a `LAYOUT` as `COLUMN_DIR` or `ROW_DIR` . If a matrix measurement is `COLUMN_DIR`, the blocks rearrange the measurement in memory and ensure that the matrix (row X, col Y) in MATLAB refers to the same entry as (row X, col Y) on the ECU. The rearrangement causes matrix entries that are contiguous on the ECU to be noncontiguous in MATLAB and Simulink.

DAQ List Priority — Specify a priority value for server device driver
priority value

Specify a priority value as an integer from 0 to 255 for the server device driver to prioritize transmission of data packets. The server can accumulate XCP packets for lower priority DAQ lists before transmission to the client. A value of 255 has the highest priority. The `SET_DAQ_LIST_MODE` command communicates the **DAQ List Priority** value from client to server. This communication method differs from the specification of the Event Channel Priority property, which comes from the A2L file.

Sample time — Specify sampling time of block
`0.01` (default)

Specify the sampling time of the block during simulation, which is the simulation time. This value defines the frequency at which the XCP UDP Data Acquisition block runs during simulation. If the block is inside a triggered subsystem or is to inherit sample time, you can specify −1 as your sample time. You can also specify a MATLAB variable for sample time. The default value is 0.01 simulation seconds. For information about simulation sample timing, see "What Is Sample Time?".

Enable Timestamp — Enable reading timestamp from incoming DTO packets
`off` (default) | `on`

When the Timestamp is enabled, the block reads the timestamp from incoming DTO packets and outputs the timestamp to Simulink. The **Enable Timestamp** check box appears in the block parameters dialog box when the parameter is supported in the A2L file.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

The XCP communication blocks support code generation with limited deployment portability that runs only on the host computer or Simulink Real-Time targets.

Code generation requires a C++ compiler that is compatible with the code generation target. For more information, see Supported and Compatible Compilers.

See Also

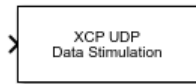### Blocks
XCP UDP Configuration | XCP UDP Data Stimulation | XCP UDP Bypass

Introduced in R2019a

XCP UDP Data Stimulation

Perform data stimulation on selected measurements
Library: Vehicle Network Toolbox / XCP Communication / UDP
              Simulink Real-Time / XCP / UDP

```
XCP UDP
Data Stimulation
```

Description

The XCP UDP Data Stimulation block sends data to the selected server connection for the event measurements that you select. The block uses the XCP UDP transport layer to output raw data for the selected measurements at the specified stimulation time step. Configure your XCP session and use the XCP UDP Data Stimulation block to select your event and measurements on the configured server connection. The block displays the selected measurements as input ports.

The XCP communication blocks support Simulink accelerator mode and rapid accelerator mode. You can speed up the execution of Simulink models by using these modes. For more information about these simulation modes, see "Design Your Model for Effective Acceleration".

Parameters

`Config name` — Specify XCP UDP session name
select from list

Select the name of XCP configuration that you want to use. This list displays all available names specified in the available XCP UDP Configuration blocks in the model. Selecting a configuration displays events and measurements available in the A2L file of this configuration. You can stimulate measurements for only one event by using an XCP UDP Data Stimulation block. Use one block for each event whose measurements you want to stimulate.

`Event name` — Select an event
select from list

Select an event from the event list. The XCP UDP Configuration block uses the specified A2L file to populate the events list. The block is configured with the corresponding event number from the A2L.

The event time cycle does not control transmission of stimulation packets. The block stimulates each time it executes. For use in Simulink simulation, consider enabling simulation pacing to avoid free-running stimulation.

`All Measurements` — List all measurements available for event
measurements list

This list displays all measurements available for the selected event. Select the measurement that you want to use and click the add button, ![add button] to move it to the selected measurements. Hold the `Ctrl` key on your keyboard to select multiple measurements.

In the block parameters dialog box, type the name of the measurement you want to use. The **All Measurements** lists displays a list of all matching names. Click the x to clear your search.

`Selected Measurements` — List selected measurements
measurement names

This list displays your selected measurements. To remove a measurement from this list, select the measurement and click the remove button, .

In the **Block Parameters** dialog box, use the toggle buttons to reorder the selected measurements.

`Block Input Settings` — Set the port input as Compu method conversion values or raw values
Raw values as double (no Compu method conversion) (default) | Raw values (no Compu method conversion) | Physical values (apply Compu method conversion)

This parameter enables support for XCP data types and dimensions as defined in the ASAP2 standard. The Block Input Settings parameter selects whether the port outputs Compu method conversion values or raw values. The options provide:

- `Physical values (apply Compu method conversion)` enables the physical-to-raw conversion of ECU measurement values. For this option, the block port settings are set either to `'double'` or `'string'`, based on the underlying Compu method conversion. For example, Compu method `IDENTICAL`, `LINEAR`, `RAT_FUNC`, `TAB_INTP`, and `TAB_NOINTP` port settings is `'double'` while Compu method `TAB_VERB` port settings is `'string'`. The maximum string length supported for Compu method conversion is 1024 as specified in the ASAM XIL specification.

  The FORM Compu method conversion is not supported. Simulink throws a warning for such a conversion and IDENTICAL conversion is applied to the underlying measurement. Also, only scalar measurement signals are supported for TAB_VERB conversion as Simulink only supports scalar strings.

  Selecting this option shows the physical units (if any) in front of the measurement name on the block mask. This physical unit is acquired from the A2L description of the measurement and Compu method. If the physical unit is not specified, only the measurement name is displayed.
- `Raw values (no Compu method conversion)` sets the port data type according to the type definition in the A2L file and supports up to three-dimensional XCP measurements in Simulink.
- `Raw values as double (no Compu method conversion)` sets the port data type as `double`, converting all internal measurement values. This selection supports up to three-dimensional XCP measurements in Simulink.

These ASAP2 data types are supported by corresponding Simulink port data types:

- `SBYTE`
- `UWORD`
- `SWORD`
- `ULONG`

- SLONG
- A_UINT64
- A_INT64
- FLOAT32_IEEE
- FLOAT64_IEEE

The dimension support in the block accommodates the different treatment of matrices by MATLAB and the ECU. The MATLAB default operation treats matrices as row-major matrices. An XCP measurement can have a LAYOUT as COLUMN_DIR or ROW_DIR . If a matrix measurement is COLUMN_DIR, the blocks rearrange the measurement in memory and ensure that the matrix (row X, col Y) in MATLAB refers to the same entry as (row X, col Y) on the ECU. The rearrangement causes matrix entries that are contiguous on the ECU to be noncontiguous in MATLAB and Simulink.

Enable Timestamp — Enable sending Simulink timestamp in STIM DTO packets
off (default) | on

When the Timestamp is enabled, the block inputs a timestamp from Simulink and sends the timestamp in the STIM DTO packets. The **Enable Timestamp** check box appears in the block parameters dialog box when the parameter is supported in the A2L file.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

The XCP communication blocks support code generation with limited deployment portability that runs only on the host computer or Simulink Real-Time targets.

Code generation requires a C++ compiler that is compatible with the code generation target. For more information, see Supported and Compatible Compilers.
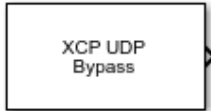
See Also

Blocks
XCP UDP Configuration | XCP UDP Data Acquisition | XCP UDP Bypass

Introduced in R2019a

XCP UDP Bypass

Connect the function-call outport to a function-call subsystem
Library: Vehicle Network Toolbox / XCP Communication / UDP
Simulink Real-Time / XCP / UDP

Description

The XCP UDP Bypass block connects the function-call outport to a function-call subsystem containing one data acquisition list. The block issues a function-call when the downstream data acquisition list has new data available.

Consider the downstream function-call subsystem as a bypass task:

In Simulink Real-Time, the bypass task is executed asynchronously with the assigned task priority.

In Simulink, the block checks for data acquisition data periodically at the assigned sample rate and executes the bypass task accordingly.

Ports

Output

Function-call — Function call for bypass
function call

Connects the function-call outport to a function-call subsystem containing one data acquistion list.

Parameters

Task Priority — Task priority in QNX Neutrino scheduler
191 (default) | integer

Select the task priority for the QNX Neutrino scheduler.

Sample Time — Sample time
-1 (default) | double

Select the sample time. For more information, see "Sample Times in Subsystems".

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

The XCP communication blocks support code generation with limited deployment portability that runs only on the host computer or Simulink Real-Time targets.

Code generation requires a C++ compiler that is compatible with the code generation target. For more information, see Supported and Compatible Compilers.

See Also

XCP UDP Data Acquisition | XCP UDP Data Stimulation | XCP UDP Configuration

Introduced in R2020b

Speedgoat Blocks Library

Speedgoat Support

Speedgoat Target Computers and Speedgoat Support

Speedgoat target computers are real-time computers fitted with a set of I/O hardware, Simulink programmable and configurable FPGAs, and communication protocol support. Speedgoat target computers are optimized for use with Simulink Real-Time and fully support the HDL Coder™ workflow.

Speedgoat real-time target machines include:

- Performance — Highest performance, cost-effective real-time system for office or lab. Supports up to 50 I/O modules.
- Mobile — Compact, rugged, fanless, and expandable real-time system. For mobile and in-vehicle use and use in confined areas. Provides an extended operating temperature range. Supports up to 14 I/O modules.
- Baseline — Small, rugged, and fanless real-time system. For mobile, in-vehicle, and classroom use and use in confined areas. Provides an extended operating temperature range. Supports up to 7 I/O modules
- Audio — Real-time system optimized for audio applications, such as hearing aids and car acoustics.
- Unit — Small, rugged real-time system for mobile, field, in-vehicle and classroom use and use in confined areas. Provides an extended operating temperature range. Supports 1 I/O module

When you install the Speedgoat I/O Blockset, the installer sets up help for the blocks in the MATLAB Help browser. To view the Speedgoat I/O Blockset documentation, open the Help browser and navigate to the home page. At the bottom right of the home page, under **Supplemental Software**, click Speedgoat I/O Blockset.

To install your Speedgoat I/O Blockset, go to www.speedgoat.com/extranet, the Speedgoat Customer Portal. Follow the instructions to download and install the Speedgoat I/O Blockset.

You can find Speedgoat real-time target machine configuration documentation online at www.speedgoat.com/help.

You can find Speedgoat real-time target machine product information online at www.speedgoat.com/products-services.

Speedgoat I/O Hardware

Speedgoat provides a wide range of I/O hardware with ready-to-use configurations for rapid control prototyping (RCP) and hardware-in-the-loop (HIL) simulations. Speedgoat I/O connectivity includes support for:

- Analog I/O: single-ended or differential inputs or outputs, with or without isolation, 16–24 bit, voltage and current controlled
- Digital I/O: LVCMOS, TTL, RS-422, RS-485, LVDS
- FPGA code modules for:

  - Interrupts
  - PWM generation and capture, pulse patterns
  - Quadrature decoding and encoding (measurement and simulation)

- SSI master, slave, and sniffer (measurement and simulation)
- SSI2 master, slave, and sniffer (measurement and simulation)
- EnDat 2.2 decoder, encoder, and sniffer (measurement and simulation)
- BiSS decoder, encoder, and sniffer (measurement and simulation)
- SPI master, slave, and sniffer
- I²C master and slave
- Cam and crank decoder and simulator (measurement and simulation)
- UART (RS-485/RS-422)
- Aurora 64B/66B master and slave
- And more
- LVDT/RVDT and synchro/resolver (measurement and simulation)
- Serial:

  - RS-232, RS-422, RS-485
  - SDLC, HDLC
- Shared memory
- Thermocouple, RTD, and strain gauge (measurement and simulation)
- Vibration measurements (IEPE/ICP transducers)
- Programmable resistors and potentiometers
- SPDT, SPST, and DPST reed relays
- Fault insertion
- Battery management systems

Speedgoat Communication Protocols

Speedgoat provides communication protocol support for I/O hardware with ready-to-use configurations. Speedgoat communication protocols include:

- CAN, CAN FD, LIN, SAE J1939, and FlexRay™
- XCP over Ethernet, XCP over CAN
- MIL-STD-1553, ARINC-429, ARINC-629, AFDX (ARINC 664 Pt7)
- EtherCAT master and EtherCAT slave
- Real-time UDP, Real-time raw Ethernet, TCP/IP
- EtherNet/IP™ Scanner (master) and EtherNet/IP Adapter (slave)
- PROFINET master and PROFINET slave
- PROFIBUS master and PROFIBUS slave
- Modbus TCP Client (master), Modbus TCP Server (slave), Modbus RTU
- POWERLINK Controlled Node (slave)
- Timing and synchronization: PTP (Precision Time Protocol, IEEE 1588), GPS, IRIG
- UART (RS-232, RS-422, RS-485)
- I2C, SPI, SSI, SSI2, EnDAT 2.2, BiSS

- Camera Link®
- Aurora 8B/10B and 64B/66B multigigabit links for FPGA

See Also

More About
- "Set Up and Configure Simulink Real-Time"

External Websites
- www.speedgoat.com/help
- www.speedgoat.com/products-services
- www.speedgoat.com