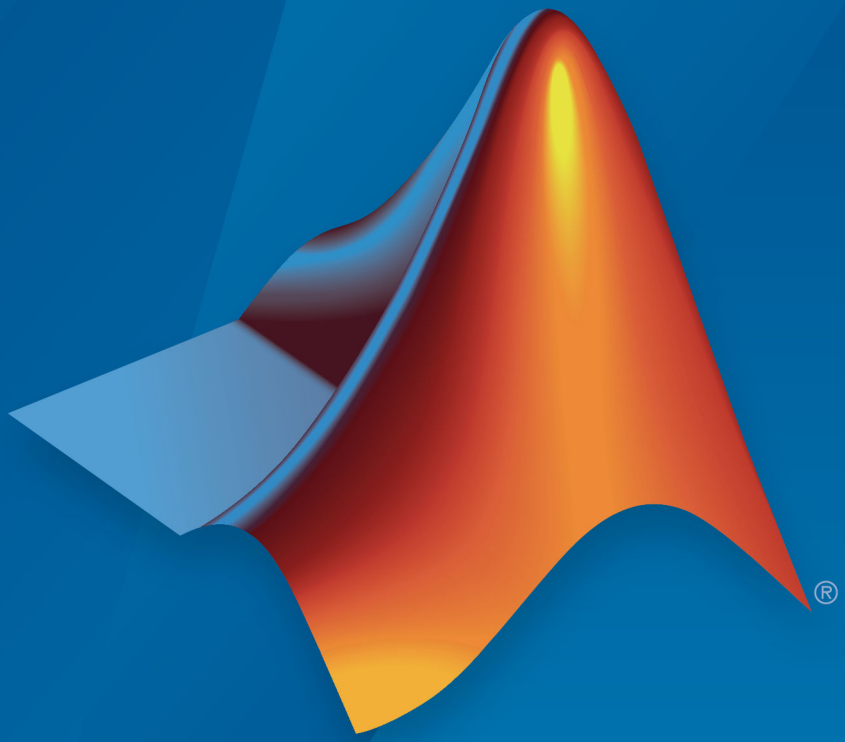
Simulink® Real-Time™

I/O Reference



MATLAB® & SIMULINK®

R2018a

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.
3 Apple Hill Drive
Natick, MA 01760-2098

Simulink® Real-Time ™ I/O Reference

Revision History

November 2000	Online only	Revised for Version 1.1 (Release 12)
June 2001	Online only	Revised for Version 1.2 (Release 12.1)
September 2001	Online only	Revised for Version 1.3 (Release 12.1+)
July 2002	Online only	Revised for Version 2 (Release 13)
September 2002	Online only	Revised for Version 2.0.1 (Release 13)
September 2003	Online only	Revised for Version 2.0.1 (Release 13SPI)
June 2004	Online only	Revised for Version 2.5 (Release 14)
August 2004	Online only	Revised for Version 2.6 (Release 14+)
October 2004	Online only	Revised for Version 2.6.1 (Release 14SP1)
November 2004	Online only	Revised for Version 2.7 (Release 14SP1+)
March 2005	Online only	Revised for Version 2.7.2 (Release 14SP2)
September 2005	Online only	Revised for Version 2.8 (Release 14SP3)
March 2006	Online only	Revised for Version 2.9 (Release 2006a)
May 2006	Online only	Revised for Version 3.0 (Release 2006a+)
September 2006	Online only	Revised for Version 3.1 (Release 2006b)
March 2007	Online only	Revised for Version 3.2 (Release 2007a)
September 2007	Online only	Revised for Version 3.3 (Release 2007b)
March 2008	Online only	Revised for Version 3.4 (Release 2008a)
October 2008	Online only	Revised for Version 4.0 (Release 2008b)
March 2009	Online only	Revised for Version 4.1 (Release 2009a)
September 2009	Online only	Revised for Version 4.2 (Release 2009b)
March 2010	Online only	Revised for Version 4.3 (Release 2010a)
September 2010	Online only	Revised for Version 4.4 (Release 2010b)
April 2011	Online only	Revised for Version 5.0 (Release 2011a)
September 2011	Online only	Revised for Version 5.1 (Release 2011b)
March 2012	Online only	Revised for Version 5.2 (Release 2012a)
September 2012	Online only	Revised for Version 5.3 (Release 2012b)
March 2013	Online only	Revised for Version 5.4 (Release 2013a)
September 2013	Online only	Revised for Version 5.5 (Release 2013b)
March 2014	Online only	Revised for Version 6.0 (Release 2014a)
October 2014	Online only	Revised for Version 6.1 (Release 2014b)
March 2015	Online only	Revised for Version 6.2 (Release 2015a)
September 2015	Online only	Revised for Version 6.3 (Release 2015b)
March 2016	Online only	Revised for Version 6.4 (Release 2016a)
September 2016	Online only	Revised for Version 6.5 (Release 2016b)
March 2017	Online only	Revised for Version 6.6 (Release 2017a)
September 2017	Online only	Revised for Version 6.7 (Release 2017b)
March 2018	Online only	Revised for Version 6.8 (Release 2018a)

Contents

3 Serial Communications Support: Blocks

A Serial Communications Support: Internal Blocks

CAN, Encoders, Ethernet, EtherCAT

4 CAN Utility Blocks

5 Model-Based Ethernet Communications Support

12 TCP Blocks

13 Real-Time UDP Communication Support

18 Precision Time Protocol Blocks

19 SAE J1939

20 SAE J1939 Blocks

21 Shared Memory Support

Video, XCP

22 Video Image Processing

UEI, Asynchronous Events

27 Asynchronous Events

28 Asynchronous Event: Blocks

Utility Drivers, Target Management, Displays and Logging

29 Utility Blocks

30 Target Management, Display, and Logging Blocks

Introduction, RS-232

Simulink Real-Time I/O Library

I/O Driver Blocks

In this section...

The Simulink Real-Time environment is a solution for prototyping and testing real-time systems using a desktop computer. To support this solution, the software allows you to 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 the physical system.

Speedgoat I/O Modules

Speedgoat real-time target machines are available with various I/O modules. See on page 26-3.

Third-Party Driver Blocks

In addition to the blocks contained in the Simulink Real-Time library, you can also use third-party driver blocks in your Simulink Real-Time model. The description of these blocks is beyond the scope of the Simulink Real-Time documentation. See the provider of the third-party driver blocks for information on those boards and driver blocks.

I/O Driver Block Library

A driver block does not represent an entire board, but an I/O section supported by a board. Therefore, 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.

Note, if your model contains I/O blocks, take I/O latency values into account for the model sample time. To find latency values for a board supported by the Simulink Real-Time block library, consult the vendor data sheet. To find a link to the vendor website, see:

`www.mathworks.com/products/simulink-real-time/supported/hardware-drivers.html`.

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

The Simulink Real-Time system supports PCI and ISA (PC/104) buses. If the bus type is not indicated in the driver block number, determine the bus type of the block by examining the block parameter dialog box. The last parameter is either a PCI slot, for PCI boards, or a base address, for ISA (PC/104) boards.

You can open the I/O device driver library with the MATLAB® command `slrtlib`. The library `slrtlib` contains sublibraries grouped by the type of I/O function they provide.

This library also contains the following blocks:

- Simulink Real-Time Driver Examples — When you double-click this block, the **Demos** tab in the MATLAB Help Navigator opens, displaying the Simulink Real-Time examples and example groups.
- Help for Simulink Real-Time — When you double-click this block, the Simulink Real-Time roadmap page is displayed. You can access the Simulink Real-Time documentation with this block.

Note The Simulink Real-Time documentation describes only the Simulink Real-Time blocks. It does not describe the actual board. Refer to the board manufacturer documentation for information about the boards.

When you double-click one of 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 (for example, A/D, D/A, Digital Inputs, and Digital Outputs).

When you double-click one of the blocks, a Block Parameters dialog box opens, allowing you to enter system-specific parameters. Parameters typically include

- Sample time
- Number of channels
- Voltage range
- PCI slot (PCI boards)
- Base address (ISA/104 boards)

Memory-Mapped Devices

Simulink Real-Time reserves a 112-kB memory space for memory-mapped devices in the address range:

```
C0000 - DBFFF
```

Drivers for some memory-mapped devices, such as the Softing CAN-AC2-104 board, support an address range higher than the range that Simulink Real-Time supports. Specify an address range supported by both the device driver and the Simulink Real-Time software.

ISA Bus I/O Devices

There are two types of ISA boards:

- Jumper addressable ISA cards
- PnP (Plug and Play) ISA cards

The Simulink Real-Time software only supports jumper addressable ISA cards (non-PnP ISA boards) where you have to set the base address manually.

PCI Bus I/O Devices

The Simulink Real-Time I/O library supports I/O boards with a PCI bus. During the boot process, the BIOS creates a conflict-free configuration of base addresses and interrupt

lines for the PCI devices in the target system. You do not need to define base address information in the dialog boxes of the drivers.

PCI device driver blocks have an additional entry in their dialog boxes. This entry is called `PCI Slot (-1 Autodetect)` and allows you to use several identical PCI boards within one target system. This entry uses a default value of `-1`, which allows the driver to search the entire PCI bus to find the board. If you specify a single number, `X`, greater than `0`, the driver uses the board in bus `0`, slot `X`. When more than one board of the same type is found, you must use a designated slot number and avoid the use of autodetection. For manually setting the slot number, you use a number greater than or equal to `0`. If the board cannot locate this slot in the target computer, your real-time application will generate an error message after downloading.

To set `PCI Slot (-1 Autodetect)` to a value equal to or greater than `0`, you must identify which board you want on the target computer. To identify the board, find the manufacturer identification number (Vendor ID) and board identification number (Device ID) of the boards supported by the I/O library. When the target is booted, the BIOS is executed and the target computer monitor shows parameters for the PCI boards installed on the target computer. For example:

Bus Number	Device Number	Function Number	Vendor ID	Device ID	Device Class	IRQ
0	4	1	8086	7111	IDE controller	14/15
0	4	2	8086	7112	Serial bus controller	10
0	11	0	1307	000B	Unknown PCI device	N/A
1	0	0	12D2	0018	Display controller	11

In this example, the third line indicates the location of the Measurement Computing™ PCI-DIO48 board. This location is known since the Measurement Computing vendor ID is `0x1307` and the device ID is `0xb`. In this case, you can see that the Measurement Computing board is plugged into PCI slot 11 (Device Number). Enter this value in the dialog box entry in your I/O device driver for each model that uses this I/O device.

Simulink Real-Time I/O Driver Structures

Properties for Simulink Real-Time I/O drivers are defined using the parameter dialog box associated with each Simulink block. However, for more advanced drivers, the available fields defined by text boxes, check boxes, and pull-down lists are inadequate to define the behavior of the driver. In such cases, you must provide a more textual description to indicate what the driver has to do at run time. *Textual* in this context refers to a programming-language-like syntax and style.

The Simulink Real-Time software currently uses a character vector description contained in message structures for the conventional RS-232 drivers.

What Is a Message Structure?

A message structure is a MATLAB array with each cell containing one complete message (command). A message consists of one or more statements.

First Message	Second Message	Third Message
`Message(1).field`	`Message(2).field`	`Message(3).field`
`Message(1).field`	`Message(2).field`	`Message(3).field`
`Message(1).field`	`Message(2).field`	`Message(3).field`

Syntax of a Message Statement

Each statement in a message has the following format:

```
Structure_name(index).field_name = <field character vector or value>
```

The driver defines the field names. Enter them with upper- and lowercase letters as defined. However, you can specify your own structure name and enter that name into the driver parameter dialog box.

Creating a Message Structure

You could enter the message structure directly in the edit field of the driver parameter dialog box. But because the message structure is a large array, direct entry becomes cumbersome easily.

A better way is to define the message structure as a variable in the MATLAB workspace and pass the variable name to the driver. For example, to initialize an external A/D module

and acquire a value during each sample interval, create a script file with the following statements:

```
Message(1).senddata='InitADConv, Channel %d'
Message(1).inputports=[1]
Message(1).recdata=''
Message(1).outputports=[]

Message(2).senddata='Wait and Read converted Value'
Message(2).inputports=[]
Message(2).recdata='%f'
Message(2).outputports=[1]
```

This approach is different from other Simulink Real-Time driver blocks:

- The script containing the definition of the message structure has to be executed before the model is opened.

  After creating your Simulink model and message script, set the preload function of the Simulink model to load the script file the next time you open the model. In the Command Window, type

  ```
  set_param(gcs, 'PreLoadFcn', 'script_name')
  ```

- When you move or copy the model file to a new folder, you must also move or copy the script defining the message structure.

During each sample interval, the driver block locates the message structure, interprets the messages, and executes the command defined by each message.

For detailed information on the fields in an RS-232 message structure, see “Simulink Real-Time RS-232 Reference” on page 2-9,

Simulink Real-Time Support and SimState

You can save complete model simulation states while simulating, on a development computer, a Simulink model that contains some Simulink Real-Time blocks. The software does not support this behavior when executing such a model on the target computer.

For this operation, set the **Save complete SimState in final state** check box in the **Data Import/Export** pane of the Configuration Parameters dialog box. If your model contains the following blocks, you cannot save complete model simulation states while simulating on the development computer.

- ASCII Encode
- ASCII Decode
- Async Buffer Read
- Async Buffer Write
- Baseboard Serial
- Baseboard Serial F
- Bit Packing (Utilities library)
- Bit Unpacking (Utilities library)
- Byte Packing (Utilities library)
- Byte Unpacking (Utilities library)
- Create Ethernet Packet (Ethernet library)
- FIFO bin read
- FIFO ASCII read
- FIFO write
- UDP Receive
- UDP Send

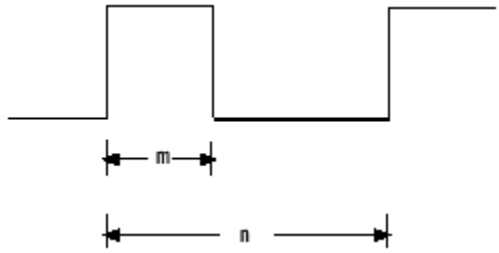
To prevent these messages, clear the **Save complete SimState in final state** check box in the **Data Import/Export** node of the Configuration Parameters dialog box.

PWM and FM Driver Block Notes

In PWM and FM driver blocks, your control over the output frequency and duty cycle is not precise. Although the base frequency value is exact, the way the base frequency is specified affects the output frequency and duty cycle.

At the beginning of each sample time, the block reads the current input signal values. It then computes two unsigned 16-bit integers, `n` and `m`, from the signal values and the block parameters. During the sample time, the block holds the output signal:

1. High for `m` cycles of the base frequency
2. Low for the next `n-m` cycles
3. High for the next `m` cycles
4. . . .



For a base frequency `b`, this algorithm results in a rectangular output signal of frequency `b/n` and duty cycle `m/n`. Because `m` and `n` must be integers, it is not possible to provide a continuous range of output frequencies and duty cycles with perfect exactness.

For example, assume that you want to configure an FM block with a duty cycle (`m/n`) of `1/2`. The input signal `f` to this block is a relative frequency that specifies an output frequency of `b × f`. However, `m` and `n` must be integers. Therefore, you cannot always find values of `m` and `n` (duty cycle `m/n = 1/2`) such that:

```
f = b/n
```

exactly and

```
n = 2 * m
```

exactly. You can find an exact match only when the input signal `f` equals `1/4`, `1/6`, `1/8`, and so forth. The output frequencies for the intervening input signal `f` values are approximate. The errors are smaller as `f` approaches `0` and larger as `f` approaches `1`.

To achieve the smallest margin of error, specify the largest possible base frequency. The fact that `n` and `m` must be 16-bit integers imposes a lower limit of:

$b / (2^{16} - 1)$

on the frequencies that can be generated using a given base frequency.

Driver Block Documentation

The typical Simulink Real-Time block documentation briefly describes the supported board, then describes the parameters for each of the blocks that support the board. Included in the documentation for each board is a board characteristics table. Board characteristics tables can include the following information:

Characteristic	Specifies...
Board name	Name of the board supported by the blocks. For example, Speedgoat IO333.
Manufacturer	Manufacturer of the board. For example, Speedgoat.
Bus type	Bus that is used by the board. For example, PCI or PC/104.
Access method	Whether the board is memory mapped or I/O mapped.
Multiple block instance support	Whether you can use multiple blocks for the same function on the same board. For example, different blocks for different channels of an A/D device.
Multiple board support	Whether you can use multiple boards of the same type in one real-time application.

Add I/O Blocks to Simulink Model

You can transform a Simulink model to a Simulink Real-Time model that accesses I/O drivers by using the Simulink Real-Time block library or by using the Simulink Real-Time: Speedgoat I/O Driver Library. 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 `ex_slrt_osc` to show how to replace Simulink blocks with Simulink Real-Time I/O blocks (see `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_osc')))`).

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

   ```
   slrtlib
   ```

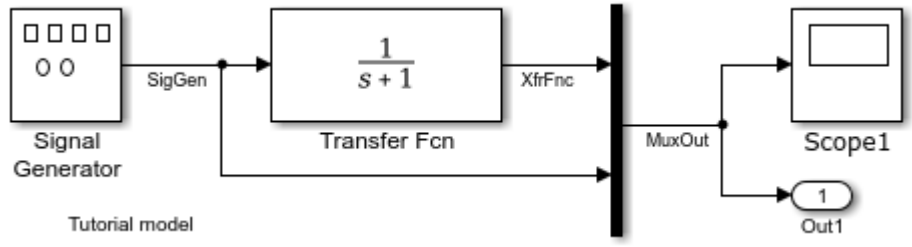
2. To browse the Simulink Real-Time: Speedgoat I/O Driver Library, open the **Library: speedgoatlib** window. In the Command Window, type

   ```
   speedgoatlib
   ```

3. In the Simulink Editor, type:

   ```
   ex_slrt_osc
   ```

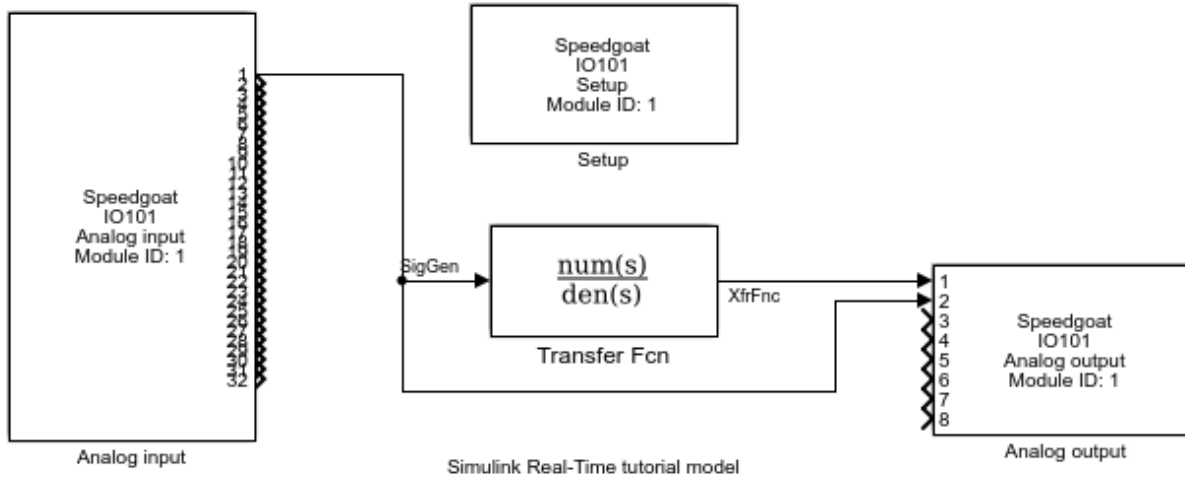
   The Simulink block diagram opens for the model `ex_slrt_osc`.



4. Open the Simulink Library Browser. Select **Simulink Real-Time: Speedgoat I/O Driver Library > 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.
6. Save the model with a new name, such as `ex_slrt_iob_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_iob_osc')))`):



You cannot run this model unless the required I/O board is installed in your target computer. However, you can substitute driver blocks for another I/O board that is installed in the target computer.

Your next task is to define the I/O block parameters. See “Defining I/O Block Parameters” on page 1-13.

Defining I/O Block Parameters

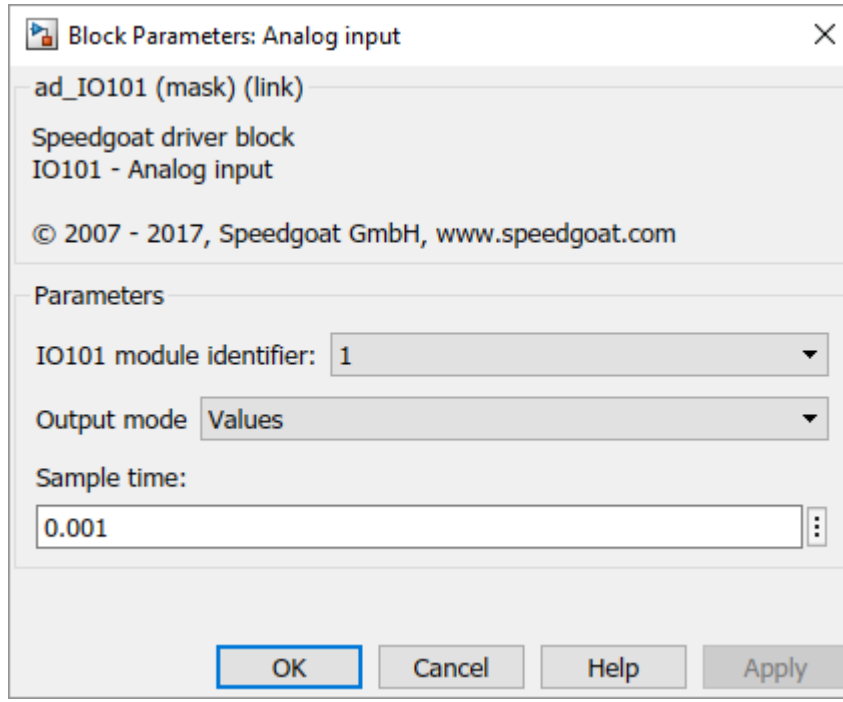
The I/O block parameters define values for your physical I/O boards. For example, I/O block parameters include channel numbers for multichannel boards, input and output voltage ranges, and sample time.

This procedure uses the Simulink model `ex_slrt_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_osc')))`). It assumes that you have added an analog input block, an analog output block, and the corresponding setup block to your model. To add an I/O block, see “Add I/O Blocks to Simulink Model” on page 1-11.

1. In the Simulink Editor, double-click the input block labeled `Speedgoat IO101 Analog Input`.

   The dialog box for the A/D converter opens.

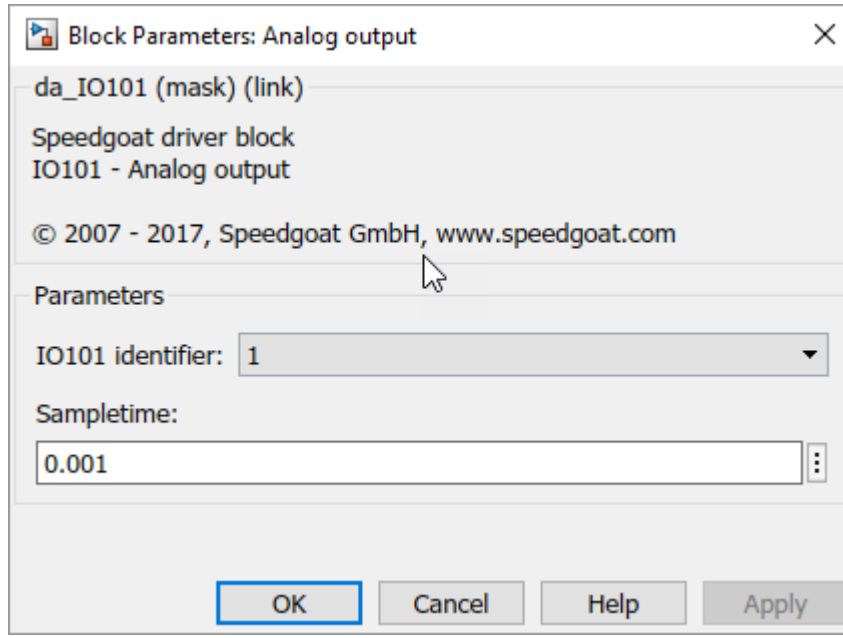
2. Fill in the dialog box. For example, enter the sample time you entered for the fixed step size in the **Solver** pane of the **Simulation > Model Configuration Parameters** dialog box.



3 In the Simulink Editor, double-click the output block labeled `Speedgoat IO101 Analog Output`.

The dialog box for the D/A converter opens.

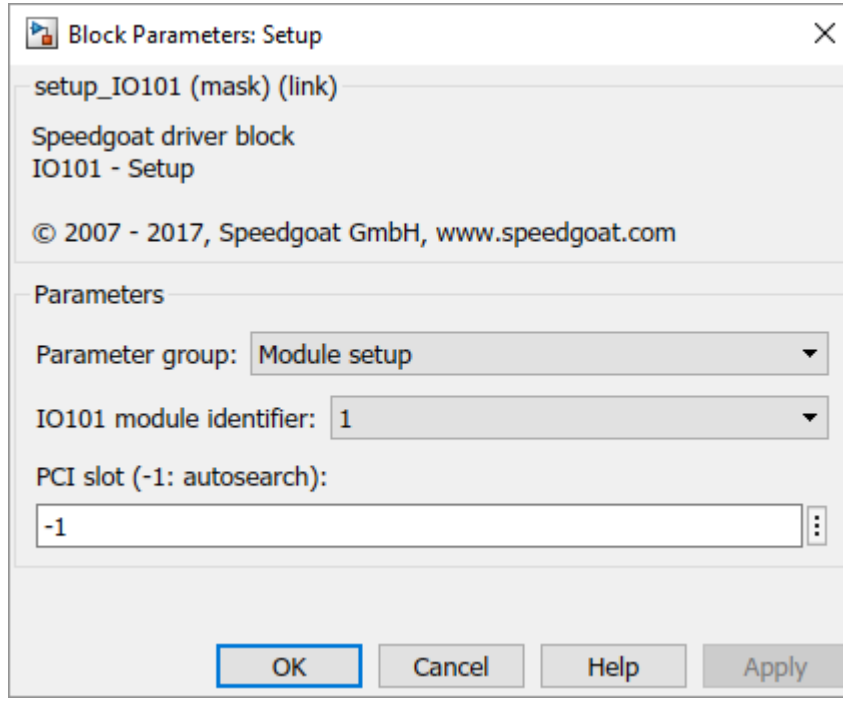
4 Fill in the dialog box. For example, enter the same sample time you entered for the fixed step size in the **Solver** pane of the **Simulation > Model Configuration Parameters** dialog box.



5 In the Simulink Editor, double-click the output block labeled `Speedgoat IO101 Setup`.

The dialog box for the converter setup opens.

6 Fill in the dialog box. For example, enter the PCI slot for this PCI-bus board.



If you change the target object property `SampleTime`, the sample times you entered in both of the I/O blocks are set to the new value. The step size you entered in the Configuration Parameters dialog box remains unchanged.

2

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 ports and the accompanying Simulink Real-Time drivers.

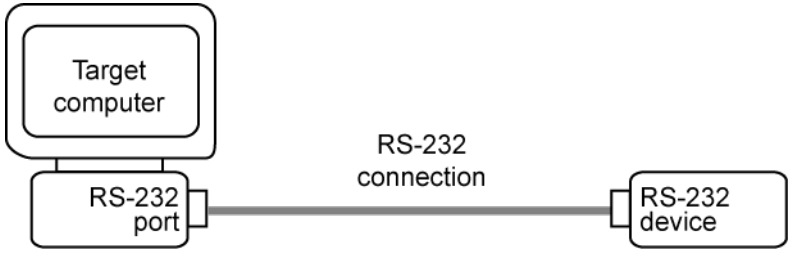
The Simulink Real-Time block library supplies composite drivers to support RS-232 communication (see “RS-232 Composite Drivers” on page 2-4). The composite drivers support RS-232 communication in asynchronous binary mode. They provide a simple ASCII encode/decode for the send and receive RS-232 blocks.

These drivers are described as composite because the library represents each functional piece of the driver as a Simulink block. They are constructed with blocks from the RS-232 internal library. The internal blocks are composite blocks for the Mainboard Baseboard series.

Do not use the RS-232 internal blocks directly. They are controlled from the mask parameters dialog box for the subsystem in which they are used.

Serial Connections for RS-232

The Simulink Real-Time software supports serial communication with 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 Decode | ASCII Decode V2 | ASCII Encode | FIFO Read | FIFO Read Binary | FIFO Read HDRS | FIFO Write | Modem Control | Modem Status | RS-232 Send/Receive | RS-232 Send/Receive FIFO | RS232 State

RS-232 Composite Drivers

This topic describes the components that make up the RS-232 composite drivers, and how you can create a model using these drivers. These drivers perform RS-232 asynchronous communications.

The Simulink Real-Time software provides composite drivers that support the target computer (main board) serial ports.

These drivers distribute the functionality of the device across several subsystems and blocks. For most RS-232 requirements, you can use these RS-232 drivers as they are implemented. However, if you must customize the Simulink Real-Time RS-232 drivers, the composite nature of the drivers enables you to do so.

In this section...
“Adding RS-232 Blocks” on page 2-4
“Building and Running the Real-Time Application” on page 2-8
“ Simulink Real-Time RS-232 Reference” on page 2-9

Adding RS-232 Blocks

You add RS-232 subsystem blocks to your Simulink model when you want to use the serial ports on the target computer for serial I/O.

After you create a Simulink model, you can add Simulink Real-Time driver blocks and configure those blocks. The following procedure describes how to use the serial ports on the target computer for I/O with the composite drivers.

Before you start, decide what COM port combinations you want to use. The example has you configure the Baseboard Send/Receive block. To configure this block, first select serial port pairs. This parameter specifies the ports for which you are defining transmit and receive. You have a choice of the following:

- `Com1/none`
- `Com2/none`
- `Com1/Com3`
- `Com2/Com4`

- `none/Com3`
- `none/Com4`
- `Custom`

If you select either the `Com1/Com3` or `Com2/Com4` pair, check that the port pair shares an interrupt. If the port pair does not share an interrupt, you cannot use the two ports as a pair.

Alternatively, you can define a `Custom` port pair. A `Custom` port pair is one that does not match the existing combinations of port pairs. When you select `Custom`, the dialog box allows you to configure your own port pair. For example, you can set the IRQ and two addresses for the port pair. If one of the ports is not used, set that address to `0`.

Normally, the ports are set to the following:

- COM1 — 0x3F8, IRQ 4
- COM2 — 0x2F8, IRQ 3
- COM3 — 0x3E8 (if present), IRQ 4
- COM4 — 0x2E8 (if present), IRQ 3

In a `Custom` port pair, either set one or both ports of the pair to addresses other than these conventions, or assign a different IRQ value. Some boards allow you to set the IRQ numbers independently.

If you select the port pairs `Com1/Com3` or `Com2/Com4`, you must include one Send/Receive subsystem block in the model. If you use COM1 and COM2, or COM1 and a custom port pair, you must include two Send/Receive blocks in the model.

The following example shows two models, one that uses a standard `Com1/Com3` port pair, and one that uses custom port pairs:

1. In the Command Window, type

   ```
   slrtlib
   ```

   The Simulink Real-Time driver block library opens.
2. Double-click the RS-232 group block.

   A window with blocks for RS-232 composite drivers opens.

   Alternatively, you can access the Simulink Real-Time block library from the Simulink Library Browser. In the Simulink Editor, and from the **View** menu, click **Show**

Library Browser. In the left pane, double-click **Simulink Real-Time**, and then click **RS232**.

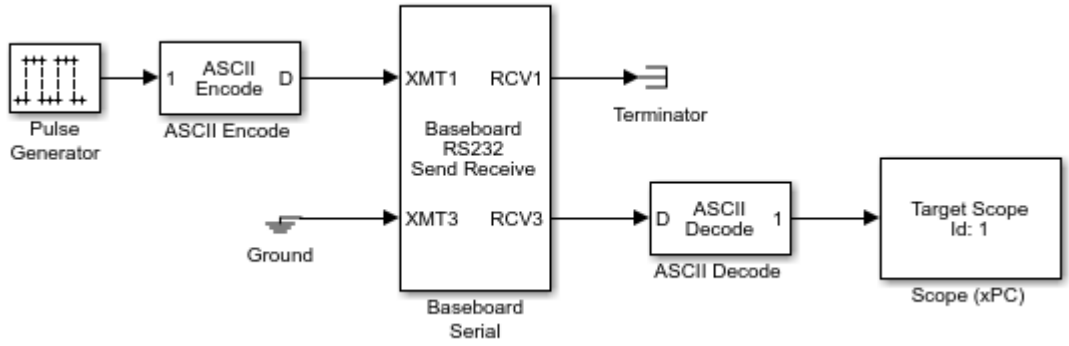
3. Drag an ASCII Encode block to your Simulink model. This block encodes input for the RS-232 Send Receive block.
4. Configure this block.
5. Drag an ASCII Decode block to your Simulink model. This block decodes output from the RS-232 Send Receive block.
6. Configure this block.
7. Double-click the Mainboard group block.
8. Depending on your port pair configuration, drag one or two Baseboard RS-232 Send/Receive blocks to your Simulink model.
9. Double-click the Baseboard RS-232 Send/Receive block.
10. Configure this block. Note the following **Parameter group** values:
    - When you select `Board Setup`, make sure that the **Configuration** value is consistent with your RS-232 serial port configuration.
    - When you select `Receive Setup`, for each channel, set the value of the **Receive Sample Time** parameter to a sample time value faster than the data being sent. Do not leave this value at `-1`. Set this parameter for all channels, including channels that you are not using; otherwise, you receive an error when generating code for the real-time application.
11. Add a Pulse Generator block and a target Scope block.
12. 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`.
13. From the Simulink Library Browser, select **Sinks**. Depending on your configuration, drag one or more Terminator blocks to your model. To suppress unused port messages, connect this block to the unused RCV1 port.

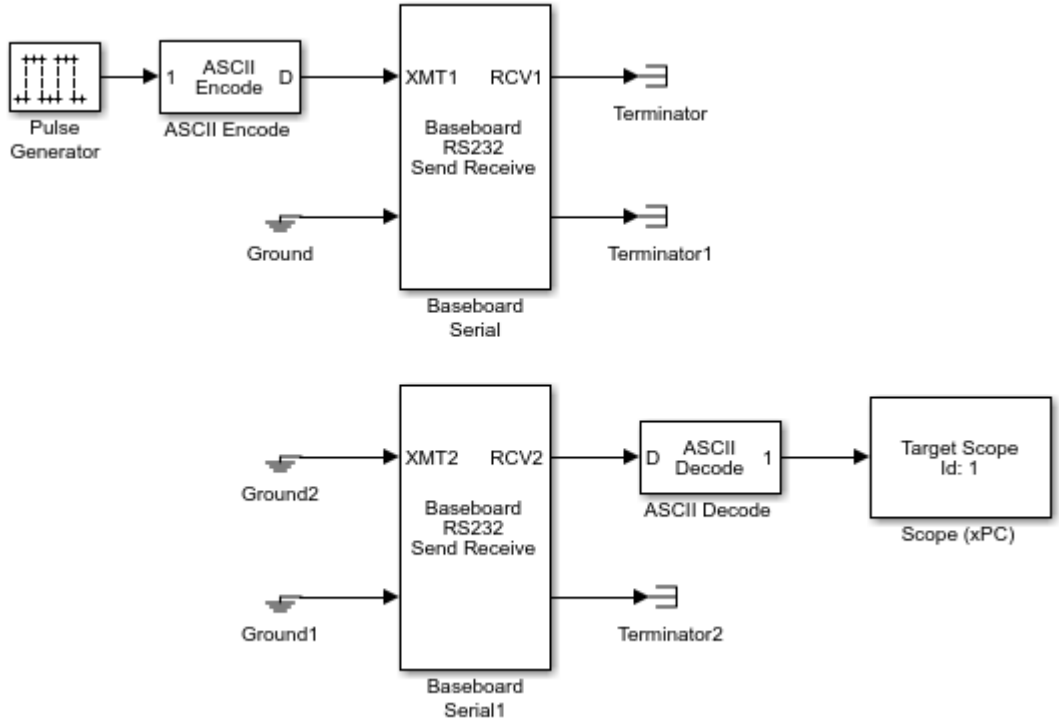
    From the Simulink Library Browser, select **Sources**. Depending on your configuration, drag the Ground block to your model. To suppress unused port messages, connect this block to the unused XMT3 port.

    Your model can use one block or two.

    The single-block model uses the `Com1/Com3` port pair:



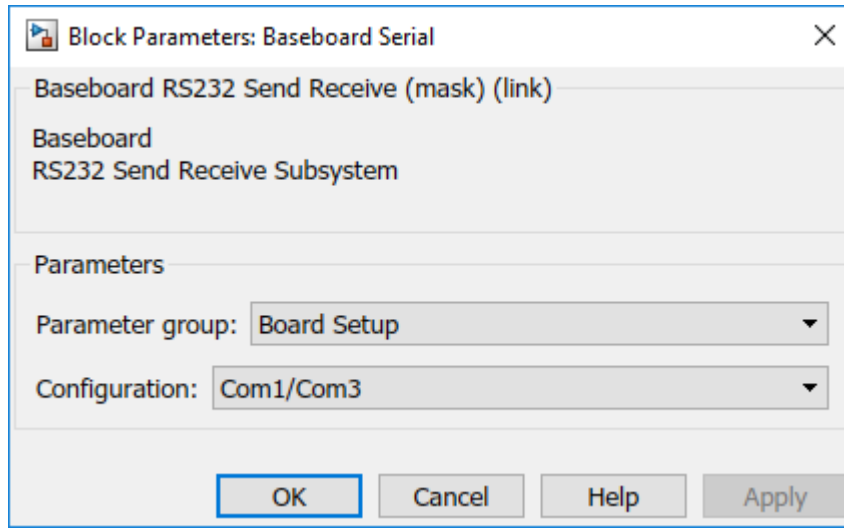
The two-block model uses two sets of `Custom` port pairs:



14 Double-click a Baseboard RS232 Send Receive block. To configure the ports on the target computer for this board, enter values.

Note This dialog box changes depending on the **Parameter group** selection.

For example, if the **Parameter group** is `Board Setup` and the target computer port is connected to COM1/COM3, your Send Receive block dialog box looks like this figure.



For more information on entering the block parameters, see RS-232 Send/Receive.

15 Click **OK**. The Send Receive block dialog box closes.

Your next task is to build and run the real-time application.

Building and Running the Real-Time Application

The Simulink Real-Time software and Simulink Coder™ create C code from your Simulink model. You can then use a C compiler to create executable code that runs on the target computer. This topic assumes that you know how to configure your model to create a real-time application. See “Build and Download Real-Time Application”.

After you have added the RS-232 blocks for the main board to your Simulink model and configured your model, you can build your real-time application.

1. In the Simulink Editor, and from the **Code** menu, click **C/C++ Code > Build Model**.
2. In the Command Window, type

   ```
   start(tg)
   ```

Simulink Real-Time RS-232 Reference

The Simulink Real-Time software supports RS-232 communication with driver blocks in your Simulink model.

Using the 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.
- More 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.

  The FIFO Read HDRS and FIFO Read Binary blocks are also useful to work with devices that can send different messages at different times.

The three FIFO read block types need their input to be of type `serialfifoptr`, which is output from F type Send Receive subsystems.

The following are examples of when you can use the FIFO Read block.

- For an instrument that sends a character vector like this:

  ```
  <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 reading and an amp reading with messages of the following 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'}`. Also configure the **Terminating string** parameter for carriage return (13) and line feed (10): `[13 10]`.

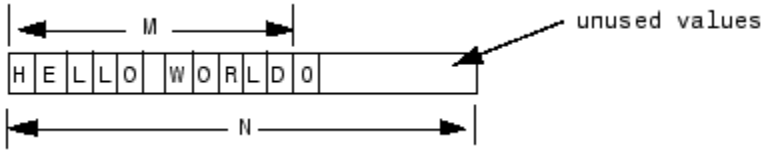
  Connect the output to multiple ASCII Decode blocks, one for each header and message. See the `xpcserialasciitest` and `xpcserialasciisplit` models in `xpcdemos` for examples of how to use this block in a model.
- For an instrument that sends a binary message, you typically know the length of each full 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. See the `xpcserialbinarytest` and `xpcserialbinarysplit` models in `xpcdemos` for further examples of how to use this block in a model.

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.

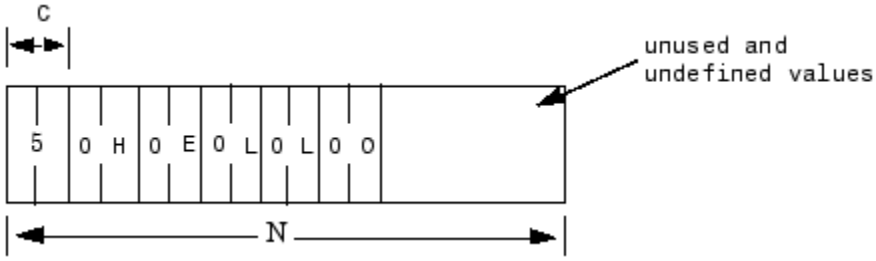
The 8-bit data types are NULL-terminated character vectors that are represented as Simulink vectors. The width is the maximum number of characters that can be stored. In

the following 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.



This character vector has 11 characters terminated with a NULL byte (`0`). This 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 following figure of a 16-bit data type, `C` is the count of the valid data, `N` is the width of the vector. This figure illustrates count + 16-bit `int` and count + 16-bit `uint` data types. It also applies to count + 32-bit `int` and count + 32-bit `uint` data types.



These 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.

Handling 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. However, 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. This condition causes a FIFO read block to perform one of the following, depending on how you configure the behavior:

- 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 with 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. Therefore, 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.

Controlling When You Send a Message

You can use the structure of both serial data types (“Signal Data Types” on page 2-10) to control when a message is sent. 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 the following:

- 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 a vector of `0` values. The Switch block only chooses between vectors of the same width. However, 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

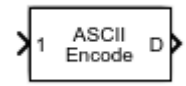
ASCII Decode | ASCII Decode V2 | ASCII Encode | FIFO Read | FIFO Read Binary | FIFO Read HDRS | FIFO Write | Modem Control | Modem Status | RS-232 Send/Receive | RS-232 Send/Receive FIFO | RS232 State

3

Serial Communications Support: Blocks

ASCII Encode

Convert Simulink values into `uint8` character vector
Library: RS232



Description

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`.

`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**.

`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.

`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`.

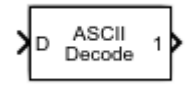
See Also
ASCII Decode

Topics

Introduced in R2008a

ASCII Decode

Parse ASCII character vector into Simulink values
Library: RS232



Description

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'
```

`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.

`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`.

See Also

ASCII Encode

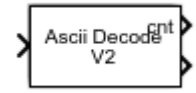
Topics

“RS-232 Serial Communication” on page 2-2
“RS-232 Composite Drivers” on page 2-4

Introduced in R2008a

ASCII Decode V2

Parse ASCII character vector into Simulink values
Library: 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

It 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`. They 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, assume that 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.

See Also

Topics

“RS-232 Serial Communication” on page 2-2
“RS-232 Composite Drivers” on page 2-4

Introduced in R2008a

FIFO Read

Read simple data streams
Library: RS232

Description

The FIFO Read block is the read side of a FIFO read/write pair. It is used to parse simple data streams. The block functions in two modes, set using the **Read to delimiter** check box.

- If you select the **Read to delimiter** check box, the block only reads 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**.

  You usually select the **Read to delimiter** check box when performing ASCII reads and clear it when performing binary reads.

The following 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 “Using the FIFO Read Blocks” on page 2-9.

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 cause this port to become 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 cause this port to become 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 cause this port to become 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 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. On the other hand, if it contains `5`, the block returns `5` characters plus the null terminator.

Dependency

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 the **Read to delimiter** check box is not selected.

`Minimum read size` — Minimum number of characters returned by block
`1` (default) | integer

Enter the smallest desired read size in bytes. The FIFO must contain at least this number of elements before elements are returned.

Dependency

If you select the parameter **Max and Min read size ports**, the value of port `MIN` supersedes this value.

`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.

`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. It works with the **Read to delimiter** parameter. By default, this block looks for a carriage return. It only returns characters when one is found. For reference, the decimal value of a carriage return is `13`, a line feed is `10`.

`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.

`Max and Min read size ports` — Enable maximum and minimum input ports
`off` (default) | `on`

When this check box is selected:

- 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 the FIFO must contain before elements can be returned. This value supersedes the value set with the **Minimum read size** parameter.

Dependency

Causes input ports `MAX` and `MIN` to become visible.

`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.

`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.

See Also

FIFO Write

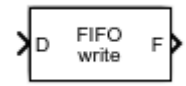
Topics

“RS-232 Serial Communication” on page 2-2
“RS-232 Composite Drivers” on page 2-4

Introduced in R2008a

FIFO Write

Write simple data streams
Library: RS232



Description

The FIFO Write block is the write side of a FIFO read/write pair. It is used 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 cause this port to become 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.

`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 Composite Drivers” on page 2-4.

`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.

Dependency

Causes output port `DP` to become visible.

`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.

`ID` — Identifier for overflow messages
character vector

Enter a user-defined identifier for FIFO overflow messages.

See Also

FIFO Read

Topics

“RS-232 Serial Communication” on page 2-2
“RS-232 Composite Drivers” on page 2-4

Introduced in R2008a

FIFO Read HDRS

Read multiple ASCII data streams according to header information
Library: 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, doing so requires a complicated state machine with the following 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. In this way, 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 `xpcdemos` folder contains the following examples that illustrate how to use the FIFO Read HDRS block: `xpcserialasciitest` and `xpcserialasciisplit`.

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 cause this port to become 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.

`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.

`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`.

`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.

`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 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. On the other hand, if it contains `5`, the block returns `5` characters plus the null terminator.

`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.

`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.

See Also

FIFO Read | FIFO Read Binary | FIFO Write

Topics

“RS-232 Serial Communication” on page 2-2
“RS-232 Composite Drivers” on page 2-4

Introduced in R2008a

FIFO Read Binary

Read multiple binary data streams according to header information
Library: 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. In this way, the block catches up with data that arrives faster than the block executes.

The `xpcdemos` folder contains the following examples that illustrate how to use the FIFO Read HDRS block: `xpcserialbinarytest` and `xpcserialbinarysplit`.

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 cause this port to become 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.

`Message Lengths` — Message lengths, in bytes

`1024` (default) | integer

Enter the message length, in bytes. Include the header in the length.

`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`.

`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.

`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 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. On the other hand, if it contains `5`, the block returns `5` characters plus the null terminator.

`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.

`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.

See Also

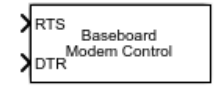
Topics

“RS-232 Serial Communication” on page 2-2
“RS-232 Composite Drivers” on page 2-4

Introduced in R2008a

Modem Control

Control state of RTS and DTR output lines on serial port
Library: RS232 / Mainboard



Description

The Modem Control block controls the state of either or both of the RTS and 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.

DTR — Enable control of DTR line for serial device
`on` (default) | `off`

Select this check box to control the DTR line for this port.

`Configuration` — Specify port of modem control line
`COM1` (default) | `COM2` | `COM3` | `COM4` | `Custom`

From the list, select a port to access:

- `COM1` — 0x3F8
- `COM2` — 0x2F8
- `COM3` — 0x3E8
- `COM4` — 0x2E8
- `Custom` — A port that is set to an address other than the addresses for `COM1`, `COM2`, `COM3`, or `COM4`.

Dependency

The value `Custom` causes the **Base address** parameter to become visible.

`Base address` — Base address for serial port
`0x3f8` (default) | integer

Use this base address to specify a `Custom` serial port.

Dependency

The value `Custom` causes the **Base address** parameter to become visible.

See Also

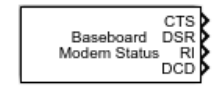
Topics

“RS-232 Serial Communication” on page 2-2
“RS-232 Composite Drivers” on page 2-4

Introduced in R2008a

Modem Status

Return state of modem control lines
Library: RS232 / Mainboard



Description

The Modem Status block reads the state of the four 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.

`RI` — Status of ring indicator line
`true` | `false`

If `true`, the modem has received an incoming ring signal.

Dependency

To make this output visible, select the **RI** 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.

Dependency

Selecting this parameter makes the `CTS` port visible.

`DSR` — Enables data set ready status output
`on` (default) | `off`

Select this check box to monitor the DSR line of the modem.

Dependency

Selecting this parameter makes the `DSR` port visible.

`RI` — Enables ring indicator status output
`on` (default) | `off`

Select this check box to monitor the RI line of the modem.

Dependency

Selecting this parameter makes the `RI` port visible.

`DCD` — Enables data carrier detect status output

`on` (default) | `off`

Select this check box to monitor the DCD line of the modem.

Dependency

Selecting this parameter makes the `DCD` port visible.

`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.

`Configuration` — Specify port of modem control line

`COM1` (default) | `COM2` | `COM3` | `COM4` | `Custom`

From the list, select a port to access:

- `COM1` — 0x3F8
- `COM2` — 0x2F8
- `COM3` — 0x3E8
- `COM4` — 0x2E8
- `Custom` — A port that is set to an address other than the addresses for `COM1`, `COM2`, `COM3`, or `COM4`.

Dependency

The value `Custom` causes the **Base address** parameter to become visible.

`Base address` — Base address for serial port

`0x3f8` (default) | integer

Use this base address to specify a `Custom` serial port.

Dependency

The value `Custom` causes the **Base address** parameter to become visible.

See Also

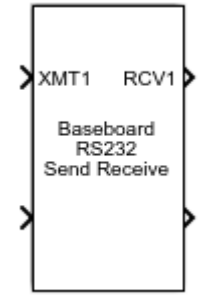
Topics

“RS-232 Serial Communication” on page 2-2
“RS-232 Composite Drivers” on page 2-4

Introduced in R2008a

RS-232 Send/Receive

Send and receive data over Mainboard Baseboard serial port
Library: RS232 / Mainboard



Description

The Send/Receive block sets up the serial interface to send and receive basic character streams. This block has basic 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.

Only one Send/Receive can exist for each COM interrupt. All ports that use that interrupt must be associated with that block. For example, if the main board is configured with four ports, COM1 and COM3 typically share an interrupt. In this case, COM1 and COM3 must then share a Send/Receive block.

Ports

Input

`XMT1` — Vector 1 of data to transmit

`[int8]` | `[uint8]` | `[int16]` | `[uint16]` | `[int32]` | `[uint32]`

Vector of data to transmit over port 1.

XMT2 — Vector 2 of data to transmit over serial port
[int8] | [uint8] | [int16] | [uint16] | [int32] | [uint32]

Vector of data to transmit over port 2.

Output

RCV1 — Vector 1 of data that has been received over serial port
[int8] | [uint8] | [int16] | [uint16] | [int32] | [uint32]

Vector containing data that has been received from serial port 1.

RCV2 — Vector 2 of data that has been received over serial port
[int8] | [uint8] | [int16] | [uint16] | [int32] | [uint32]

Vector containing data that has been received from serial port 2.

Parameters

Parameter group — Select groups of parameters
Board Setup (default) | Basic Setup | Transmit Setup | Receive Setup

To configure a group of parameters, select a group.

Board Setup

Configuration — Specify ports for transmitting and receiving
Com1/none (default) | Com2/none | Com1/Com3 | Com2/Com4 | none/Com3 | none/Com4 | Custom

This parameter specifies the ports for which you are defining transmit and receive. For example, Com1/Com3 specifies that port 1 uses COM1 and port 2 uses COM3. On the Simulink block, the upper port is port 1 and the lower port is port 2.

A Custom configuration is one that does not match the existing combinations of port pairs. For example, assume that your target computer BIOS disables port 1 and reconfigures port 2 to use base address 0x220, IRQ 11. Then you can make the following settings:

- **Configuration** — `Custom`
- **IRQ number** — `11`
- **First port address** — `0`
- **Second port address** — `0x220`

In this case, port `1` is unused.

Dependency

The value `Custom` makes the **IRQ number**, **First port address**, and **Second port address** parameters visible.

`IRQ number` — Base address for serial port 2

`4` (default) | integer

Use this IRQ to specify a `Custom` serial port configuration.

Dependency

The value `Custom` causes the **IRQ number** parameter to become visible.

`First base address` — Base address for serial port 1

`0x3f8` (default) | integer

Use this base address to specify a `Custom` serial port configuration.

Dependency

The value `Custom` causes the **First base address** parameter to become visible.

`Second base address` — Base address for serial port 2

`0x3f8` (default) | integer

Use this base address to specify a `Custom` serial port configuration.

Dependency

The value `Custom` causes the **Second base address** parameter to become visible.

Basic Setup

`Port to modify` — Specify 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.

`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.

`Parity` — Parity for checking data transfer
`None` (default) | `Even` | `Odd` | `Mark` | `Space`

Select a parity for checking data integrity.

`Data bits` — Number of bits per character
8 (default) | 7 | 6 | 5

Select the number of bits that encode a character.

`Stop bits` — Number of stop bits for port
1 (default) | 2

Select the number of stop bits for the character stream.

`Hardware FIFO size` — Specify FIFO depth of UART
`16 deep` (default) | `64 deep` | `1 deep`

Depth of hardware FIFO, in characters. The capability of the UART limits the depth of the FIFO.

`Receive FIFO interrupt level` — Number of characters in hardware FIFO before interrupt
`half full` (default) | `1` | `quarter full` | `almost full`

This parameter specifies the number of characters in the receive hardware FIFO before an interrupt occurs.

Receive interrupts occur at least as often as this parameter specifies. Each interrupt calls the interrupt service routine, causing overhead. Interrupt level 1 produces much higher overhead than the other settings. Consider interrupt level 1 only for applications that have low latency.

If both of the following are true, the UART requests an interrupt for the receiver regardless of the value of **Receive FIFO interrupt level**:

- The FIFO contains at least 1 character.
- A gap of at least 4 character times (the time required to transfer four characters) occurs in a data stream.

`Auto RTS/CTS` — Enable RTS/CTS handshake
`off` (default) | `on`

To enable the RTS/CTS handshake of the 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. However, if your model gets FIFO overruns, select this check box.

Transmit Setup

`Port to modify` — Specify 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.

`Transmit software FIFO size` — Transmitter FIFO size, in bytes
`1024` (default) | integer

Enter the transmit software FIFO size, in bytes. This parameter specifies the size of the software FIFO that the block uses to buffer transmitted characters.

`Transmit FIFO data type` — Data type of transmitter
`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`

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 allows 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 would terminate the character vector.

Receive Setup

`Port to modify` — Specify 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.

`Receive software FIFO size` — Receiver FIFO size, in bytes
integer

Enter the receive software FIFO size, in bytes. This parameter specifies the size of the software FIFO that the block uses to buffer characters between interrupt service and periodic execution.

`Receive maximum read` — 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.

Dependency

If the **Read to delimiter** check box is selected and if the block does not find the delimiter before it reads **Receive maximum read** characters, the output vector is empty.

`Receive minimum read` — 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.

`Read to delimiter` — Return characters including message delimiter
`on` (default) | `off`

Select this check box to have this block return all characters in the FIFO, up to and including the specified delimiter.

If the buffer has errors, such as framing errors, the modem returns characters regardless of the presence of the delimiter. This special case helps diagnose errors such as mismatched baud rates.

Dependency

If the block does not find the delimiter before it reads **Receive maximum read** characters, the output vector is empty.

`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).

`Receive FIFO data type` — Data type of receiver

`count+16 bit uint` (default) | `8 bit uint null terminated` | `count+32 bit int` | `count+32 bit uint` | `count+16 bit int` | `8 bit int null terminated`

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.

`Receive Sample Time` — Sample time of receiver

`-1` (default) | numeric

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

See Also

RS-232 Send/Receive FIFO

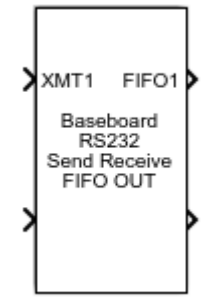
Topics

“RS-232 Serial Communication” on page 2-2
“RS-232 Composite Drivers” on page 2-4

Introduced in R2008a

RS-232 Send/Receive FIFO

Send and receive data over Mainboard Baseboard serial port with FIFO
Library: RS232 / Mainboard



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 FIFO outputs.

A model that contains a Send/Receive FIFO block with the FIFO Read block provides the same capability as the Send/Receive block. A model that contains a Send/Receive FIFO block with a FIFO Read HDRS or FIFO Read Binary block provides greater capability than the Send/Receive block.

Only one Send/Receive can exist for each COM interrupt. All ports that use that interrupt must be associated with that block. For example, if the main board is configured with four ports, COM1 and COM3 typically share an interrupt. In this case, COM1 and COM3 must then share a Send/Receive block.

Ports

Input

`XMT1` — Vector 1 of data to transmit

`[int8]` | `[uint8]` | `[int16]` | `[uint16]` | `[int32]` | `[uint32]`

Vector of data to transmit over port 1.

XMT2 — Vector 2 of data to transmit over serial port
[int8] | [uint8] | [int16] | [uint16] | [int32] | [uint32]

Vector of data to transmit over port 2.

Output

FIF01 — FIFO 1 of data that has been received over serial port
serialfifoptr

FIFO containing data that has been received from serial port 1.

FIF02 — FIFO 2 of data that has been received over serial port
serialfifoptr

FIFO containing data that has been received from serial port 2.

Parameters

Parameter group — Select groups of parameters
Board Setup (default) | Basic Setup | FIFO Setup

To configure a group of parameters, select a group.

Board Setup

Configuration — Specify ports for transmitting and receiving
Com1/none (default) | Com2/none | Com1/Com3 | Com2/Com4 | none/Com3 | none/Com4 | Custom

This parameter specifies the ports for which you are defining transmit and receive. For example, Com1/Com3 specifies that port 1 uses COM1 and port 2 uses COM3. On the Simulink block, the upper port is port 1 and the lower port is port 2.

A Custom configuration is one that does not match the existing combinations of port pairs. For example, assume that your target computer BIOS disables port 1 and reconfigures port 2 to use base address 0x220, IRQ 11. Then you can make the following settings:

- **Configuration** — `Custom`
- **IRQ number** — `11`
- **First port address** — `0`
- **Second port address** — `0x220`

In this case, port `1` is unused.

Dependency

The value `Custom` makes the **IRQ number**, **First port address**, and **Second port address** parameters visible.

`IRQ number` — Base address for serial port 2

`4` (default) | integer

Use this IRQ to specify a `Custom` serial port configuration.

Dependency

The value `Custom` causes the **IRQ number** parameter to become visible.

`First base address` — Base address for serial port 1

`0x3f8` (default) | integer

Use this base address to specify a `Custom` serial port configuration.

Dependency

The value `Custom` causes the **First base address** parameter to become visible.

`Second base address` — Base address for serial port 2

`0x3f8` (default) | integer

Use this base address to specify a `Custom` serial port configuration.

Dependency

The value `Custom` causes the **Second base address** parameter to become visible.

Basic Setup

`Port to modify` — Specify 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.

`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.

`Parity` — Parity for checking data transfer
`None` (default) | `Even` | `Odd` | `Mark` | `Space`

Select a parity for checking data integrity.

`Data bits` — Number of bits per character
8 (default) | 7 | 6 | 5

Select the number of bits that encode a character.

`Stop bits` — Number of stop bits for port
1 (default) | 2

Select the number of stop bits for the character stream.

`Hardware FIFO size` — Specify FIFO depth of UART
`16 deep` (default) | `64 deep` | `1 deep`

Depth of hardware FIFO, in characters. The capability of the UART limits the depth of the FIFO.

`Receive FIFO interrupt level` — Number of characters in hardware FIFO before interrupt
`half full` (default) | `1` | `quarter full` | `almost full`

This parameter specifies the number of characters in the receive hardware FIFO before an interrupt occurs.

Receive interrupts occur at least as often as this parameter specifies. Each interrupt calls the interrupt service routine, causing overhead. Interrupt level 1 produces much higher overhead than the other settings. Consider interrupt level 1 only for applications that have low latency.

If both of the following are true, the UART requests an interrupt for the receiver regardless of the value of **Receive FIFO interrupt level**:

- The FIFO contains at least 1 character.
- A gap of at least 4 character times (the time required to transfer four characters) occurs in a data stream.

`Auto RTS/CTS` — Enable RTS/CTS handshake
`off` (default) | `on`

To enable the RTS/CTS handshake of the 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. However, if your model gets FIFO overruns, select this check box.

FIFO Setup

`Port to modify` — Specify 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.

`Transmit software FIFO size` — Transmitter FIFO size, in bytes
`1024` (default) | integer

Enter the transmit software FIFO size, in bytes. This parameter specifies the size of the software FIFO that the block uses to buffer transmitted characters.

`Transmit FIFO data type` — Data type of transmitter
`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`

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 allows 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 would terminate the character vector.

`Receive software FIFO size` — Receiver FIFO size, in bytes
integer

Enter the receive software FIFO size, in bytes. This parameter specifies the size of the software FIFO that the block uses to buffer characters between interrupt service and periodic execution.

See Also

FIFO Read | FIFO Read Binary | FIFO Read HDRS | RS-232 Send/Receive

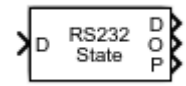
Topics

“RS-232 Serial Communication” on page 2-2
“RS-232 Composite Drivers” on page 2-4

Introduced in R2008a

RS232 State

Monitor board state information from send/receive block
Library: RS232



Description

The RS232 State block monitors the UART status that comes from a receive port of a send/receive block. The driver puts the UART status in 16-bit or 32-bit data streams. The RS232 State block looks at this status. Only the FIFO Read block passes this status information to its output port.

The RS232 State block accumulates errors over the whole input vector. An output error state is true if it is true for any byte in the input vector.

Ports

Input

`D` — Input vector from send/receive block

`[int8]` | `[uint8]` | `[int16]` | `[uint16]`

The error status depends upon the data type of input vector `D`:

- `int16`, `uint16` — The upper byte contains the error status bits from the UART.
- `int8`, `uint8` — No error status is available. The Boolean outputs are `false`.

Output

`D` — Passthrough of input vector

`int8` | `uint8` | `int16` | `uint16`

Passes through the input vector `D`.

0 — Overrun error status

`true` | `false`

If the hardware FIFO in the UART is full when a character on the serial port enters the UART, this output is `true`.

Dependency

To make this output visible, select the **Overrun error output** parameter.

P — Parity error status

`true` | `false`

If any byte in the input vector fails the parity check, this output is `true`.

Dependency

To make this output visible, select the **Parity error output** parameter.

F — Framing error status

`true` | `false`

If a framing error occurs on any character in this vector, this output is `true`. For example, a framing error can occur if the baud rates between the transmitter and receiver do not match.

Dependency

To make this output visible, select the **Framing error output** parameter.

B — Line break interrupt status

`true` | `false`

If the UART detects a serial line break condition, this output is `true`. A line break interrupt is not an error, but the UART treats it like an error state.

To detect a line break condition, the UART checks how long the serial line remains at voltage `0` (not `mark` and not `space`). If the line is at voltage `0` for longer than the time required to receive one character, the UART detects a line break. For some serial I/O port modules, disconnecting the serial cable does not cause a line break.

Dependency

To make this output visible, select the **Break interrupt output** parameter.

Parameters

`Overrun error output` — Enable overrun error check
`on` (default) | `off`

Select this check box to retrieve overrun error output.

Dependency

Selecting this parameter makes the `0` port visible.

`Parity error output` — Enable parity error check
`on` (default) | `off`

Select this check box to retrieve parity error output.

Dependency

Selecting this parameter makes the `P` port visible.

`Framing error output` — Enable framing error check
`off` (default) | `on`

Select this check box to retrieve framing error output.

Dependency

Selecting this parameter makes the `F` port visible.

`Break interrupt output` — Enable break interrupt check
`off` (default) | `on`

Select this check box to retrieve break interrupt output.

Dependency

Selecting this parameter makes the `B` port visible.

See Also

Topics

“RS-232 Serial Communication” on page 2-2

“RS-232 Composite Drivers” on page 2-4

Introduced in R2008a

A

Serial Communications Support: Internal Blocks

RS-232 Enable TX Interrupt

RS-232 Enable TX Interrupt Mainboard Baseboard block

Library

Simulink Real-Time Library for RS-232

Description

The Enable TX Interrupt block enables the transmitter buffer empty interrupt when data is present in the software FIFO.

The input port for controlling the interrupt is a Boolean value. If the input port value is true, the Enable Transmit Interrupt block enables the transmitter buffer empty interrupt in the UART. After the interrupt service routine empties the software FIFO, the interrupt is disabled.

Block Parameters

Base address

Enter the base address of the UART for which you want to enable the transmitter buffer empty interrupt.

See Also

Topics

“RS-232 Serial Communication” on page 2-2
“RS-232 Composite Drivers” on page 2-4

Introduced before R2006a

RS-232 Filter Interrupt Reason

RS-232 Filter Interrupt Reason Mainboard Baseboard block

Library

Simulink Real-Time Library for RS-232

Description

The Filter Interrupt Reason block filters the output of the Read Interrupt Status block.

If the condition that the interrupt query block reads from the IRR register matches the one specified here, the output is true.

This block is used exclusively inside the interrupt service subsystem for this board.

Block Parameters

Port

From the list, choose a port. This parameter specifies the port from which this block gets control data.

Filter value

From the list, choose `Receive data`, `Transmitter empty`, or `Modem status change`. This parameter specifies the interrupt reason that this filter block is looking for.

Note that `Modem status change` currently does nothing because the interrupt is not enabled.

See Also

Topics

“RS-232 Serial Communication” on page 2-2
“RS-232 Composite Drivers” on page 2-4

Introduced before R2006a

RS-232 Read Hardware FIFO

RS-232 Read Hardware FIFO Mainboard Baseboard block

Library

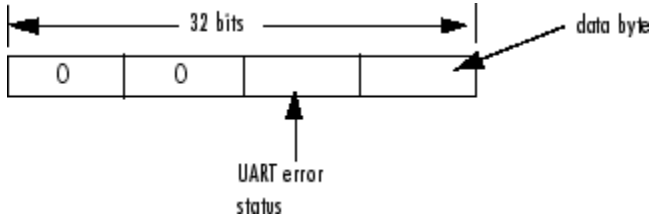
Simulink Real-Time Library for RS-232

Description

The Read Hardware FIFO block reads characters from the I/O module FIFO in the UART.

It then outputs those characters as the low-order byte of an unsigned 32-bit integer vector with a width of 65. This output vector is large enough to hold the maximum number of characters that the FIFO can hold. The first element of the vector specifies the number of data elements in the remainder of the vector.

If the input to the enable port (input port, labeled E) is not true, this block outputs a zero-length vector. The following illustrates the vector.



The UART error status can contain one of the following error values:

`0x02` — Overrun error

`0x04` — Parity error

`0x08` — Framing error

`0x01` — Break interrupt

The data byte ranges from 0 to 255.

The dialog box for the RS-232 FIFO Read block contains the following fields.

Block Parameters

Flush HW FIFO on startup

Select this check box to flush the FIFO when the device starts up.

Base address

Enter the base address of the UART for which you want to read the FIFO.

See Also

Topics

“RS-232 Serial Communication” on page 2-2
“RS-232 Composite Drivers” on page 2-4

Introduced before R2006a

RS-232 Read Interrupt Status

RS-232 Read Interrupt Status Mainboard Baseboard block

Library

Simulink Real-Time Library for RS-232

Description

The Read Interrupt Status block reads the interrupt status for the boards in the system.

The output for this block is a vector with one 32-bit element for each port. Each element contains two pieces of information for that port, where the 4 bytes are:

```
[0, 0, IRR, Reason]
```

The Read Interrupt Status block has signal output with the following format:

This output is a vector of integers. The values in the reason byte and their definitions are:

- `0` — This UART did not cause this interrupt.
- `1` — Receive characters are available.
- `2` — Transmit holding register is empty.
- `3` — Modem status has changed (ignored).

The second byte is the value read from the Interrupt Reason Register (IRR). This register is specific to the 16450, 16550, and 16750 types of UARTs. Some bytes in this register give the active FIFO depth. Other bytes give the maximum number of characters that the transmitter empty interrupt handlers can write to the transmit FIFO.

Block Parameters

Base address 1

Enter the base address of the first UART for which you want to read the interrupt status.

Base address 2

Enter the base address of the second UART for which you want to read the interrupt status.

See Also

Topics

“RS-232 Serial Communication” on page 2-2
“RS-232 Composite Drivers” on page 2-4

Introduced before R2006a

RS-232 Setup

RS-232 Setup Mainboard Baseboard block

Library

Simulink Real-Time Library for RS-232

Description

A setup block is a subsystem block that sets up the interface characteristics for the board.

Block Parameters

Baud rate

From the list, choose a baud.

Number of data bits

From the list, choose either 5, 6, 7 or 8 to define the number of data bits for the port.

Number of stop bits

From the list, choose either 1 or 2 to define the number of stop bits for the port.

Parity

From the list, choose `None`, `Even`, `Odd`, `Mark` or `Space`. This parameter defines the receive and transfer parity.

Fifo mode

From the list, choose `64 deep`, `16 deep`, or `1 deep`. This parameter sets the transmit and receive FIFO depth. The UART can operate with a FIFO depth of 1 character (1 deep), 16 characters (16 deep), or 64 characters (64 deep).

Receive trigger level

From the list, choose `1`, `quarter full`, `half full`, or `almost full`. This parameter defines a trigger level for a receive data available interrupt. When the FIFO reaches the level specified in this parameter, the driver asserts the receive data available interrupt.

Enable auto RTS/CTS

Select this check box to enable handshaking using the RTS and CTS modem control lines. If this is not checked, handshaking is not done.

Base Address

Enter the base address of the board that you are setting up.

See Also

Topics

“RS-232 Serial Communication” on page 2-2
“RS-232 Composite Drivers” on page 2-4

Introduced before R2006a

RS-232 Write Hardware FIFO

RS-232 Write Hardware FIFO Mainboard Baseboard block

Library

Simulink Real-Time Library for RS-232

Description

The Write Hardware FIFO block writes the data from the input port (labeled E) to the FIFO in the I/O module UART for this port.

The following pseudocode describes the behavior of this FIFO.

```
if (enable is false)
     return
else
{
     if (input data empty)
          disable transmitter buffer empty interrupt
          return
     else
          copy input data to HW FIFO
}
```

In words: if the enable port (input port E) becomes `true` and the input data has length `0`, then the block turns off the transmitter buffer empty interrupt. Otherwise, the block adds input data to the FIFO.

Block Parameters

Base address

Enter the base address of the UART for which you want to write the FIFO.

See Also

Topics

“RS-232 Serial Communication” on page 2-2
“RS-232 Composite Drivers” on page 2-4

Introduced before R2006a

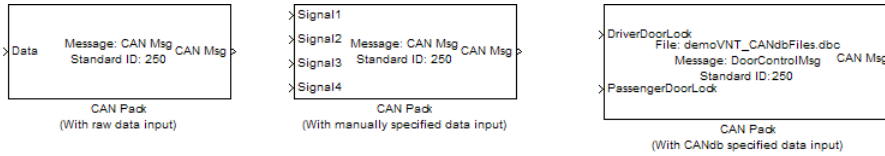
CAN, Encoders, Ethernet, EtherCAT

4

CAN Utility Blocks

CAN Pack

Pack individual signals into CAN message



Library

CAN Communication

Embedded Coder®/ Embedded Targets/ Host Communication

Description

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

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

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

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

Other Supported Features

The CAN Pack block supports:

- The use of Simulink Accelerator™ 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.
- Code generation to deploy models to targets.

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

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

Dialog Box

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

Parameters

Data is input as

Select your data signal:

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

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

CANdb file

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

Note 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.

Message list

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

Message

Name

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

Identifier type

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

Identifier

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

Length (bytes)

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

Remote frame

Specify the CAN message as a remote frame.

Output as bus

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” (Simulink).

Signals Table

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

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

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

Name

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

Start bit

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

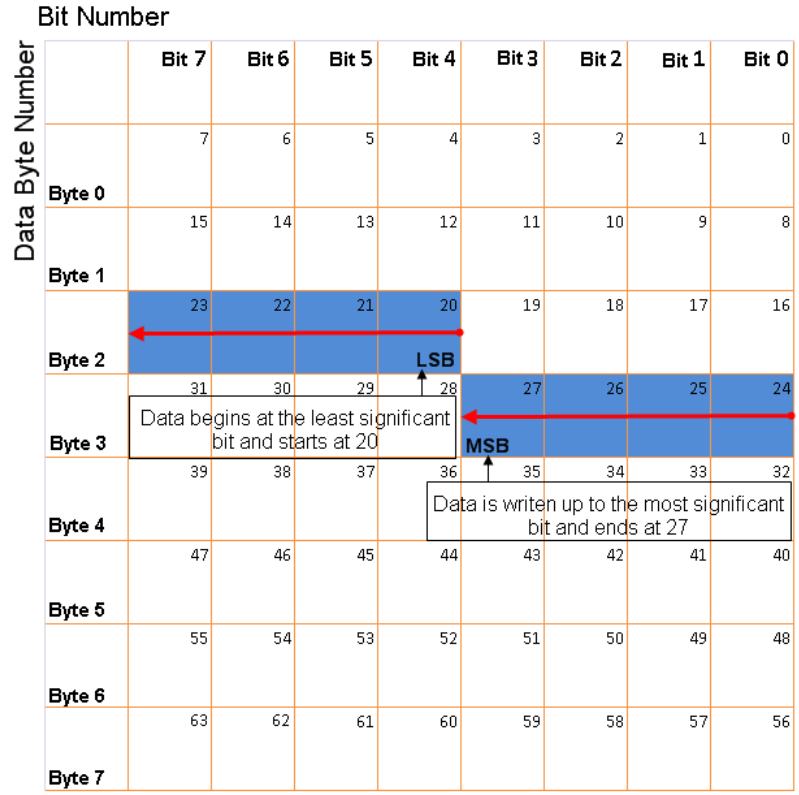
Length (bits)

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

Byte order

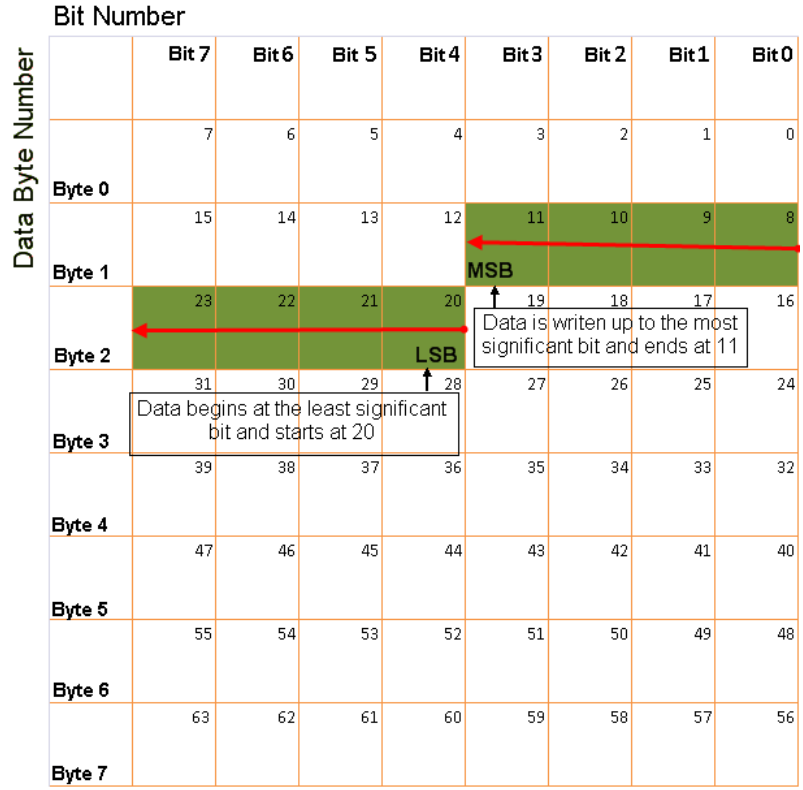
Select either of the following options:

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



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

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



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

Data type

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

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

Multiplex type

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

- `Standard`: The signal is packed at each timestep.
- `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 the following types and values.

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

In this example:

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

Multiplex value

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

Factor

Specify the **Factor** value to apply to convert the physical value (signal value) to the raw value packed in the message. See “Conversion Formula” on page 4-9 to understand how physical values are converted to raw values packed into a message.

Offset

Specify the **Offset** value to apply to convert the physical value (signal value) to the raw value packed in the message. See “Conversion Formula” on page 4-9 to understand how physical values are converted to raw values packed into a message.

Min

Specify the minimum physical value of the signal. The default value is `-inf` (negative infinity). You can specify a number for the minimum value. See “Conversion Formula” on page 4-9 to understand how physical values are converted to raw values packed into a message.

Max

Specify the maximum physical value of the signal. The default value is `inf`. You can specify a number for the maximum value. See “Conversion Formula” on page 4-9 to understand how physical values are converted to raw values packed into a message.

Conversion Formula

The conversion formula is

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

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

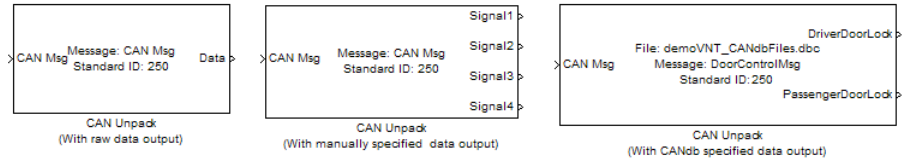
See Also

Blocks
CAN Unpack

Introduced in R2009a

CAN Unpack

Unpack individual signals from CAN messages



Library

CAN Communication

Embedded Coder/ Embedded Targets/ Host Communication

Description

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

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

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

Other Supported Features

The CAN Unpack block supports:

- The use of Simulink Accelerator 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.
- Code generation to deploy models to targets.

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

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

Dialog Box

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

Parameters

Data to be output as

Select your data signal:

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

  The number of output ports on your block depends on the number of signals you specify. For example, if you specify four signals, your block has four output ports.
- **CANdb specified signals**: Allows you to specify a CAN database file that contains data signals. If you select this option, select a CANdb file.

  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.

Note 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.

CANdb file

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

Note 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.

Message list

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

Message

Name

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

Identifier type

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

Identifier

Specify your CAN message ID. This number must be a integer from 0 through 2047 for a standard identifier and from 0 through 536870911 for an extended identifier. If

you specify `-1`, the block unpacks 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.

Length (bytes)

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

Signals Table

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

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

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

Name

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

Start bit

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

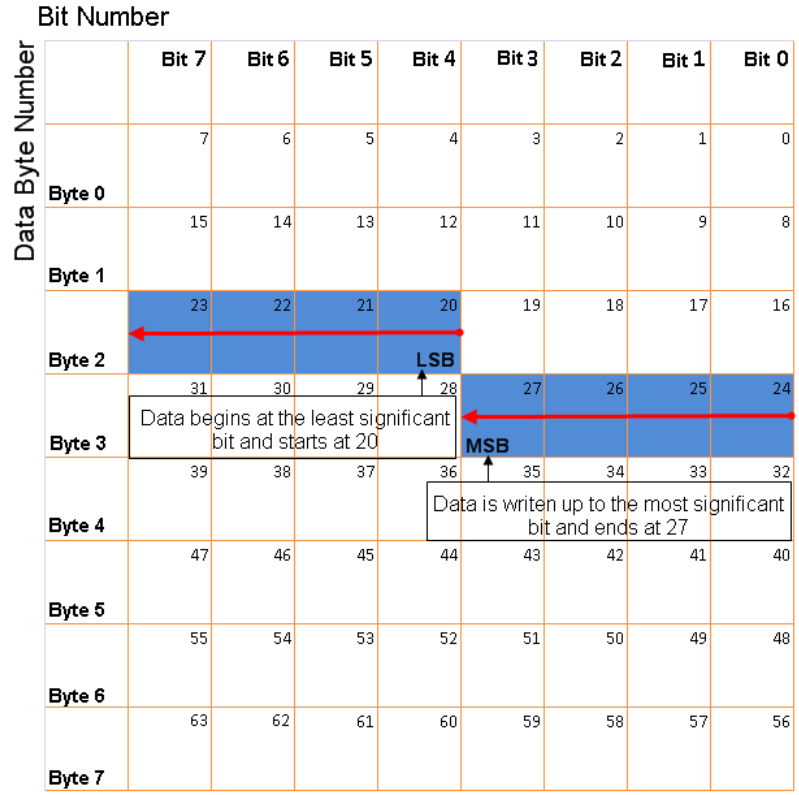
Length (bits)

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

Byte order

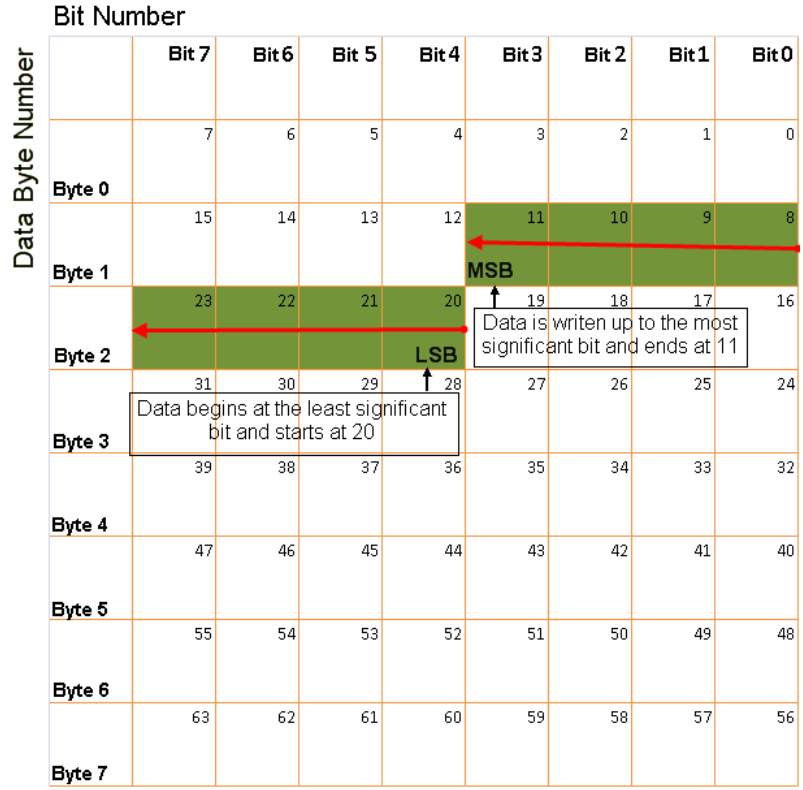
Select either of the following options:

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



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

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



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

Data type

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

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

Multiplex type

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

- `Standard`: The signal is unpacked at each timestep.
- `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 the following values.

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

In this example:

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

Multiplex value

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

Factor

Specify the **Factor** value applied to convert the unpacked raw value to the physical value (signal value). See “Conversion Formula” on page 4-18 to understand how unpacked raw values are converted to physical values.

Offset

Specify the **Offset** value applied to convert the physical value (signal value) to the unpacked raw value. See “Conversion Formula” on page 4-18 to understand how unpacked raw values are converted to physical values.

Min

Specify the minimum raw value of the signal. The default value is `-inf` (negative infinity). You can specify a number for the minimum value. See “Conversion Formula” on page 4-18 to understand how unpacked raw values are converted to physical values.

Max

Specify the maximum raw value of the signal. The default value is `inf`. You can specify a number for the maximum value. See “Conversion Formula” on page 4-18 to understand how unpacked raw values are converted to physical values.

Output Ports

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

Output identifier

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

Output remote

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

Output timestamp

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

Output length

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

Output error

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

Output status

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

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

Conversion Formula

The conversion formula is

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

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

See Also

Blocks
CAN Pack

Introduced in R2009a

5

Model-Based Ethernet Communications Support

Model-Based Ethernet Communications

In this section...

What Is Model-Based Ethernet Communications?

The Simulink Real-Time software supports communication from the target computer to other systems or devices using raw Ethernet (Ethernet packets). Raw Ethernet is a direct method to send and receive packets with the real-time application using the Ethernet protocol. To transfer data using Ethernet packets, you must manually create Ethernet frames. This topic assumes that you are knowledgeable about the IEEE® 802.3 standard.

By itself, raw Ethernet does not implement the TCP/IP or UDP standards. For information about modeling protocols built upon raw Ethernet, see “Real-Time UDP”.

Ethernet Hardware

Before you start, provide a dedicated Ethernet card on your target computer. A dedicated Ethernet card is to be used only for model-based Ethernet communications and not for communication between the development and target computers. Therefore, your target computer must have at least two Ethernet cards, one to connect the development and target computers, and one for model-based Ethernet communication. The Simulink Real-Time model-based Ethernet communication blocks support selected members of the following Intel (Vendor ID 0x8086) chip families:

- Intel 8255X
- Intel Gigabit

PCI Bus and Slot Numbers

To use the model-based Ethernet blocks, specify the PCI bus and slot number of the dedicated Ethernet card in the Real-Time Ethernet Configuration block. To identify which Ethernet card is available:

1. Boot the target computer with which you want to perform model-based Ethernet communications.
2. Examine the startup screen on the target computer. Note the PCI bus and slot information on the bottom right of the status window. This information represents the Ethernet card that is installed on the target computer for dedicated communication between the development and target computers.
3. In the MATLAB Command Window, type

   ```
   tg = slrt;
   getPCIInfo(tg, 'ethernet')
   ```

   This command determines which Ethernet cards are installed in the target computer.
4. In the list, find the Ethernet card with a bus and slot different from the bus and slot that are displayed on the target computer monitor.
5. Note the PCI bus and slot of the free Ethernet card. Use the card for model-based Ethernet communications.

MAC Addresses

Several Ethernet blocks require you to enter MAC addresses. The MAC address must be vector-based. To obtain the vector-based version of a MAC address, use the `macaddr` command. This command converts a character vector-based MAC address to a vector-based one. For example:

```
macaddr('01:23:45:67:89:ab')

[1 35 69 103 137 171]
```

When an Ethernet block requires a MAC address, you can enter either of the following in the address field:

- Command `macaddr('`*`MAC address character vector`*`')`, for example:

  ```
  macaddr('01:23:45:67:89:ab')
  ```

- Vector-based output from the `macaddr` command, for example:

  ```
  [1 35 69 103 137 171]
  ```

Network Buffer Pointers

The Simulink Real-Time Ethernet block library uses pointers to refer to network buffers. Blocks can pass pointers to these buffers as single `uint32`pointers. They can also refer to a chain of network buffer packets.

Filter Type and Filter Address Blocks

The Filter Type and Filter Address blocks accept a chain of network buffers as input. These blocks specify criteria that the drivers use while parsing each buffer on the chain. Based on these criteria, the drivers either pass the packets through the port or drop the packets. When using these blocks, create your models with filter blocks to pass data only from expected sources.

Execution Priority

The raw Ethernet blocks have the following execution priority, from first to last:

1. Real-Time Ethernet Configuration
2. The remaining raw Ethernet and network buffer library blocks

Simulink Real-Time Ethernet Block Library

To access the Simulink Real-Time Ethernet library blocks, in the Simulink Real-Time block library, double-click Ethernet. The Simulink Real-Time Ethernet library is displayed.

The Simulink Real-Time Ethernet library contains commonly used Ethernet blocks at the top level of the library. Use these blocks to create your models.

The Ethernet library also has a sublibrary, Network Buffers, which contains blocks specific to the management of Ethernet network buffers. The blocks in this sublibrary are core blocks for use in creating other subsystems. However, the top-level Ethernet blocks provide enough functionality for model-based Ethernet communications.

See Also

More About

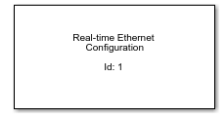
- “Real-Time Transmit and Receive over Ethernet”
- “Filtering on MAC Address”
- “Filtering on EtherType”

6

Ethernet Blocks

Real-Time Ethernet Configuration

Configure network interface for real-time raw Ethernet communication
Library: Ethernet



Description

To initialize the network and network buffers, use the Real-Time Ethernet Configuration block.

Parameters

Device

`Device ID` — Ethernet board identifier
`1-8`

From the list, select a unique number to identify the Ethernet board.

`Driver` — Drivers for chip families that this block supports
`Intel 8255X` (default) | `Intel Gigabit`

Identifies the drivers for the development computer Ethernet chip families that the block supports.

`PCI bus` — PCI bus number of Ethernet card
`0` (default) | integer

Enter the PCI bus number for the Ethernet card.

`PCI slot` — PCI slot number of Ethernet card
`0` (default) | integer

Enter the PCI slot number for the Ethernet card.

`PCI function` — PCI function number of Ethernet card
`0` (default) | integer

Enter the PCI function number for the Ethernet card.

`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.

Addressing

`Address source` — Source of MAC address
`EEPROM` (default) | `Specify`

From the list, select:

- `EEPROM` — The block gets the Ethernet card MAC address that is built into the Ethernet card.
- `Specify` — Explicitly enter a MAC address for the Ethernet card.

Dependency

To see the **MAC** parameter, select `Specify`.

`MAC` — MAC address for Ethernet card
`macaddr('00:00:00:00:00:00')` (default) | `macaddr('`*`xx:xx:xx:xx:xx:xx`*`)`

Enter the MAC address for the Ethernet card.

Dependency

To make this parameter visible, set **Address source** to `Specify`.

`Rx promiscuous` — Receive all packets regardless of their destination address
`off` (default) | `on`

To direct the model to receive all packets regardless of their destination address, select this check box.

`Multicast address list` — List of multicast address vectors
`{}` (default) | cell array

Enter a list of multicast address vectors as a cell array. The Ethernet Rx block uses these addresses and the broadcast and unicast addresses.

Advanced

`Rx bad frames` — Receive all packets, including erroneous ones
`off` (default) | `on`

To direct the model to receive all packets, including erroneous ones (such as CRC error and alignment error), select this check box.

`Rx short frames` — Receive all packets, including short ones
`off` (default) | `on`

To direct the model to receive all packets, including frames that are less than 64 bytes in length, select this check box.

The Intel Gigabit Ethernet controller does not distinguish between bad packets and short packets. Therefore, selecting either **Rx Bad Frames** or **Rx Short Frames** produces the same results for **Driver** type `Intel Gigabit`.

`Max MTU` — Maximum transmission unit number
1518 (default) | numeric

Specify a maximum transmission unit number (MTU). With this parameter, you can specify a smaller maximum transmission unit number.

`Tx threshold` — Determine when device begins DMA on packets from memory
224 (default) | numeric

Enter a value that controls when the Ethernet device begins to perform direct memory access (DMA) on packets from memory.

This parameter applies only to **Driver** type `Intel 8255X`. Before you change this parameter, see *Intel 8255x 10/100 Mbps Ethernet Controller Family — Open Source Software Developer Manual*.

`Tx buffers` — Maximum number of queued transmit buffers
128 (default) | numeric

Enter the maximum number of buffers that the driver holds in the queue before it drops new transmit requests.

The number of buffers must be a multiple of 8.

`Rx buffers` — Maximum number of queued receive buffers
64 (default) | numeric

Enter the maximum number of buffers that the driver holds in the queue before it drops new receive packets.

The number of buffers must be a multiple of 8.

`Display tuning information` — Display statistical data
`off` (default) | `on`

To enable a display of statistical data collected during the run of the model, select this check box.

See Also

External Websites

www.iso.org

Introduced in R2014b

Create Ethernet Packet

Create Ethernet packet based on the MAC address and EtherType provided
Library: Ethernet

Description

To create the Ethernet packets that you want to transfer, use the Create Ethernet Packet block.

Ports

Input

`Data` — Payload data for Ethernet packet
vector

Data Types: `uint8`

`Length` — Number of bytes in data vector
scalar

Output

`Network Buffer` — Network buffer containing packet data
scalar

The parameter is a reference to the network buffer.

Parameters

`Destination MAC` — MAC address of the target computer to receive the data
`macaddr(00:1B:21:85:37:6C)` (default) | `macaddr(`*`xx:xx:xx:xx:xx:xx`*`)`

Enter the MAC address of the target computer that receives the data.

`EtherType (use 0 for length)` — EtherType or the use of Ethernet length
`hex2dec('0000')` (default) | numeric

Enter a value that represents either the EtherType or the use of Ethernet length:

- EtherType — If you are creating Ethernet packets that use EtherType values, to specify which prototype the Ethernet frame transfers, enter a valid EtherType value.
- Ethernet length — If you are creating Ethernet packets that use Ethernet lengths, enter `0`.

See Also

Extract Ethernet Packet

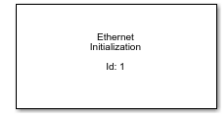
External Websites

www.iso.org

Introduced in R2008b

Ethernet Init

Initialize network card for real-time raw Ethernet communication
Library: Ethernet



Description

To initialize the Ethernet communication channel, use the Ethernet Init block. Use a separate Ethernet Init block for each Ethernet board.

Note The Ethernet Init and Buffer Mngmt blocks are combined in the Real-Time Ethernet Configuration block. For new development, use this block.

Parameters

Device

`Device ID` — Ethernet board identifier
`1-8`

From the list, select a unique number to identify the Ethernet board.

`Driver` — Drivers for chip families that this block supports
`Intel 8255X` (default) | `Intel Gigabit`

Identifies the drivers for the development computer Ethernet chip families that the block supports.

`PCI bus` — PCI bus number of Ethernet card
`0` (default) | integer

Enter the PCI bus number for the Ethernet card.

`PCI slot` — PCI slot number of Ethernet card
`0` (default) | integer

Enter the PCI slot number for the Ethernet card.

`PCI function` — PCI function number of Ethernet card
`0` (default) | integer

Enter the PCI function number for the Ethernet card.

`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.

Addressing

`Address source` — Source of MAC address
`EEPROM` (default) | `Specify`

From the list, select:

- `EEPROM` — The block gets the Ethernet card MAC address that is built into the Ethernet card.
- `Specify` — Explicitly enter a MAC address for the Ethernet card.

Dependency

To see the **MAC** parameter, select `Specify`.

`MAC` — MAC address for Ethernet card
`macaddr('00:00:00:00:00:00')` (default) | `macaddr('`*`xx:xx:xx:xx:xx:xx`*`)`

Enter the MAC address for the Ethernet card.

Dependency

To make this parameter visible, set **Address source** to `Specify`.

`Rx promiscuous` — Receive all packets regardless of their destination address
`off` (default) | `on`

To direct the model to receive all packets regardless of their destination address, select this check box.

`Multicast address list` — List of multicast address vectors
`{}` (default) | cell array

Enter a list of multicast address vectors as a cell array. The Ethernet Rx block uses these addresses and the broadcast and unicast addresses.

Advanced

`Rx bad frames` — Receive all packets, including erroneous ones
`off` (default) | `on`

To direct the model to receive all packets, including erroneous ones (such as CRC error and alignment error), select this check box.

`Rx short frames` — Receive all packets, including short ones
`off` (default) | `on`

To direct the model to receive all packets, including frames that are less than 64 bytes in length, select this check box.

The Intel Gigabit Ethernet controller does not distinguish between bad packets and short packets. Therefore, selecting either **Rx Bad Frames** or **Rx Short Frames** produces the same results for **Driver** type `Intel Gigabit`.

`Max MTU` — Maximum transmission unit number
1518 (default) | numeric

Specify a maximum transmission unit number (MTU). With this parameter, you can specify a smaller maximum transmission unit number.

`Tx threshold` — Determine when device begins DMA on packets from memory
224 (default) | numeric

Enter a value that controls when the Ethernet device begins to perform direct memory access (DMA) on packets from memory.

This parameter applies only to **Driver** type `Intel 8255X`. Before you change this parameter, see *Intel 8255x 10/100 Mbps Ethernet Controller Family — Open Source Software Developer Manual*.

`Tx buffers` — Maximum number of queued transmit buffers
128 (default) | numeric

Enter the maximum number of buffers that the driver holds in the queue before it drops new transmit requests.

The number of buffers must be a multiple of 8.

`Rx buffers` — Maximum number of queued receive buffers
64 (default) | numeric

Enter the maximum number of buffers that the driver holds in the queue before it drops new receive packets.

The number of buffers must be a multiple of 8.

`Display tuning information` — Display statistical data
`off` (default) | `on`

To enable a display of statistical data collected during the run of the model, select this check box.

See Also

Real-time Ethernet Configuration

Topics

“Real-Time Transmit and Receive over Ethernet”

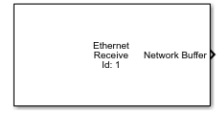
External Websites

www.iso.org

Introduced in R2008b

Ethernet Rx

Receive data over Ethernet network
Library: Ethernet



Description

To receive Ethernet packets and to filter on the received packets, use the Ethernet Rx block. You can filter packets by EtherType or length. You can use multiple Ethernet Rx blocks with the same device ID. However, you must configure each block to filter a unique set of packets.

Ports

Output

`Network Buffer` — Network buffer containing packet data
scalar

The parameter is a reference to the network buffer.

Parameters

Rx

`Device ID` — Ethernet board identifier
1-8

From the list, select a unique number to identify the Ethernet board. Select the same **Device ID** as the ID that you selected for the Real-Time Ethernet Configuration block.

`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.

Filter

`Filter criteria` — Filter on EtherTypes or Ethernet lengths
`Receive all unmatched types [0 to 65535]` (default) | `Receive unmatched lengths [0 to 1500]` | `Receive unmatched EtherTypes [150 to 65535]` | `Specify types to match`

From the list, select how you want to filter on EtherTypes (Ethernet II framing standard) or Ethernet lengths (IEEE 802.3 framing standard).

- `Receive all unmatched types [0 to 65535]` — Output all unmatched packets, both Ethernet II framing and IEEE 802.3 framing standards.
- `Receive unmatched lengths [0 to 1500]` — Output all packets with IEEE 802.3 framing standard.
- `Receive unmatched EtherTypes [150 to 65535]` — Output all output packets with Ethernet II framing standard.
- `Specify types to match` — Explicitly enter the EtherTypes to output.

Dependency

To see the **Receive these types (vector of types 0-65535)** parameter, select `Specify types to match`.

`Receive these types (vector of types 0–65535)` — EtherTypes to output
`[hex2dec('0000')]` (default) | vector

Enter a vector of EtherTypes that you want to output.

Dependency

To make this parameter visible, set **Filter criteria** to `Specify types to match`.

See Also

Ethernet Tx

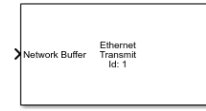
External Websites

www.iso.org

Introduced in R2008b

Ethernet Tx

Transmit data over Ethernet network
Library: Ethernet



Description

To send network packets, use the Ethernet Tx block.

Ports

Input

`Network Buffer` — Network buffer containing packet data
scalar

The parameter is a reference to the network buffer.

Parameters

`Device ID` — Ethernet board identifier
1-8

From the list, select a unique number to identify the Ethernet board. Select the same **Device ID** as the ID that you selected for the Real-Time Ethernet Configuration block.

`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.

See Also

Ethernet Rx

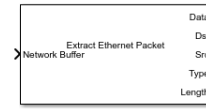
External Websites

www.iso.org

Introduced in R2008b

Extract Ethernet Packet

Extract data from Ethernet packet
Library: Ethernet



Description

To extract data from an Ethernet packet, use the Extract Ethernet Packet block.

Ports

Input

`Network Buffer` — Network buffer containing packet data
scalar

The parameter is a reference to the network buffer.

Output

`Data` — Packet payload data
vector

Data Types: `uint8`

`Dst` — Ethernet address of packet destination
`'xxx.xxx.xxx.xxx'`

`Src` — Ethernet address of packet source
`'xxx.xxx.xxx.xxx'`

`Type` — EtherType of data
scalar

`Length` — Number of bytes in data vector
numeric

Parameters

`Data Size` — Number of bytes to extract
`1500` (default) | numeric

Enter the data size (in bytes) for the data that you want to extract from an Ethernet packet.

See Also

Create Ethernet Packet

External Websites

www.iso.org

Introduced in R2008b

Filter Address

Filter Ethernet packets based on MAC address
Library: Ethernet

Description

To filter network buffer packets by their MAC addresses, use the Filter Address block. See “Filter Type and Filter Address Blocks” on page 5-4 for cautions on setting the parameters for this block.

Ports

Input

`Network Buffer` — Network buffer containing packet data
scalar

The parameter is a reference to the network buffer.

Output

`Match [Addr (#)]` — Network buffer containing packets that match filter
vector

If you specify one MAC address, one port appears with the name `Match`. The block directs packets with this MAC address to port `Match`.

If you specify more than one MAC address, multiple ports appear, matched to the values in parameter **MAC Address**. The block directs packets with the first address to `Match Addr (1)`, with the second address to `Match Addr (2)`, and so on.

Dependency

The port count depends on how many MAC addresses appear in parameter **MAC Address**.

`Remainder` — Network buffer chain containing packets that do not match filter
vector of network buffers

Packets that do not meet the filter criteria appear at this port.

Dependency

To activate this port, clear **Drop non-matches**.

Parameters

`MAC Address` — MAC addresses to filter
`{[macaddr('00:00:00:00:00:00')]}` (default) | cell array

Enter a cell array that contains the MAC addresses for the filter.

`Drop non-matches` — Discard packets that do not match filter criteria
`'off'` (default) | `'on'`

To discard packets that do not match the filter criteria, select this parameter.

To output packets that do not match the filter criteria, clear this parameter.

`Filter on destination address` — Filter packets that match destination address
`'off'` (default) | `'on'`

To filter addresses for the source address, clear this check box (default).

To filter addresses for the destination address, select this check box.

See Also
Filter Type

Topics

“Filter Type and Filter Address Blocks” on page 5-4

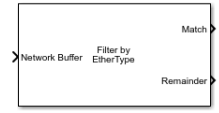
External Websites

www.iso.org

Introduced in R2008b

Filter Type

Filter Ethernet packets based on EtherType
Library: Ethernet



Description

To filter network buffer packets by their EtherType values, use the Filter Type block. See “Filter Type and Filter Address Blocks” on page 5-4 for cautions on setting the parameters for this block.

Ports

Input

`Network Buffer` — Network buffer containing packet data

scalar

The parameter is a reference to the network buffer.

Output

`Match Length` — Network buffer chain containing packets that match length filter

vector of network buffers

One port receives packets with EtherType values within `1–1500`.

Dependency

To activate this port, select parameter **Match Length (1-1500)**.

`Match [Type (#)]` — Network buffer chain containing packets that match filter
vector of network buffers

If you specify one Ethertype, one port appears with the name `Match`. The block directs packets with this EtherType to port `Match`.

If you specify more than one EtherType, multiple ports appear, matched to the values in parameter **EtherType**. The block directs packets with the first **EtherType** to `Match Type (1)`, with the second **EtherType** to `Match Type (2)`, and so on.

Dependency

The port count depends on how many EtherTypes appear in parameter **EtherType**.

`Remainder` — Network buffer chain containing packets that do not match filter
vector of network buffers

Packets that do not meet the filter criteria appear at this port.

Dependency

To activate this port, clear **Drop non-matches**.

Parameters

`Match Length (1-1500)` — Match packets with EtherType values within 1–1500
`'on'` (default) | `'off'`

To match packets whose **EtherType** values fall within the range `1–1500`, select this check box.

EtherType — EtherTypes on which to filter
`[hex2dec('0000')]` (default) | vector

Enter a vector of EtherTypes on which you want to filter.

`Drop non-matches` — Discard packets that do not match filter criteria
`'off'` (default) | `'on'`

To discard packets that do not match the filter criteria, select this parameter.

To output packets that do not match the filter criteria, clear this parameter.

See Also

Filter Type

Topics

“Filter Type and Filter Address Blocks” on page 5-4

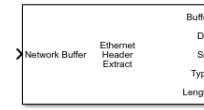
External Websites

www.iso.org

Introduced in R2008b

Header Extract

Extract header data from Ethernet packet
Library: Ethernet



Description

To extract the header data of network buffer packets, use the Header Extract block.

Ports

Input

`Network Buffer` — Network buffer containing packet data
scalar

The parameter is a reference to the network buffer.

Output

`Data` — Packet payload data
vector

Data Types: `uint8`

`Dst` — Ethernet address of packet destination
`'xxx.xxx.xxx.xxx'`

`Src` — Ethernet address of packet source
`'xxx.xxx.xxx.xxx'`

`Type` — EtherType of data
scalar

Length — Number of bytes in data vector
numeric

See Also

Extract Ethernet Packet

External Websites

www.iso.org

Introduced in R2008b

7

Network Buffer Library for Model-Based Ethernet Communications Support

Network Buffer Blocks

The Ethernet library includes a sublibrary, Network Buffers, that contains blocks for managing Ethernet network buffers. The blocks in this sublibrary are core blocks that you can use to create other subsystems.

The Ethernet drivers use a set of buffers, *Ethernet network buffers*, that it uses to store data that is sent and received over the network. The block organizes these buffers into several pools, each with different values of maximum data size. The buffers include information about the data itself. The block allocates the buffer pools during initialization and does not change the buffer pools during run time. When the block sends, receives, or processes data, it allocates a buffer. When the operation is done, it frees the buffer.

You can control the number of buffers allocated for each allowable value of data size by using the Buffer Mngmt block parameter **Buffer pool sizes**. Allocate enough buffers for the maximum number of data packets that you anticipate receiving, sending, or processing at one time. You can send and receive more data by allocating many more buffers. However, each allocation reserves more memory, which you cannot then use for other purposes. Running out of buffers means that data cannot be sent and received until the block frees allocated buffers.

Monitor the buffer pool statistics at run time to find the optimal values that an application requires. To monitor the buffer pool statistics, select the **Display tuning information** check box in the Buffer Mngmt block parameters dialog box.

See Also

Buffer Mngmt

Network Buffer Library Blocks

Buffer Mngmt

Initialize network buffer pools

Library

Simulink Real-Time Library for Ethernet

Description

To initialize network buffers, use the Buffer Mngmt block.

Block Parameters

This block has two tabs, **Main** and **Advanced**.

Main

Sample time

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

Advanced

Modify the values of the parameters in this tab only if you have a thorough understanding of the Ethernet protocol. Changing the values of these parameters can change the behavior of your system.

Buffer pool sizes (256, 512, 1024, 2048)

Enter a vector of the number of buffers for each pool size (`256`, `512`, `1024`, or `2048`).

Display tuning information

Select this check box to enable a display of statistical data collected during the run of the model.

See Also

Topics

“Network Buffer Blocks” on page 7-2

External Websites

www.iso.org

Introduced in R2008b

Chain Size

Determine the number of network buffers in the chain

Library

Simulink Real-Time Library for Ethernet

Description

To determine the number of buffers that are on the chain, use the Chain Size block.

See Also

Topics

“Network Buffer Blocks” on page 7-2

External Websites

www.iso.org

Introduced in R2008b

Compose

Create a network buffer from raw input data

Library

Simulink Real-Time Library for Ethernet

Description

To create a network buffer, use the Compose block. This block creates a pointer to a network buffer.

See Also

Topics

“Network Buffer Blocks” on page 7-2

External Websites

www.iso.org

Introduced in R2008b

Extract

Extract raw data from network buffer

Library

Simulink Real-Time Library for Ethernet

Description

To extract network buffer packets, use the Extract block.

Block Parameters

Packet size (-1: inherit)

Enter the packet size for the network buffer packet to extract. Enter `-1` (default) to inherit the packet size.

See Also

Topics

“Network Buffer Blocks” on page 7-2

External Websites

www.iso.org

Introduced in R2008b

Link

Link vector of network buffers into a chain

Library

Simulink Real-Time Library for Ethernet

Description

To convert a vector of network buffer signals into a linked list of signals, use the Link block.

See Also

Topics

“Network Buffer Blocks” on page 7-2

External Websites

www.iso.org

Introduced in R2008b

Manage

Output or buffer packets as indicated by the parameters

Library

Simulink Real-Time Library for Ethernet

Description

Output or buffer packets as indicated by the parameters.

Block Parameters

Chain size

Specify the queuing (output) behavior of the block as packets are received.

Value	Description
`inf`	Output all packets. No queuing occurs.
`0`	Delete all packets, no packets pass through.
Positive number, *C*	Pass through the first *C* packets
Negative number, *C*	Pass through the last *C* packets

Buffer size

Specify the buffering behavior of the block as packets are received.

Value	Description
`inf`	Buffer all remaining packets. Delete no packets.
`0`	Do not buffer packets. Delete all remaining packets.

Value	Description
Positive number, *B*	Buffer the remaining first *B* packets.
Negative number, *B*	Buffer the remaining last *B* packets.

Threshold

Enter a minimum threshold before which this block begins to output buffers.

Value	Description
`0`	Specifies no threshold.
Negative number,*T*	Delay passing buffer packets until *T* packets are buffered.
Positive number, *T*	Passes buffer packets if *T* packets are buffered.

See Also

Topics

“Network Buffer Blocks” on page 7-2

External Websites

www.iso.org

Introduced in R2015a

Merge

Merge the incoming network buffer chains into one

Library

Simulink Real-Time Library for Ethernet

Description

To combine signal pointers to a linked list, use the Merge block.

Block Parameters

Number of inputs

Enter the number of network buffer signal pointers to combine into a linked list.

See Also

Topics

“Network Buffer Blocks” on page 7-2

External Websites

www.iso.org

Introduced in R2008b

Split

Split a network buffer chain

Library

Simulink Real-Time Library for Ethernet

Description

To separate a linked list of buffer pointers into separate individual pointers, use the Split block.

Block Parameters

Number of outputs

Enter the number of pointers the input linked list should be separated into.

- If the number of buffers is the same as this value, this block splits them and outputs them in the order they appear in the vector, or in reverse order (depending on the setting of the **Split in reverse order** parameter).
- If the number of buffers is less than **Number of outputs**, the block outputs zeros (0s) for the extra output ports.
- If the number of buffers is greater than **Number of outputs**, the block either deletes the extra buffers, or chains the remaining buffers together (depending on the setting of the **Allow chaining for last signal** parameter).

Split in reverse order

Select this check box to split out the network buffers in the reverse order in which they are received.

Allow chaining for last signal

Select this check box to chain together remaining network buffers. There might be remaining buffers if the incoming linked list contains more buffers than the number in **Number of outputs**.

Clear this check box to delete the remaining buffers.

See Also

Topics

“Network Buffer Blocks” on page 7-2

External Websites

www.iso.org

Introduced in R2008b

Unlink

Unlink a chain into a vector of network buffers

Library

Simulink Real-Time Library for Ethernet

Description

To convert a linked list of signals into a vector of network buffer signal, use the Unlink block.

Block Parameters

Vector length (-1: inherit)

Enter the number of signals in the linked list of signals that you want to separate. Enter `-1` (default) to inherit the vector length.

See Also

Topics

“Network Buffer Blocks” on page 7-2

External Websites

www.iso.org

Introduced in R2008b

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 for 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 using the blocks in the EtherCAT sublibrary. However, you can use these blocks to prototype multiple EtherCAT networks with multiple Ethernet cards.

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

The Beckhoff ET9000 configurator is no longer supported.

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

- “EtherCAT Protocol Mapping” on page 9-24
- “EtherCAT Configurator Component Mapping” on page 9-25

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 with 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 using Acontis EC-Engineer, the software creates one default task linked to all slave device input/output channels.

When you know the input/output cycle ticks, set the **Fixed-step size** in the Model Configuration Parameters dialog box to a value that is consistent with the following 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 9-23.

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 Upload and SDO Download blocks to your model. The SDO blocks come in two types, synchronous and asynchronous. From the EtherCAT perspective, there is little difference in behavior of these types. The difference arises during the execution of the real-time application. The Sync SDO blocks halt execution while they wait for a response. The Async SDO blocks continue executing and poll the I/O module for a response.

To avoid a CPU overload, set the sample time for the synchronous SDO blocks to a value at least three times 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 two phases:

1. *Phase 1* — Reads data from EtherCAT variables from the last received frame into EtherCAT PDO Receive blocks.
2. Either of the following blocks, in arbitrary 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. Each of the following blocks, in arbitrary order:
   - 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.

  Synchronous upload and download take at least three ticks of the fastest PDO cycle tick to complete processing.
- EtherCAT Get State — Reads current state of EtherCAT network.
- EtherCAT Set State — Queues request to change current state of EtherCAT network.

4 *Phase 2* — Sends the PDO frames, followed by the next available queued SDO frames.

See Also

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

More About

- “Fixed-Step Size Derivation” on page 9-23
- “EtherCAT Protocol Mapping” on page 9-24
- “EtherCAT Configurator Component Mapping” on page 9-25
- “Ethernet Chip Sets Compatible with EtherCAT” on page 9-26
- “EtherCAT Data Types” on page 9-29
- “EtherCAT Init Block DC Error Values” on page 9-30

Install TwinCAT 3

To install the EtherCAT network and configuration software, execute the following steps. For requirements, see “Hardware Setup Requirements for TwinCAT 3” on page 9-6.

1. Install a dedicated, EtherCAT compatible Ethernet card on the development computer.
2. Download or purchase the Beckhoff TwinCAT 3 configurator (`www.beckhoff.com`).

   The Beckhoff ET9000 configurator is no longer supported.
3. Install Microsoft® Visual Studio® on your development computer.

   TwinCAT 3 uses the Microsoft Visual Studio IDE desktop as its user interface. See the TwinCAT 3 documentation for the required version.
4. Install the TwinCAT 3 software on your development computer.

The next task is “Configure EtherCAT Network with TwinCAT 3” on page 9-7.

See Also

External Websites

- www.beckhoff.com
- www.acontis.com/eng

Hardware Setup Requirements for TwinCAT 3

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

- Each Ethernet card must be compatible with EtherCAT communication. For a list of supported chip sets, see “Ethernet Chip Sets Compatible with EtherCAT” on page 9-26.
- To keep non EtherCAT traffic from interfering with the protocol timing, assign each Ethernet card a static IP address and a nonroutable subnet.

  For information on setting up the dedicated Ethernet card, see your network administrator.
- On the target computer, install two Ethernet cards. Dedicate one card to linking the development and target computers. Dedicate the other to model-based 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 to EtherCAT network configuration.
- Configure the development computer Ethernet card that you are using for EtherCAT to enable only the Internet Protocol Version 4 (TCP/IPv4) driver. See the TwinCAT 3 documentation for information on manually creating an EtherCAT configuration file.

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 with TwinCAT 3

To configure the EtherCAT network using TwinCAT 3, execute the following steps.

This procedure assumes that you are familiar with TwinCAT 3 and its documentation.

Before configuring the network, carry out the steps in “Install TwinCAT 3” on page 9-5.

Scan EtherCAT Network

The rest of this example assumes that your EtherCAT network consists of Beckhoff `EK1100`, `EL3062`, and `EL4002` modules connected in that order, followed by a terminator.

To scan an EtherCAT network with 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, check that the scanner downloaded the required information about your EtherCAT devices.

Configure EtherCAT Master Node Data

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

To configure the master node, execute the following steps.

Create EtherCAT Task

To create and configure an EtherCATtask:

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

   Provide a name for the task, for example `Task 1` and configure `Task 1` as a task with image.
2. In the task list, select `Task 1` and set its cycle ticks value to a value in milliseconds, such as `10` for `10` milliseconds.
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 allow 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`.

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`.

Export and Save EtherCAT Configuration with 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 withTwinCAT 3.

To export the ENI file from TwinCAT 3, execute the following steps.

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. Save the Microsoft Visual Studio TwinCAT project file.

   In the file save dialog box, enter an SLN file name, such as `BeckhoffAIOconfig`.

To review or modify your configuration, open the project SLN file 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 9-10.

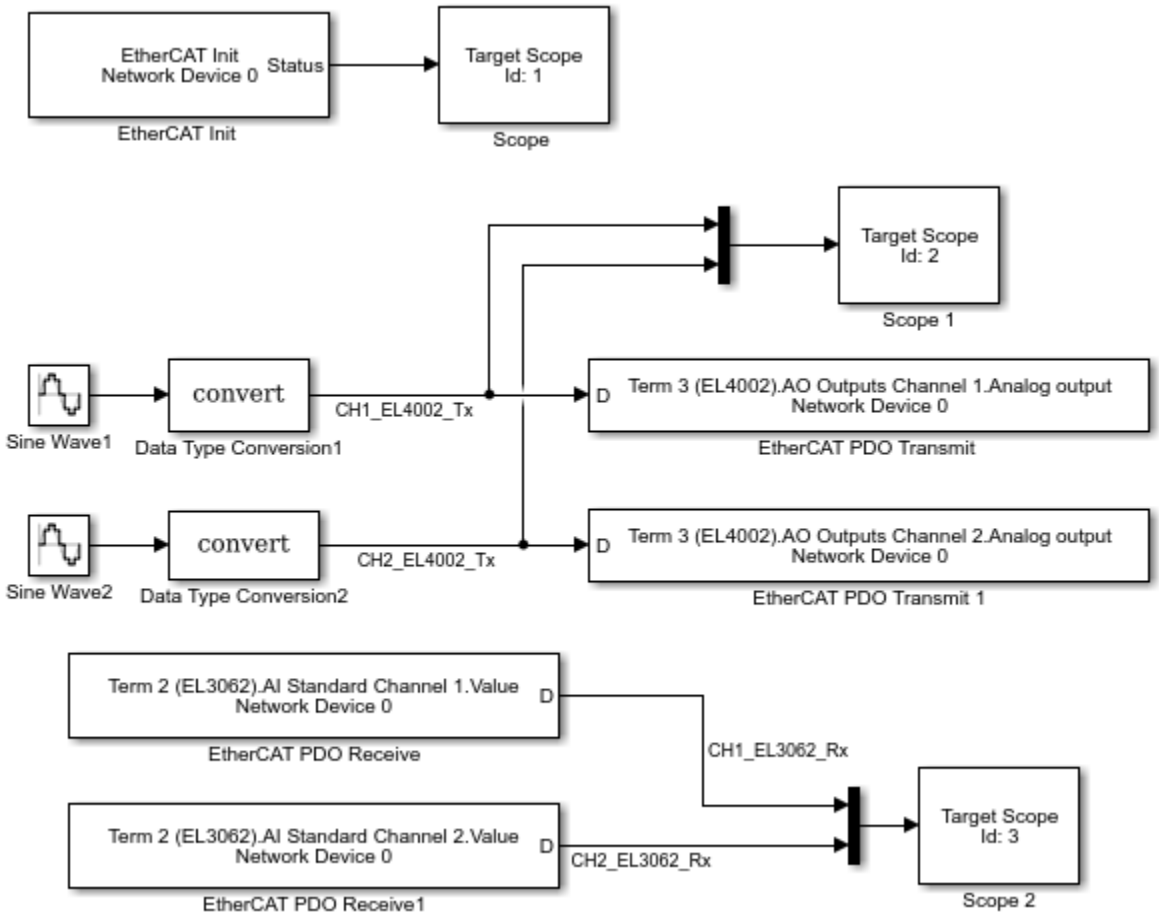
Install EtherCAT Network for Execution

To install the EtherCAT Network for execution using the target computer as master node, execute the following steps. For requirements, see “Hardware Setup Requirements for TwinCAT 3” on page 9-6.

1. Record the PCI bus and PCI slot for the existing Ethernet cards in the target computer. For more on identifying and selecting Ethernet cards for linking the development and target computers, see “Ethernet Card Selection by Index”.
2. Install a dedicated, EtherCAT compatible Ethernet card on the target computer.
3. Record the PCI bus and PCI slot for the new Ethernet card. Check the PCI bus and PCI slot for the existing card. To select the Ethernet card required for the Ethernet link, update the Simulink Real-Time environment settings.
4. 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 9-11.

Configure EtherCAT Master Node Model



Before configuring the model, carry out the steps in “Configure EtherCAT Network with TwinCAT 3” on page 9-7.

To configure model `xpcEthercatBeckhoffAIO` for execution using the target computer as master node, execute the following steps.

Configure EtherCAT Init Block

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

This procedure assumes that you are familiar with TwinCAT 3 and its documentation.

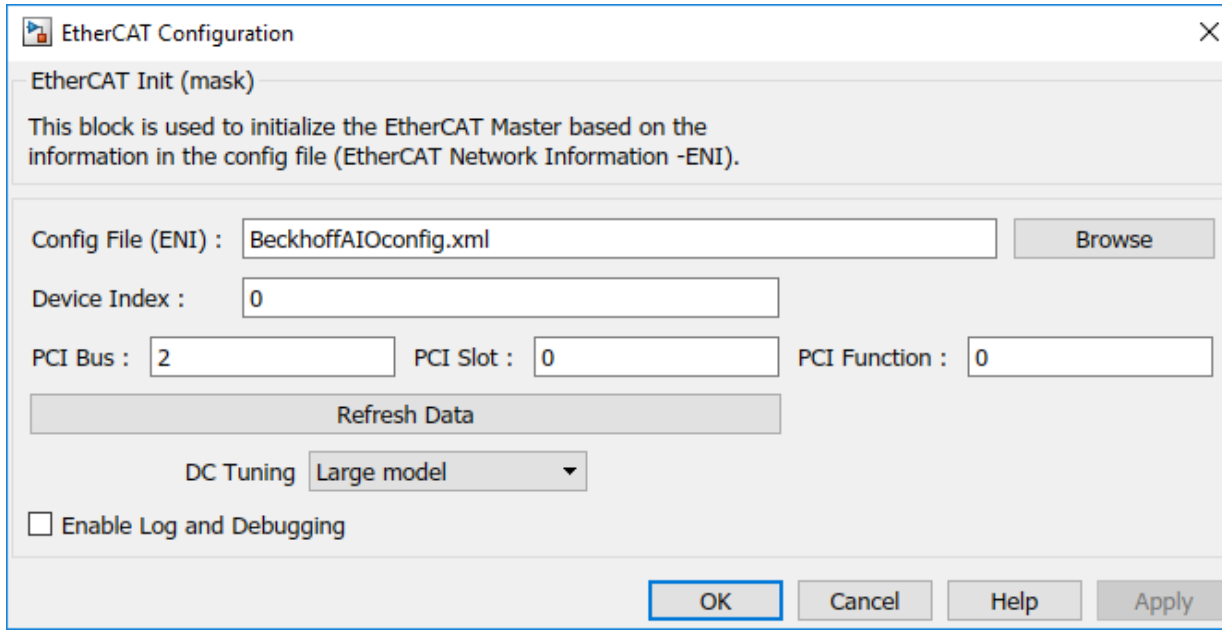
As part of the configuration process, create and save an EtherCAT Network Information (ENI) file. See “Configure EtherCAT Network with TwinCAT 3” on page 9-7.

If you configure EtherCAT distributed clocks in master shift mode, using the IEEE 1588 Sync Execution block in the same model produces a build error. To include EtherCAT distributed clocks and IEEE 1588 synchronized execution in the same model, use EtherCAT bus shift mode.

To configure the EtherCAT Init block of model `xpcEthercatBeckhoffAIO`, execute the following steps.

1. Open model `xpcEthercatBeckhoffAIO`.
2. Double-click the EtherCAT Init block.
3. At 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. Take 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 **PCI bus** and **PCI slot** for the EtherCAT port that you are connecting to your EtherCAT network. See “Install EtherCAT Network for Execution” on page 9-10.
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

To configure the EtherCAT PDO Receive blocks of model `xpcEthercatBeckhoffAIO`, execute the following steps. You must have selected a valid ENI file in the EtherCAT Init block.

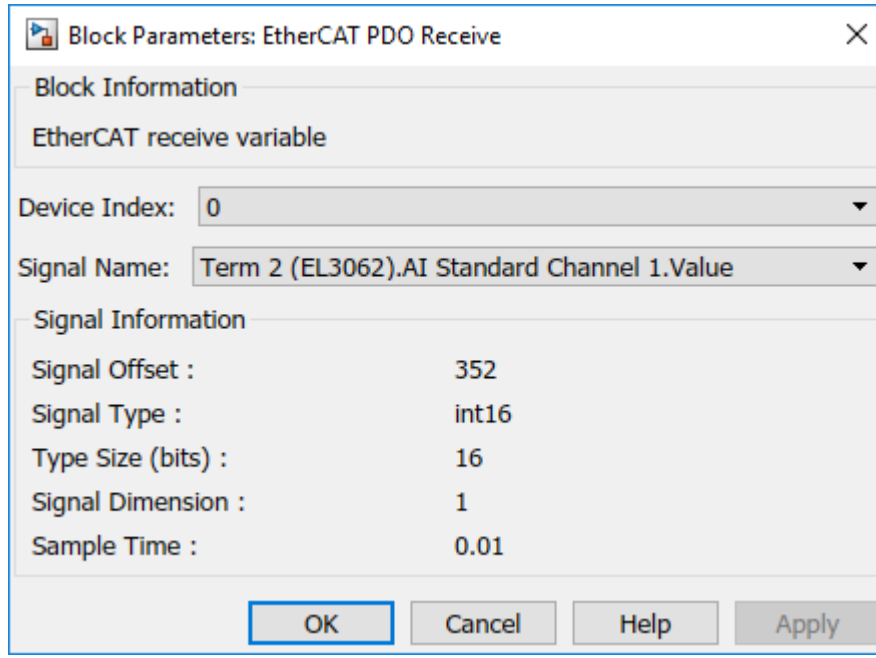
This procedure assumes that you are familiar with TwinCAT 3 and its documentation.

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 EtherCAT network being accessed, here `Term 2 (EL3062).AI Standard Channel 1.Value`.

4 Note the value of parameter **Sample Time**, which is in seconds.



5 Click **OK**.

Execute steps 5–9 for the EtherCAT PDO Receive block labeled `EtherCAT PDO Receive 1`.

Configure EtherCAT PDO Transmit Blocks

To configure the EtherCAT PDO Transmit blocks of model `xpcEthercatBeckhoffAIO`, execute the following steps. You must have selected a valid ENI file in the EtherCAT Init block.

This procedure assumes that you are familiar with TwinCAT 3 and its documentation.

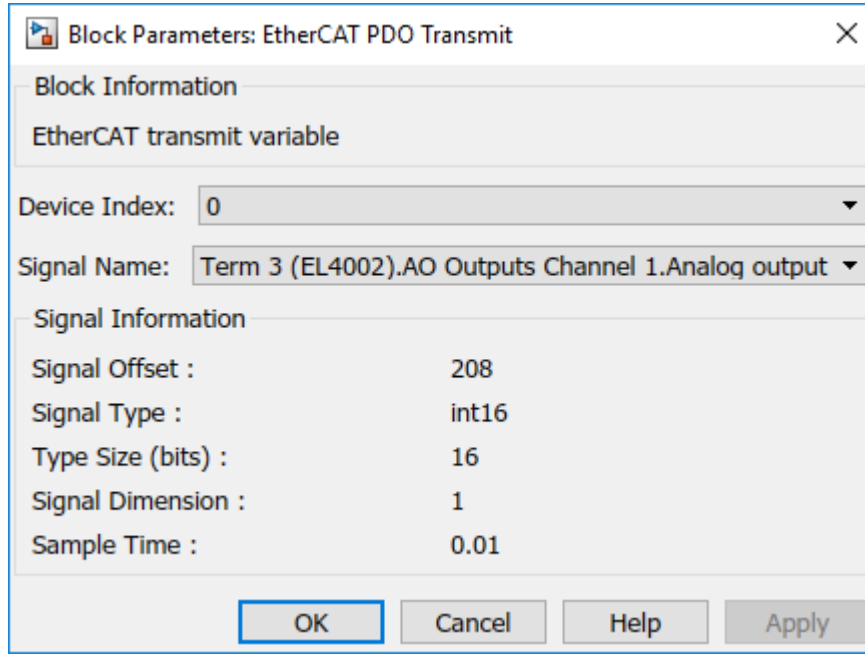
1 Open model `xpcEthercatBeckhoffAIO`.

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** value consistent with the EtherCAT network being accessed, here `Term 3 (EL4002).AO Outputs Channel 1.Analog output`.

5 Note the value of parameter **Sample Time**, which is in seconds.



6 Click **OK**.

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

Configure EtherCAT Model Configuration Parameters

To configure the configuration parameters for model `xpcEthercatBeckhoffAIO`, execute the following steps. You must have selected a valid ENI file in the EtherCAT Init block. For more information, see “Fixed-Step Size Derivation” on page 9-23.

1 Open model `xpcEthercatBeckhoffAIO`.

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`.

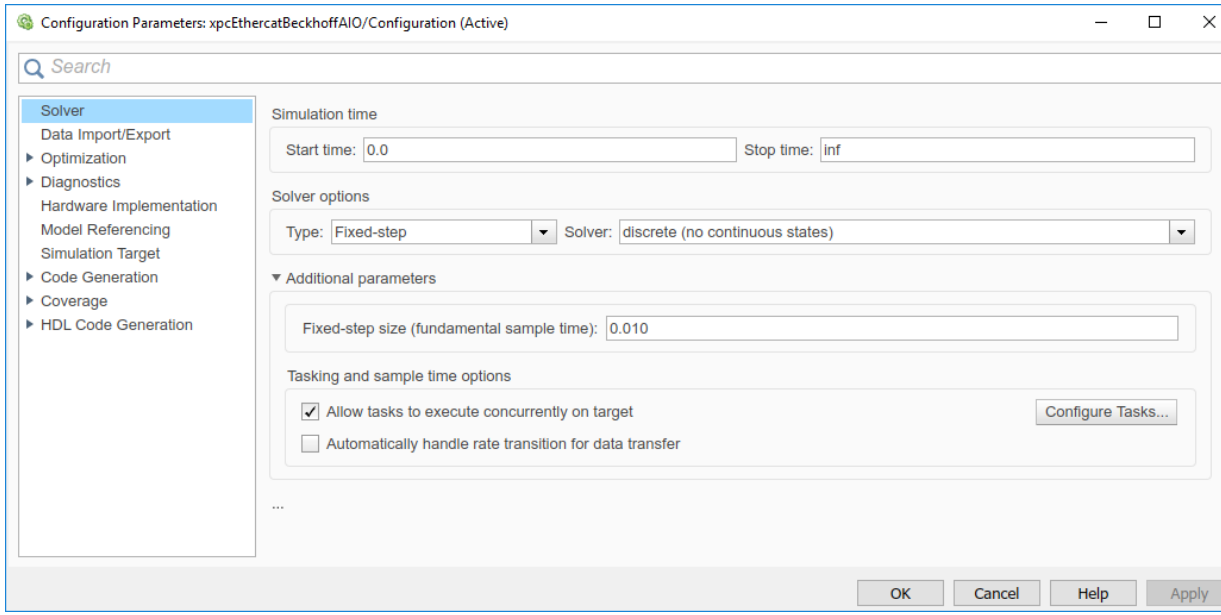
3 In the Model Editor, click **Simulation** > **Model Configuration Parameters** and click the **Solver** tab.

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 it to `0.010`.

The model configuration parameters dialog box looks like this figure.



5 Click **OK**.

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

EtherCAT Distributed Clock Algorithm

In this section...

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, 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.

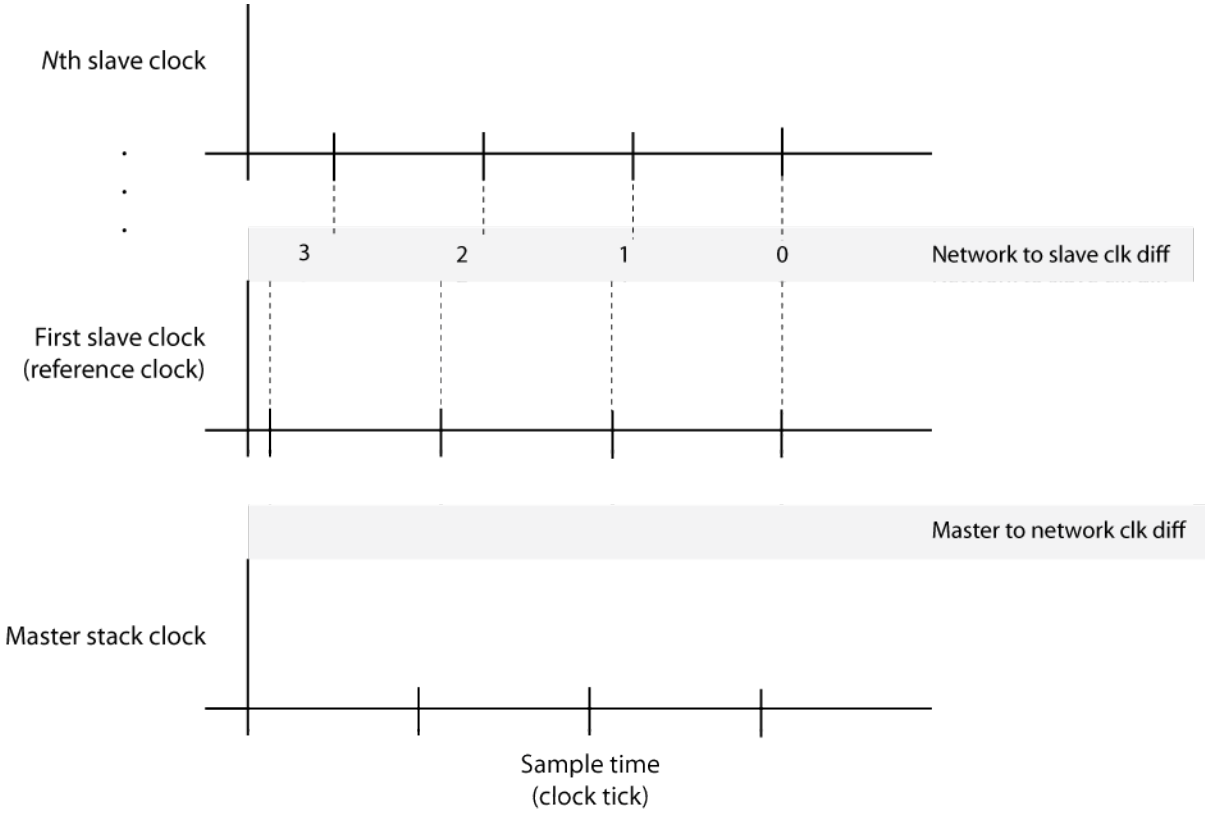
The DC algorithm operates in two phases. In phase 1, the algorithm aligns the clocks of DC-enabled network nodes other than the master node with the clock of the first DC-enabled slave node. In phase 2, the algorithm aligns the remaining unaligned clock with the reference clock.

Do not manually adjust the sample time of the real-time application in either master shift mode or bus shift mode.

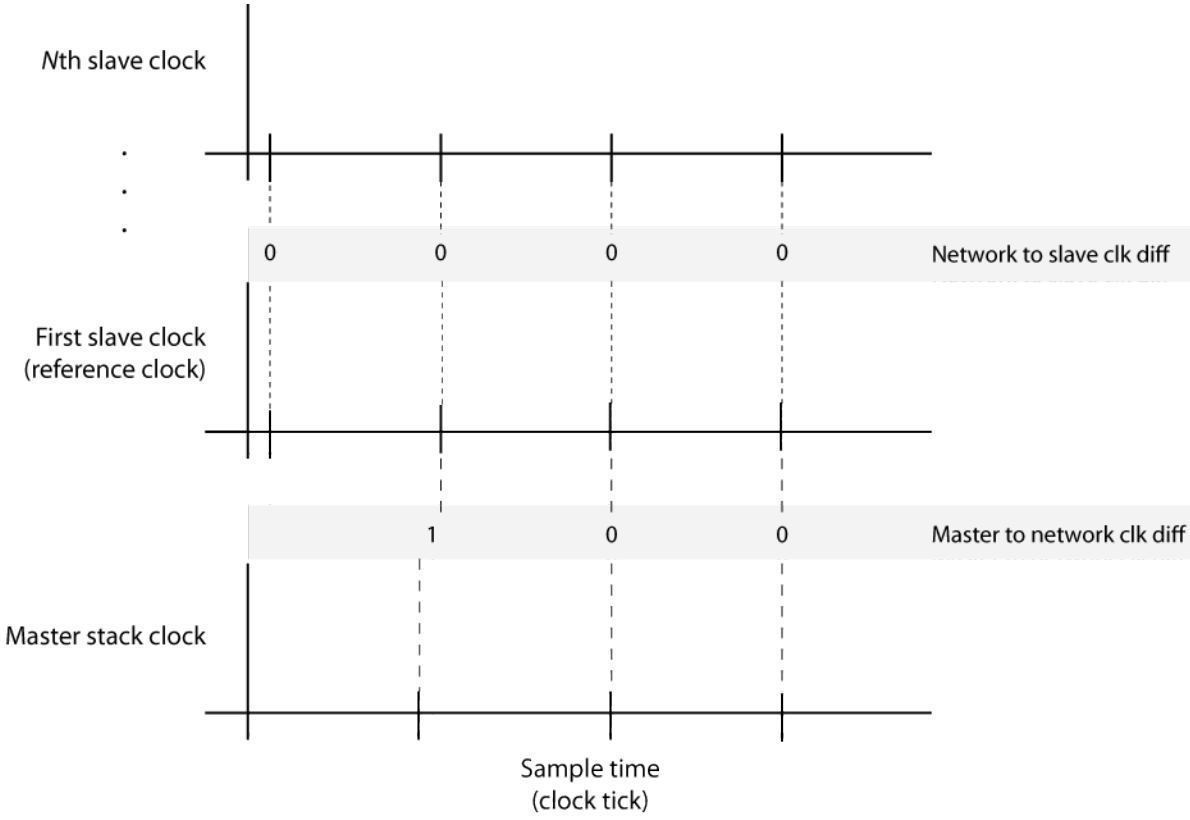
Master Shift Mode

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.



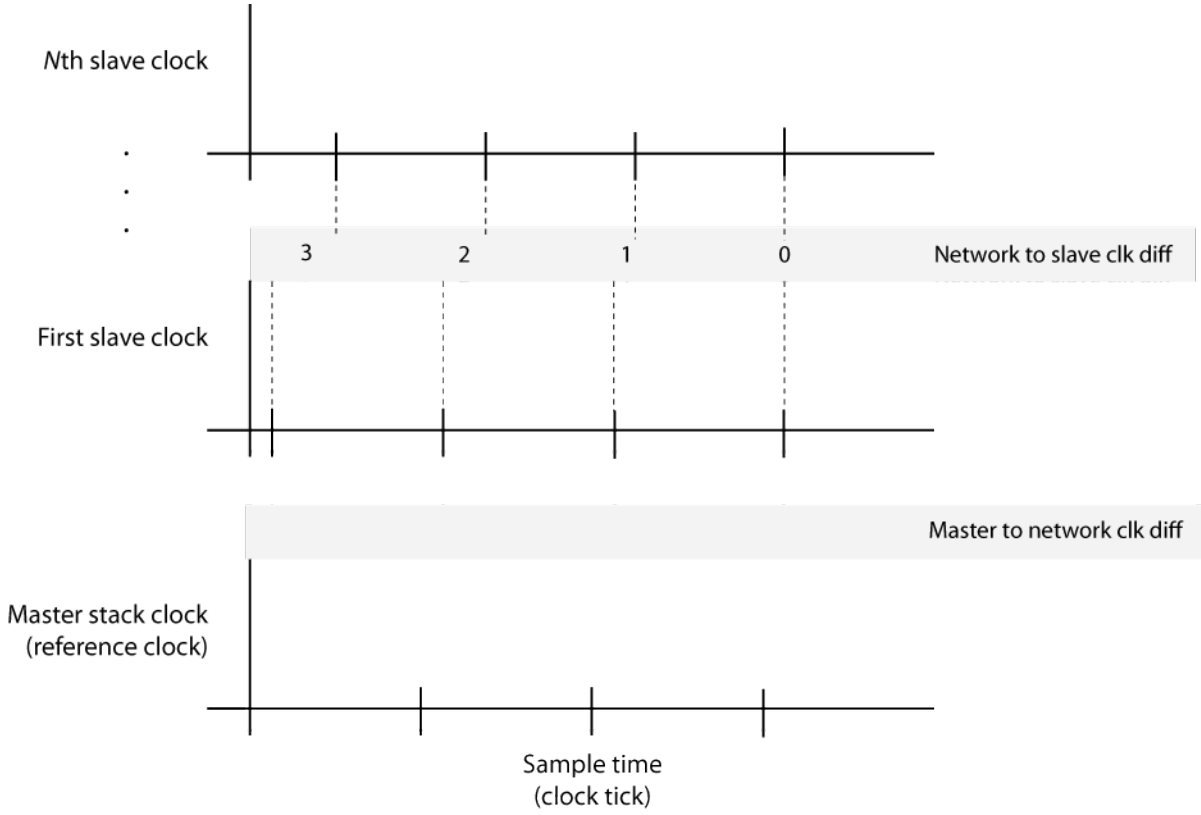
In phase 2, the algorithm shifts the sample time of the master stack running on the target computer to align with the first slave node clock. In that process, the EtherCAT Init block output value `MasterToNetworkClkDiff` decreases to near zero.



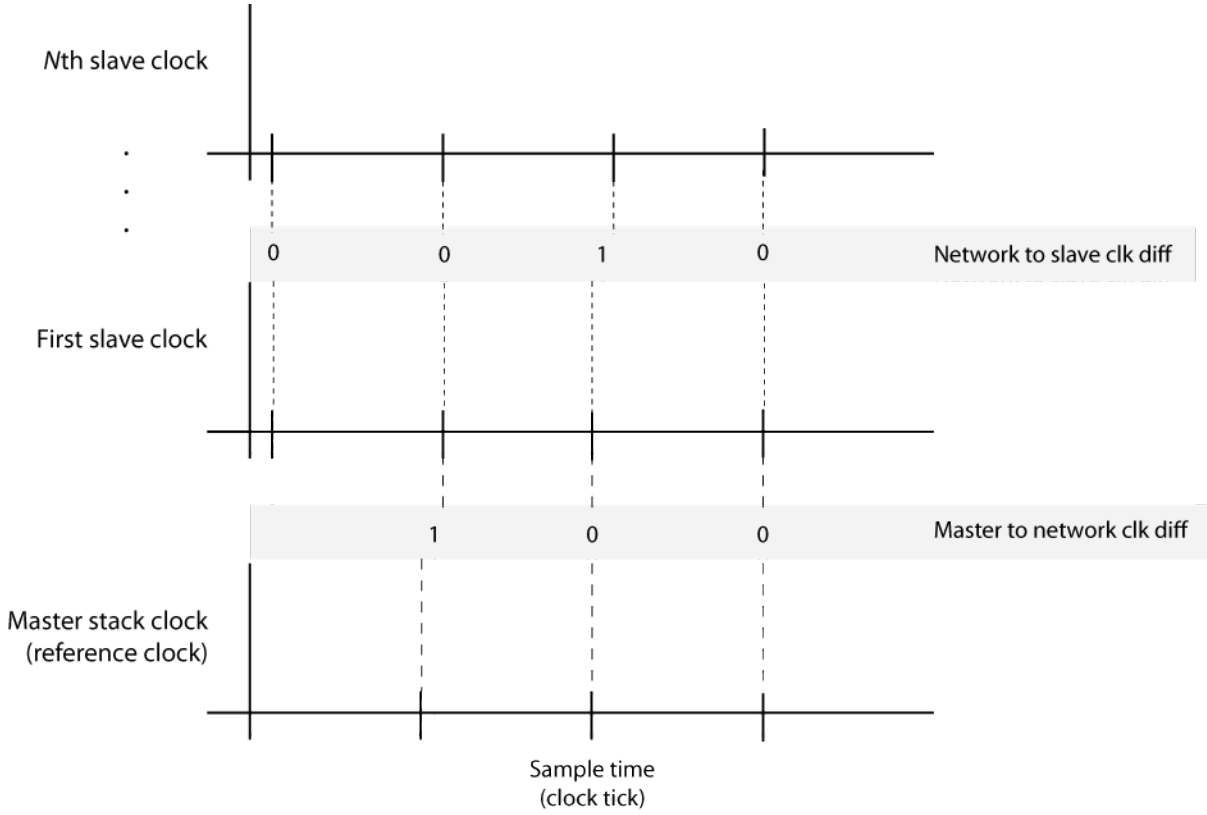
Bus Shift Mode

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.



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` first increases, then decreases to near zero.



Limitations

If you configure EtherCAT distributed clocks in master shift mode, using the IEEE 1588 Sync Execution block in the same model produces a build error. To include EtherCAT distributed clocks and IEEE 1588 synchronized execution in the same model, use EtherCAT bus shift mode.

See Also

EtherCAT Init

More About

- “EtherCAT Init Block DC Error Values” on page 9-30

Fixed-Step Size Derivation

To configure the sample time for an EtherCAT model, set the fixed-step size for the entire model in the model Configuration Parameters **Solver** pane. You can also specify the sample times for key blocks.

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 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]. Then 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, from the Simulink **Display** menu, set **Sample Time > Colors** to reveal 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 ignores some others.

Overlay Protocol	Protocol Description	Support Type	Means of Support
CANopen over EtherCAT (CoE)	Implements CAN functionality using EtherCAT	Direct	Model CoE 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	Ignored	Update the EtherCAT slave ROM with TwinCAT 3.
Functional Safety over EtherCAT (FSoE)	Sends asynchronous ‘safety’ messages over the network.	Ignored	
Servo over EtherCAT (SoE)	Wraps vendor-specific servo commands in a common protocol.	Ignored	

EtherCAT Configurator Component Mapping

The following 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 linked to a variable in a task	All PDO variables included in default task, with no linking required.	EtherCAT PDO Receive **Signal Name**
Device variables in Process Image entity	EtherCAT PDO Receive or EtherCAT PDO Transmit block	EtherCAT PDO Receive or EtherCAT PDO Transmit block

See Also

EtherCAT PDO Receive | EtherCAT PDO Transmit

More About

- “EtherCAT Data Types” on page 9-29

Ethernet Chip Sets Compatible with EtherCAT

For target computer Ethernet chipsets, the Simulink Real-Time EtherCAT blocks support the Intel Gigabit Ethernet and Intel Fast Ethernet (10/100) chip sets. The Intel vendor ID is 0x8086.

Intel Gigabit Ethernet Chip Sets

Device ID	Chip Number	Chip Description
0x100E	02000	Intel PRO/1000 MT Desktop Adapter
0x1010	82546EB	Intel PRO/1000 MT Dual Port Server Adapter
0x1013	82541EI	Gigabit Ethernet Controller (Copper)
0x1019	82547EI	Gigabit Ethernet Controller (LOM)
0x1026	82545EP	Gigabit Ethernet Controller
0x104A	82566DM	Intel 82566DM Gigabit Ethernet
0x104D	82566MC	Intel 82566MC Gigabit Ethernet
0x105E	82579V	Intel PRO/1000 PT Dual Port Server Adapter
0x1075	82547EI	Gigabit Ethernet Controller
0x1076	82541EI	Gigabit Ethernet Controller
0x1078	82541ER	Gigabit Ethernet Controller
0x1079	82546EB	Dual Port Gigabit Ethernet Controller
0x107C	82541PI	Gigabit Ethernet Controller (Copper) rev 5
0x107D	82572EI	Intel 82572EI Gigabit Ethernet Controller
0x108B	PC82573V	Intel network controller (PCIE Gigabit Ethernet)
0x108C	82573E	Intel 82573E Gigabit Ethernet Controller (Copper)
0x109A	82573L	Intel PRO/1000 PL Network Adaptor
0x10A4	82571GB	Intel PRO/1000 PT Quad Port Server Adapter
0x10A7	82575EB	82575EB Gigabit Network Connection
0x10B9	82572GI	Intel PRO/1000 PT Desktop
0x10BC	82571GB	Intel PRO/1000 PT Low Profile Quad Port Server Adapter

Device ID	Chip Number	Chip Description
0x10BD	82566DM	Intel 82566DM Gigabit Ethernet Adapter
0x10C9	82576	82576 Gigabit ET Dual Port Server Adapter
0x10CE	82567V-2	Intel 82567V-2 Gigabit Network Connection
0x10D3	82574L	Intel 82574L Gigabit Ethernet Controller
0x10DE	02761028	Intel Gigabit network connection
0x10EA	82577LM	Intel 82577LM Gigabit LAN Controller
0x10EB	82577LC	Intel 82577 Gigabit Ethernet
0x10EF	82578DM	Intel 82578DM Gigabit Ethernet
0x10F0	82578DC	Intel 82578DC Gigabit Ethernet
0x10F5	82567LM	Intel 82567LM-2 Gigabit Network Connection
0x1501	82567V3	Intel 82567V3 Gigabit Network Connections
0x1502	0x04931028	Intel 82579LM Gigabit Network Card
0x1503	82579V	Gigabit Network Connection
0x150C	82583V	Intel 82583V Gigabit Ethernet Controller
0x150E	82580 QUAD	82580 Gigabit Network Connection
0x1521	I350	i350 Gigabit Network Connection
0x1526	82576	Intel Gigabit ET2 Quad Port Server Adapter
0x1527	82580 QUAD FIBRE	Intel Ethernet Server Adapter I340-F4
0x1533	210_COPPER	Intel I210 Gigabit Network Connection
0x1539	211AT	Intel Ethernet Controller I211-AT
0x153A	217LM	
0x153B	217V	
0x155A	218LM	
0x157B	210_COPPER_FLASHLESS	Intel I210 Gigabit Network Connection
0x1559	218V	
0xABB1		

Device ID	Chip Number	Chip Description
0xABB2		

Intel Fast Ethernet (10/100) Chip Sets

Device ID	Chip Number	Chip Description
0x1039	10011734	LAN Controller: 82562ET, 82562EZ, 82562VE, 82562VM
0x103A	82801DB	LAN Controller with 82562ET/EZ (CNR)
0x1050	82562EZ	Pro/100 VE Network Connection
0x1059	82551QM	Fast Ethernet PCI/CardBus Controller
0x1092	27DA	PRO/100 VE Network Controller
0x1209	8255xER/IT	Fast Ethernet Controller for XP PC
0x1229	IE22	Intel PRO/100 M Desktop Adapter: 82557, 82558, 82559, 82550, 82551
0x2449	82559ER	82559ER Integrated 10Base-T/100Base-TX Ethernet Controller
0x27DC	336C1462	Intel PRO/100 VE Desktop Adapter

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 conversion 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
uint64	64	uint64
float	32	real32_T
double	64	real_T

EtherCAT Init Block DC Error Values

The Simulink Real-Time EtherCAT Init block returns the following EtherCAT distributed clock (DC) error values. The value `0` indicates that no error occurred.

Error Value	Description
1 (0x1)	Initialization function not called or not successful
2 (0x2)	Controller error — synchronization out of limit
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

10

EtherCAT Blocks

EtherCAT Init

Initialize EtherCAT Master node with data in the EtherCAT Network Information (ENI) file

Library: EtherCAT

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 with TwinCAT 3” on page 9-7.

The Beckhoff ET9000 configurator is no longer supported.

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.

If you configure EtherCAT distributed clocks in master shift mode, using the IEEE 1588 Sync Execution block in the same model produces a build error. To include EtherCAT distributed clocks and IEEE 1588 synchronized execution in the same model, use EtherCAT bus shift mode.

Ports

Output

`Status` — Status information about the EtherCAT network

vector

The `Status` vector contains six 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 `SimulinkRealTime.etherCAT.filterNotifications` 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 slave 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 master sends input data to the slave device. The slave device output remains in a safe state.
`OP`	8	Operational — The network is in full operation. The master sends input data to the slave device. The slave device responds with output data.

- `DCErrVal` — DC error status:

  - No DC Error: `0`
  - DC error: Value from “EtherCAT Init Block DC Error Values” on page 9-30.

  The value `0` appears both if the distributed clock is turned off and if no error occurs.
- `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 or not initialized: `0`
  - DC has been started: `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 full path name or a partial path name. 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 where you select the file that you want.

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

`Device index` — EtherCAT Ethernet card 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.

`PCI bus` — PCI bus number of Ethernet card
`0` (default) | integer

Enter the PCI bus number for the Ethernet card.

`PCI slot` — PCI slot number of Ethernet card
`0` (default) | integer

Enter the PCI slot number for the Ethernet card.

`PCI function` — PCI function number of Ethernet card
`0` (default) | integer

Enter the PCI function number for the Ethernet card.

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

Enter the distributed clock initialization parameter, 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. Check that the devices are synchronized closely enough for your application.

`Enable Log and Debugging` — Access to debugging and logging block parameters
`off` (default) | `on`

Dependency

Selecting **Enable Log and Debugging** makes these parameters visible: **Log link layer error messages**, **Log master state changes**, **Log all state changes**, **Log base clock changes**, **Log master config changes**, and **Target log filename**.

`Log link layer error messages`, `Log master state changes`, `Log all state changes`, `Log base clock changes`, `Log master config changes` — Generate driver-level debug messages
`off` (default) | `on`

To generate driver-level messages for driver and network debugging, select these check boxes.

For a high-speed model, turning on these options can cause CPU overloads.

Dependency

To make these parameters visible, select **Enable Log and Debugging**.

`Target log filename` — Name of log file on target computer
character vector

Enter the name of the log file on the target computer, in single quotes. The default value is `'c:\dbglog.txt'`.

If the target computer does not have a usable disk partition, the software does not create the log file.

Dependency

To make these parameters visible, select **Enable Log and Debugging**.

See Also

"EtherCAT Init Block DC Error Values" on page 9-30 | EtherCAT Get Notifications | `SimulinkRealTime.etherCAT.filterNotifications`

Topics

"EtherCAT® Communication with Beckhoff® Analog IO Slave Devices EL3062 and EL4002"

“EtherCAT® Communication with Beckhoff® Digital IO Slave Devices EL1004 and EL2004”
“EtherCAT® Communication - Motor Velocity Control with Accelnet™ Drive”
“EtherCAT® Communication - Motor Position Control with an Accelnet™ Drive and Beckhoff® Analog IO Devices”
“Configure EtherCAT Network with TwinCAT 3” on page 9-7
“Configure EtherCAT Master Node Model” on page 9-11

External Websites

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

Introduced in R2010b

EtherCAT Get Notifications

Collect notifications from the EtherCAT bus
Library: EtherCAT

Description

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 `SimulinkRealTime.etherCAT.filterNotifications` without an argument.

Data Types: `int32`

Parameters

`Device index` — EtherCAT Ethernet card 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.

`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.

Tips

To collect notifications:

1. Add the EtherCAT Get Notifications block to your model.
2. Connect the EtherCAT Get Notifications block to an Outport block. If possible, make this Outport block Outport block `1`. If the EtherCAT Get Notifications block is connected to the first Outport block, the 21 notification signals appear in the first 21 columns `tg.OutputLog` matrix. Otherwise, you must specify the columns with an offset.
3. Increase the value of **Signal logging data buffer size in doubles** by at least a factor of 100 in the **Simulink Real-Time Options** pane. The EtherCAT Get Notifications block can quickly increase the size of the output log.

4 To print the notifications for this model, pass the relevant 21 columns into the `SimulinkRealTime.etherCAT.filterNotifications` function.

See Also

EtherCAT Init | `SimulinkRealTime.etherCAT.filterNotifications`

External Websites

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

Introduced in R2017a

EtherCAT PDO Receive

Receive data from slave device represented by process data object
Library: 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 shows 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 Ethernet card 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.

`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 9-25.

For a mapping of Simulink data types to EtherCAT data types, see “EtherCAT Data Types” on page 9-29.

`Signal Offset` — Location in the process of signal data
integer

This property is read-only.

Location in the process image from which the data is available after the execution of the EtherCAT Init block.

`Signal Type` — Data type for EtherCAT data
character vector

This property is read-only.

Simulink data type for the EtherCAT data.

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

This property is read-only.

Size in bits of the EtherCAT data type.

`Signal Dimension` — Dimension of the signal
integer

This property is read-only.

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

`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 Beckhoff TwinCAT configurator.

See Also

EtherCAT Init | EtherCAT PDO Transmit

Topics

“EtherCAT Configurator Component Mapping” on page 9-25
“EtherCAT Data Types” on page 9-29

External Websites

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

Introduced in R2010b

EtherCAT PDO Transmit

Send data to slave device represented by process data object
Library: 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 shows 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 Ethernet card 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.

`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 9-25.

For a mapping of Simulink data types to EtherCAT data types, see “EtherCAT Data Types” on page 9-29.

`Signal Offset` — Location in the process of signal data
integer

This property is read-only.

Location in the process image from which the data is available after the execution of the EtherCAT Init block.

`Signal Type` — Data type for EtherCAT data
character vector

This property is read-only.

Simulink data type for the EtherCAT data.

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

This property is read-only.

Size in bits of the EtherCAT data type.

`Signal Dimension` — Dimension of the signal
integer

This property is read-only.

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

`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 Beckhoff TwinCAT configurator.

See Also

EtherCAT Init | EtherCAT PDO Receive

Topics

“EtherCAT Configurator Component Mapping” on page 9-25
“EtherCAT Data Types” on page 9-29

External Websites

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

Introduced in R2010b

EtherCAT Get State

Get state of EtherCAT network
Library: 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 slave 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 master sends input data to the slave device. The slave device output remains in a safe state.
`OP`	8	Operational — The network is in full operation. The master sends input data to the slave device. The slave device responds with output data.

Parameters

`Device index` — EtherCAT Ethernet card 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.

`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.

See Also

EtherCAT Init | EtherCAT Set State

Topics

“EtherCAT Configurator Component Mapping” on page 9-25
“EtherCAT Data Types” on page 9-29

External Websites

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

Introduced in R2010b

EtherCAT Set State

Set state of EtherCAT network
Library: 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 slave 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 master sends input data to the slave device. The slave device output remains in a safe state.
`OP`	8	Operational — The network is in full operation. The master sends input data to the slave device. The slave device responds with output data.

Output

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

This port transmits 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 transmits `0`. Otherwise, it transmits a nonzero value.

Parameters

`Device index` — EtherCAT Ethernet card 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.

`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.

`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.

See Also

EtherCAT Get State | EtherCAT Init

Topics

“EtherCAT Configurator Component Mapping” on page 9-25
“EtherCAT Data Types” on page 9-29

External Websites

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

Introduced in R2010b

EtherCAT Sync SDO Upload

Read data synchronously from slave device represented by service data object
Library: 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 can take 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.

Parameters

`Index` — Index of CANopen register

integer

Specify the decimal index of the CANopen register.

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

`Subindex` — Subindex of CANopen register
integer

Specify the decimal subindex of the CANopen register.

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

`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.

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

Specify the row and column dimension of the CANopen register.

Enter a value of `1`. EtherCAT blocks support only scalars and vectors.

`Device index` — EtherCAT Ethernet card 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.

`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.

`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.

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

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

See Also

EtherCAT Init | EtherCAT Sync SDO Download

Topics

“EtherCAT Configurator Component Mapping” on page 9-25
“EtherCAT Data Types” on page 9-29

External Websites

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

Introduced in R2010b

EtherCAT Sync SDO Download

Transmit data synchronously to slave device represented by service data object
Library: EtherCAT

Description

The EtherCAT Sync SDO Upload 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 can take 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.

Parameters

`Index` — Index of CANopen register
integer

Specify the decimal index of the CANopen register.

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

`Subindex` — Subindex of CANopen register
integer

Specify the decimal subindex of the CANopen register.

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

`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.

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

Specify the row and column dimension of the CANopen register.

Enter a value of `1`. EtherCAT blocks support only scalars and vectors.

`Device index` — EtherCAT Ethernet card 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.

`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.

`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.

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

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

See Also

EtherCAT Init | EtherCAT Sync SDO Upload

Topics

“EtherCAT Configurator Component Mapping” on page 9-25
“EtherCAT Data Types” on page 9-29

External Websites

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

Introduced in R2010b

EtherCAT Async SDO Upload

Read data asynchronously from slave device represented by service data object
Library: 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

boolean

When `true`, the block uploads data.

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

Parameters

`Index` — Index of CANopen register
integer

Specify the decimal index of the CANopen register.

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

`Subindex` — Subindex of CANopen register
integer

Specify the decimal subindex of the CANopen register.

If you specify an invalid subindex, the block returns the value 3 through the `Status` output.

`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 the value 3 through the `Status` output.

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

Specify the row and column dimension of the CANopen register.

Enter a value of 1. EtherCAT blocks support only scalars and vectors.

`Device index` — EtherCAT Ethernet card 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.

`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.

`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.

See Also

EtherCAT Async SDO Download | EtherCAT Init

Topics

“EtherCAT Configurator Component Mapping” on page 9-25
“EtherCAT Data Types” on page 9-29

External Websites

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

Introduced in R2010b

EtherCAT Async SDO Download

Transmit data asynchronously to slave device represented by service data object
Library: 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
boolean

When `true`, the block downloads 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

Parameters

`Index` — Index of CANopen register
integer

Specify the decimal index of the CANopen register.

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

`Subindex` — Subindex of CANopen register
integer

Specify the decimal subindex of the CANopen register.

If you specify an invalid subindex, the block returns the value 3 through the `Status` output.

`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 the value 3 through the `Status` output.

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

Specify the row and column dimension of the CANopen register.

Enter a value of 1. EtherCAT blocks support only scalars and vectors.

`Device index` — EtherCAT Ethernet card 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.

`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.

`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.

See Also
EtherCAT Async SDO Upload | EtherCAT Init

Topics
“EtherCAT Configurator Component Mapping” on page 9-25
“EtherCAT Data Types” on page 9-29

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

Introduced in R2010b

TCP, UDP

Real-Time TCP Communication Support

TCP Transport Protocol

The Simulink Real-Time software supports communication from the target computer to other systems or devices 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 long stream of data that flows from one end of the network to the other. Another long 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 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 to the other side at the beginning of communication. The receiver acknowledges each packet, and the acknowledgment contains the sequence number so that the sender knows which packet was acknowledged. Therefore, packets lost on the way 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 is like a telephone conversation. A continuous connection is required, and two-way streaming data (the words spoken by each party) are exchanged.

When describing TCP, the words *Reliable* and *Unreliable* have a specific meaning.

Note *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 can then be used in a Simulink model.

The preceding discussion 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 using a shared Ethernet board. Shared TCP communication shares bandwidth with the link between the development and target computers.

See Also

Byte Unpacking | Byte Packing | Byte Reversal/Change Endianess | TCP Receive | TCP Client Configure | TCP Send | TCP Server Configure

External Websites

- www.ietf.org/rfc/rfc793.txt

Troubleshoot TCP Block Configuration

I seek to resolve TCP block configuration problems.

What This Means

TCP is a transport protocol layered on top of the Internet Protocol (IP). It is commonly known as TCP/IP. If the block configuration or signal connections for TCP blocks do not follow best practices, the blocks generate errors. Apply these guidelines:

- TCP Blocks Run Only on Target Computer

  The Simulink Real-Time TCP blocks function only when executed on the target computer. When simulated on the development computer, they do nothing.

- Excluded Ports When Using Host-Target Connection

  When you select the **Use host-target connection** parameter in the TCP configure blocks, you cannot use ports 22222 and 22223. Simulink Real-Time reserves these ports for its own use.

- Order of Operation of TCP Blocks

  The real-time application must execute the TCP configure blocks before it executes the TCP Send or TCP Receive blocks.

  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.

Try This

You can use a dedicated Ethernet card for TCP communication while using another card for communicating between the development and target computers. If there is a duplicate subnet calculated in a TCP block, you can get the following error during model initialization:

```
The subnet in this block is the same as or is a subset of the subnet
calculated in ''block''. The block calculates the
subnet by ANDing the IP address bitwise with the subnet mask.
```

Check the IP address and subnet you assigned to the target computer Ethernet card in the configuration block. The TCP implementation requires that the two communication channels use separate subnets.

The block calculates the subnet by ANDing the IP address bitwise with the subnet mask for each card. For example, the following specifications result in the same subnet for both cards.

```
E1 (development-target): IP address:        192.168.0.25
                         Subnet mask:       255.255.255.0
                                            -------------
                         Calculated Subnet: 192.168.0.0

E2 (TCP):                IP address:        192.168.0.26
                         Subnet mask:       255.255.255.0
                                            -------------
                         Calculated Subnet: 192.168.0.0
```

Try a configuration such as the following:

```
E1 (development-target): IP address:        192.168.0.25
                         Subnet mask:       255.255.255.0
                                            -------------
                         Calculated Subnet: 192.168.0.0

E2 (TCP):                IP address:        192.168.0.26
                         Subnet mask:       255.255.255.2
                                            -------------
                         Calculated Subnet: 192.168.0.2
```

In some networks, the development computer must also be in the subnet where the TCP communication occurs. You can either add a second network card to the development computer or provide a gateway device to create a dedicated network for TCP communication.

See Also

TCP Client | TCP Client Configure | TCP Receive | TCP Send | TCP Server | TCP Server Configure

More About

- “TCP/IP and UDP Interface” (Instrument Control Toolbox)
- “TCP/IP Communication” (MATLAB)

TCP Blocks

IP Config

Initialize Ethernet network interface to use for IP communication in real-time applications
Library: TCP

Description

The IP Config block configures a dedicated Ethernet network for real-time operation.

The combination of **Local IP Address** and **Subnet mask** must be unique across all Ethernet cards in the target computer, including the card for communicating between the development and target computers. Distinguish cards by specifying a different subnet for each. The subnet is the IP address masked by the subnet mask.

Parameters

`Local IP Address` — IP address for the Ethernet interface
`x.x.x.x`

Enter the IP address for the dedicated Ethernet board.

The addresses `0.0.0.0` and `255.255.255.255` are invalid IP addresses.

`Subnet mask` — Subnet mask for interface
`255.255.255.0` (default) | *`x.x.x.x`*

Mask that designates a logical subdivision of a network.

`Gateway` — IP address for gateway interface
`0.0.0.0` (default) | *`x.x.x.x`*

The gateway must be within the network.

To indicate that a gateway is not being used, enter `0.0.0.0` (the default). The address `255.255.255.255` is an invalid gateway IP address.

`PCI bus` — PCI bus number of Ethernet card
`0` (default) | `0`–`31`

Enter the PCI bus number for the Ethernet card.

`PCI slot` — PCI slot number of Ethernet card
`0` (default) | `0–31`

Enter the PCI slot number for the Ethernet card.

See Also

`SimulinkRealTime.target.getPCIInfo`

Topics

“Troubleshoot TCP Block Configuration” on page 11-4
“Troubleshoot UDP Block Configuration” on page 13-15

Introduced in R2017a

TCP Client

Configure TCP client
Library: TCP

Description

Configure a TCP client application. You must have already configured a network interface for IP by the IP Config block.

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.

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

`Client IP address` — IP address of the client device that is being configured
`x.x.x.x`

If you are using the Ethernet connection between the development and target computers, this value must match the value of the `TcpIpTargetAddress` target setting. 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 addresses `0.0.0.0` and `255.255.255.255` are invalid IP addresses.

`Client local port` — IP port of the client device that is being configured
1–65535

The combination of **Client IP address** and **Client local port** must be unique.

Dependency

When you select the **Use host-target connection** parameter in the TCP configure blocks, you cannot use ports 22222 and 22223. Simulink Real-Time reserves these ports for its own use.

`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.

The addresses `0.0.0.0` and `255.255.255.255` are invalid IP addresses.

`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.

Dependency

When you select the **Use host-target connection** parameter in the TCP configure blocks, you cannot use ports 22222 and 22223. Simulink Real-Time reserves these ports for its own use.

See Also

IP Config | Target Settings

Topics

“Troubleshoot TCP Block Configuration” on page 11-4

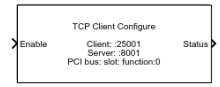
External Websites

www.ietf.org/rfc/rfc793.txt

Introduced in R2017a

TCP Client Configure

Configure a TCP client application that uses the specified Ethernet interface
Library: TCP



Description

Configure a TCP client application and initialize a network interface for the application.

The combination of **Client IP Address** and **Subnet mask** must be unique across all Ethernet cards in the target computer, including the card for communicating between the development and target computers. Distinguish cards by specifying a different subnet for each. The subnet is the IP address masked by the subnet mask.

The Simulink Real-Time TCP blocks function only when executed on the target computer. When simulated on the development computer, they do nothing.

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.

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**, **Subnet mask**, **Gateway**, **PCI bus**, and **PCI slot** parameters and excludes the ports 22222 and 22223 from use by TCP.

`Client IP address` — IP address of the client device that is being configured
`x.x.x.x`

The addresses `0.0.0.0` and `255.255.255.255` are invalid IP addresses.

Dependency

To activate this parameter, clear the **Use host-target connection** parameter.

`Client local port` — IP port of the client device that is being configured
`1`–`65535`

The combination of **Client IP address** and **Client local port** must be unique.

Dependency

When you select the **Use host-target connection** parameter in the TCP configure blocks, you cannot use ports 22222 and 22223. Simulink Real-Time reserves these ports for its own use.

`Subnet mask` — Subnet mask for interface
`255.255.255.0` (default) | *`x.x.x.x`*

Mask that designates a logical subdivision of a network.

Dependency

To activate this parameter, clear the **Use host-target connection** parameter.

`Gateway` — IP address for gateway interface

`0.0.0.0` (default) | *`x.x.x.x`*

Gateway to access a different subnet. The gateway must be within the network.

To indicate that a gateway is not being used, enter `0.0.0.0` (the default). The address `255.255.255.255` is an invalid gateway IP address.

Dependency

To activate this parameter, clear the **Use host-target connection** parameter.

`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.

The addresses `0.0.0.0` and `255.255.255.255` are invalid IP addresses.

`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.

Dependency

When you select the **Use host-target connection** parameter in the TCP configure blocks, you cannot use ports 22222 and 22223. Simulink Real-Time reserves these ports for its own use.

`PCI bus` — PCI bus number of dedicated Ethernet card

`0` (default) | `0`–`31`

Enter the PCI bus number for the dedicated Ethernet card.

Dependency

To activate this parameter, clear the **Use host-target connection** parameter.

`Slot` — PCI slot number of dedicated Ethernet card

`0` (default) | `0`–`31`

Enter the PCI slot number for the dedicated Ethernet card.

Dependency

To activate this parameter, clear the **Use host-target connection** parameter.

`Function` — PCI function number of Ethernet card
`0` (default) | integer

Enter the PCI function number for the Ethernet card.

See Also

`SimulinkRealTime.target.getPCIInfo` | TCP Receive | TCP Send

Topics

“Troubleshoot TCP Block Configuration” on page 11-4

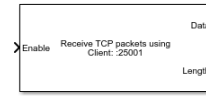
External Websites

www.ietf.org/rfc/rfc793.txt

Introduced in R2017a

TCP Receive

Receive data over TCP network from a remote device
Library: TCP



Description

Receive 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.

`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.

`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.

See Also

TCP Client Configure | TCP Server Configure

Topics

“Troubleshoot TCP Block Configuration” on page 11-4

External Websites

www.ietf.org/rfc/rfc793.txt

Introduced in R2017a

TCP Send

Send data over TCP network to a remote device
Library: TCP

Description

Send 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 `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.

`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.

See Also

TCP Client Configure | TCP Server Configure

Topics

“Troubleshoot TCP Block Configuration” on page 11-4

External Websites

www.ietf.org/rfc/rfc793.txt

Introduced in R2017a

TCP Server

Configure TCP server application
Library: TCP

Description

Configure a TCP server application. This block assumes that a network interface has been configured for IP by the IP Config block.

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.

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

`Server IP address` — IP address of the server device that is being configured
`x.x.x.x`

If you are using the Ethernet connection between the development and target computers, this value must match the value of the `TcpIpTargetAddress` target setting. 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 addresses `0.0.0.0` and `255.255.255.255` are invalid IP addresses.

`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.

Dependency

When you select the **Use host-target connection** parameter in the TCP configure blocks, you cannot use ports 22222 and 22223. Simulink Real-Time reserves these ports for its own use.

See Also

IP Config | Target Settings

Topics

“Troubleshoot TCP Block Configuration” on page 11-4

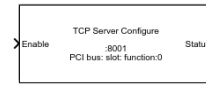
External Websites

www.ietf.org/rfc/rfc793.txt

Introduced in R2017a

TCP Server Configure

Configure TCP server application that uses the specified Ethernet interface
Library: TCP



Description

Configure a TCP server application and initialize a network interface for the application.

The combination of **Server IP Address** and **Subnet mask** must be unique across all Ethernet cards in the target computer, including the card for communicating between the development and target computers. Distinguish cards by specifying a different subnet for each. The subnet is the IP address masked by the subnet mask.

The Simulink Real-Time TCP blocks function only when executed on the target computer. When simulated on the development computer, they do nothing.

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.

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

Selecting the **Use host-target connection** parameter disables the **Server IP address**, **Subnet mask**, **PCI bus**, and **PCI slot** parameters and excludes the ports 22222 and 22223 from use by TCP.

`Server IP address` — IP address of the server device that is being configured

`x.x.x.x`

The addresses `0.0.0.0` and `255.255.255.255` are invalid IP addresses.

Dependency

To enable this parameter, clear the **Use host-target connection** parameter.

`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.

Dependency

When you select the **Use host-target connection** parameter in the TCP configure blocks, you cannot use ports 22222 and 22223. Simulink Real-Time reserves these ports for its own use.

`Subnet mask` — Subnet mask for interface

`255.255.255.0` (default) | *`x.x.x.x`*

Mask that designates a logical subdivision of a network.

Dependency

To activate this parameter, clear the **Use host-target connection** parameter.

`Gateway` — IP address for gateway interface

`0.0.0.0` (default) | *`x.x.x.x`*

Gateway to access a different subnet. The gateway must be within the network.

To indicate that a gateway is not being used, enter `0.0.0.0` (the default). The address `255.255.255.255` is an invalid gateway IP address.

Dependency

To activate this parameter, clear the **Use host-target connection** parameter.

`PCI bus` — PCI bus number of dedicated Ethernet card

`0` (default) | `0`–`31`

Enter the PCI bus number for the dedicated Ethernet card.

Dependency

To activate this parameter, clear the **Use host-target connection** parameter.

`Slot` — PCI slot number of dedicated Ethernet card

`0` (default) | `0`–`31`

Enter the PCI slot number for the dedicated Ethernet card.

Dependency

To activate this parameter, clear the **Use host-target connection** parameter.

`Function` — PCI function number of Ethernet card

`0` (default) | integer

Enter the PCI function number for the Ethernet card.

See Also

`SimulinkRealTime.target.getPCIInfo` | TCP Receive | TCP Send

Topics

“Troubleshoot TCP Block Configuration” on page 11-4

External Websites

www.ietf.org/rfc/rfc793.txt

Introduced in R2017a

Real-Time UDP Communication Support

UDP Transport Protocol

The Simulink Real-Time software supports communication from the target computer to other systems or devices with 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 is like sending letters by mail, without a return address. If the other party is not found, or the letter is lost in transit, it 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 as long as 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 can then be used 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 using a shared Ethernet board. Shared UDP communication shares bandwidth with the link between the development and target computers.

See Also

Byte Unpacking | Byte Packing | Byte Reversal/Change Endianess | UDP Configure | UDP Receive | UDP Send

More About

- “Target to Host Transmission using UDP”
- “Target to Target Transmission using UDP”
- “UDP Communication Setup” on page 13-11
- “UDP and Variable-Size Signals” on page 13-13

UDP Data Exchange with Shared Ethernet Board

In this section...

This example shows how to set up two-way data exchange with 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 a UDP Configure block because the example uses the connection between the development and target computers. To perform real-time UDP data transfer with a dedicated Ethernet board, see “Target to Target Transmission using UDP”.

The example models are named `udpsendreceiveA` and `udpsendreceiveB`. Replace the port and IP address examples with ports and addresses as required by your network.

Data Transferred

The models transfer two different data sets between them, one data set from `udpsendreceiveA` to `udpsendreceiveB` and another data set in the opposite direction.

For this example, the inputs are generated with 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.

udpsendreceiveA to udpsendreceiveB

The UDP data send is 75 bytes wide. The data to transfer is in the following 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.

udpsendreceiveB to udpsendreceiveA

The UDP data to be sent is 79 bytes wide. The data to transfer is in the following formats.

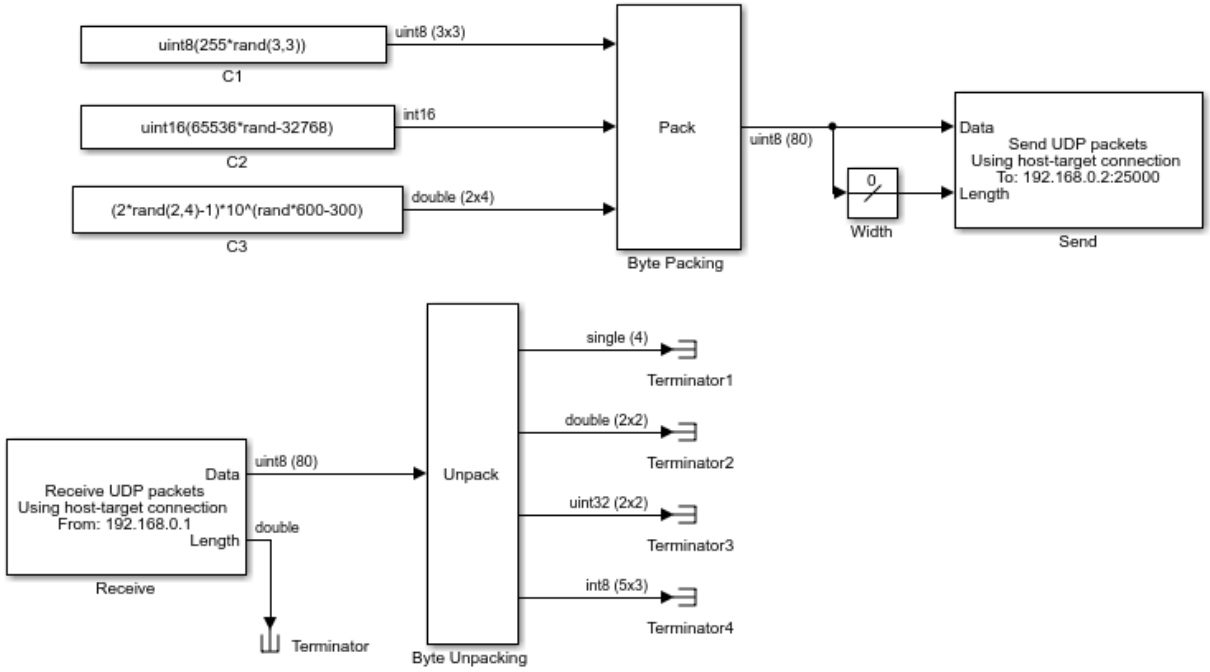
- `[4 1]` of `single` (16 bytes)
- `[2 2]` of `double` (32 bytes)
- `[2 2]` of `uint32` (16 bytes)
- `[5 3]` of `int8` (15 bytes)

When packed, the data is aligned on 2-byte boundaries. A zero-valued pad byte is added during packing.

Set Up udpsendreceiveA

The final `udpsendreceiveA` is shown in the figure.

The tables list the parameters for the send and receive sides of the model.



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`
	Local port	`-1` (autoselect)
	To IP address	`192.168.0.2`
	To port	`25000`

Block	Parameter	Value
	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 `udpsendreceiveB`.

udpsendreceiveA Receive Side

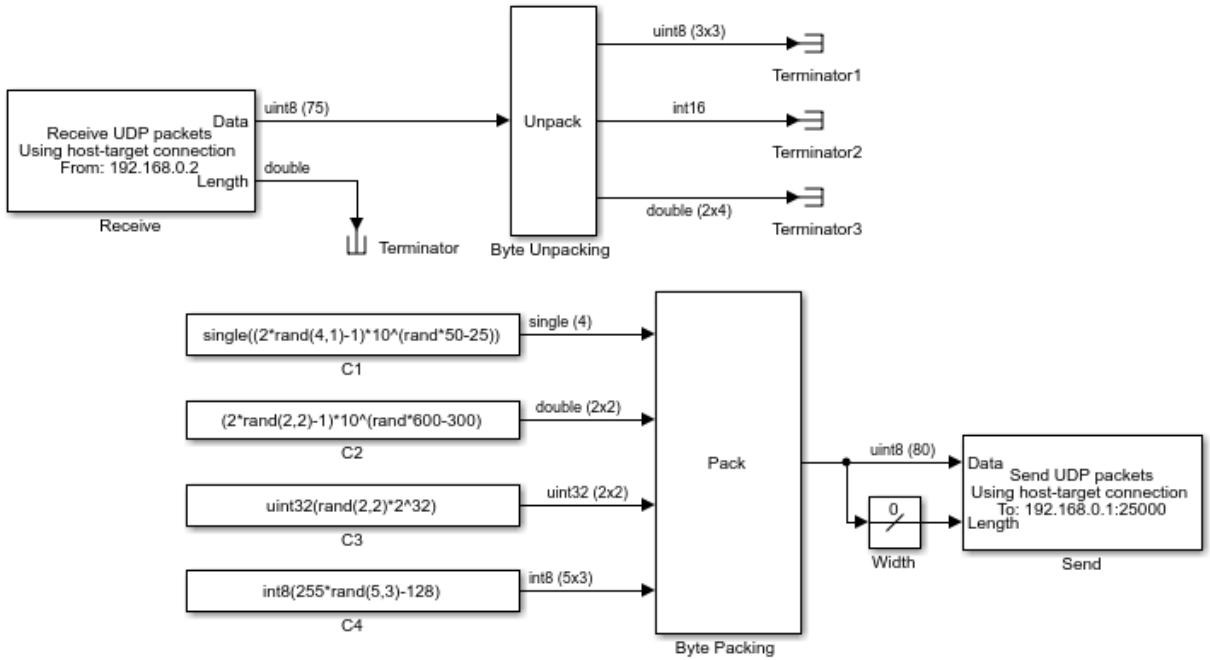
Block	Parameter	Value
UDP Receive	**Local IP address**	`Use host-target connection`
	Local port	`25000`
	Receive width	`80`
	Receive from any source	`off`
	From IP address	`192.168.0.1`
	Sample time (-1 for inherited)	`0.01`
Byte Unpacking	**Output port (unpacked) data types (cell array)**	`{'single', '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, which in a real model would be used in the model.
- The **Receive width** of the UDP Receive block matches the output port width of the Byte Packing block in `udpsendreceiveB`.
- The Byte Unpacking block settings match the settings of the Byte Packing block of `udpsendreceiveB`.
- The number of unpacked bytes is `79`. The byte alignment is `2`, so the Byte Unpacking block assumes that the input vector includes a pad `0` to align the vector on an even-numbered boundary.

Set Up udpsendreceiveB

The final `udpsendreceiveB` model is shown in the figure.

The tables list the parameters for the receive side and the send side of the model.



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	`192.168.0.2`

Block	Parameter	Value
	Sample time (-1 for inherited)	`0.01`
Byte Unpacking	**Output port (unpacked) data types (cell array)**	`{'uint8', 'int16', '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, which in a real model would be used in the model.
- The **Receive width** of the UDP Receive block matches the output port width of the Byte Packing block in `udpsendreceiveA`.
- The Byte Unpacking block settings match the Byte Packing block in `udpsendreceiveA`.

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	`192.168.0.1`
	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 `udpsendreceiveA`.

- The number of unpacked bytes is `79`. The byte alignment is `2`, so the Byte Packing block pads the output vector with `0` to align on an even-numbered boundary.

See Also

Byte Packing | Byte Unpacking | UDP Configure | UDP Receive | UDP Send

More About

- “Target to Host Transmission using UDP”
- “Target to Target Transmission using UDP”
- “UDP Transport Protocol” on page 13-2
- “UDP Communication Setup” on page 13-11
- “UDP and Variable-Size Signals” on page 13-13

UDP Communication Setup

The infrastructure provided in the Simulink Real-Time Library for UDP communication consists mainly 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:

```
slrtlib
```

The blocks are located under the `Real-Time UDP` heading in the library. The UDP Send block takes as input a vector of type `uint8`, which it sends. 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 `uint8`s back into arbitrary Simulink data types, use a Byte Unpacking block.

If you are using a dedicated Ethernet port for UDP communication, use a UDP Configure block to configure the Ethernet interface.

You can have up to 32 UDP blocks in a model—UDP Send and UDP Receive blocks combined in arbitrary order, plus the optional UDP Configure block.

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 kernel 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 cautious 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

Byte Packing | Byte Unpacking | UDP Configure | UDP Receive | UDP Send

More About

- “Target to Host Transmission using UDP”
- “Target to Target Transmission using UDP”
- “UDP Transport Protocol” on page 13-2

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.

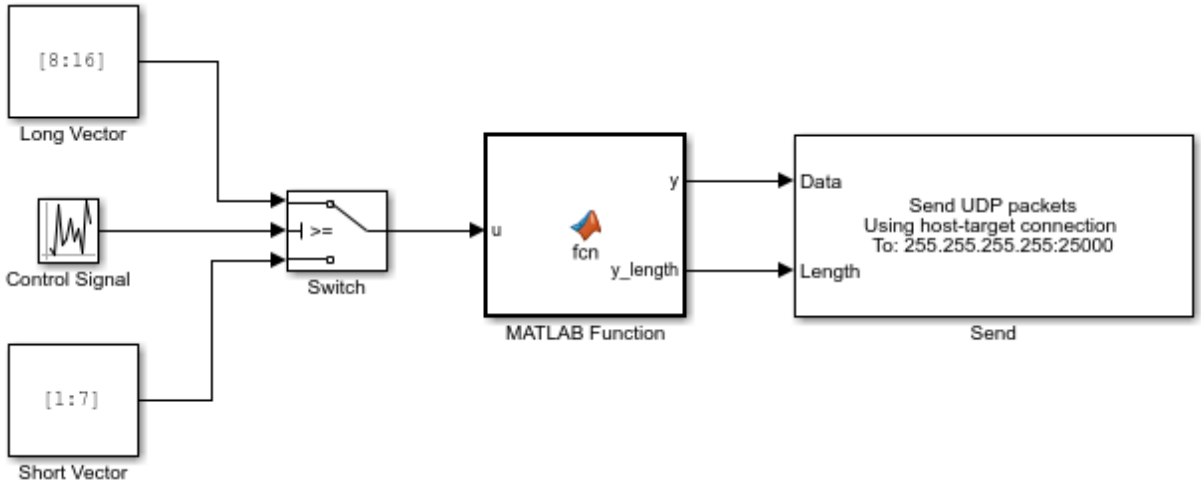
To send variable-size signals though UDP, determine the maximum number of elements of a fixed-size input signal that you expect to connect to the block. Then use the second input, `Length`, to specify the number of elements of this input signal to send through UDP.

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, select **Tools** > **Edit Data/Ports**. 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

MATLAB Function | UDP Send

Troubleshoot UDP Block Configuration

I seek to resolve UDP Configure block configuration problems.

What This Means

The Real-Time UDP Configure block configures a dedicated Ethernet network for real-time UDP operation. If the block configuration does not distinguish cards by specifying a different subnet for each, errors occur.

Note There is a limitation on the number of UDP Send and UDP Receive blocks in a model. The total number of these blocks in a model is limited to 2048.

Try This

To identify UDP Configure block configuration problems, check for the following issues.

Duplicate Subnet Calculated in Block

You can use a dedicated Ethernet card for TCP communication while using another card for communicating between the development and target computers. During model initialization, you get this error:

```
The subnet in this block is the same as or is a subset of the subnet
calculated in ''block''. The block calculates the subnet by ANDing the
IP address bitwise with the subnet mask.
```

Check the IP address and subnet that you assigned to the target computer Ethernet card in the configuration block. The UDP implementation requires that the two communication channels use separate subnets.

The block calculates the subnet by ANDing the IP address bitwise with the subnet mask for each card. For example, these specifications result in the same subnet for both cards:

```
E1 (development-target): IP address:        192.168.0.25
                         Subnet mask:       255.255.255.0
                                            -------------
                         Calculated Subnet: 192.168.0.0

E2 (RT-UDP):             IP address:        192.168.0.130
```

```
                          Subnet mask:       255.255.255.0
                                             -------------
                          Calculated Subnet: 192.168.0.0
```

Try a configuration such as the following:

```
E1 (development-target): IP address:        192.168.0.25
                         Subnet mask:       255.255.255.0
                                            -------------
                         Calculated Subnet: 192.168.0.0

E2 (RT-UDP):             IP address:        192.168.0.130
                         Subnet mask:       255.255.255.128
                                            -------------
                         Calculated Subnet: 192.168.0.128
```

In some networks, the development computer must also be in the subnet where the TCP communication occurs. You can either add a second network card to the development computer or provide a gateway device to create a dedicated network for TCP communication.

Excluded Ports When Using Development-Target Computer Connection

When you use the same IP address as the communication channel between the development and target computers, you cannot use ports `22222` and `22223`. Simulink Real-Time reserves these ports for its own use.

ENOPKTS Error

During real-time execution with a UDP model, you sometimes see the error `ENOPKTS`. This error stops model execution. When too many packets are received and queued at the UDP socket and too few packets are removed, this error occurs.

To address this issue, decrease the sample time of your UDP Receive block.

See Also

More About

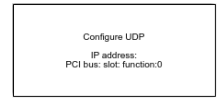
- “TCP/IP and UDP Interface” (Instrument Control Toolbox)

Real-Time UDP Blocks

UDP Configure

Initialize Ethernet network interface to use for UDP communication in real-time applications

Library: Real-Time UDP



Description

The Real-Time UDP Configuration block configures a dedicated Ethernet network for real-time UDP operation.

The combination of **Local IP Address** and **Subnet mask** must be unique across all Ethernet cards in the target computer, including the card for communicating between the development and target computers. Distinguish cards by specifying a different subnet for each. The subnet is the IP address masked by the subnet mask.

Parameters

General Parameters

`Local IP Address` — IP address for the Ethernet interface
`x.x.x.x`

Enter the IP address for the dedicated Ethernet board.

The addresses `0.0.0.0` and `255.255.255.255` are invalid local IP addresses.

`Subnet mask` — Subnet mask for interface
`255.255.255.0` (default) | *`x.x.x.x`*

Mask that designates a logical subdivision of a network.

`Gateway` — IP address for gateway interface
`0.0.0.0` (default) | *`x.x.x.x`*

The gateway must be within the network.

To indicate that a gateway is not being used, enter `0.0.0.0` (the default). The address `255.255.255.255` is an invalid gateway IP address.

`PCI bus` — PCI bus number of Ethernet card
`0` (default) | `0`–`31`

Enter the PCI bus number for the Ethernet card.

`Slot` — PCI slot number of Ethernet card
`0` (default) | `0`–`31`

Enter the PCI slot number for the Ethernet card.

`Function` — PCI function number of Ethernet card
`0` | integer

Enter the PCI function number for the Ethernet card.

Multicast Parameters

`Enable multicast` — Enables multicast UDP parameters and operations
`off` (default) | `on`

When you select **Enable multicast**, the UDP multicast parameters become visible.

Example: `on`

`Multicast IP address list` — Selects list of UDP multicast addresses
`{'x.x.x.x'}` | cell array of character vectors

Each IP address and corresponding mask parameter set configure the UDP multicast operations. The multicast IP address and corresponding mask values are AND'ed, and the packets to the destination addresses that match this value are sent through this interface. In the example values, the two pairs of address and mask indicate that all packets sent to the IP address corresponding to `224.0.1.xxx` and `224.0.2.xxx` are sent out through the card at `[5,0,0]`.

A limitation is that the combination of multicast IP and masks (multicast subnets) cannot overlap with those values in other configuration blocks. Validation for this overlap follows validation of the Local IP Address and subnet mask combinations.

Example: `{'224.0.1.129','224.0.2.107'}`

`Mask` — Selects the mask for UDP multicast addresses
`{'x.x.x.x'}` | cell array of character vectors

See description of **Multicast IP address list** parameter.

Example: `{'255.255.255.0','255.255.255.0'}`

See Also

`SimulinkRealTime.target.getPCIInfo` | UDP Receive | UDP Send

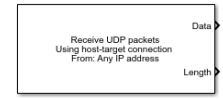
Topics

“UDP Transport Protocol” on page 13-2
“Troubleshoot UDP Block Configuration” on page 13-15

Introduced in R2016b

UDP Receive

Receive data over UDP network from a remote device
Library: Real-Time UDP



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, add a UDP Configure block to your model.

The parameter **Local IP address** applies only when the block executes on a target 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

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

`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. Otherwise, the block uses the value that you set in the **Local IP address** parameter of the UDP Configure block.

`Local port` — Destination UDP port through which to receive data
`1–65535`

Specifies UDP port through which to receive data.

`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.

`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**.

Dependency

To make the **From IP address** parameter visible, clear **Receive from any source**.

`From IP address` — Source from which to receive data
`0.0.0.0` (default) | *`x.x.x.x`*

Enter a valid IP address as a dotted decimal character vector, for example, `10.10.10.3`. You can also use a MATLAB expression that returns a valid IP address as a character vector.

The default address, `0.0.0.0`, causes the block to accept UDP packets from any accessible device. If you set **From IP address** to a specific IP address, only packets arriving from that IP address are received.

The address `255.255.255.255` is an invalid IP address.

Dependency

To make this parameter visible, clear **Receive from any source**.

`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.

Multicast Parameters

`Receive multicast` — Enables multicast UDP parameters and receive operations
`off` (default) | `on`

When you select **Enable multicast**, the UDP multicast parameters become visible.

Example: `on`

`Multicast group address` — Multicast group to join
`x.x.x.x` | dotted decimal character vector

Enter a valid IP address as a dotted decimal character vector, for example, `224.0.0.0`.

The UDP 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`

See Also

Byte Reversal/Change Endianess | Byte Unpacking | UDP Configure | UDP Send

Topics

Introduced in R2016b

UDP Send

Send data over UDP network to a remote device
Library: Real-Time UDP

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, add a UDP Configure block to your model.

The parameter **Local IP address** applies only when the block executes on a target 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.

To broadcast to all devices, set **To IP address** to `255.255.255.255`, otherwise set **To IP address** to a valid IP address.

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

`Local IP address` — Source IP address for sending 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. Otherwise, the block uses the value that you set in the **Local IP address** parameter of the UDP Configure block.

If the UDP Configure block settings enable multicast operation, the Send block sends to the IP address that is set to the group IP address. For real time multicast send capability, the model requires a UDP Configure block. If the model does not include this block, a warning about route unavailability is issued on the target.

`Local port` — Source UDP port through which to transmit data
`1–65535` | `-1`

Specifies local UDP port through which to transmit data.

The value `−1` means that the block transmits using any available port.

`To IP address` — IP address of target device
`255.255.255.255` (default) | *`x.x.x.x`*

Specifies IP address of target device. To broadcast, send to `255.255.255.255`.

`To port` — UDP port of target device
`1–65535`

Specify the UDP port of target device. With **To IP address**, this parameter defines the destination of the data transmission.

`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.

See Also

Byte Reversal/Change Endianess | Byte Packing | UDP Configure | UDP Receive

Topics

“UDP Transport Protocol” on page 13-2
“Troubleshoot UDP Block Configuration” on page 13-15

Introduced in R2016b

Parallel Ports, PTP, SAE J1939, Shared Memory

Parallel Ports

Using Parallel Ports

In this section...

Introduction

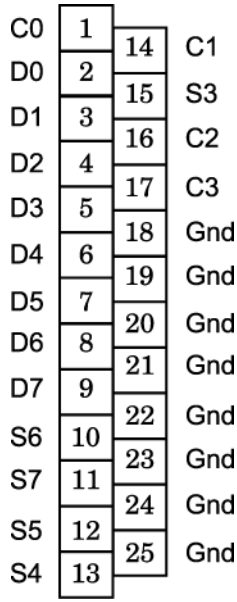
Most target computers have a parallel port that you can use for various devices. The Simulink Real-Time block library provides blocks that enable you to use the parallel ports of a target computer for digital input and output, and source interrupts.

Caution The parallel port is part of the motherboard on many computers. Be careful when configuring the port and when connecting external devices to the port. Incorrect connections to the port can damage your computer.

The Simulink Real-Time parallel port blocks assume that the connector to the parallel port has one 25-pin connector whose pins have the following designations:

- Eight data pins
- Five status pins
- Four control pins
- Eight ground pins

Function	Channel	1	2	3	4	5	6	7	8	Additional Pins
	Bit	0	1	2	3	4	5	6	7	
Digital Input		02	03	04	05	06	07	08	09	
Digital Output		02	03	04	05	06	07	08	09	
Digital Input (Status)		15	13	12	10	11				
Digital Output (Control)		01	14	16	17					
Interrupt										10



Using the Parallel Port as an Interrupt Source

To use the parallel port as an interrupt source, use pin 10 of the parallel port as the interrupt source. Configure the Simulink Real-Time model as follows:

1. Select **Simulation > Model Configuration Parameters**.
2. Under node **Code Generation**, select node **Simulink Real-Time Options**.
3. In the **Execution options** pane:
   - From **Execution mode**, select `Real-Time`.
   - From **Real-time interrupt source**, select the IRQ level (typically 7).
   - From **I/O board generating the interrupt**, select `Parallel_Port`.
   - In **PCI slot (-1: autosearch) or ISA base address**, enter the base address of the parallel port (typically `0x378`).

If you want to use the Async IRQ Source block, you do not have to configure the model. Instead, you can set the Async IRQ Source block parameters as follows:

- **IRQ line number** — Select the IRQ level (typically 7).

- **I/O board generating the interrupt** — Select `Parallel_Port`.
- **PCI slot** — Enter the base address of the parallel port (typically `0x378`).

Using Add-On Parallel Port Boards

To use an add-on parallel port board with the parallel port blocks, configure the base address for the board as follows:

1. To get the base address of a board, in the MATLAB Command Window, call the function `SimulinkRealTime.target.getPCIInfo` with the `'verbose'` option. For example:

   ```
   tg = slrt;
   getPCIInfo(tg, 'verbose')
   ```

2. Identify the base address for the add-on parallel port board.
3. In your model, open the parallel port block and set the value of the **Base address** parameter to `Other`.

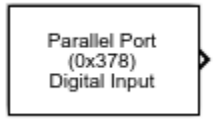
   The **Alternate base address** parameter is displayed.
4. In the **Alternate base address** parameter, enter the base address you identified in step 2.
5. Configure the rest of the block as desired.

Note You cannot use add-on parallel port boards as interrupt sources. You also cannot trigger the execution of a model with these boards.

Parallel Port Blocks

Parallel Port Digital Input

Parallel Port Digital Input block



Library

Simulink Real-Time Library for Parallel Port

Scaling Input to Output

I/O Module Input	Block Output Data Type	Scaling
TTL	Double (**Format**: `8 1-bit Channels`)	Double: TTL low = 0.0 TTL high = 1.0
	uint8 (**Format**: `One 8-bit Port`)	uint8: TTL low corresponding bit is clear TTL high corresponding bit is set

Block Parameters

Base address

Select a parallel port base address. This address depends on the target computer BIOS. From the list, select one of the following. If your base address is not one of the supplied standard base addresses, select `Other` and enter your base address in **Alternate base address**.

- `0x3bc`
- `0x378`
- `ox278`
- `Other`

Alternate base address

Enter an alternate parallel port base address, in hexadecimal. This parameter appears only if you select `Other` for **Base address**. For example,

```
0x300
```

Format

From the list, select one of the following modes to specify how to treat data:

- `8 1-bit Channels`

  Treats data as individual bits. Configures block to accept up to eight 1-bit channels.

- `One 8-bit Port`

  Treats data as a single byte. Configures block to accept one 8-bit port.

Channels

Enter a vector of numbers between 1 and 8. This parameter appears only if you select `8 1-bit Channels` for **Format**. For example,

```
[1, 3]
```

Sample time

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

See Also

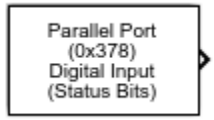
Topics

“Using Parallel Ports” on page 15-2

Introduced in R2007a

Parallel Port Digital Input Status Bits

Parallel Port Digital Input Status Bits block



Library

Simulink Real-Time Library for Parallel Port

Scaling Input to Output

I/O Module Input	Block Output Data Type	Scaling
TTL	Double (**Format**: `5 1-bit Channels`)	Double: TTL low = 0.0 TTL high = 1.0
	uint8 (**Format**: `One 5-bit Port`)	uint8: TTL low corresponding bit is clear TTL high corresponding bit is set

Block Parameters

Base address

Select a parallel port base address. This address depends on the target computer BIOS. From the list, select one of the following. If your base address is not one of the supplied standard base addresses, select `Other` and enter your base address in **Alternate base address**.

- `0x3bc`
- `0x378`
- `ox278`
- `Other`

Alternate base address

Enter an alternate parallel port base address, in hexadecimal. This parameter appears only if you select `Other` for **Base address**. For example,

```
0x300
```

Format

From the list, select one of the following modes to specify how to treat data:

- `5 1–bit Channels`

  Treats data as individual bits. Configures block to accept up to five 1-bit channels.

- `One 5–bit Port`

  Treats data as a single byte. Configures block to accept one 5-bit port.

Channels

Enter a vector of numbers between 1 and 5. This parameter appears only if you select `5 1–bit Channels` for **Format**. For example,

```
[1, 3]
```

Sample time

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

See Also

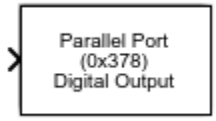
Topics

“Using Parallel Ports” on page 15-2

Introduced in R2007a

Parallel Port Digital Output

Parallel Port Digital Output block



Library

Simulink Real-Time Library for Parallel Port

Scaling Output to Input

I/O Module Output	Block Input Data Type	Scaling
TTL	Double (**Format**: `8 1-bit Channels`)	Double: < 0.5 = TTL low > 0.5 = TTL high
	uint8 (**Format**: `One 8-bit Port`)	uint8: Bit clear = TTL low Bit set = TTL high

Block Parameters

Base address

Select a parallel port base address. This address depends on the target computer BIOS. From the list, select one of the following. If your base address is not one of the supplied standard base addresses, select `Other` and enter your base address in **Alternate base address**.

- `0x3bc`
- `0x378`
- `ox278`
- `Other`

Alternate base address

Enter an alternate parallel port base address, in hexadecimal. This parameter appears only if you select `Other` for **Base address**. For example,

```
0x300
```

Format

From the list, select one of the following modes to specify how to treat data:

- `8 1–bit Channels`

  Treats data as individual bits. Configures block to accept up to eight 1-bit channels.

- `One 8–bit Port`

  Treats data as a single byte. Configures block to accept one 8-bit port.

Channels

Enter a vector of numbers between 1 and 8. This parameter appears only if you select `5 1–bit Channels` for **Format**. For example,

```
[1, 3]
```

Initial value vector

The initial value vector contains the initial voltage values for the output channels. Enter a scalar or a vector that is the same length as the channel vector. If you specify a scalar value, that value is replicated as the initial value over the channel vector. The channels are set to the initial values between the time the model is downloaded and the time it is started.

Final action vector

The final action vector controls the behavior of the channel at model termination. Enter a scalar or a vector that is the same length as the channel vector. If you specify a scalar value, that setting is replicated over the channel vector. If you specify a value of 1, the corresponding channel is reset to the value specified in the initial value

vector. If you specify a value of `-1`, the block sets the channel to the value specified in the **Final value vector** value for that channel. If you specify a value of `0`, the channel remains at the last value attained while the model was running.

Final value vector

The final value vector contains the final value for each output channel. Enter a scalar or a vector that is the same length as the channel vector. If you specify a scalar value, that setting is replicated over the channel vector. If the **Final action vector** is `-1`, the block sets the channel to this value on model termination.

Sample time

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

See Also

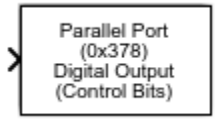
Topics

“Using Parallel Ports” on page 15-2

Introduced in R2007a

Parallel Port Digital Output Control Bits

Parallel Port Digital Output Control Bits block



Library

Simulink Real-Time Library for Parallel Port

Scaling Output to Input

I/O Module Output	Block Input Data Type	Scaling
TTL	Double (**Format**: `4 1-bit Channels`)	Double: < 0.5 = TTL low > 0.5 = TTL high
	uint8 (**Format**: `One 4-bit Port`)	uint8: Bit clear = TTL low Bit set = TTL high

Block Parameters

Base address

Select a parallel port base address. This address depends on the target computer BIOS. From the list, select one of the following. If your base address is not one of the supplied standard base addresses, select `Other` and enter your base address in **Alternate base address**.

- `0x3bc`
- `0x378`
- `ox278`
- `Other`

Alternate base address

Enter an alternate parallel port base address, in hexadecimal. This parameter appears only if you select `Other` for **Base address**. For example,

```
0x300
```

Format

From the list, select one of the following modes to specify how to treat data:

- `4 1–bit Channels`

  Treats data as individual bits. Configures block to accept up to four 1-bit channels.

- `One 4–bit Port`

  Treats data as a single byte. Configures block to accept one 4-bit port.

Channels

Enter a vector of numbers between 1 and 4. This parameter appears only if you select `4 1–bit Channels` for **Format**. For example,

```
[1, 3]
```

Initial value vector

The initial value vector contains the initial voltage values for the output channels. Enter a scalar or a vector that is the same length as the channel vector. If you specify a scalar value, that value is replicated as the initial value over the channel vector. The channels are set to the initial values between the time the model is downloaded and the time it is started.

Final action vector

The final action vector controls the behavior of the channel at model termination. Enter a scalar or a vector that is the same length as the channel vector. If you specify a scalar value, that setting is replicated over the channel vector. If you specify a value of `1`, the corresponding channel is reset to the value specified in the initial value vector. If you specify a value of `-1`, the block sets the channel to the value specified in

the **Final value vector** value for that channel. If you specify a value of `0`, the channel remains at the last value attained while the model was running.

Final value vector

The final value vector contains the final value for each output channel. Enter a scalar or a vector that is the same length as the channel vector. If you specify a scalar value, that setting is replicated over the channel vector. If the **Final action vector** is `-1`, the block sets the channel to this value on model termination.

Sample time

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

See Also

Topics

“Using Parallel Ports” on page 15-2

Introduced in R2007a

Precision Time Protocol

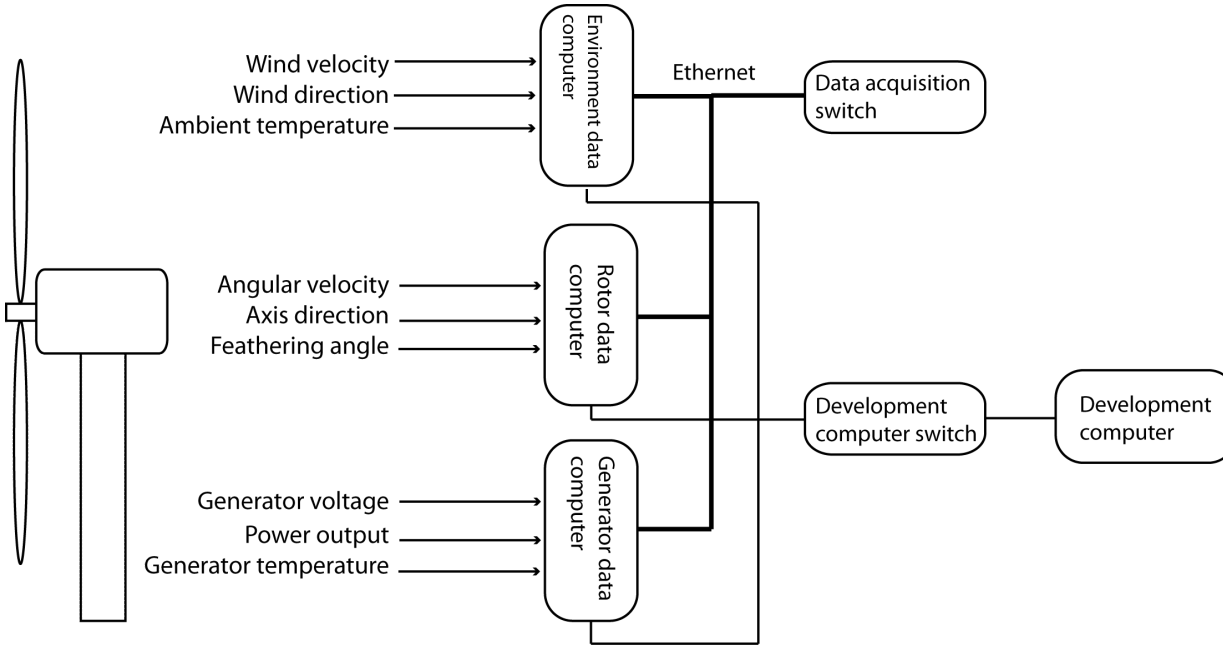
Precision Time Protocol

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 PTP clock of each computer to a reference time. The PTP clock of a Simulink Real-Time target computer is the clock on the PTP network card.

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, you connect the sensors to a set of data acquisition target computers. You 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, you connect the target computers to a development computer through another Ethernet network and switch. On the development computer, you 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 have invalid (unsynchronized) timestamps.
- Integrating values of measured data collected at the same time from sensors connected to different computers.

To synchronize the target computer PTP clocks, you create a Simulink Real-Time model for each target computer. Each model uses the following PTP blocks:

- IEEE 1588 Ethernet — Run PTP protocol with Raw Ethernet as transport protocol. This block communicates with the corresponding blocks on the other target computers and determines the time offset that synchronizes them.

- IEEE 1588 Read Parameter — Output a Precision Time Protocol parameter value. Of the possible output values, you select `PTP time (nanosecond)`.

  For debugging, you can configure a separate IEEE 1588 Read Parameter block to read other values, such as `Protocol state`.
- IEEE 1588 Sync Execution — Synchronize model execution to Precision Time Protocol clock. You can now make measurements at the same time step.
- IEEE 1588 Sync Status — Output the synchronization status of the Precision Time Protocol. When the value is `true`, the data timestamps are synchronized to the required precision.

As a best practice, for each model, you enclose the sensor block and the IEEE 1588 Read Parameter and IEEE 1588 Sync Status blocks in an Atomic Subsystem block. By using the Atomic Subsystem block, you bring the PTP timestamp as close as possible to the time of the data measurement.

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.

See Also

Atomic Subsystem | IEEE 1588 Read Parameter | IEEE 1588 Ethernet | IEEE 1588 Sync Execution | IEEE 1588 Sync Status

More About

- “Synchronize Timestamps Across Data-Gathering Network” on page 17-5
- “IEEE® 1588™ Precision Time Protocol - Execution Synchronization”
- “Data Acquisition and Data Analysis Example Description” on page 17-18
- “Prerequisites, Limitations, and Unsupported Features” on page 17-31

External Websites

- standards.ieee.org

Synchronize Timestamps Across Data-Gathering Network

This example shows a data acquisition target computer that transmits timestamped data to a second target computer that analyzes the data.

Required Products: Simulink®, Simulink Real-Time™

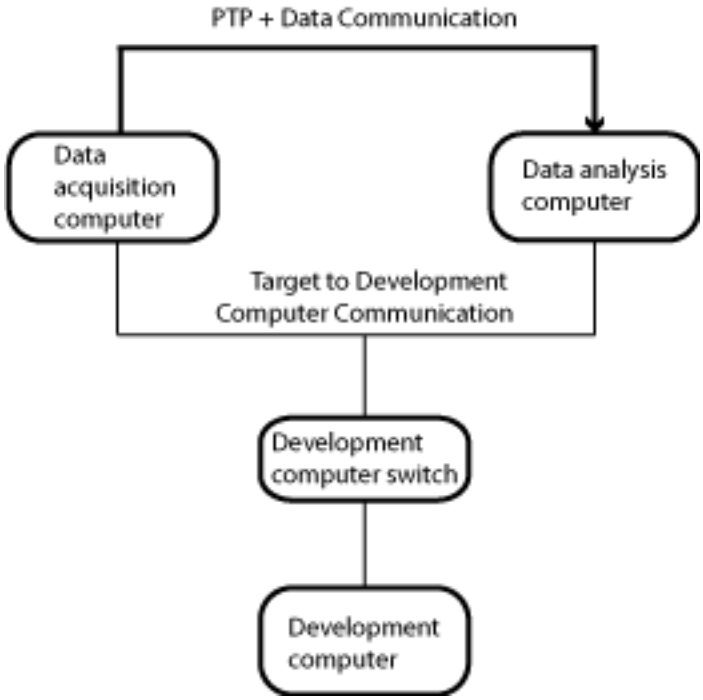
Other Requirements:

- One Windows® development computer with an Ethernet card.
- Two Speedgoat target computers.
- At least one Intel® 82574 Ethernet card on each target computer
- One Ethernet card on each target computer dedicated to communication between the development and target computers.
- One Ethernet switch.
- Four crossover Ethernet cables.

The real-time applications use Precision Time Protocol blocks to synchronize the Intel 82574 Ethernet card PTP clocks and the kernel clocks for each computer. You can log the PTP timestamps from both computers and use them to associate transmitted data with reference data.

Configure Hardware

Your Speedgoat target machines include at least two Ethernet cards installed, one of them an Intel 82574 Ethernet card. Use the Intel 82574 Ethernet card for the PTP network. Use the other to connect the two target computers through the Ethernet switch to the development computer. The network looks like this figure.



Required Information

To configure the network and your models for this example, collect the following information for each target computer:

- Identifiers
- *PTP card:* Device name, PCI bus and slot numbers, MAC address that you assign to the PTP card that is transmitting non-PTP data
- *COM card:* Device name, PCI bus and slot numbers, Ethernet index of the card

You can find the built-in MAC address of the PTP card from a source such as a bill of materials or a label on the hardware. You can find the device name and the PCI bus and slot numbers by using SimulinkRealTime.target.getPCIInfo. You can find the Ethernet index of the communication card by using SimulinkRealTime.getTargetSettings.

Example Information

TargetPC1

Identifier — TargetPC1

PTP card

- Device name — `Intel 82574L`
- PCI bus — `5`
- PCI slot — `0`
- MAC Address — [EEPROM}

COM card

- Device name — `Intel 82579LM`
- Ethernet index — `0`
- PCI bus — `0`
- PCI slot — `25`
- MAC address — N/A

TargetPC2

Identifier — TargetPC2

PTP card

- Device name — `Intel 82574L`
- PCI bus — `0`
- PCI slot — `52`
- MAC Address — `68:05:CA:31:B9:EF`

COM card

- Device name — `Intel 82541GI_LF`
- Ethernet index — `0`
- PCI bus — `16`
- PCI slot — `4`
- MAC address — N/A

Hardware Configuration

1. Connect an Ethernet cable between the PTP card in `TargetPC1` and the PTP card in `TargetPC2`. This connection creates the PTP network. To configure the IEEE 1588 Ethernet blocks in the two real-time applications, you must have the PCI bus and PCI slot of these cards.
2. Connect an Ethernet cable from the Comm card in `TargetPC1` to the Ethernet switch.
3. Connect an Ethernet cable from the Comm card in `TargetPC2` to the switch.
4. Connect an Ethernet cable from the switch to the development computer. These connections complete the communication network between the development computer and the target computers.
5. Start the two target computers. Use slrtexplr to connect to them. If you cannot establish communication with a target computer, verify the Ethernet index assigned to the communication port.

Configure Real-Time Applications

In this example, the system contains two real-time applications running on separate target computers. The data acquisition application transmits PTP and non-PTP data to the data analysis application. To configure the data acquisition application, you must have the MAC address of the PTP card that is installed in the data analysis target computer. The data analysis application transmits only PTP data to the data acquisition application. To configure the data analysis application, you can use the MAC address stored in the EEPROM of the PTP card in the data acquisition target computer.

Configure Data Acquisition Application

To configure this application, first perform the steps in Configure Hardware. Collect the PCI bus, PCI slot, and role of the Ethernet cards installed in the target computers.

The data acquisition application is `ex_ptp_sync_src` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_ptp_sync_src')))`). It runs on `TargetPC1`. To configure the application:

Set Configuration Parameters

1. Open `ex_ptp_sync_src`.
2. In the Configuration Parameters dialog box, open the **Solver** pane.

3. Verify that **Type** is `Fixed-step`.
4. Verify that **Fixed-step size (fundamental sample time)** is set to an explicit value and not to `auto` (best practice). Use the same sample time as in the data analysis application.
5. Verify that **Allow tasks to execute concurrently on target** is set.
6. Take the defaults for the other settings.
7. Open the Simulink Real-Time Options pane.
8. Verify that **Build for default target computer** is cleared.
9. Verify that **Specify target computer name** is `TargetPC1`.
10. Verify that **Execution mode** is `Real-Time`.
11. Take the defaults for the other settings.

Configure PTP Blocks

1. Open the IEEE 1588 Ethernet block
2. Open the General pane.
3. From the information for the `TargetPC1` PTP card, enter the values `5` and `0` for **PCI bus** and **PCI slot**.
4. Verify that the IEEE 1588 Ethernet **Sample time** value is a multiple of the **Fixed-step size (fundamental sample time)** value.
5. Verify that **Sample time** has the same value as in the data analysis model.
6. Open the **Network parameters** pane.
7. From the information for the `TargetPC1` PTP card, in the **Source MAC address** box, select `EEPROM`.
8. Verify that **Destination MAC address** is `Standard PTP multicast`.
9. Open the **Clock parameters** pane.
10. Verify that **Timescale (epoch)** is `PTP (1970-01-01)`.
11. Verify that **Delay measurement mechanism** is `Request-response`.
12. Set the **Slave only** check box. This setting prevents the software from making this node the master PTP clock node.
13. Open the **Time intervals** pane.
14. Verify that **Announce interval (second)**, **Sync interval (second)**, and **Min delay or pdelay request interval (second)** are at least three times the **Sample time** value.

15 Verify that the intervals are integral multiples of **Sample time**.

16 Verify that the intervals have the same settings as in the data analysis model.

17 In the remaining top-level PTP blocks, verify that the **Sample time** value matches that in the IEEE 1588 Ethernet block.

18 Open PTP Clock-Data Subsystem. For each block in the subsystem, verify that the **Sample time** value matches that in the IEEE 1588 Ethernet block.

Configure Data Communication Blocks

1 From the information for the `TargetPC2` PTP card, in the Create Ethernet Packet block dialog box, set **Destination MAC** to `macaddr('68:05:CA:31:B9:EF')`.

2 Set **EtherType (use 0 for length)** to `hex2dec('0010')`. Using this type distinguishes the data-specific messages from the PTP-specific messages, which use the same Ethernet card.

3 In the Ethernet Tx block, verify that the **Sample time** value matches that in the IEEE 1588 Ethernet block.

4 Save the updated model in your working folder. You cannot build and run a real-time application in the examples folder.

Configure Data Analysis Application

To configure this application, first perform the steps in Configure Hardware. Collect the PCI bus, PCI slot, and role of the Ethernet cards that are installed in the target computers. To configure the PTP card in the data analysis target computer, use the MAC address that you specified in the Create Ethernet Packet block of the data acquisition application.

The data analysis application is `ex_ptp_sync_sink` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_ptp_sync_sink')))`). It runs on `TargetPC2`. To configure the application:

Set Configuration Parameters

1 Open `ex_ptp_sync_sink`.

2 In the Configuration Parameters dialog box, open the **Solver** pane.

3 Verify that **Type** is `Fixed-step`.

4 Verify that **Fixed-step size (fundamental sample time)** is set to an explicit value (best practice) and not to `auto`. Use the same sample time as in the data acquisition application.

5. Verify that **Allow tasks to execute concurrently on target** is set.
6. Take the defaults for the other settings.
7. Open the **Simulink Real-Time Options** pane.
8. Verify that **Build for default target computer** is cleared.
9. Verify that **Specify target computer name** is `TargetPC2`.
10. Verify that **Execution mode** is `Real-Time`.
11. Take the defaults for the other settings.

Configure PTP Blocks

1. Open the IEEE 1588 Ethernet block.
2. Open the **General** pane.
3. From the information for the `TargetPC2` PTP card, enter the values `52` and `0` for **PCI bus** and **PCI slot**.
4. Verify that the IEEE 1588 Ethernet Sample time value is a multiple of the Fixed-step size (fundamental sample time) value.
5. Verify that Sample time has the same value as in the data acquisition IEEE 1588 Ethernet block.
6. Open the **Network parameters** pane.
7. In the **Source MAC address** box, select `Specify`.
8. From the information for the `TargetPC2` PTP card, in the **Specify source MAC address** text box, enter macaddr('68:05:CA:31:B9:EF'). You can enter an arbitrary MAC address in this box, provided it is unique in the PTP network.
9. Verify that **Destination MAC address** is `Standard PTP multicast`.
10. Open the **Clock parameters** pane.
11. Verify that **Timescale (epoch)** is `PTP (1970-01-01)`.
12. Verify that **Delay measurement mechanism** is `Request-response`.
13. Verify that **Slave only** is cleared. This setting allows the software to make this node the master PTP clock node.
14. Open the **Time intervals** pane.
15. Verify that **Announce interval (second)**, **Sync interval (second)**, and **Min delay or pdelay request interval (second)** are at least three times the Sample time value.

16 Verify that the intervals are integral multiples of **Sample time**.

17 Verify that the intervals have the same settings as in the data acquisition model.

18 In the remaining top-level PTP blocks, verify that the **Sample time** value matches that in the IEEE 1588 Ethernet block.

19 Open PTP Clock-Data Subsystem. For each block in the subsystem, verify that the **Sample time** value matches that in the IEEE 1588 Ethernet block.

Configure Data Communication Blocks

1 Open PTP Clock-Data Subsystem

2 Open the Ethernet Rx block.

3 Open the **Rx** pane and verify that the **Sample time** value matches that in the IEEE 1588 Ethernet block.

4 Open the **Filter** pane.

5 Verify that **Filter criteria** is `Specify types to match`.

6 Verifty that **Receive these types (vector of types 0-65535)** is `[hex2dec('0010')]`.

7 Save the updated model in your working folder. You cannot build and run a real-time application in the examples folder.

Build, Download, and Run Real-Time Applications

In this example, the data acquisition application builds and is downloaded to `TargetPC1`. The data analysis application builds and is downloaded to `TargetPC2`. To run these applications, first perform the steps in Configure Real-Time Applications. MATLAB® and Simulink Real-Time Explorer must be running in your working folder.

Build, download, and run the applications:

1 Start `TargetPC1` and `TargetPC2`.

2 In the Explorer Targets pane, connect to `TargetPC1` and `TargetPC2`.

3 In your working folder, open `ex_ptp_sync_src` and `ex_ptp_sync_sink`.

4 Build and download `ex_ptp_sync_src` to `TargetPC1`.

5 Build and download `ex_ptp_sync_sink` to `TargetPC2`.

6 In the Explorer **Applications** pane, for `TargetPC1/ex_ptp_sync_src` and `TargetPC2/ex_ptp_sync_sink`, change the property **Stop Time** to `Inf`.

7. In the Explorer **Applications** pane, start `ex_ptp_sync_src` and `ex_ptp_sync_sink`.
8. For both applications, waveform data starts streaming in the target scopes labeled `Data`. However, the timestamps displayed as signal `Time` are not initially valid. The two applications go through the following sequence of `Sync Status` and `State` values:

1. Initialization:

- `ex_ptp_sync_src State → 4` (LISTENING)
- `ex_ptp_sync_src Sync Status → 0` (not synchronized)
- `ex_ptp_sync_sink State → 4` (LISTENING)
- `ex_ptp_sync_sink Sync Status → 0` (not synchronized)

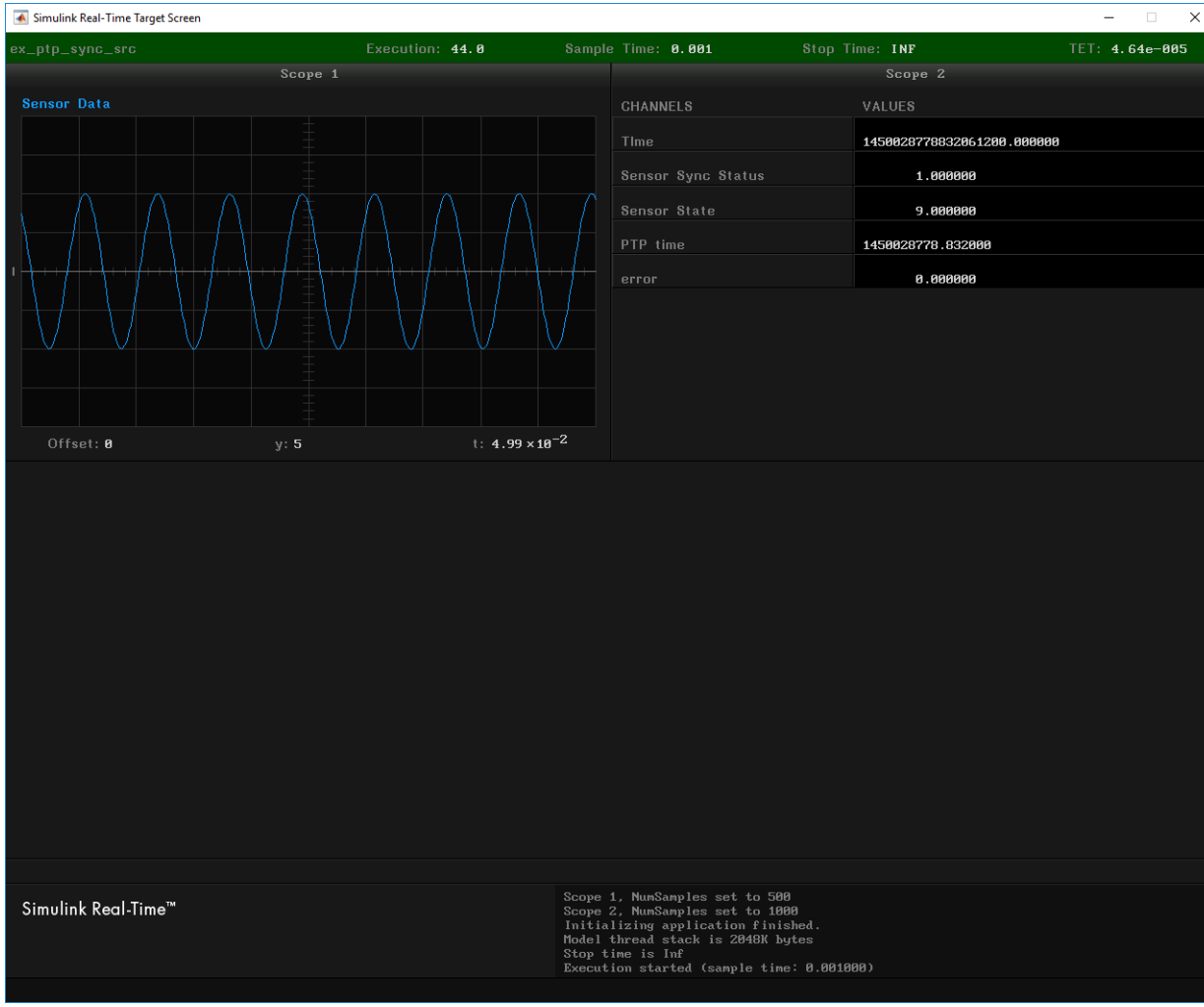
2. Master allocation and synchronization

- `ex_ptp_sync_sink State → 6` (MASTER)
- `ex_ptp_sync_sink Sync Status → 1` (synchronized)

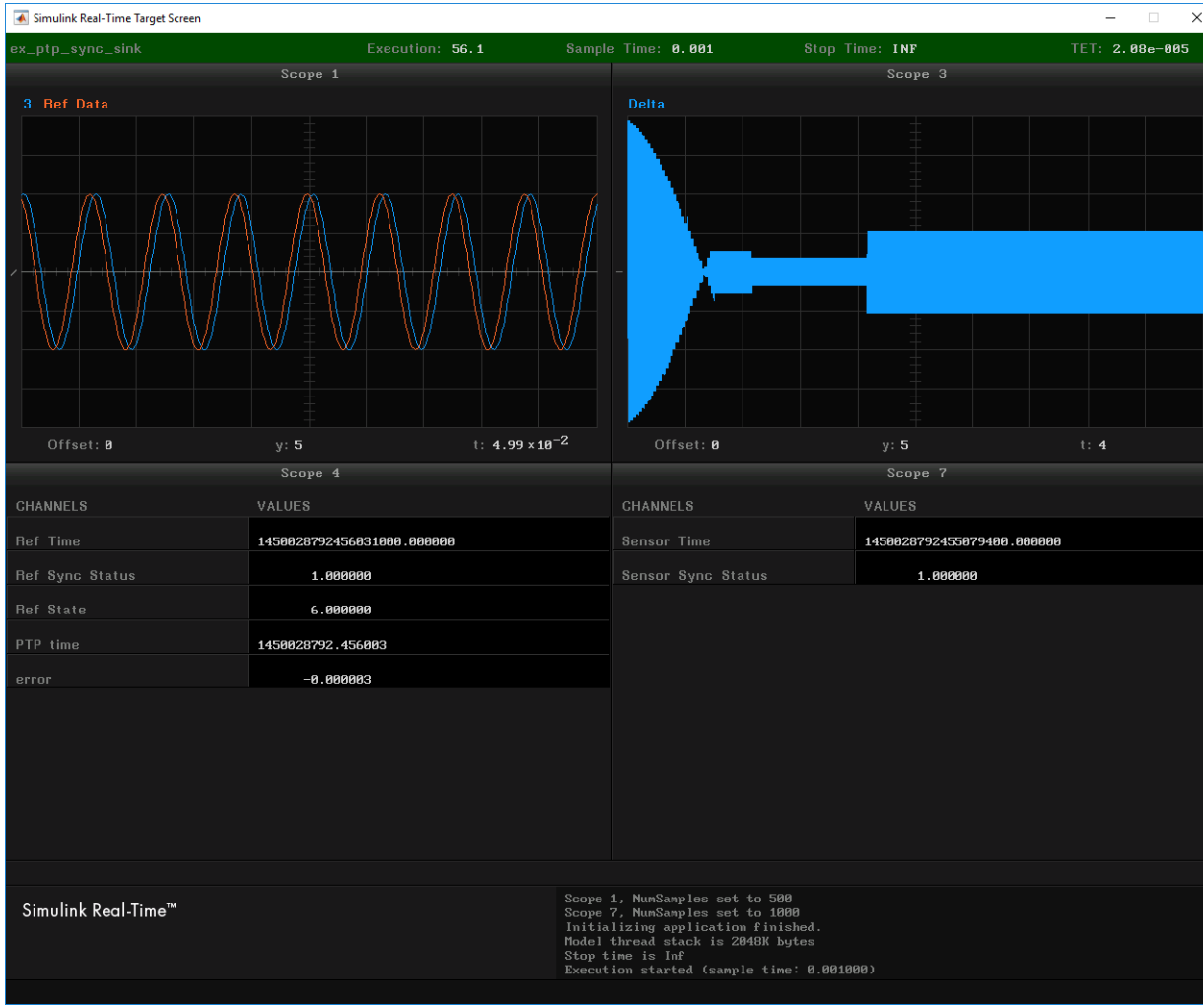
3. Slave allocation and synchronization

- `ex_ptp_sync_src State → 9` (SLAVE)
- `ex_ptp_sync_src Sync Status → 1` (synchronized)

For the data acquisition node (the slave node), the final state looks like this figure.



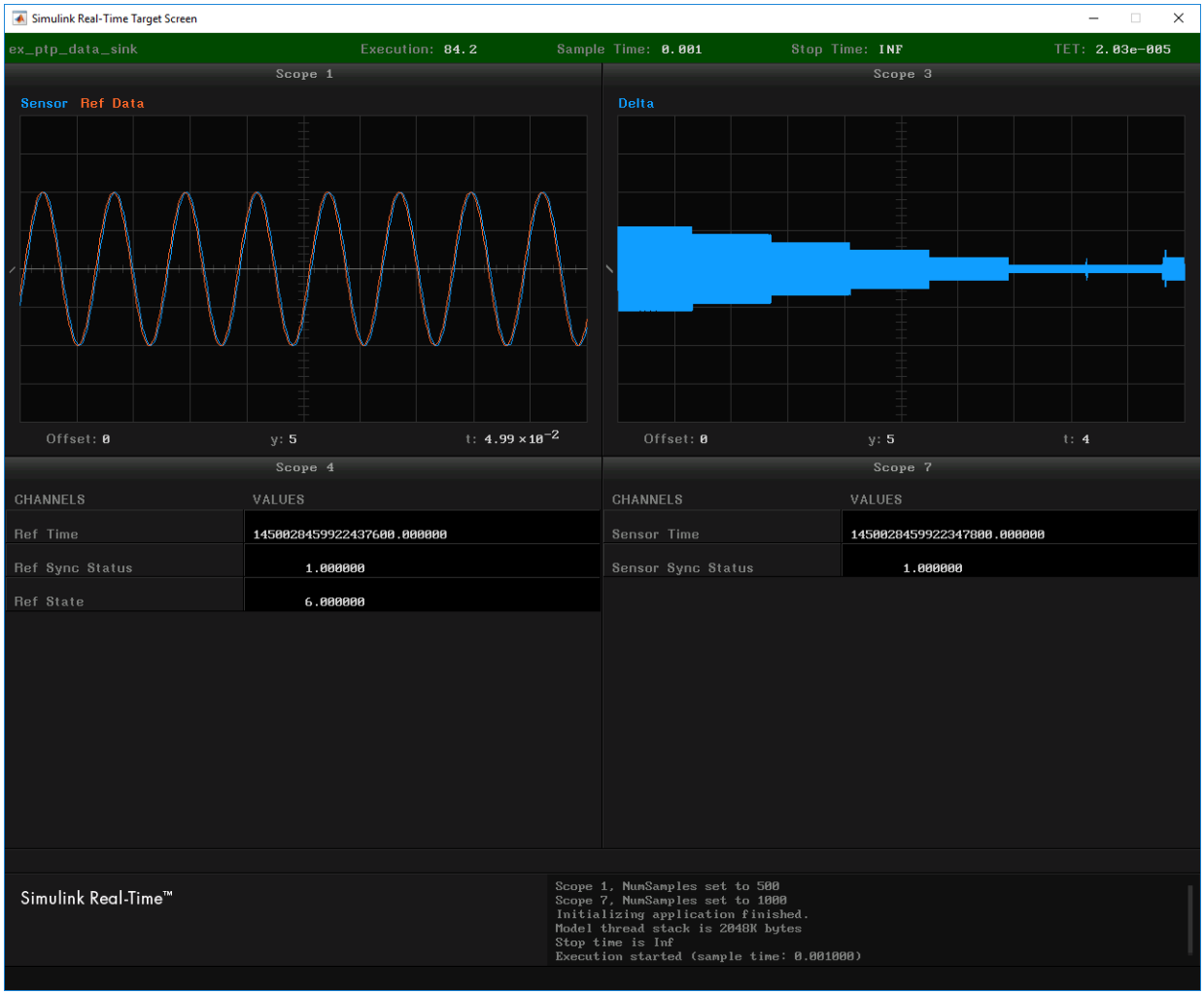
For the data analysis node (the master node), the final state looks like this figure.



For this example, the sensor and reference Sine Wave blocks are set to the same frequency and amplitude, but start at arbitrary times. The difference in start time causes a phase difference between the sine waves. The phase difference appears on the `Delta` scope as a waveform that settles to a constant amplitude.

The phase difference is constant because the IEEE 1588 Sync Execution blocks synchronize the kernel clocks on the two target computers. If you do not include these

blocks, the kernel clocks of the two target computers drift apart. As a result, the `Delta` waveform shows a beat frequency.



With the IEEE 1588 Sync Execution block, you can make measurements across multiple target computers at a synchronized time step. However, the kernel interrupt clock

controller can shorten some time steps up to 10% of the fundamental sample time, resulting in a CPU overload.

See Also

IEEE 1588 Sync Execution | IEEE 1588 Ethernet | IEEE 1588 Read Parameter | IEEE 1588 Sync Status | open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_ptp_data_sink'))) | open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_ptp_data_src'))) | open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_ptp_sync_sink'))) | open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_ptp_sync_src')))

More About

- “Data Acquisition and Data Analysis Example Description” on page 17-18
- “IEEE® 1588™ Precision Time Protocol - Execution Synchronization”
- “Precision Time Protocol” on page 17-2

Data Acquisition and Data Analysis Example Description

This example features a data acquisition target computer that transmits timestamped sensor data to a second target computer. The second target computer processes the sensor data and displays the data and the difference between the sensor data and reference data. Both target computers connect to a development computer. The development computer runs Simulink and Simulink Real-Time Explorer to build, download, and run real-time applications on each target computer. The real-time applications use Precision Time Protocol (PTP) blocks to synchronize the PTP clocks and kernel clocks on each computer.

In this section...
“Data Acquisition Application” on page 17-18
“Data Analysis Application” on page 17-21

Data Acquisition Application

The data acquisition application is a PTP slave node that acquires data from a sensor, which a Sine Wave block represents. The application transmits the sensor data to the data analysis application.

Top-Level Model

The PTP initialization block and the blocks that create and transmit the Ethernet packet are in the top-level model:

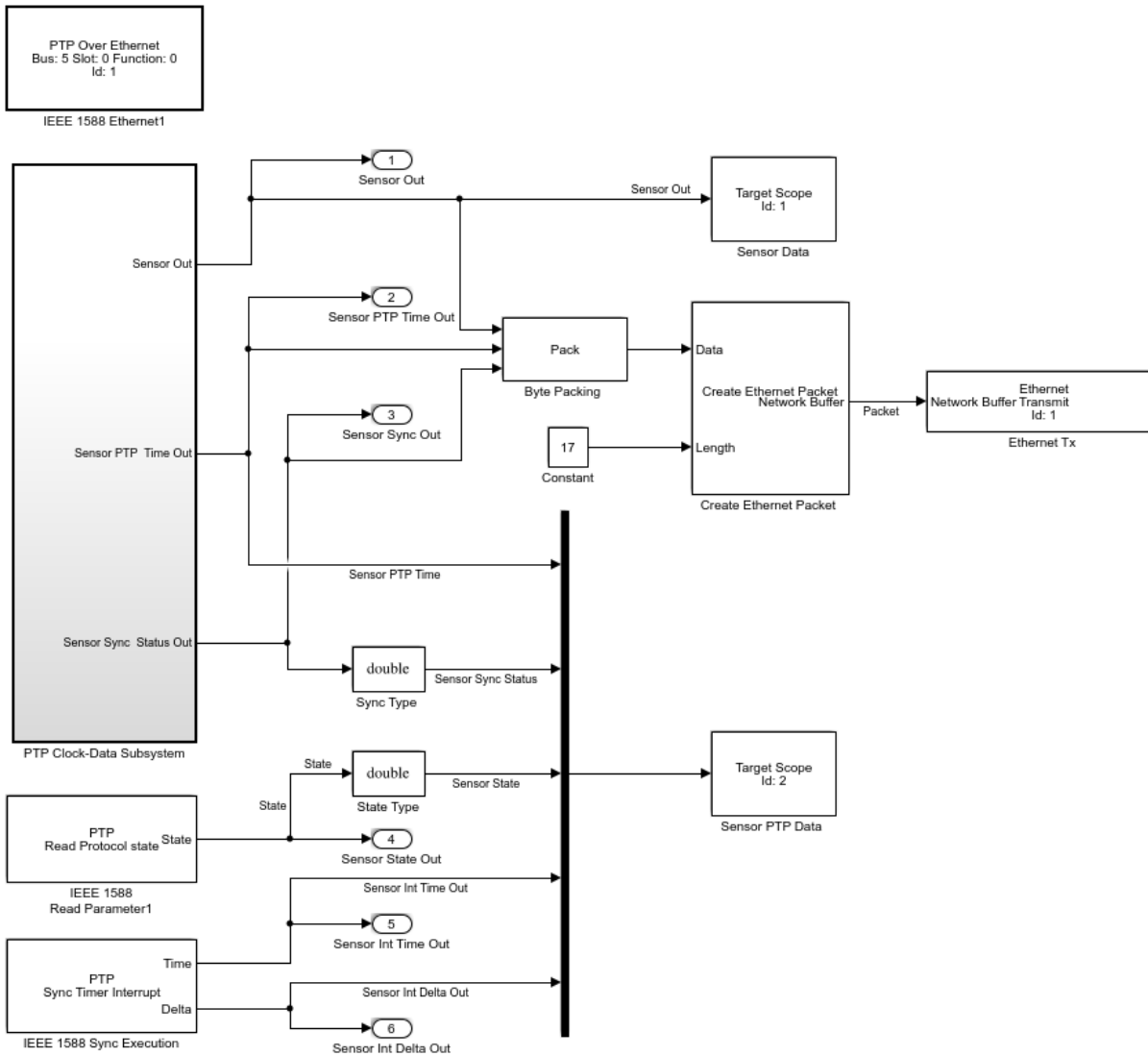
- IEEE 1588 Ethernet — Configures the PTP network card clock as a slave clock.
- IEEE 1588 Read Parameter — Shows when the PTP clock has been allocated as a slave clock (value 9). Configured as `Read Protocol state`.
- IEEE 1588 Sync Execution — Aligns the kernel clocks across multiple target computers. The block output shows the difference between the following times:
  - The PTP time at the real-time interrupt
  - The nearest PTP time that is a multiple of the fundamental sample time
- Byte Packing — Packs the sensor data, timestamp, and synchronization status into an Ethernet packet.
- Create Ethernet Packet — Addresses the Ethernet packet to the MAC address of the data analysis application.

- Ethernet Tx — Transmits the Ethernet packet to the data-analysis target computer.

  The Ethernet Tx block sends data packets through the same Ethernet connection as the PTP blocks use to send PTP messages. To distinguish data packets from PTP messages, the model assigns to the data packets Ethernet type `hex2dec('0010')`. This Ethernet type is different from the default Ethernet type of PTP packets (`hex2dec('88F7')`).
- Outport — For data logging purposes, the top-level model propagates the major signals to Outport blocks.

For debugging purposes, the top-level model includes two real-time Scope blocks:

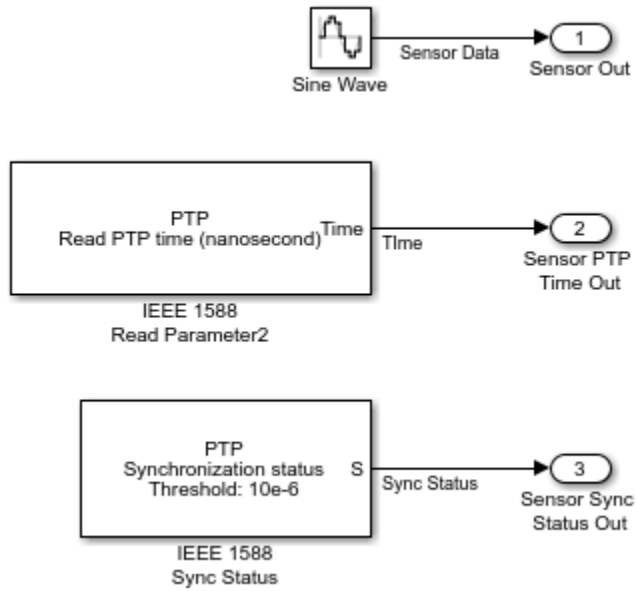
- `Sensor Data` — Displays the sensor data in a graphic scope.
- `Sensor PTP Data` — Displays the PTP time, PTP synchronization status, PTP state, and synchronization delta of the data acquisition model.


PTP Over Ethernet
Bus: 5 Slot: 0 Function: 0
Id: 1
IEEE 1588 Ethernet1
1
Sensor Out
Sensor Out
Target Scope
Id: 1
Sensor Data
Sensor Out
2
Sensor PTP Time Out
Pack
Byte Packing
Data
Create Ethernet Packet
Network Buffer
Length
Packet
Ethernet
Network Buffer Transmit
Id: 1
Ethernet Tx
3
Sensor Sync Out
Sensor PTP Time Out
17
Constant
Create Ethernet Packet
Sensor PTP Time
Sensor Sync Status Out
double
Sensor Sync Status
Sync Type
PTP Clock-Data Subsystem
State
double
Sensor State
State Type
Target Scope
Id: 2
Sensor PTP Data
State
PTP
Read Protocol state
State
4
Sensor State Out
IEEE 1588
Read Parameter1
Sensor Int Time Out
5
Sensor Int Time Out
Time
PTP
Sync Timer Interrupt
Delta
Sensor Int Delta Out
6
Sensor Int Delta Out
IEEE 1588 Sync Execution

Atomic Subsystem

The PTP timestamp must align as closely as possible with the data source. For better alignment, the model wraps the sensor data block and the lower-level PTP blocks in an atomic subsystem:

- Sine Wave — Represents sensor data.
- IEEE 1588 Read Parameter — Generates the timestamp, configured as `PTP Time (nanosecond)`.
- IEEE 1588 Sync Status — Generates the synchronization status. When the PTP clock is synchronized with the master PTP clock, the block output becomes `1`.



Data Analysis Application

The data analysis application is a PTP master node that gets sensor data from an emulator, a Sine Wave block. The application gets reference data from an emulator, a Sine Wave block, and sensor data from an Ethernet Rx block. The application calculates the difference between the reference data and the sensor data.

Top-Level Model

The PTP initialization block and the blocks that receive and process the data are in the top-level model:

- IEEE 1588 Ethernet — Configures the PTP network card clock as a master clock.
- IEEE 1588 Read Parameter — Shows when the PTP clock has been allocated as a master clock (value `6`). Configured as `Read Protocol state`.
- IEEE 1588 Sync Execution — Aligns the kernel clocks across multiple target computers. The block output shows the difference between the following times:
  - The PTP time at the real-time interrupt
  - The nearest PTP time that is a multiple of the fundamental sample time
- Extract Ethernet Packet — Extracts the Ethernet packet that is carrying the sensor data.
- Byte Unpacking — Unpacks the sensor data, timestamp, and synchronization status from the Ethernet packet.
- Sum — Calculates the difference between the sensor data and the reference data.

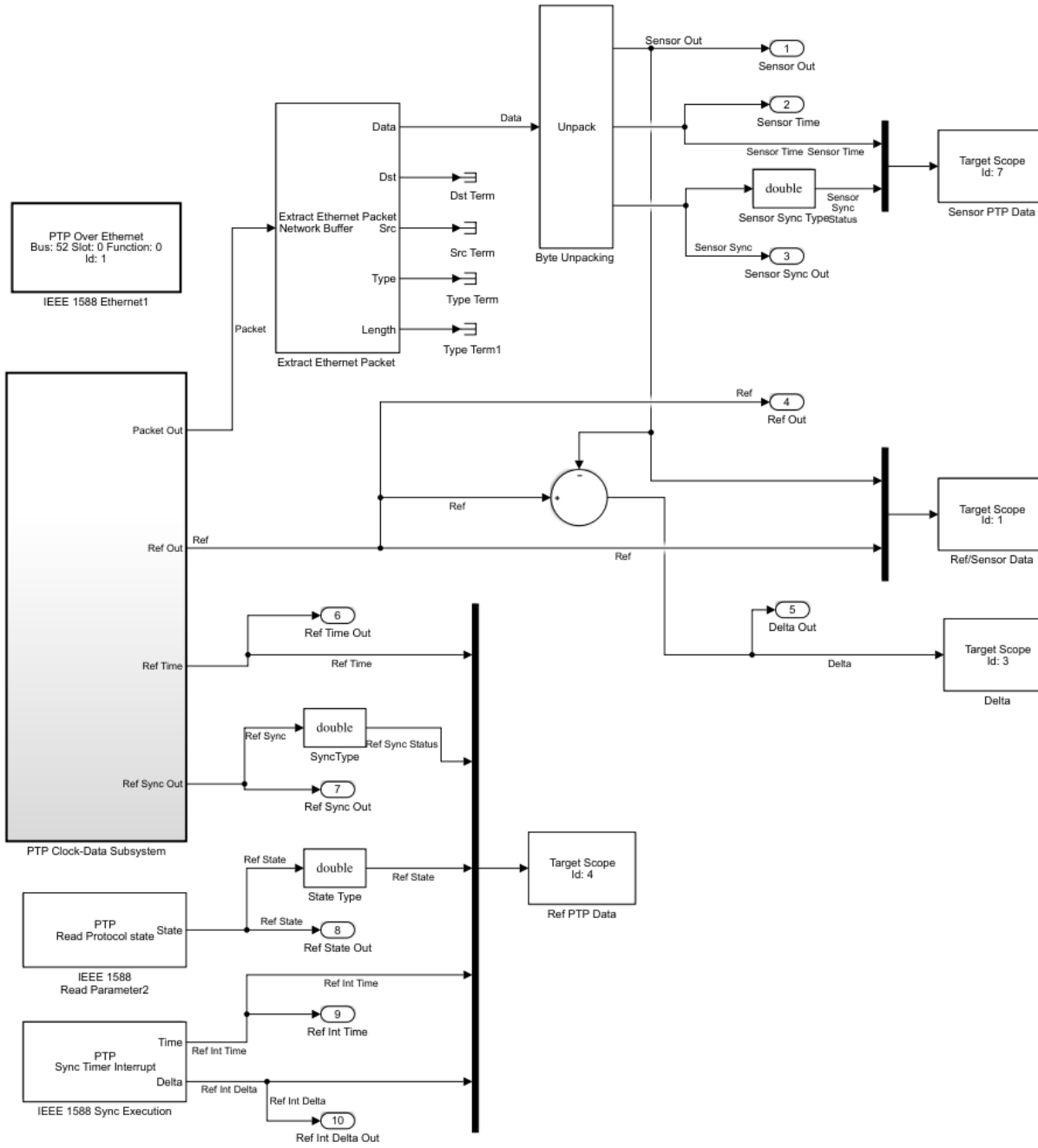
  The Sum block provides input data for further processing. For example, you can plot the sensor data, reference data, and difference against the timestamp to assess the real-time behavior. You can also feed the difference data back through a control system to change an actuator setting at the data acquisition site.
- Outport — For data logging purposes, the top-level model propagates the major signals to Outport blocks.

For debugging purposes, the top-level model includes four real-time Scope blocks:

- `Ref/Sensor Data` — Displays the reference data and the sensor data together in a graphic scope.
- `Delta` — Displays the difference between the reference data and the sensor data in a graphic scope.

  The `Delta` scope is configured with a long sample time. It captures long-period differences between the sensor and reference data. If the frequency, phase, and amplitude differences are constant, the scope displays a rectangular area. If the differences are periodic, the scope displays a beat frequency.
- `Ref PTP Data` — Displays the PTP time, PTP synchronization status, PTP state, and synchronization delta of the data analysis model.

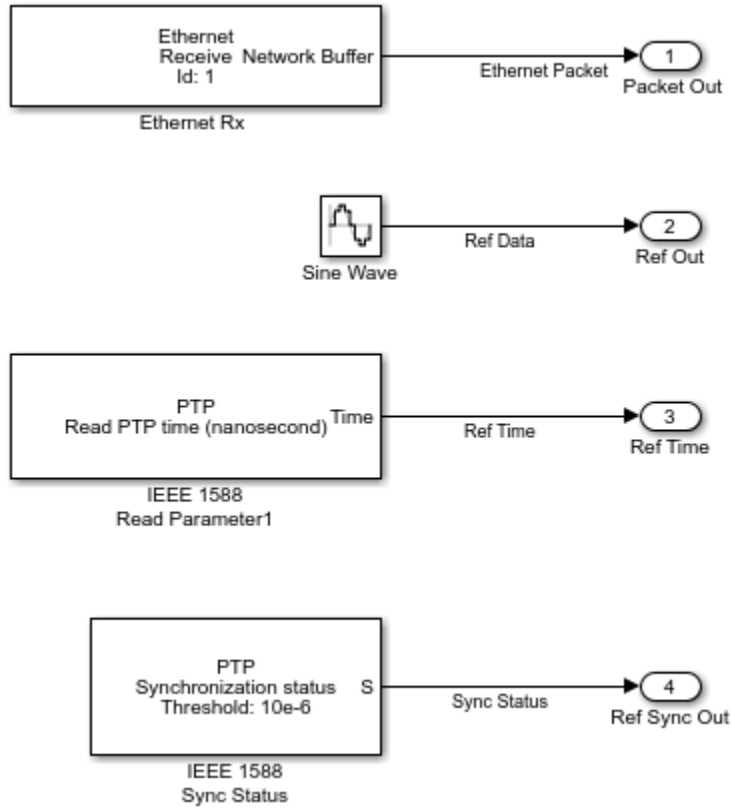
- `Sensor PTP Data` — Displays the PTP time, PTP synchronization status, and synchronization delta of the data acquisition model.


PTP Over Ethernet
Bus: 52 Slot: 0 Function: 0
Id: 1
IEEE 1588 Ethernet1
Extract Ethernet Packet
Network Buffer
Data
Dst
Src
Type
Length
Dst Term
Src Term
Type Term
Type Term1
Extract Ethernet Packet
Packet
Data
Unpack
Byte Unpacking
Sensor Out
1
Sensor Out
2
Sensor Time
Sensor Time Sensor Time
Target Scope
Id: 7
Sensor PTP Data
double
Sensor Sync Type
Sensor Sync Status
Sensor Sync
3
Sensor Sync Out
Packet Out
Ref Out
Ref
4
Ref Out
Ref
Target Scope
Id: 1
Ref/Sensor Data
5
Delta Out
Target Scope
Id: 3
Delta
Ref Time
6
Ref Time Out
Ref Time
Ref Sync
double
SyncType
Ref Sync Status
Ref Sync Out
7
Ref Sync Out
PTP Clock-Data Subsystem
Ref State
double
State Type
Ref State
Target Scope
Id: 4
Ref PTP Data
PTP
Read Protocol state
State
IEEE 1588
Read Parameter2
Ref State
8
Ref State Out
Ref Int Time
9
Ref Int Time
Time
PTP
Sync Timer Interrupt
Delta
Ref Int Time
Ref Int Delta
IEEE 1588 Sync Execution
Ref Int Delta
10
Ref Int Delta Out

Atomic Subsystem

The PTP timestamp must align as closely as possible with the Ethernet receiver. For better alignment, the model wraps the blocks representing the reference data source and the lower-level PTP blocks in an atomic subsystem:

- Sine Wave — Represents reference data.
- IEEE 1588 Read Parameter — Generates the timestamp, configured as `PTP Time (nanosecond)`.
- IEEE 1588 Sync Status — Generates the synchronization status. When the PTP clock is synchronized with the master PTP clock, the block output becomes `1`.
- Ethernet Rx — Receives sensor data from the acquisition target computer. The configured block filters out all packets except packets of Ethernet type `hex2dec('0010')`. The default Ethernet type of PTP packets is `hex2dec('88F7')`.



See Also

IEEE 1588 Sync Execution | IEEE 1588 Ethernet | IEEE 1588 Read Parameter | IEEE 1588 Sync Status | open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_ptp_sync_sink'))) | open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_ptp_sync_src')))

More About

- “Synchronize Timestamps Across Data-Gathering Network” on page 17-5
- “IEEE® 1588™ Precision Time Protocol - Execution Synchronization”

Troubleshoot Precision Time Protocol Configuration

I seek to resolve `IEEE 1588` block precision time protocol (PTP) configuration problems.

What This Means

To troubleshoot your model, first familiarize yourself with the PTP standard, and then with the specialized requirements of the Simulink Real-Time implementation. For more information, see “Precision Time Protocol” on page 17-2.

Try This

To identify PTP model or configuration problems, check for the following issues.

PTP Block Configuration in Model

- Configure all PTP nodes in a network with the same **Delay measurement mechanism**. If you configure a slave node with a different setting from the master node, the slave node enters the `FAULTY` state
- Configure all PTP nodes in a network with the same **Timescale** or **Arbitrary timescale epoch** value. If you configure the master and slave nodes with differing timescales, the representation of time in time-of-day format differs for the two nodes.
- Configure all nodes in the PTP networks with the same **Announce interval** and **Announce receipt timeout**. Differing values of these parameters in a PTP network can lead to unpredictable behavior.
- Avoid using inherited sample time everywhere in your model. Inherited sample time occurs throughout your model when you make the following settings:

  - **Fixed step size** → `auto` in the Configuration Parameters dialog box
  - **Sample time** → `-1` in all of the blocks of your model.

  The sample time that Simulink calculates can be greater than the PTP message transmission intervals, resulting in an unusable model.
- The PTP configuration subsystems include configuration blocks for the associated transport protocol. If you use a separate Ethernet card for data transmission, include a separate network configuration block. Assign it a **Device ID** different from the one already in use by the PTP configuration block. Multiple network configuration blocks with the same **Device ID** cause a build error.

- The PTP Over Ethernet block creates PTP messages with **Ether type** set to `hex2dec('88F7')`. To use the same Ethernet card for PTP as for data transmission:
  - In the Create Ethernet Packet block, set **Ether type** to a nonzero value that is different from `hex2dec('88F7')` (for example, `hex2dec('0010')`.
  - In the Ethernet Rx block, set **Filter criteria** to `Specify types to match`. Set **Receive these types** to the value that you set in the Create Ethernet Packet block (for example, `[hex2dec('0010')]`).
- If you include more than one slave node in the network, configure the master node to use the standard PTP multicast address for transmitting messages. The master node must transmit the same synchronization message to all the slaves.
- Using the IEEE 1588 Sync Execution block to make measurements across multiple target computers at the same simulation step can lead to a CPU overload. Also, the kernel interrupt clock controller can shorten some time steps up to 10% of the model fundamental sample time.

  If you get CPU overloads, consider decreasing the value of the **Proportional gain** parameter of the IEEE 1588 Sync Execution block or increasing the sample time of your real-time application.
- If you use the IEEE 1588 Sync Execution block in your model, configuring EtherCAT distributed clocks in master shift mode in the same model produces a build error. To include IEEE 1588 synchronized execution and EtherCAT distributed clocks in the same model, use EtherCAT bus shift mode.

PTP Synchronization Accuracy

- The synchronization accuracy depends upon the value of **Sync interval**. The smaller the value, the more accurate the synchronization. If your model fails to meet your required synchronization accuracy, try decreasing the value of **Sync interval**.
- You can use IEEE 1588 Sync Execution block to synchronize two PTP models with differing fundamental sample times. Their execution is synchronous at a PTP time equal to the least common multiple of the two rates.

PTP Faulty States

- When a slave node enters the `FAULTY` state, look for one of the following conditions:
  - The slave node is configured with a different **Delay measurement mechanism** setting from the master node setting.

  - The slave node model sample time setting is greater than the master node **Sync interval** setting.
  - The slave node **Announce interval** setting is shorter than the master node **Announce interval** setting.
  - The slave is not receiving a response from the master to delay request messages sent by the slave. This behavior occurs, for example, if the slave node is configured to use a delay measurement mechanism setting different from the master node setting.
- If the master node fails to read a required timestamp from the Ethernet card due to contention for the timestamp register, the transmission can fail. After a master node fails five consecutive times to transmit a `Follow_Up`, `Delay_Resp`, `Pdelay_Resp`, or `Pdelay_Resp_Follow_Up` message to its slave nodes, it enters the `FAULTY` state. Try the following:

  - Reduce the number of slave nodes in the network.
  - Shorten the sample time for the subsystem that represents the master node. The master node cycles through the slave messages faster and reads the timestamp register more often.
  - Increase the **Min delay or pdelay request interval** of the slave nodes. The slave nodes generate messages less often.
  - Connect a peer-to-peer transparent PTP clock between the master and slave nodes. Set **Delay measurement mechanism** to `Peer-delay` for all of the nodes. The peer-to-peer transparent PTP clock has a separate timestamp register for each port, taking the load off the master node.

    For more information, see IEEE Std 1588-2008 Clause 10.

See Also

IEEE 1588 Adjust Time | IEEE 1588 Create Message | IEEE 1588 Ethernet | IEEE 1588 Process Message | IEEE 1588 Read Parameter | IEEE 1588 Real-Time UDP | IEEE 1588 Setup | IEEE 1588 Sync Error | IEEE 1588 Sync Execution | IEEE 1588 Sync Status

More About

- “Prerequisites, Limitations, and Unsupported Features” on page 17-31
- “Synchronize Timestamps Across Data-Gathering Network” on page 17-5

- “IEEE® 1588™ Precision Time Protocol - Execution Synchronization”

External Websites

- standards.ieee.org

Prerequisites, Limitations, and Unsupported Features

The Simulink Real-Time implementation of PTP enforces specific requirements and limitations.

In this section...

Prerequisites

- PTP functionality is available only with a Speedgoat target computer. If you have not installed the Speedgoat library, attempting to build a real-time application with PTP causes a build error.
- Simulink Real-Time supports PTP only with Intel 82574 Ethernet cards. To check that you have the required card, start your target computer. In the Command Window, type:

  ```
  tg = slrt;
  getPCIInfo(tg, 'Ethernet')
  ```

  Check that you see an entry like this entry:

  ```
  Intel                    82574L
       Bus 5, Slot 0, IRQ 10
       Ethernet controller
       VendorID 0x8086, DeviceID 0x10d3, SubVendorID 0x15bd,
             SubDeviceID 0x100a
       Released in: R2010a
       Notes: Intel 8254x Gigabit Ethernet series
  ```

Limitations

- The PTP network card clock acts as PTP clock. Only one clock is allowed per node.
- Run the model in `Real-Time` execution mode, not in `Freerun` mode or driven by an external interrupt. In the latter two cases, the PTP message transmission intervals can violate the PTP standard.

- You can include only one PTP configuration block in a model. You can run only one real-time application on a target computer. If you have installed multiple PTP Ethernet cards on your target computer, you can use only one of them for PTP at a time. You can use the other PTP Ethernet cards for non-PTP purposes.
- The PTP message transmission intervals (**Announce interval**, **Sync interval**, and **Min delay or pdelay request interval**) must be greater than the block sample time. Too small a message transmission interval causes a model update error.
- Simulink Real-Time can transmit PTP messages only at a multiple of the block sample time. If a transmission interval is not a multiple of the block sample time, PTP transmits the messages at the nearest multiple to the specified transmission time. As a best practice, specify all transmission intervals as integral multiples of the block sample time.
- The specification requires that a PTP node issue messages within ±30% of the message transmission intervals at least 90% of the time. To meet this requirement, specify message transmission intervals (**Announce interval**, **Sync interval**, and **Min delay or pdelay request interval**) at least three times the base sample time.
- The following factors limit accuracy:

  - Network protocol stack delay fluctuation
  - Network technology component delay fluctuations (switches, routers)
  - Clock timestamp accuracy
  - Clock oscillator stability

  Use components that minimize these factors. For example, you can use a transparent or boundary PTP clock to increase synchronization accuracy.

  - A transparent PTP clock tracks the amount of time a PTP message takes to go through the device. It passes that information to nodes receiving the message.
  - A boundary PTP clock has multiple PTP ports that can act as a master clock or a slave clock.
- Some systems require a PTP time source that is traceable to an International Atomic Time (TAI) clock, such as a GPS signal. To support traceability, acquire a third-party grandmaster PTP clock that provides this capability. In that case, a Simulink Real-Time target computer running PTP acts only as a slave clock.

Unsupported Features

- Simulink Real-Time supports only PTP version 2, as defined in IEEE Std 1588-2008. If a Simulink Real-Time PTP node receives a PTP version 1 message as defined in IEEE Std 1588-2002, it ignores it.
- The Simulink Real-Time implementation of PTP does not support the following functionality defined in IEEE Std 1588-2008:
  - PTP variance computation, as described in IEEE Std 1588-2008 Clause 7.6.3.
  - PTP nodes configured as one-step PTP clocks, as described in IEEE Std 1588-2008 Clause 3.1.21.

    You can configure the master node as a two-step clock only, as described in IEEE Std 1588-2008 Clause 3.1.47.
  - PTP management messages, as described in IEEE Std 1588-2008 Clause 15.

    A Simulink Real-Time PTP node cannot transmit a PTP management message. When a Simulink Real-Time PTP node receives a PTP management message, it ignores it.
  - PTP signaling messages, as described in IEEE Std 1588-2008 Clause 13.12.

    A Simulink Real-Time PTP node cannot transmit a PTP signaling message. When a Simulink Real-Time PTP node receives a PTP signaling message, it ignores it.
  - Optional features, as described in IEEE Std 1588-2008 Clause 16 and Clause 17.

See Also

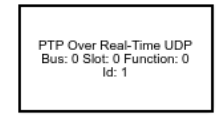
Related Examples

- “Troubleshoot Precision Time Protocol Configuration” on page 17-27

Precision Time Protocol Blocks

IEEE 1588 Real-Time UDP

Execute IEEE 1588 Precision Time Protocol
Library: IEEE 1588



Description

IEEE 1588 Real-Time UDP executes the PTP protocol, using UDP to send and receive the protocol messages. The block communicates with the corresponding blocks on the other target computers, determines the time offset that synchronizes them, and adjusts the time offset.

Parameters

General

`Device ID` — Ethernet board identifier
`1-8`

From the list, select a unique number to identify the Ethernet board.

`PCI bus` — PCI bus number of Ethernet card
`0` (default) | integer

Enter the PCI bus number for the Ethernet card.

`PCI slot` — PCI slot number of Ethernet card
`0` (default) | integer

Enter the PCI slot number for the Ethernet card.

`PCI function` — PCI function number of Ethernet card
`0` (default) | integer

Enter the PCI function number for the Ethernet card.

`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.

Network Parameters

`IP address of port` — IP address of PTP clock board
`x.x.x.x`

IP address of the Ethernet board, or node, carrying the PTP clock.

The addresses `0.0.0.0` and `255.255.255.255` are invalid IP addresses.

`Subnet mask` — Subnet mask for interface
`255.255.255.0` (default) | *`x.x.x.x`*

Mask that designates a logical subdivision of a network.

`Gateway` — IP address for gateway interface
`0.0.0.0` (default) | *`x.x.x.x`*

The gateway must be within the network.

To indicate that a gateway is not being used, enter `0.0.0.0` (the default). The address `255.255.255.255` is an invalid gateway IP address.

`Source IP address of receive packets (set to 0.0.0.0 to receive all)` — IP address in receive blocks
`0.0.0.0` (default) | *`x.x.x.x`*

IP address in UDP Receive blocks. The default value (`0.0.0.0`) specifies that the node is to receive all packets sent to the ports assigned to PTP messages (ports `319` and `320`).

Use a specific value for one-to-one communication. If the node is a master PTP clock node, use a specific value only if exactly one slave is connected to the master clock node.

The address `255.255.255.255` is an invalid IP address.

`Destination IP address of transmit packets` — IP address in transmit blocks
`Standard PTP Multicast (224:0:1:129, 224:0:0:107)` (default) | *`x.x.x.x`*

IP address in UDP Transmit blocks. Specifies the IP address of the other PTP computers or devices to which to send the PTP packets. Select one of:

- `Standard PTP Multicast (224:0:1:129, 224:0:0:107)` (default) — Default standard multicast IP address assigned to PTP. If you select this option, the PTP packets are broadcast to all computers listening on the PTP ports (ports `319` and `320`). The destination IP addresses are:
  - `224.0.1.129` for non-peer-delay measurement mechanism messages (`Announce`, `Sync`, `Follow_up`, `Delay_Req`, `Delay_Resp`)
  - `224.0.0.107` for peer-delay measurement mechanism messages (`Pdelay_Req`, `Pdelay_Resp`, `Pdelay_Resp_Follow_up`)
- `Specify` — Explicitly specify the destination IP address.

Dependency

Selecting `Specify` makes the **Specify destination IP address** parameter visible.

`Specify Destination IP address` — IP address of transmit packets

`255.255.255.255` (default) | *`x.x.x.x`*

The default value (`255.255.255.255`) specifies that the node is to broadcast the packets to all listening nodes of the network. Use a specific value for one-to-one communication. If the node is a master PTP clock node, use a specific value only if exactly one slave is connected to the master clock node.

Dependency

To make this parameter visible, set **Destination IP address of transmit packets** to `Specify`.

Clock Parameters

`Timescale (epoch)` — Origin point of the PTP timescale

`PTP (1970-01-01)` (default) | `GPS (1980-06-01)` | `NTP (1900-01-01)` | `Specify`

Specify the origin point of the PTP timescale. Select one of:

- `PTP (1970-01-01)` — Precision Time Protocol standard epoch, starting January 1, 1970.
- `GPS (1980-06-01)` — Global Positioning System standard epoch, starting June 1, 1980.

- `NTP (1900-01-01)` — Network Time Protocol standard epoch, starting January 1, 1900.
- `Specify` — Explicitly specify the timescale epoch.

Dependency

Selecting `Specify` makes the **Arbitrary timescale epoch (yyyy mm dd hh)** parameter visible.

`Arbitrary timescale epoch [yyyy mm dd hh]` — Explicit origin point for PTP timescale

`[1970 01 01 00]` (default) | `[yyyy mm dd hh]`

Specify the origin point for the PTP timescale, in year, month, day, and hour.

Dependency

To make this parameter visible, set **Timescale (epoch)** to `Specify`.

`Delay measurement mechanism` — Method of measuring link delays

`Request-response` (default) | `Peer-delay`

Specify the method of measuring link delays. Configure all PTP network nodes to use the same link delay measurement mechanism.

For more information, see IEEE Std Clause 7.5.4.

`Slave only` — Node that cannot be allocated as master PTP clock

`off` (default) | `on`

When you select this check box, you cannot allocate the PTP Ethernet card that this block represents as a master PTP clock.

In **Slave only** mode, the values of the advanced parameters (**Priority 1**, **Clock class**, **Clock accuracy**, and **Priority 2**) are set to their highest values. When the parameters have these settings, all of the other nodes must have the same configuration. If a node has a different configuration, the Best Master Clock Algorithm (BMCA) cannot allocate the node as best master clock. If the BMCA selects a **Slave only** node as best clock, the node remains in the `LISTENING` state.

`Show advanced configuration parameters` — Enable low-level PTP configuration parameters

`off` (default) | `on`

For more information, see IEEE Std 1588-2008.

Dependency

Selecting this check box makes advanced configuration parameters visible: **Domain number**, **Current UTC offset**, **Priority 1**, **Clock class**, **Clock accuracy**, and **Priority 2**.

`Domain number` — Domain number of PTP network

`0` (default) | `0`–`127`

Specify the domain number of the PTP network to which the node belongs.

A Simulink Real-Time PTP node can belong to only one PTP domain at a given time. If the node receives a PTP message with a different domain number, it ignores it. For more information, see IEEE Std Clause 7.1.

Dependency

To make this parameter visible, select the **Show advanced configuration parameters** check box.

`Current UTC offset` — Current offset from Coordinated Universal Time

35 (default) | integer

The current UTC offset, in seconds.

If you specify a nonzero value, that value is considered valid. The `UTCOffsetValid` flag is set to `true`. Otherwise, the flag is set to `false`. For more information, see IEEE Std 1588-2008 Clause 7.2.3.

Dependency

To make this parameter visible, select the **Show advanced configuration parameters** check box.

`Priority 1` — Priority of PTP node

`128` (default) | `0`–`255`

Specify an integer value encoding the priority of the PTP node in the network. When the value is `0`, the node has the highest priority. When it is `255`, the node has the lowest priority.

To assess the quality of two PTP clocks, the Best Master Clock Algorithm compares the following parameters, in order:

1. **Priority 1**
2. **Clock class**
3. **Clock accuracy**
4. **Priority 2**

For each parameter, the algorithm selects the clock with the smaller value as the best clock. If all four parameters are equal for both clocks, the algorithm compares the MAC addresses of the nodes.

For more information, see IEEE Std 1588-2008 Clause 7.6.2.2.

Dependency

To make this parameter visible, select the **Show advanced configuration parameters** check box.

`Clock class` — Clock class designator

`248` (default) | `0`–`255`

Specify a nonreserved integer value. If **Clock class** is less than `128`, the node cannot enter the `SLAVE` state. If **Clock class** is less than `128` and the node is not selected as the best clock, the node enters the `PASSIVE` state.

If you specify a reserved integer value, the block produces an error during model update. For more information, see IEEE Std 1588-2008 Clause 7.6.2.4. For a list of reserved and nonreserved **Clock class** values, see IEEE Std 1588-2008 Table 5.

Dependency

To make this parameter visible, select the **Show advanced configuration parameters** check box.

`Clock accuracy` — Accuracy code for clock

`hex2dec('FE')` (default) | `0`–`254`

Specify a nonreserved integer value. For more information, see IEEE Std 1588-2008 Clause 7.6.2.5. For a list of reserved and nonreserved **Clock accuracy** values, see IEEE Std 1588-2008 Table 6.

Dependency

To make this parameter visible, select the **Show advanced configuration parameters** check box.

`Priority 2` — Secondary priority of PTP node

128 (default) | `0`–`255`

Specify secondary priority of PTP node. When the value is `0`, the node has the highest priority. When it is `255`, the node has the lowest priority.

For more information, see IEEE Std 1588-2008 Clause 7.6.2.3.

Dependency

To make this parameter visible, select the **Show advanced configuration parameters** check box.

Time Intervals

`Announce interval (second)` — Period of master node Announce message

2 (default) | numeric

The period, in seconds, of an Announce message transmitted by a node in master state.

For more information, see IEEE Std 1588-2008 Clause 9.5.8.

`Sync interval (second)` — Period of master node Sync message

`0.1` (default) | numeric

The period, in seconds, of a Sync message transmitted by a node in master state.

For more information, see IEEE Std 1588-2008 Clause 9.5.9.

`Min delay or pdelay request interval (second)` — Period of slave node request message

`0.1` (default) | numeric

Period of delay request message or of peer-delay request message transmitted by a node in the slave state. When the delay measurement mechanism is `Request-response`, the block transmits delay request messages. When the mechanism is `Peer-delay`, it transmits peer-delay request messages.

For more information, see IEEE Std 1588-2008 Clauses 9.5.11 and 9.5.13.

`Announce receipt timeout (in announce intervals)` — Timeout for Announce message response
3 (default) | integer

Specifies the number of announce intervals a node not in the master state has to wait without receiving an announce message. After the timeout passes, the node enters the master state.

For more information, see IEEE Std 1588-2008 Clause 9.2.6.11.

See Also

`SimulinkRealTime.target.getPCIInfo`

Topics

“Troubleshoot Precision Time Protocol Configuration” on page 17-27

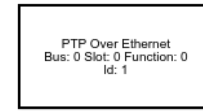
External Websites

standards.ieee.org

Introduced in R2015b

IEEE 1588 Ethernet

Execute IEEE 1588 Precision Time Protocol
Library: IEEE 1588



Description

IEEE 1588 Ethernet executes the PTP protocol, using raw Ethernet to send and receive the protocol messages. The block communicates with the corresponding blocks on the other target computers, determines the time offset that synchronizes them, and adjusts the time offset.

Parameters

General

`Device ID` — Ethernet board identifier
`1-8`

From the list, select a unique number to identify the Ethernet board.

`PCI bus` — PCI bus number of Ethernet card
`0` (default) | integer

Enter the PCI bus number for the Ethernet card.

`PCI slot` — PCI slot number of Ethernet card
`0` (default) | integer

Enter the PCI slot number for the Ethernet card.

`PCI function` — PCI function number of Ethernet card
`0` (default) | integer

Enter the PCI function number for the Ethernet card.

`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.

Network Parameters

`Source MAC address` — MAC address of Ethernet card for message source
`EEPROM` (default) | `Specify`

Source MAC address in Ethernet transport protocol. From the list, select:

- `EEPROM` — Allow the block to get the Ethernet card MAC address that is built into the Ethernet card. Use this option if you use separate Ethernet connections to transmit data and to synchronize PTP clocks.
- `Specify` — Explicitly specify the source MAC address. Use this option if both of these conditions are true:
  - You use the same Ethernet connection to transmit data as you use to synchronize PTP clocks.
  - You do not know the built-in MAC address of the Ethernet card.

Dependency

Selecting `Specify` makes the **Specify source MAC address** parameter visible.

`Specify source MAC address` — Explicit MAC address of Ethernet card for message source
`00:00:00:00:00:00` (default) | *`xx:xx:xx:xx:xx:xx`*

Enter the MAC address for the Ethernet card. Use the MAC address that is built into the Ethernet card or an arbitrary MAC address that is unique within the PTP network. Do not use one of the standard PTP multicast MAC addresses.

Dependency

To make this parameter visible, set **Source MAC address** to `Specify`.

`Destination MAC address` — MAC address of Ethernet card for message destination
`Standard PTP Multicast (01:1B:19:00:00:00, 01:80:C2:00:00:0E)` (default) | `Specify`

Destination MAC address in Ethernet transport protocol. Select one of:

- `Standard PTP Multicast (01:1B:19:00:00:00, 01:80:C2:00:00:0E)` — Default multicast MAC address assigned to the PTP protocol. If you select this option, the destination MAC addresses are:
  - `01:1B:19:00:00:00` for non-peer-delay measurement mechanism messages (`Announce`, `Sync`, `Follow_up`, `Delay_Req`, `Delay_Resp`)
  - `01:80:C2:00:00:00:0E` for peer-delay measurement mechanism messages (`Pdelay_Req`, `Pdelay_Resp`, `Pdelay_Resp_Follow_up`)
- `Specify` — Explicitly specify the destination MAC address.

You do not have to specify a source MAC address. The block uses the unique MAC address of the PTP Ethernet card.

Dependency

Selecting `Specify` makes the **Specify destination MAC address** parameter visible.

`Specify destination MAC address` — Explicit MAC address of Ethernet card for message destination

`00:00:00:00:00:00` (default) | *`xx:xx:xx:xx:xx:xx`*

Specify a MAC address for the message destination. Use this option for **Slave only** nodes. Specify the MAC address of the master node Ethernet card. The master node uses the standard PTP multicast MAC address to transmit messages to all slave nodes.

Dependency

To make this parameter visible, set **Destination MAC address** to `Specify`.

Clock Parameters

`Timescale (epoch)` — Origin point of the PTP timescale

`PTP (1970-01-01)` (default) | `GPS (1980-06-01)` | `NTP (1900-01-01)` | `Specify`

Specify the origin point of the PTP timescale. Select one of:

- `PTP (1970-01-01)` — Precision Time Protocol standard epoch, starting January 1, 1970.
- `GPS (1980-06-01)` — Global Positioning System standard epoch, starting June 1, 1980.

- `NTP (1900-01-01)` — Network Time Protocol standard epoch, starting January 1, 1900.
- `Specify` — Explicitly specify the timescale epoch.

Dependency

Selecting `Specify` makes the **Arbitrary timescale epoch (yyyy mm dd hh)** parameter visible.

`Arbitrary timescale epoch [yyyy mm dd hh]` — Explicit origin point for PTP timescale
`[1970 01 01 00]` (default) | `[yyyy mm dd hh]`

Specify the origin point for the PTP timescale, in year, month, day, and hour.

Dependency

To make this parameter visible, set **Timescale (epoch)** to `Specify`.

`Delay measurement mechanism` — Method of measuring link delays
`Request-response` (default) | `Peer-delay`

Specify the method of measuring link delays. Configure all PTP network nodes to use the same link delay measurement mechanism.

For more information, see IEEE Std Clause 7.5.4.

`Slave only` — Node that cannot be allocated as master PTP clock
`off` (default) | `on`

When you select this check box, you cannot allocate the PTP Ethernet card that this block represents as a master PTP clock.

In **Slave only** mode, the values of the advanced parameters (**Priority 1**, **Clock class**, **Clock accuracy**, and **Priority 2**) are set to their highest values. When the parameters have these settings, all of the other nodes must have the same configuration. If a node has a different configuration, the Best Master Clock Algorithm (BMCA) cannot allocate the node as best master clock. If the BMCA selects a **Slave only** node as best clock, the node remains in the `LISTENING` state.

`Show advanced configuration parameters` — Enable low-level PTP configuration parameters
`off` (default) | `on`

For more information, see IEEE Std 1588-2008.

Dependency

Selecting this check box makes advanced configuration parameters visible: **Domain number**, **Current UTC offset**, **Priority 1**, **Clock class**, **Clock accuracy**, and **Priority 2**.

`Domain number` — Domain number of PTP network

`0` (default) | `0`–`127`

Specify the domain number of the PTP network to which the node belongs.

A Simulink Real-Time PTP node can belong to only one PTP domain at a given time. If the node receives a PTP message with a different domain number, it ignores it. For more information, see IEEE Std Clause 7.1.

Dependency

To make this parameter visible, select the **Show advanced configuration parameters** check box.

`Current UTC offset` — Current offset from Coordinated Universal Time

35 (default) | integer

The current UTC offset, in seconds.

If you specify a nonzero value, that value is considered valid. The `UTCOffsetValid` flag is set to `true`. Otherwise, the flag is set to `false`. For more information, see IEEE Std 1588-2008 Clause 7.2.3.

Dependency

To make this parameter visible, select the **Show advanced configuration parameters** check box.

`Priority 1` — Priority of PTP node

`128` (default) | `0`–`255`

Specify an integer value encoding the priority of the PTP node in the network. When the value is `0`, the node has the highest priority. When it is `255`, the node has the lowest priority.

To assess the quality of two PTP clocks, the Best Master Clock Algorithm compares the following parameters, in order:

1. **Priority 1**
2. **Clock class**
3. **Clock accuracy**
4. **Priority 2**

For each parameter, the algorithm selects the clock with the smaller value as the best clock. If all four parameters are equal for both clocks, the algorithm compares the MAC addresses of the nodes.

For more information, see IEEE Std 1588-2008 Clause 7.6.2.2.

Dependency

To make this parameter visible, select the **Show advanced configuration parameters** check box.

`Clock class` — Clock class designator

`248` (default) | `0`–`255`

Specify a nonreserved integer value. If **Clock class** is less than `128`, the node cannot enter the `SLAVE` state. If **Clock class** is less than `128` and the node is not selected as the best clock, the node enters the `PASSIVE` state.

If you specify a reserved integer value, the block produces an error during model update. For more information, see IEEE Std 1588-2008 Clause 7.6.2.4. For a list of reserved and nonreserved **Clock class** values, see IEEE Std 1588-2008 Table 5.

Dependency

To make this parameter visible, select the **Show advanced configuration parameters** check box.

`Clock accuracy` — Accuracy code for clock

`hex2dec('FE')` (default) | `0`–`254`

Specify a nonreserved integer value. For more information, see IEEE Std 1588-2008 Clause 7.6.2.5. For a list of reserved and nonreserved **Clock accuracy** values, see IEEE Std 1588-2008 Table 6.

Dependency

To make this parameter visible, select the **Show advanced configuration parameters** check box.

`Priority 2` — Secondary priority of PTP node

128 (default) | `0`–`255`

Specify secondary priority of PTP node. When the value is `0`, the node has the highest priority. When it is `255`, the node has the lowest priority.

For more information, see IEEE Std 1588-2008 Clause 7.6.2.3.

Dependency

To make this parameter visible, select the **Show advanced configuration parameters** check box.

Time Intervals

`Announce interval (second)` — Period of master node Announce message

2 (default) | numeric

The period, in seconds, of an Announce message transmitted by a node in master state.

For more information, see IEEE Std 1588-2008 Clause 9.5.8.

`Sync interval (second)` — Period of master node Sync message

`0.1` (default) | numeric

The period, in seconds, of a Sync message transmitted by a node in master state.

For more information, see IEEE Std 1588-2008 Clause 9.5.9.

`Min delay or pdelay request interval (second)` — Period of slave node request message

`0.1` (default) | numeric

Period of delay request message or of peer-delay request message transmitted by a node in the slave state. When the delay measurement mechanism is `Request-response`, the block transmits delay request messages. When the mechanism is `Peer-delay`, it transmits peer-delay request messages.

For more information, see IEEE Std 1588-2008 Clauses 9.5.11 and 9.5.13.

`Announce receipt timeout (in announce intervals)` — Timeout for Announce message response
3 (default) | integer

Specifies the number of announce intervals a node not in the master state has to wait without receiving an announce message. After the timeout passes, the node enters the master state.

For more information, see IEEE Std 1588-2008 Clause 9.2.6.11.

See Also

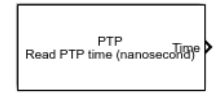
Topics
“Troubleshoot Precision Time Protocol Configuration” on page 17-27

External Websites
standards.ieee.org

Introduced in R2015b

IEEE 1588 Read Parameter

Output Precision Time Protocol status parameter value
Library: IEEE 1588



Description

Read the parameter that you select and send its value to the block output. The block output name changes based on the parameter that you select.

Ports

Output Arguments

`Time` — Time into epoch, in nanoseconds

`double` | `[uint]`

Current number of nanoseconds, counting from the beginning of the epoch.

When **Parameter to read** is:

- `PTP time (nanosecond)` — The output is a double.
- `PTP time (nanosecond vector)` — The output is a `uint` vector.

  To compute the difference in nanoseconds between two vector time values, pass both time values to the Time Stamp Delta block. To convert a single time value to nanoseconds, pass one time value to a Time Stamp Delta block and ground the other input.

Dependency

When **Parameter to read** is `PTP time (nanosecond)` or `PTP time (nanosecond vector)`, output `Time` is visible.

When you select the **Time at block start** check box, the value is measured at the beginning of block execution. When you clear the **Time at block start** check box, the value is measured at the end of block execution.

`Date` — Current date and time

[year, month, day of week, day of month, hour, minute, second, millisecond]

Current time in time-of-day format. The value is a vector of size `8`, data type `uint16`, containing: year, month (1–12), day of week (0–6), day of month (0–31), hour (0–23), minute (0–59), second (0–59), and millisecond (0–999).

Dependency

When **Parameter to read** is `PTP time (time-of-day)`, output `Date` is visible.

`Offset` — Offset from master PTP clock, in nanoseconds

double

Last computed offset from master PTP clock node, in nanoseconds.

Dependency

When **Parameter to read** is `Offset from Master`, output `Offset` is visible.

`Pdelay` — Mean path delay, in nanoseconds

double

Last computed mean path delay, in nanoseconds.

Dependency

When **Parameter to read** is `Path delay`, output `PDelay` is visible.

`State` — Current state of protocol

`1-9`

Current state of the protocol state machine. Returns one of:

- `1` = INITIALIZING — Initializing data set and communication protocol
- `2` = FAULTY — Occurrence of serious fault
- `3` = DISABLED — Management message disables the node
- `4` = LISTENING — Waiting for announce receipt timeout period to expire

- 5 = PRE_MASTER — Intermediate state before moving to MASTER state after execution of Best Master Clock Algorithm (BMCA)
- 6 = MASTER — Node is the master PTP clock node
- 7 = PASSIVE — BCMA designates node as passive
- 8 = UNCALIBRATED — Intermediate state before moving to SLAVE state after execution of BMCA
- 9 = SLAVE — Node is a slave node

For more information, see IEEE Std 1588-2008 Clause 9.2.5.

Dependency

When **Parameter to read** is `Protocol state`, output `State` is visible.

Parameters

`Parameter to read` — Parameter to display at output
`PTP time (nanosecond)` (default) | `PTP time (nanosecond vector)` | `PTP time (time-of-day)` | `Offset from Master` | `Path delay` | `Protocol state`

Specify parameter to read and make corresponding output port visible. Select one of:

- `PTP time (nanosecond)` — Reveals port `Time` and parameter **Time at block start**
- `PTP time (nanosecond vector)` — Reveals port `Time` and parameter **Time at block start**
- `PTP time (time-of-day)` — Reveals port `Date`
- `Offset from Master` — Reveals port `Offset`
- `Path delay` — Reveals port `PDelay`
- `Protocol state` — Reveals port `State`

`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.

`Time at block start` — Place of time value in execution
`0` (default) | `1`

When you select this check box, the `Time` output contains the time at the beginning of block execution. When you clear this check box (the default), the `Time` output contains the time at the end of block execution.

Dependency

Setting **Parameter to read** to `PTP time (nanosecond)` makes this check box visible.

See Also

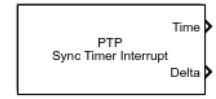
Topics

“Troubleshoot Precision Time Protocol Configuration” on page 17-27

Introduced in R2015b

IEEE 1588 Sync Execution

Synchronize model execution to Precision Time Protocol clock
Library: IEEE 1588



Description

When the PTP time is a multiple of the fundamental step size of the model, this block causes a real-time interrupt.

Make measurements across multiple target computers at the same time step by using the IEEE 1588 Sync Execution block. The block uses a control loop to adjust the step size toward the synchronization objective. During this process, the control loop decreases or increases the step size. When the control loop decreases the step size, the CPU can become overloaded. You can decrease the maximum adjustment value by decreasing the **Proportional gain** parameter. The upper bound of the adjustment value is `10%` of the model fundamental sample time, regardless of the **Proportional gain** value.

Use this block in every model that requires synchronized execution, whether it is a PTP master or slave model. To use this block, in the Simulink Real-Time options, set the real-time interrupt source to `Timer`. As a best practice, for all models, use the same fundamental sample time. Set the sample time in this block to that fundamental sample time.

If you use the IEEE 1588 Sync Execution block in your model, configuring EtherCAT distributed clocks in master shift mode in the same model produces a build error. To include IEEE 1588 synchronized execution and EtherCAT distributed clocks in the same model, use EtherCAT bus shift mode.

Ports

Output

`Time` — PTP time of real-time interrupt

scalar

PTP time value at which the interrupt occurs, in seconds.

Data Types: `double`

`Delta` — Difference between interrupt time and nearest PTP sample time

scalar

Current difference, in seconds, between the PTP time at the interrupt and the nearest PTP time that is a multiple of the fundamental sample time.

Data Types: `double`

Parameters

`Proportional gain` — Proportional gain of the kernel clock adjustment controller

`0.1` (default) | double

The current value of output port Delta is multiplied by the proportional gain to get the first part of the controller output.

`Low pass filter pole` — Pole of the low-pass filter of the kernel clock adjustment controller

`0.7` (default) | double

The low-pass filter is a discrete-time, first-order transfer function. The low-pass filter tracks the rate difference between the kernel and PTP clocks and provides the second part of the controller output.

`PTP clock synchronization threshold (seconds)` — Threshold value at which the controller begins to adjust the kernel clock

`1e-3` (default) | double

The effect of this value depends on the PTP node state:

- Slave node — The controller starts the kernel adjustment when the slave PTP clock offset from the master clock is less than or equal to this parameter.
- Master node — The controller starts the kernel clock adjustment immediately after it enters the master state, regardless of the value of this parameter.

It is a best practice to start adjusting the kernel clock only when the PTP clock is stable. Keep this value less than or equal to a millisecond.

`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.

See Also

Transfer Fcn First Order

Topics

“Troubleshoot Precision Time Protocol Configuration” on page 17-27

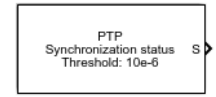
External Websites

standards.ieee.org

Introduced in R2016a

IEEE 1588 Sync Status

Output synchronization status of Precision Time Protocol
Library: IEEE 1588



Description

When the absolute value of the offset from the master PTP clock is less than or equal to the threshold, the output port value is `true`. If the node is selected as the master clock, the output port value becomes true when it enters the MASTER state.

Ports

Output Arguments

S — Detects if block is synchronized with master PTP clock
`true` | `false`

When the absolute value of the offset from the master PTP clock is less than or equal to the specified threshold, returns `true`.

Parameters

`Offset threshold (second)` — Synchronization threshold value
10e-6 (default) | double

Threshold value, in seconds, from which the node is considered well synchronized to the master PTP clock node. The default value is 10 µs. The minimum allowed value is 1 µs.

`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.

See Also

Topics

“Troubleshoot Precision Time Protocol Configuration” on page 17-27

External Websites

standards.ieee.org

Introduced in R2015b

IEEE 1588 Setup

Configure node for Precision Time Protocol execution

Library

Simulink Real-Time Library for Precision Time Protocol

Description

Sets up the Precision Time Protocol for the specified transport protocol (Ethernet or UDP). Exposes as outputs the state of the protocol, the delay measurement mechanism, and triggers for sending messages.

The PTP internal block descriptions are for informational purposes only. You cannot use these blocks by themselves in a model. The subsystem mask controls the block parameters. Do not edit the parameters directly.

Block Outputs

Name	Description
`State`	Current state of the protocol state machine. Returns one of: • 1 = INITIALIZING — Initializing data set and communication protocol • 2 = FAULTY — Occurrence of serious fault • 3 = DISABLED — Management message disables the node • 4 = LISTENING — Waiting for announce receipt timeout period to expire • 5 = PRE_MASTER — Intermediate state before moving to MASTER state after execution of Best Master Clock Algorithm (BMCA) • 6 = MASTER — Node is the master PTP clock node • 7 = PASSIVE — BCMA designates node as passive • 8 = UNCALIBRATED — Intermediate state before moving to SLAVE state after execution of BMCA • 9 = SLAVE — Node is a slave node For more information, see IEEE Std 1588-2008 Clause 9.2.5.
`DM`	Value of **Delay measurement mechanism**. Returns one of: • 1 = `Request-response` • 2 = `Peer-delay`
`ST`	Value of synchronization trigger, `true` every **Sync interval**

Name	Description
AT	Value of announce trigger, `true` every **Announce interval**
DT	Value of delay request trigger, `true` every **Min delay or pdelay request interval**

Block Parameters

General

Device ID

From the list, select a unique number to identify the Ethernet board. Select the same **Device ID** as the one that you selected for the protocol configuration block.

Transport protocol

The network protocol for communicating messages. Select one of `Real-Time UDP` and `Raw Ethernet`.

IP address

IP address of Ethernet card, or node, represented by the PTP Setup block.

Board time increment value

Value that changes the PTP clock.

PCI bus

Enter the PCI bus number for the Ethernet card.

PCI slot

Enter the PCI slot number for the Ethernet card.

Time Properties

Time source

Source of PTP clock signal. Select one of:

- `Precise System Time` — Synchronization of system time without hardware timestamp

- `Tick Counter` — Synchronization of tick counter read without hardware timestamp
- `Ethernet board` — Clock on PTP Ethernet board

Timescale (epoch)

Specify origin point of the PTP timescale. Select one of:

- `PTP (1970-01-01` — Precision Time Protocol standard epoch, starting January 1, 1970.
- `GPS (1980-06-01)` — Global Positioning System standard epoch, starting June 1, 1980.
- `NTP (1900-01-01)` — Network Time Protocol standard epoch, starting January 1, 1900.
- `Specify` — Selecting this value makes the **Arbitrary timescale epoch (yyyy mm dd hh)** parameter visible.

Arbitrary timescale epoch (yyyy mm dd hh)

Specify origin point for PTP timescale, in year, month, day, and hour.

When **Timescale (epoch)** is `Specify`, **Arbitrary timescale epoch (yyyy mm dd hh)** is visible.

Delay measurement mechanism

Method of measuring link delays. Select one of `Request-response` and `Peer-delay`.

In a PTP network, you must configure all nodes to use the same link delay measurement mechanism.

For more information, see IEEE Std 1588-2008 Clause 7.5.4.

Sample time (-1 for inherited)

Enter the base sample time or a multiple of the base sample time.

Slave only

When you select this check box, you cannot allocate the PTP Ethernet card that this block represents as a master PTP clock.

In **Slave only** mode, the values of the advanced parameters (**Priority 1**, **Clock class**, **Clock accuracy**, and **Priority 2**) are set to their highest values. When the parameters have these settings, the Best Master Clock Algorithm (BMCA) cannot

choose the node as best master clock unless all of the other nodes have the same configuration. If the BMCA selects a **Slave only** node as best clock, the node remains in the `LISTENING` state.

Time Intervals

Announce interval (second)

The period, in seconds, of an Announce message transmitted by a node in master state.

For more information, see IEEE Std 1588-2008 Clause 9.5.8.

Sync interval (second)

The period, in seconds, of a Sync message transmitted by a node in master state.

For more information, see IEEE Std 1588-2008 Clause 9.5.9.

Min delay or pdelay request interval (second)

Period of delay request message or of peer-delay request message transmitted by a node in the slave state. When the delay measurement mechanism is `Request-response`, the block transmits delay request messages. When the mechanism is `Peer-delay`, it transmits peer-delay request messages.

For more information, see IEEE Std 1588-2008 Clauses 9.5.11 and 9.5.13.

Announce receipt timeout (in announce intervals)

Specifies the number of announce intervals a node not in the master state has to wait without receiving an announce message before the node enters the master state.

For more information, see IEEE Std 1588-2008 Clause 9.2.6.11.

Advanced

Domain number

Specify the domain number of the PTP network to which the node belongs.

A Simulink Real-Time PTP node can belong to only one PTP domain at a given time. If the node receives a PTP message with a different domain number, it ignores it. For more information, see IEEE Std 1588-2008 Clause 7.1.

When you select the **Show advanced configuration parameters** check box, **Domain number** is visible.

Current UTC offset

The current UTC offset, in seconds.

If you specify a nonzero value, that value is considered valid. The `UTCOffsetValid` flag is set to `true`. Otherwise, the flag is set to `false`. For more information, see IEEE Std 1588-2008 Clause 7.2.3.

When you select the **Show advanced configuration parameters** check box, **Current UTC offset** is visible.

Priority 1

Specify an integer value in the range `0`—`255`. When the value is `0`, the node has the highest priority. When it is `255`, the node has the lowest priority.

To assess the quality of two PTP clocks, the Best Master Clock Algorithm compares the following parameters, in order:

1. **Priority 1**
2. **Clock class**
3. **Clock accuracy**
4. **Priority 2**

If a parameter for one PTP clock has a smaller value than that parameter for the other clock, the algorithm selects the clock with the smaller value as the best clock. If all four parameters are equal for both clocks, the algorithm compares the MAC addresses of the nodes.

For more information, see IEEE Std 1588-2008 Clause 7.6.2.2.

When you select the **Show advanced configuration parameters** check box, **Priority 1** is visible.

Clock class

Specify a nonreserved integer value in the range `0`—`255`. If **Clock class** is less than `128`, the node cannot enter the `SLAVE` state. If **Clock class** is less than `128` and the node is not selected as the best clock, the node enters the `PASSIVE` state.

If you specify a reserved integer value, the block produces an error during model update. For more information, see IEEE Std 1588-2008 Clause 7.6.2.4. For a list of reserved and nonreserved **Clock class** values, see IEEE Std 1588-2008 Table 5.

When you select the **Show advanced configuration parameters** check box, **Clock class** is visible.

Clock accuracy

Specify a nonreserved integer value in the range `0—254`.

For more information, see IEEE Std 1588-2008 Clause 7.6.2.5. For a list of reserved and nonreserved **Clock accuracy** values, see IEEE Std 1588-2008 Table 6.

When you select the **Show advanced configuration parameters** check box, **Clock accuracy** is visible.

Priority 2

Specify an integer value in the range `0—255`. When the value is `0`, the node has the highest priority. When it is `255`, the node has the lowest priority.

For more information, see IEEE Std 1588-2008 Clause 7.6.2.3.

When you select the **Show advanced configuration parameters** check box, **Priority 2** is visible.

See Also

Topics

“Troubleshoot Precision Time Protocol Configuration” on page 17-27

External Websites

standards.ieee.org

Introduced in R2015b

IEEE 1588 Adjust Time

Run Precision Time Protocol clock correction servo

Library

Simulink Real-Time Library for Precision Time Protocol

Description

When `Trigger` is `true`, IEEE 1588 Adjust Time performs time offset correction at every sample time until the offset drops below a threshold value. It then switches to rate correction. During rate correction, the block adjusts the time increment value of the Ethernet card to track the master PTP clock rate.

The PTP internal block descriptions are for informational purposes only. You cannot use these blocks by themselves in a model. The subsystem mask controls the block parameters. Do not edit the parameters directly.

Block Inputs

Name	Description
`Trigger`	While `Trigger` is `true`, the block runs the time adjustment algorithm at every time step.

Block Parameters

Sample time (-1 for inherited)

Enter the base sample time or a multiple of the base sample time.

See Also

Topics

“Troubleshoot Precision Time Protocol Configuration” on page 17-27

External Websites

standards.ieee.org

Introduced in R2015b

IEEE 1588 Create Message

Pack a Precision Time Protocol message for transmission

Library

Simulink Real-Time Library for Precision Time Protocol

Description

Creates and packs an IEEE 1588 message of a type specified by parameter **Message type**. Sends the message and the message length to the `Data` and `Length` outputs, respectively.

By default, IEEE 1588 Create Message creates a message at every sample time. If you select the **Enable trigger port** check box, IEEE 1588 Create Message creates a message at every sample time only when the trigger is `true`.

The PTP internal block descriptions are for informational purposes only. You cannot use these blocks by themselves in a model. The subsystem mask controls the block parameters. Do not edit the parameters directly.

Block Inputs and Outputs

Inputs

Name	Description
Trigger	While Trigger is `true`, the block creates a message at every sample time. When **Enable trigger port** is `true`, this input is visible.

Outputs

Name	Description
Data	Message data in a `uint8` array.
Length	Message data length (`double`).

Block Parameters

Message type

Select the PTP message type to pack. Select one of `Sync`, `Delay_Req`, `Pdelay_Req`, `Pdelay_Resp`, `Follow_up`, `Delay_Resp`, `Pdelay_Resp_Follow_up`, and `Announce`.

For more information, see IEEE Std 1588-2008 Clause 7.3.3.

Sample time (-1 for inherited)

Enter the base sample time or a multiple of the base sample time.

Enable trigger port

Selecting this check box makes input port Trigger visible.

See Also

Topics

“Troubleshoot Precision Time Protocol Configuration” on page 17-27

External Websites

standards.ieee.org

Introduced in R2015b

IEEE 1588 Process Message

Process a received Precision Time Protocol message

Library

Simulink Real-Time Library for Precision Time Protocol

Description

Unpack a received PTP message and execute the actions that the message data requires. For example, get timestamps and calculate offset.

The PTP internal block descriptions are for informational purposes only. You cannot use these blocks by themselves in a model. The subsystem mask controls the block parameters. Do not edit the parameters directly.

Block Inputs and Outputs

Inputs

Name	Description
`Data`	Message data in a `uint8` array.
`Length`	Message data length (`double`).

Outputs

Name	Description
`Flag`	If `true`, the two-steps flag bit in the message is set, otherwise, the bit is cleared.
`Type`	Message type (`uint8`). Return one of: • `0` = `Sync` • `1` = `Delay_Req` • `2` = `Pdelay_Req` • `3` = `Pdelay_Resp` • `8` = `Follow_up` • `9` = `Delay_Resp` • `10` = `Pdelay_Resp_Follow_up` • `11` = `Announce` For more information, see IEEE Std 1588-2008 Clause 7.3.3.

Block Parameters

Sample time (-1 for inherited)

Enter the base sample time or a multiple of the base sample time.

See Also

Topics

“Troubleshoot Precision Time Protocol Configuration” on page 17-27

External Websites

standards.ieee.org

Introduced in R2015b

IEEE 1588 Sync Error

Output block execution step size and offset delta of real-time interrupt

Library

Simulink Real-Time Library for Precision Time Protocol

Description

The block returns the block execution step size as measured by the PTP clock. It also returns the following information:

- The PTP time when the real-time interrupt triggers.
- The difference between the PTP time at the real-time interrupt and the nearest PTP time that is a multiple of the fundamental sample time.

To use this block, in the Simulink Real-Time options, set the real-time interrupt source to `Timer`.

The PTP internal block descriptions are for informational purposes only. You cannot use these blocks by themselves in a model. The subsystem mask controls the block parameters. Do not edit the parameters directly.

Block Outputs

Name	Description
`Time`	PTP time value when the real-time interrupt occurs, in seconds
`Step`	Actual block execution step size, in seconds, as calculated from PTP clock measurements

Name	Description
`Delta`	Current difference, in seconds, between the PTP time at the interrupt and the nearest PTP time that is a multiple of the fundamental sample time

Block Parameters

Sample time (-1 for inherited)

Enter the base sample time or a multiple of the base sample time.

See Also

Topics

“Troubleshoot Precision Time Protocol Configuration” on page 17-27

External Websites

standards.ieee.org

Introduced in R2016a

Adjust Step Size

Adjust block step size during execution

Library

Simulink Real-Time Library for Precision Time Protocol

Description

The block sets the execution step size of the block. The block execution step size is equal to the fundamental sample time plus the value of `Adj`, scaled to the block sample time. For example, if the block sample time is twice the fundamental sample time, each fundamental execution step gets half of the `Adj` value.

When the block changes the execution step size, the new value remains active until the block changes the value again.

The PTP internal block descriptions are for informational purposes only. You cannot use these blocks by themselves in a model. The subsystem mask controls the block parameters. Do not edit the parameters directly.

Block Inputs

Name	Description
`Adj`	Step size increment

Block Parameters

Sample time (-1 for inherited)

Enter the base sample time or a multiple of the base sample time.

See Also

Topics

“Troubleshoot Precision Time Protocol Configuration” on page 17-27

External Websites

standards.ieee.org

Introduced in R2016a

Current Step Size

Output current block execution step size

Library

Simulink Real-Time Library for Precision Time Protocol

Description

The block returns the current block execution step size. When you select **Enable base rate output port**, output port BR shows the base rate or model fundamental step size.

The PTP internal block descriptions are for informational purposes only. You cannot use these blocks by themselves in a model. The subsystem mask controls the block parameters. Do not edit the parameters directly.

Block Outputs

Name	Description
CS	Current block execution step size, in seconds
BR	Base rate, or model fundamental step size, in seconds When you select **Enable base rate output port**, this port becomes visible.

Block Parameters

Sample time (-1 for inherited)

Enter the base sample time or a multiple of the base sample time.

Enable base rate output port

To make the BR output visible, select this check box.

See Also

Topics

“Troubleshoot Precision Time Protocol Configuration” on page 17-27

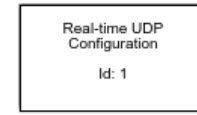
External Websites

standards.ieee.org

Introduced in R2016a

Real-Time UDP Configuration

Configure network interface for real-time UDP communication
Library: IEEE 1588 / PTP UDP



Description

The Real-Time UDP Configuration block configures the network for real-time UDP operation.

IP fragmentation is not supported in Simulink Real-Time PTP UDP blocks. The packet payload is limited to 1472 bytes (1500 bytes UDP packet size − 28 bytes combined packet header size).

Parameters

`Device ID` — Identify this Ethernet board
1 (default) | 1–8

From the list, select a unique integer to identify the Ethernet board. Use this ID to associate the other UDP blocks with this board.

`IP Address` — IP address for interface
`x.x.x.x`

The addresses `0.0.0.0` and `255.255.255.255` are invalid local IP addresses.

`Subnet mask` — Subnet mask for interface
`255.255.255.0` (default) | *`x.x.x.x`*

Mask that designates a logical subdivision of a network.

`Gateway` — IP address for gateway interface
`0.0.0.0` (default) | *`x.x.x.x`*

The gateway must be within the network.

To indicate that a gateway is not being used, enter `0.0.0.0` (the default). The address `255.255.255.255` is an invalid gateway IP address.

`Ethernet Driver` — Driver for each chip family
`Intel 8255X` (default) | `Intel Gigabit`

Identifies the driver for each chip family that the block supports.

`PCI bus` — PCI bus number of Ethernet card
`0` (default) | integer

Enter the PCI bus number for the Ethernet card.

`PCI slot` — PCI slot number of Ethernet card
`0` (default) | integer

Enter the PCI slot number for the Ethernet card.

`PCI function` — PCI function number of Ethernet card
`0` (default) | integer

Enter the PCI function number for the Ethernet card.

See Also

Receive | Send

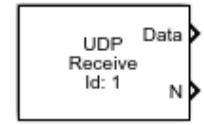
External Websites

www.iso.org

Introduced in R2014b

Receive

Receive data over UDP network on a dedicated network interface
Library: IEEE 1588 / PTP UDP



Description

The Receive block receives UDP data on the specified local (destination) port. To receive all data sent to this port, set **Source IP address** to `0.0.0.0`, otherwise set **Source IP address** to a valid IP address.

The default block behavior is to keep the previous output when there is no new data.

IP fragmentation is not supported in Simulink Real-Time PTP UDP blocks. The packet payload is limited to 1472 bytes (1500 bytes UDP packet size − 28 bytes combined packet header size).

Ports

Output Arguments

`Data` — Data received

[uint8]

Vector of `uint8` containing data received.

`N` — Number of bytes received

`0–1472`

Number of new bytes received, and otherwise `0`.

Parameters

`Device ID` — Device ID for Ethernet board
1 (default) | 1–8

From the list, select a unique number to identify the Ethernet board. Select the same **Device ID** as the one you selected for the Real-Time UDP Configuration block.

`Source IP address` — IP address from which device accepts packets
`0.0.0.0` (default) | *`x.x.x.x`*

Enter a valid IP address as a dotted decimal character vector, for example, `10.10.10.3`. You can also use a MATLAB expression that returns a valid IP address as a character vector. With **Local (destination) port**, this parameter defines the source address.

The default address, `0.0.0.0`, causes the block to accept UDP packets from any accessible computer. If **Source IP address** is set to a specific IP address, packets arriving from only that IP address are received.

The address `255.255.255.255` is an invalid IP address.

`Local (destination) port` — Port from which device accepts packets
1–65535

Specify the port of the target computer or device from which to receive the UDP packets. With **Source IP address**, this parameter defines the source address.

`Output port width` — Width of output vector, in bytes
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.

`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.

See Also

Real-Time UDP Configuration | Send

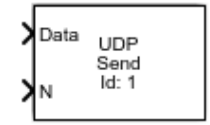
External Websites

www.iso.org

Introduced in R2011a

Send

Send data over UDP network on a dedicated network interface
Library: IEEE 1588 / PTP UDP



Description

The Send block sends UDP packets from **Local (source) port** to **Destination port**. To broadcast to all devices, set **Destination IP address** to `255.255.255.255`, otherwise set **Destination IP address** to a valid IP address.

IP fragmentation is not supported in Simulink Real-Time PTP UDP blocks. The packet payload is limited to 1472 bytes (1500 bytes UDP packet size − 28 bytes combined packet header size).

Ports

Input

`Data` — Data to transmit

[uint8]

Vector of `uint8` containing data to send.

`N` — Number of bytes to transmit

`0–1472`

Number of bytes to send.

Parameters

`Device ID` — Device ID for Ethernet board
1 (default) | 1–8

From the list, select a unique number to identify the Ethernet board. Select the same **Device ID** as the one you selected for the Real-Time UDP Configuration block.

`Destination IP address` — IP address to which device sends packets
`255.255.255.255` (default) | *`x.x.x.x`*

Specify the IP address of the target computer or other device to which you want to send the UDP packets. To broadcast the packets to all listening computers or devices, enter `255.255.255.255`. With **Destination port**, this parameter defines the destination address.

`Destination port` — Port to which device sends packets
1–65535

Specify the target computer port to which you want to send the UDP packets. With **Destination IP address**, this parameter defines the destination address.

`Local (source) port` — Target computer port that sends packets
–1 (default) | 1–65535

Specify the target computer port from which you want to send the UDP packets.

Enter `-1` to automatically assign a port for the target computer.

`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.

See Also

Real-Time UDP Configuration | Receive

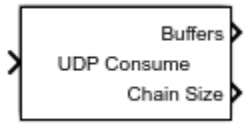
External Websites

www.iso.org

Introduced in R2011a

UDP Consume

Consume a UDP packet



Library

Simulink Real-Time Library for PTP UDP

Description

The UDP Consume block outputs a network buffer with raw data that you can output to a Network Buffer library block. To create this output, the block:

1. Receives as input a network buffer that contains a UDP header.
2. Removes the UDP header.
3. Outputs the updated network buffer.

The block has two output ports:

- Buffers

  Chain of network buffers.
- Chain size

  Number of buffers on the chain.

Block Parameters

IP Group

Enter a number in the range `0` to `7`. This value identifies the IP Init block inside the Real-Time UDP Configuration subsystem that is associated with this block.

Output port width

Enter the width of the port. A value other than `0` creates the following output ports:

- Source IP Address
- Destination IP Address
- Local UDP Port
- Remote UDP Port

Sample time

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

See Also

External Websites

www.iso.org

Introduced in R2011a

UDP Produce

Produce UDP packet by adding a UDP header to the input data

Library

Simulink Real-Time Library for PTP UDP

Description

The UDP Produce block receives a network buffer and adds a header to that buffer. It then outputs that updated buffer.

Block Parameters

IP Group

Enter a number in the range `0` to `7`. This value identifies the IP Init block inside the Real-Time UDP Configuration subsystem that is associated with this block.

IP address to send to (255.255.255.255 for broadcast)

Specify IP address of the target computer to which to send the UDP packets. To broadcast the packets to all listening computers or devices, enter `255.255.255.255`. With **Remote IP port to send to**, this parameter defines the destination address.

Remote IP port to send to

Specify the target computer port to which to send the UDP packets. With **IP address to send to (255.255.255.255 for broadcast)**, this parameter defines the destination address.

Use the following local IP port (-1 for automatic assignment)

Specify the target computer port from which to send the UDP packets.

Enter `-1` to automatically assign a port for the target computer.

Sample time

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

See Also

External Websites

www.iso.org

Introduced in R2011a

UDP Rx

Receive UDP packet

Library

Simulink Real-Time Library for PTP UDP

Description

The UDP Rx block outputs a network buffer with a UDP header.

Block Parameters

IP Group

Enter a number in the range `0` to `7`. This value identifies the IP Init block inside the Real-Time UDP Configuration subsystem that is associated with this block.

IP address to receive from (0.0.0.0 for accepting all)

Enter a valid IP address as a dotted decimal character vector. For example, `10.10.10.3`. You can also use a MATLAB expression that returns a valid IP address as a character vector. With **IP port to receive from**, this parameter defines the source address.

The default address, `0.0.0.0`, enables the acceptance of all UDP packets from any accessible computer. If set to a specific IP address, only packets arriving from that IP address are received.

IP port to receive from

Specify the port of the target computer or device from which to receive the video frames. With **IP address to receive from (0.0.0.0 for accepting all)**, this parameter defines the source address.

Sample time

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

See Also

External Websites

www.iso.org

Introduced in R2011a

UDP Tx

Transmit UDP packet

Library

Simulink Real-Time Library for PTP UDP

Description

The UDP Tx block receives a network buffer with a UDP header and sends it.

Block Parameters

IP Group

Enter a number in the range `0` to `7`. This value identifies the IP Init block inside the Real-Time UDP Configuration subsystem that is associated with this block.

Sample time

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

See Also

External Websites

www.iso.org

Introduced in R2011a

SAE J1939

SAE J1939 Blocks

The Simulink Real-Time J1939 blocks enable you to send and receive messages over a FIFO-mode CAN network using the SAE J1939 message protocol. See “CAN”.

Before you start, provide a J1939 database in `.dbc` format.

See Also

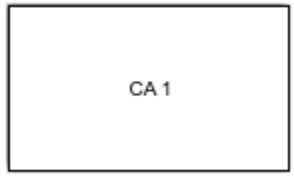
More About

- “CAN”

SAE J1939 Blocks

J1939 Controller Application

J1939 Controller Application



Library

Simulink Real-Time Library for J1939

Description

The J1939 Controller Application block supports address claiming. It enables the dynamic exchange of address information with other nodes in the J1939 bus, resolving conflicts. Once conflicts are resolved, the blocks on the CAN network each have unique addresses and IDs.

Use this block to register your J1939 device on the CAN bus. Associate this block with the J1939 Transmit Message and J1939 Receive Message blocks.

The block has two tab groups, **General** and **NAME**. The **General** tab contains general block information. The **NAME** tab has 10 parameters that, when combined, create a unique identification address (NAME) for this J1939 device. Refer to the SAE J1939-81 and base SAE J1939 specifications for details.

Block Outputs

This block has the following output port:

Status output (Optional)

Outputs a status signal.

Current node address output (Optional)

Outputs current node address.

Block Parameters

General tab:

CA ID

Enter the controller application ID.

Protocol Stack ID

Enter the protocol stack ID.

Node address

From a range of 0 to 253, specify the node address of the node for which J1939 communication is to occur. This value is the source address for the J1939 communication. Set this address to the same as that in the J1939 Transmit Message block.

Show status output

Select this check box to enable an output status signal.

Show current node address as output

Select this check box to enable an output current node address signal.

NAME tab:

Identity Number

Enter the identity number for the controller application. Use the identify number provided by the ECU manufacturer.

Manufacturer Code

Enter the code of the electronic control unit (ECU) manufacturer.

ECU Instance

Enter a number to identify the particular ECU associated with **Function**.

If this number identifies the first or only instance, enter `0`.

Function Instance

Enter a number to identify an instance of the function for the **Vehicle System** in the CAN network.

Function

Enter an 8-bit value for the function for this controller application. See Appendix B of the base SAE J1939 specification for a list of allowed function values.

If you enter a value between `0` and `127`, the block independently evaluates the function value.

If you enter a value greater than 127, the block takes into account the value of the **Vehicle System** parameter when evaluating the function value.

Reserved

Reserved. Leave set to `0`.

Vehicle System

Enter a 7-bit value for the vehicle system for this controller application. See Appendix B of the base SAE J1939 specification for a list of allowed vehicle system values.

Vehicle System Instance

Enter a number to identify an instance of the vehicle system in the CAN network.

If this number identifies the first or only instance, enter `0`.

Industry Group

Enter a 3-bit value for the industry group for this controller application. See Appendix B of the base SAE J1939 specification for a list of allowed industry group values.

Arbitrary Address Capable

Select this check box to allow the controller application to resolve address claim conflicts with arbitrary source addresses.

See Also

External Websites

www.sae.org/

Introduced in R2009b

J1939 Database (CANdb) Setup

J1939 Database Setup

Library

Simulink Real-Time Library for J1939

Description

The J1939 Database Setup block is where you specify the user-supplied database for the J1939 block set. Use one block per model.

Block Parameters

J1939 database file

Specify the J1939 database location and file name. For example, 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`.

This file defines the J1939 message set and is in a format defined by Vector Informatik GmbH.

Note You must supply the database file. The Simulink Real-Time software does not supply this file.

J1939 Database Format

From the list, select:

- **J1939 PG — Full Extended CAN ID**

  Specifies that the database format is the new version of CANdb. The block extracts the Parameter Group Number (PGN) from the full extended CAN ID (29–bit).

- **J1939 PG**

  Specifies that the database format is simplified ID (CANDB++ 2.7 SP7 and before).

See Also

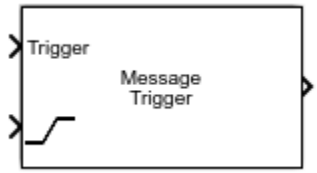
External Websites

www.sae.org/

Introduced in R2009a

J1939 Message Trigger

J1939 Message Triggering



Library

Simulink Real-Time Library for J1939

Description

The J1939 Message Trigger block triggers the transmission of a J1939 message under conditions that the block specifies. When using this block in your model, connect its Trigger Out port to the Trigger input port of the J1939 Transmit Message block. The possible triggering conditions are:

- Nonzero input at the Trigger Message input port
- Expiration of the repetition interval
- A detected change in the input signal

Block Inputs

This block has the following input ports:

Trigger Message

Manually triggers the message transmission when the input value is 1.

Signal For Change Detection

Detects changes in the input.

Block Outputs

This block has the following output ports:

Trigger Out

Outputs an active signal when a condition for triggering a message occurs. Connect this output to the Trigger input port of the J1939 Transmit Message block.

Block Parameters

Repeat on interval

Select this check box to enable periodic triggering at the interval specified in the **Repetition Interval** parameter.

Repetition Interval (ms)

Specify the enabled repetition interval in milliseconds. The repetition interval must be an integer multiple of the model update rate.

Send on a change in signal

Select this check box to enable triggering when both of the following are true:

- Input signal wired to the Signal For Change Detection port input port changes by at least the value in the **Minimum Change Threshold** parameter.
- Specified time in the **Minimum Change Interval** parameter has expired.

Minimum Change Threshold (%)

Specify the threshold change to trigger a message. The trigger event is a change in the input signal relative to the signal value from the previous triggering.

Minimum Change Interval (ms)

Specify the minimum time for the block to wait after a message transmission occurs because of a change in the signal.

See Also

J1939 Transmit Message

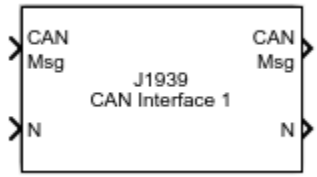
External Websites

www.sae.org/

Introduced in R2009a

J1939 Protocol Stack

J1939 Protocol Stack instance



Library

Simulink Real-Time Library for J1939

Description

The J1939 Protocol Stack block defines a J1939 protocol stack instance that you can associate with the CAN boards. This block is useful for the transmission and reception of CAN data to or from the bus. For each selected CAN board, you must have a corresponding CAN FIFO Setup block in the model. The protocol stack handles both the regular J1939 communication and the (optional) Transport Protocol functionality. Use this block with the J1939 Transmit Message and J1939 Receive Message blocks.

Block Parameters

General tab:

Protocol Stack ID

Enter the protocol stack ID.

Max CAN Message Receive (Per Sample Time)

Specify maximum number of CAN messages that the block can receive (process) in a single sample time.

Max CAN Message Transmit (Per Sample Time)

Specify maximum number of CAN messages the block can send in a single sample time.

Sample time

Specify the sample rate for the protocol stack.

Transport Protocol tab:

Enable Transport Protocol

Select this check box to enable the transport protocol. Enabling the transport protocol allows for the sending and receiving of J1939 data, whose size is greater than 8 bytes, over the CAN bus.

Maximum Concurrent Sessions

Specify maximum transport protocol sessions that can be active at a given time.

Network Management tab

Enable Address Claiming

Select this check box to enable address claiming. Selecting this check box enables J1939 Controller Application blocks to negotiate node addresses with other nodes on the bus.

See Also

J1939 Transmit Message | J1939 Receive Message

External Websites

www.sae.org/

Introduced in R2009a

J1939 Receive Message

J1939 Receive Message

Library

Simulink Real-Time Library for J1939

Description

The J1939 Receive Message block receives a J1939 message. The J1939 database file defines the message type. You specify the J1939 database with the J1939 Database (CANdb) Setup block.

Block Outputs

Signal Output(s)

Depending on the J1939 message defined in the J1939 database file, the block can have multiple output signal ports. If the bit length of a signal exceeds 32, the output is a byte array; otherwise, the block output data type is double.

New Message Received (Optional)

Outputs `1` when a new message is received from the CAN bus; otherwise, outputs `0`.

Block Parameters

PGN

From the list, select the parameter group number. The contents of this list vary depending on the messages that the J1939 database file specifies.

CA ID

Specify the ID of the controller application that this block maps to. The ID must match the **CA ID** value of the corresponding J1939 Controller Application block.

Source Address Filter (255:all)

Specify the source node address from which the block is to expect messages. Type `255` to receive messages from any node.

Destination Address Filter

From the list, select the node destination of the expected message:

- `global and specific`

  Receive all messages for all types of destination nodes.
- `global only`

  Receive only broadcast messages.
- `CA specific only`

  Receive only messages sent to this node.

Show 'New Message Received' output port

Select this check box to create a New Message Received output port.

See Also

J1939 Transmit Message | J1939 Database (CANdb) Setup

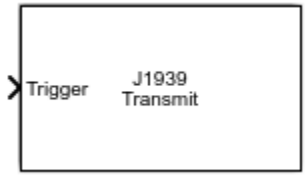
External Websites

www.sae.org/

Introduced in R2009a

J1939 Transmit Message

J1939 Transmit



Library

Simulink Real-Time Library for J1939

Description

The J1939 Transmit Message block transmits a J1939 message The J1939 database file defines the message type. You specify the J1939 database with the J1939 Database (CANdb) Setup block.

Block Inputs

- `Trigger`

  Enables the transmission of the message for that sample (a value of `1` indicates send, a value of `0` indicates do not send).

- `Signal Input(s)`

  Depending on the J1939 message defined in the J1939 database file, the block can have multiple input signal ports. If the bit length of a signal exceeds 32, the input is a byte array; otherwise, the block input data type is double.

Block Parameters

PGN

From the list, select the parameter group number. The contents of this list vary depending on the messages that the J1939 database file specifies.

CA ID

Identifies the controller application that this block maps to. The ID must match the **CA ID** value of the corresponding J1939 Controller Application block.

Priority (0-7; highest to lowest

From a range of 0 to 7, specify the priority for the message transmission. `0` is the highest, `7` is the lowest.

Destination Address (0–253 or 255)

Specify the node address of the destination node. Type `255` for broadcast.

Note For message-oriented PGNs, this block ignores this parameter.

J1939 Database (CANdb) Setup | J1939 Receive Message
<examplesgroup role="model"></examplesgroup>

External Websites

- www.sae.org/

Introduced in R2009a

Shared Memory Support

This topic describes implementations of reflective (shared) memory by various manufacturers.

Create GE Fanuc Shared Partitions

The Simulink Real-Time software uses a model for reflective (shared) memory that includes Simulink blocks for shared memory driver functions. To define node initialization and shared memory partitions, the driver functions use MATLAB structures. This topic describes the Simulink Real-Time support of the GE Fanuc Embedded Systems shared memory boards.

To use the GE Fanuc Embedded Systems shared memory blocks, you must define shared memory partition structures. A partition structure describes how you want to allocate (partition) the shared memory. The Simulink Real-Time software allocates shared memory as segments of data that are packed into memory regions or partitions. Along with the Shared Memory Pack and Shared Memory Unpack blocks, the GE Fanuc Embedded Systems shared memory blocks use shared memory partition structures.

After defining the shared memory partitions, you can add shared memory driver blocks to your Simulink model. See “GE Fanuc Shared Partition Structure” on page 21-5 for the complete list of fields in a partition.

The following description refers to the `completepartitionstruct` command. Type

```
help completepartitionstruct
```

for more information.

- Create a partition structure in one of the following ways. Using the `completepartitionstruct` command at the MATLAB Command Window, create a default partition structure. For example, type

  ```
  Partition = completepartitionstruct([],'5565')

  Partition =

        Address: '0x0'
           Type: 'uint32'
           Size: '1'
      Alignment: '4'
       Internal: [1x1 struct]
  ```

- At the MATLAB Command Window, create a user-defined partition structure. Partially define a structure in a script file, run that script in the MATLAB workspace, and fill in the resulting structure with the `completepartitionstruct` function. For example:

```
Partition(1).Address='0x5000';
Partition(1).Type='int8';
Partition(1).Size='10';
Partition(2).Type='uint16';
Partition(2).Size='5';
Partition(3).Type='uint8';
Partition(3).Size='1';
Partition(3).Alignment='8';
Partition(4).Type='double';
Partition(4).Size='3';
```

This example defines a partition with four segments.

- The `Address` field is optional. Only specify this field for the first segment of a partition. The elements of a partition are defined as a continuous memory block from the first address. The function extrapolates segment addresses from the first segment definition. If you have or require fragmented memory, use multiple partitions.
- The `Type` and `Size` fields are required for all segments in the partition structure.
- The `Alignment` value is optional. It is `'4'` by default, which forces segments that do not have alignment specifications to start on 4 byte (32 bit) boundaries. In the preceding partition definition, the third segment (`Partition(3)`) has an alignment of `'8'`.
- The data type, size, and alignment of the preceding segment define the base addresses of subsequent segments.
- To populate the partition structure, call the `completepartitionstruct()` command.

  ```
  Partition = completepartitionstruct(Partition,'5565');
  ```

Initialize GE Fanuc Shared Nodes

In addition to GE Fanuc Embedded Systems shared memory partitions, you must also define a node initialization structure before using the shared memory blocks. A node initialization structure describes the shared memory partitions (see “Create GE Fanuc Shared Partitions” on page 21-2) and the board configuration, including interrupt settings.

After defining the node initialization structure, you can add shared memory driver blocks to your Simulink model. See “GE Fanuc Shared Node Initialization Structure” on page 21-7 for the complete list of fields in a node initialization structure.

The following description refers to the `completenodestruct` command. Type

```
help completenodestruct
```

for more information.

- Create a node initialization structure in one of the following ways. Using the `completenodestruct` command at the MATLAB Command Window, create a default node initialization structure. For example, type

  ```
  node=completenodestruct([],'5565')

  node =

       Interface: [1x1 struct]
      Partitions: [1x1 struct]
  ```

- Fill in the structure fields. For example:

  ```
  node.Interface.NodeID = '128';
  node.Partitions = Partition;
  ```

- A user-defined node structure, created with MATLAB code or from the MATLAB Command Window and supplement the resulting structure with a call to the `completenodestruct` function. For example:

  ```
  node.Interface.NodeID = '128';
  node.Partitions = Partition;
  node.completenodestruct(node,'5565');
  ```

GE Fanuc Shared Partition Structure

You do not need to use all the fields of a partition initialization structure. However, knowing the possible structure fields is helpful when you are setting up to use shared memory.

A shared memory partition structure has the following fields:

```
      Address: '0x0'
         Type: 'uint32'
         Size: '1'
    Alignment: '4'
     Internal: [1x1 struct]
```

where

Partition Fields	Description
`Address`	Specifies the base address (in hexadecimal) of the memory partition within the shared memory space of the node. The default value is `'0x0'`, the first location in shared memory. Align partition addresses on 32-bit word boundaries (for example, `0x0`, `0x4`, `0x8`).

Partition Fields	Description
`Type`	Specifies the data type of the memory segment. Specify one of the following types: • `single` (IEEE Single Precision) • `double` (IEEE Double Precision) • `uint8` • `int8` • `uint16` • `int16` • `uint32` • `int32` • `Boolean` (a single byte represents a boolean value) The default value is `'uint32'`. The minimum partition size is 32 bits.
`Size`	Specifies the dimension and size of the memory segment. You can enter a scalar value or a value with the `[m,n]` format. The default value is `'1'`. • scalar — Treats the `Size` entry as the specification of the length of a non-oriented array or vector • `[m,n]` — Treats the `Size` entry as an array dimension. The total number of elements in this segment is `m*n`.
`Alignment`	If another partition precedes this partition, this field defines the byte alignment of this segment. Specify one of the following alignment values: 1, 2, 3, 4, or 8. The default value is `'4'`. This value forces a double word boundary alignment.
`Internal`	Reserved for internal use.

GE Fanuc Shared Node Initialization Structure

A node initialization structure has the following fields:

```
     Interface: [1x1 struct]
    Partitions: [1x1 struct]
```

where

Node Structure Fields	Description
`Interface`	Specifies how the board is configured. The Interface structure has the following fields, three of which are structures: • `Mode` — Configures board registers (see “Board Mode” on page 21-7) • `Interrupts` — Enables the board to generate PCI interrupts from network events that have been broadcast from other nodes, or in response to error conditions (see “Board Interrupts” on page 21-8) • `NodeID` — Specifies the node ID for the board (see “Board Node ID” on page 21-10) • `Internal` — Reserved for internal use
`Partitions`	Stores the shared memory segments (see “Create GE Fanuc Shared Partitions” on page 21-2)

Board Mode

The shared memory board has several registers that you can set through the `Interface.Mode` field. To display the board mode fields, type:

```
>> node.Interface.Mode

ans =
          StatusLEDOff: 'off'
    TransmitterDisable: 'off'
      DarkOnDarkEnable: 'off'
        LoopbackEnable: 'off'
     LocalParityEnable: 'off'
```

```
MemoryOffset: '0'
  MemorySize: '64MByte'
```

The mode values interact with the board setting of the LSR1 (Local Control and Status Register 1) and LIER (Local Interrupt Enable Register) registers. Refer to the board product documentation for further details on these two registers. To monitor the status of these modes, select the **Error Status Port** check box of the read or write blocks.

Of particular note are the following modes:

Board Modes	Description
`StatusLEDOff`	Turns the board status LED on and off. Setting this value to 'off' turns off the LED when the Simulink Real-Time model runs, setting this value to 'on' turns on the LED when the Simulink Real-Time model runs. When the Simulink Real-Time software terminates, the LED status reverses in both cases. The default value is `'off'`.
`MemoryOffset`	Applies a global offset to the network data transfers coming from the board. The following table lists offset values and the resulting offset. The default value is `'0'`.
`MemorySize`	Specifies the minimum memory size required, in the format `'`*`size`*`MByte'`. The board driver checks this value against the memory size of the board. If you enter a size in this field that is larger than the actual board memory size, the driver returns an error.

This table lists the values for `MemoryOffset`:

Value	Offset Produced
`'0'`	0
`'1'`	0x4000000
`'2'`	0x8000000
`'3'`	0xC000000

Board Interrupts

The board can generate PCI interrupts in response to network events that have been broadcast from other nodes, or in response to error conditions. For example, you can configure two Simulink Real-Time Simulink models, one as master, and one as a slave of

the broadcast node in the master Simulink Real-Time model. In such a configuration, the broadcast node interrupt triggers the model time steps.

To display the interrupt mode fields, type:

```
node.Interface.Interrupts
```

```
ans =
       LocalMemoryParity: 'off'
    MemoryWriteInhibited: 'off'
         LatchedSyncLoss: 'off'
              RXFifoFull: 'off'
        RXFifoAlmostFull: 'off'
                 BadData: 'off'
             PendingInit: 'off'
             RoguePacket: 'off'
        ResetNodeRequest: 'off'
             PendingInt3: 'off'
             PendingInt2: 'off'
             PendingInt1: 'off'
```

Each field corresponds to a bit in the LIER register of the GE Fanuc Embedded Systems shared memory board. Each bit enables the specified interrupt source on the board. Refer to the board product documentation for further details on this register.

To enable a node to generate a network interrupt source, add the broadcast block to a model (the master model). This block issues network interrupts at the model sample rate. To enable other nodes of the network (the slave models) to accept broadcast interrupts, configure the slave models to expect the broadcast interrupt.

The following procedure describes how to configure an entire Simulink Real-Time model to accept a broadcast interrupt from the board.

1. From the MATLAB Command Window, type

   ```
   tg = slrt;
   getPCIInfo(tg, 'installed')
   ```

   This command lists board information for the installed PCI devices that the Simulink Real-Time software knows about.

2. Find the IRQ specified for the shared memory board.

   To specify in the **Real-Time interrupt source** field of the **Simulink Real-Time Options** pane, use this interrupt source number.

3 Edit your script and add a line like the following.

```
node.Interface.Interrupts.PendingInt1='on'
```

This line directs the model to expect an interrupt. It assumes that the value of the broadcast block **Interrupt parameter** is 1.

4 From the MATLAB Command Window, type the name of your Simulink model.

The Simulink model appears.

5 Select **Simulation > Model Configuration Parameters**

6 Select node **Code Generation**.

7 Select node **Simulink Real-Time Options**.

8 Set the **Execution mode** field to `Real-Time`.

9 Click the **Real-Time interrupt source** list.

10 Select the interrupt source number to which the board is set.

11 Click the **I/O board generating the interrupt** list and select `GE_Fanuc(VMIC)_PCI-5565` from the list.

12 Click **OK**.

Note If you have a larger model, to localize control of the interrupt within that model, use the IRQ Source block from the Asynchronous Event sublibrary.

Board Node ID

The jumpers of the board specify the board node ID. Correspondingly, you can also configure the block with the board node ID using the `Interface.NodeID` field. Enter values according to the following:

NodeID Value	Description
`'any'`	Allows the board driver to work with a node regardless of the board node ID jumper setting
value from `'0'` to `'255'`	Specifies the particular node that the driver must look for. If this value does not match the jumpered value on the board, the driver returns an error.

The default value of 'any' meets requirements in most instances. If you have multiple shared-memory boards in your system, to identify the driver for a particular node, specify a particular NodeID value.

Create Curtiss-Wright Shared Partitions

The Simulink Real-Time software uses a model for reflective (shared) memory that includes Simulink blocks for shared memory driver functions. To define node initialization and shared memory partitions, the driver functions use MATLAB structures. This topic describes Simulink Real-Time support of the Systran® shared memory board.

To use the Simulink Real-Time shared memory blocks, you must define shared memory partition structures. A partition structure describes how you want to allocate (partition) the shared memory. The Simulink Real-Time software allocates shared memory as segments of data that are packed into memory regions or partitions. Along with the Shared Memory Pack and Shared Memory Unpack blocks, the Systranshared memory blocks use shared memory partition structures.

After defining the shared memory partitions, you can add shared memory driver blocks to your Simulink model. See “Curtiss-Wright Shared Partition Structure” on page 21-15 for the complete list of fields in a partition structure.

The following description refers to the `completepartitionstruct` command. Type

```
help completepartitionstruct
```

for more information.

- Create a partition structure in one of the following ways. Using the `completepartitionstruct` command at the MATLAB Command Window, create a default partition structure. For example, type

  ```
  Partition = completepartitionstruct([],'scramnet')

  Partition =
          Address: '0x0'
             Type: 'uint32'
             Size: '1'
        Alignment: '4'
              RIE: 'off'
              TIE: 'off'
      ExtTrigger1: 'off'
      ExtTrigger2: 'off'
            HIPRO: 'off'
         Internal: [1x1 struct]
  ```

- At the MATLAB Command Window, create a user-defined partition structure. Partially define a structure in a script file, run that script in the MATLAB workspace, and fill in the resulting structure with the `completepartitionstruct` function. For example:

  ```
  Partition(1).Address='0x5000';
  Partition(1).Type='int8';
  Partition(1).Size='10';
  Partition(2).Type='uint16';
  Partition(2).Size='5';
  Partition(3).Type='double';
  Partition(3).Size='3';
  Partition(4).Type='uint8';
  Partition(4).Size='[2, 3]';
  ```

  This example defines a partition with four segments.

  - The `Address` field is optional. Only specify this field for the first segment of a partition. The elements of a partition are defined as a continuous memory block from the first address. The function extrapolates segment addresses from the first segment definition. If you have or require fragmented memory, use multiple partitions.
  - The `Type` and `Size` fields are required for all segments in the partition structure.
  - The `Alignment` value is optional. It is `'4'` by default. This value forces segments that do not have alignment specifications to start on 4 byte (32–bit) boundaries.
  - The data type, size, and alignment of the preceding segment define the base addresses of subsequent segments.
  - To populate the partition structure, call the `completepartitionstruct()` command.

    ```
    Partition = completepartitionstruct(Partition,'scramnet');
    ```

Initialize Curtiss-Wright Shared Nodes

In addition to shared memory partitions, you must also define a node initialization structure before using the Systran shared memory blocks. A node initialization structure describes the shared memory partitions (see “Create Curtiss-Wright Shared Partitions” on page 21-12) and the board configuration, including interrupt settings. The initialization block requires a shared memory node initialization structure.

After defining the node initialization structure, you can add shared memory driver blocks to your Simulink model. See “Curtiss-Wright Shared Node Initialization Structure” on page 21-21 for the complete list of fields in a node initialization.

The following description refers to the `completenodestruct` command. Type

```
help completenodestruct
```

for more information.

- Create a node initialization structure in one of the following ways. Using the `completenodestruct` command at the MATLAB Command Window, create a default node initialization structure. For example, type

  ```
  node=completenodestruct([],'scramnet')

  node =

       Interface: [1x1 struct]
      Partitions: [1x1 struct]
  ```

- Fill in the structure fields. For example:

  ```
  node.interface.NodeID = '128';
  node.Partitions = Partition;
  ```

- A user-defined node structure, created with MATLAB code or from the MATLAB Command Window and supplement the resulting structure with a call to the `completenodestruct` function. For example:

  ```
  node.Interface.NodeID = '128';
  node.Partitions = Partition;
  node = completenodestruct(node,'scramnet');
  ```

Curtiss-Wright Shared Partition Structure

A shared memory partition structure has the following fields. You do not need to use all the fields of a partition or node initialization structure. However, knowing the possible structure fields is helpful when you are setting up to use shared memory.

```
        Address: '0x0'
           Type: 'uint32'
           Size: '1'
      Alignment: '4'
            RIE: 'off'
            TIE: 'off'
    ExtTrigger1: 'off'
    ExtTrigger2: 'off'
          HIPRO: 'off'
       Internal: [1x1 struct]
```

where

Partition Fields	Description
Address	Specifies the base address (in hexadecimal) of the memory partition within the node shared memory space. The default value is `'0x0'`, the first location in shared memory. The base address is byte aligned.

Partition Fields	Description
Type	Specifies the data type of the memory segment. Specify one of the following types: • `double` • `float` • `uint8` • `int8` • `uint16` • `int16` • `uint32` • `int32` • `boolean` (a single byte represents a boolean value) The minimum partition size is 32 bits. The default value is `'uint32'`.
Size	Specifies the dimension and size of the memory segment. You can enter a scalar value or a value with the `[m,n]` format. The default value is `'1'`. • scalar — Treats the `Size` entry as the specification of the length of a non-oriented array or vector • `[m,n]` — Treats the `Size` entry as an array dimension. The total number of elements in this segment is `m*n`.
Alignment	Specifies the byte alignment of the next partition (if one is defined). Enter alignment value in bytes: `1`, `2`, `3`, `4`. The alignment value defines the end of the current segment, and therefore the beginning alignment of the next segment. The default value is `'4'`, forcing a double word boundary alignment. See “Alignment Examples” on page 21-19.

Partition Fields	Description
RIE	Specifies whether this partition can receive interrupts (Receive Interrupt Register (RIE)). Specify one of: • `'off'` (default) — Prevents the partition from receiving interrupts. • `'first'` — Allows only the first double word of the memory segment to be marked with the corresponding Auxiliary Control RAM bit. • `'all'` — Allows all memory locations of the memory segment to be marked with the corresponding Auxiliary Control RAM bit. • `'last'` — Allows only the last double word of the memory segment to be marked with the corresponding Auxiliary Control RAM bit.
TIE	Specifies whether this partition can transmit interrupts (Transmit Enable (TIE)). Specify one of: • `'off'` (default) — Prevents the partition from transmitting interrupts. • `'first'` — Allows only the first double word of the memory segment to be marked with the corresponding Auxiliary Control RAM bit. • `'all'` — Allows all memory locations of the memory segment to be marked with the corresponding Auxiliary Control RAM bit. • `'last'` — Allows only the last double word of the memory segment to be marked with the corresponding Auxiliary Control RAM bit.

Partition Fields	Description
ExtTrigger1	If this partition receives a write access, specifies whether this partition can generate a trigger signal to an external connector. Specify one of: • `'off'` (default) — Prevents the partition from generating signals. • `'first'` — Allows only the first double word of the memory segment to be marked with the corresponding Auxiliary Control RAM bit. • `'all'` — Allows all memory locations of the memory segment to be marked with the corresponding Auxiliary Control RAM bit. • `'last'` — Allows only the last double word of the memory segment to be marked with the corresponding Auxiliary Control RAM bit.
ExtTrigger2	If this partition receives a write access, specifies whether this partition can generate a trigger signal to an external connector. Specify one of: `'off'` (default) — Prevents the partition from generating signals. `'first'` — Allows only the first double word of the memory segment to be marked with the corresponding Auxiliary Control RAM bit. `'all'` — Allows all memory locations of the memory segment to be marked with the corresponding Auxiliary Control RAM bit. `'last'` — Allows only the last double word of the memory segment to be marked with the corresponding Auxiliary Control RAM bit.
HIPRO	Specifies whether the elements in this partition can be transmitted as one network message. Specify one of: `'off'` (default) — Prevents the partition from transmitting the elements as one message `'on'` — Allows the partition to transmit the elements as one message
Internal	Reserved for internal use.

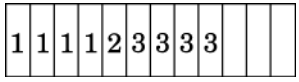
Alignment Examples

This example shows the shared memory map with default alignment values.

```
Partition1(1).Type='int32';
Partition1(1).Size='1';

Partition1(2).Type='boolean';
Partition1(2).Size='1';

Partition1(3).Type='uint32';
Partition1(3).Size='1';
Partition1 = completepartitionstruct(Partition1,'scramnet');
```

This example shows the shared memory map with alignment value changed from 4 to 1 in the second partition.

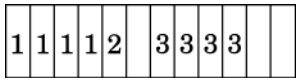
```
Partition1(1).Type='int32';
Partition1(1).Size='1';
Partition1(1).Alignment='4';

Partition1(2).Type='boolean';
Partition1(2).Size='1';
Partition1(2).Alignment='1';

Partition1(3).Type='uint32';
Partition1(3).Size='1';
Partition1 = completepartitionstruct(Partition1,'scramnet');
```



This example shows the shared memory map with alignment value changed from 4 to 2 in the second partition.

```
Partition1(1).Type='int32';
Partition1(1).Size='1';
Partition1(1).Alignment='4';

Partition1(2).Type='boolean';
```

```
Partition1(2).Size='1';
Partition1(2).Alignment='2';

Partition1(3).Type='uint32';
Partition1(3).Size='1';
Partition1 = completepartitionstruct(Partition1,'scramnet');
```



Curtiss-Wright Shared Node Initialization Structure

A node initialization structure has the following fields:

```
     Interface: [1x1 struct]
    Partitions: [1x1 struct]
```

where

Node Structure Fields	Description
Interface	Specifies settings for the board Control/Status Register (CSR). The Interface structure has the following fields. Refer to the board product documentation for a description of the CSR and its operation modes. • `Mode` — Configures board modes (see "Board Mode" on page 21-21) • `Timeout` — Enables the board to set the timeout value (see "Board Timeout" on page 21-23) • `DataFilter` — Controls the data filtering operation (see "Board Data Filter" on page 21-23) • `VirtualPaging` — Controls the board virtual paging operation (see "Virtual Paging" on page 21-24) • `Interrupts` — Enables the board to generate and receive interrupts from the network (see "Board Interrupts" on page 21-24) • `Internal` — Reserved for internal use
Partitions	Stores the shared memory segments (see "Create Curtiss-Wright Shared Partitions" on page 21-12)

Board Mode

The Systranshared memory board has modes that you can set through the `Interface.Mode` field. The Interface Mode fields set the corresponding bits in the CSR. To display the board mode fields, type:

```
>> node.Interface.Mode
ans =
```

```
          NetworkCommunicationsMode: 'TransmitReceive'
                         InsertNode: 'on'
          DisableFiberOpticLoopback: 'on'
                 EnableWireLoopback: 'off'
           DisableHostToMemoryWrite: 'off'
                 WriteOwnSlotEnable: 'off'
                 MessageLengthLimit: '256'
VariableLengthMessagesOnNetwork: 'off'
                       HIPROEnable: 'off'
                    MultipleMessages: 'on'
           NoNetworkErrorCorrection: 'on'
           MechanicalSwitchOverride: 'on'
                     DisableHoldoff: 'on'
```

These modes have the following values:

Field	Values	Default	CSR
NetworkCommunications Mode	`'none'`, `'receiveonly'`, `'transmitonly'`, `'transmit receive'`	`'transmit receive'`	CSR3[8..15]
InsertNode	`'off'`,`'on'`	`'on'`	CSR0[0..1]
DisableFiberOptic Loopback	`'off'`,`'on'`	`'on'`	CSR2[6]
EnableWire Loopback	`'off'`,`'on'`	`'off'`	CSR2[7]
DisableHost ToMemory Write	`'off'`,`'on'`	`'off'`	CSR2[8]
WriteOwnSlotEnable	`'off'`,`'on'`	`'off'`	CSR2[9]
Message LengthLimit	`'256'`,`'1024'`	`'256'`	CSR2[11]
Variable Length MessagesOn Network	`'off'`,`'on'`	`'off'`	CSR212]
HIPROEnable	`'off'`,`'on'`	`'off'`	CSR2[13]
Multiple Messages	`'off'`,`'on'`	`'on'`	CSR2[14]
NoNetwork Error Correction	`'off'`,`'on'`	`'on'`	CSR2[15]
Mechanical Switch Override	`'off'`,`'on'`	`'on'`	CSR8[11]
Disable Holdoff	`'off'`,`'on'`	`'on'`	CSR8[11]

Board Timeout

The Systranshared memory board allows you to set the network timeout through the `Interface.Timeout` field. The Interface Timeout fields set the corresponding bits in the CSR.

To display the timeout fields, type:

```
>> node.Interface.Timeout
ans =
       NumOfNodesInRing: '2'
    TotalCableLengthInM: '2'
```

These fields have the following values:

Field	Values	Default	CSR
NumOfNodes InRing	`'0' to '255'`	`'2'`	CSR5
TotableCable LengthInM	`'0' to '`*n*`'`	`'2'`	CSR5

Refer to the board product documentation for a description of these fields.

Board Data Filter

The Systranshared memory board allows you to set the data filter operation through the `Interface.DataFilter` field. The Interface DataFilter fields set the corresponding bits in the CSR.

```
>> node.Interface.DataFilter
ans =
           EnableTransmitDataFilter: 'off'
    EnableLower4KBytesForDataFilter: 'off'
 >>
```

These fields have the following values:

Field	Values	Default	CSR
Enable TransmitData Filter	`'off'`, `'on'`	`'off'`	CSR0[10]
EnableLower4KBytesFor DataFilter	`'off'`, `'on'`	`'off'`	CSR0[11]

Virtual Paging

The Systran shared memory board allows you to set the bits of the Virtual Paging Register operation through the `Interface.VirtualPaging` field. The Interface VirtualPaging fields set the corresponding bits in the CSR.

```
>> node.Interface.VirtualPaging
ans =
    VirtualPagingEnable: 'off'
      VirtualPageNumber: '0'
```

These fields have the following values:

Field	Values	Default	CSR
VirtualPagingEnable	`'off'`, `'on'`	`'off'`	CSR12[0]
VirtualPage Number	`'0'` to `'2047'`	`'0'`	CSR12[5..15]

Board Interrupts

The Systran shared memory board allows you to specify the interrupt sources transmitted and received between the nodes of the network. You can set these bits through the `Interface.Interrupts` field. The Interface Interrupts fields set the corresponding bits in the CSR.

```
>> node.Interface.Interrupts
ans =
                HostInterrupt: 'off'
    InterruptOnMemoryMaskMatch: 'off'
      OverrideReceiveInterrupt: 'off'
              InterruptOnError: 'off'
              NetworkInterrupt: 'off'
     OverrideTransmitInterrupt: 'off'
            InterruptOnOwnSlot: 'off'
      ReceiveInterruptOverride: 'off'
```

These fields have the following values:

Field	Values	Default	CSR
HostInterrupt	`'off'`, `'on'`	`'off'`	CSR0[3]
InterruptOn MemoryMask Match	`'off'`, `'on'`	`'off'`	CSR0[5]

Field	Values	Default	CSR
Override Receive Interrupt	`'off','on'`	`'off'`	CSR0[6]
InterruptOn Error	`'off', 'on'`	`'off'`	CSR0[7]
Network Interrupt	`'off','on'`	`'off'`	CSR0[8]
Override Transmit Interrupt	`'off', 'on'`	`'off'`	CSR0[9]
InterruptOn OwnSlot	`'off','on'`	`'off'`	CSR2[10]
Receive Interrupt Override	`'off','on'`	`'off'`	CSR8[10]

Video, XCP

Video Image Processing

Process Video Images with Simulink Real-Time

The Simulink Real-Time software supports webcams compliant with the USB Video Class (UVC) standard and cameras compliant with the Automated Imaging Association Camera Link® standard.

Note UVC-compliant cameras are often referred to as "driverless webcams". For more information, see your camera documentation.

Using blocks from the Simulink Real-Time Video library, on your target computer, you can do the following:

- Acquire real-time video frames from cameras connected to the target computer.
- Display the real-time image on the target computer monitor.
- Process or reduce the real-time image, for instance to specify a region of interest.
- Compress video frames acquired on the target computer.
- Stream video frames acquired on the target computer to the development computer.

Using blocks from the Computer Vision System Toolbox™ or Image Processing Toolbox™, on your development computer, you can do the following:

- Receive images from the target computer.
- Decompress video frames on the development computer.
- Process or reduce images.
- Display images on the development computer.

USB Video Display on Development Computer

The following workflow acquires and displays video frames using a USB Video Class (UVC) compliant webcam. To view the images on the development computer, compress the video frames on the target computer and transmit them to the development computer. You then decompress and display the video frames.

1. Enable USB general support on the target computer. See Speedgoat target computer documentation.
2. Acquire a USB Video Class (UVC) compliant webcam, and then connect it to the target computer USB port.
3. To query the available camera configurations, add the USB Video Device List block.
4. Add the image input block From USB Video Device to the portion of the model that runs on the target computer. Configure the block as required.
5. If your USB camera does not support on-chip JPEG compression, add the JPEG Compression block to the target computer portion. Connect this block to the output of the image input block.
6. Add the Image Transmit block to the target computer portion. Connect this block to the output of the compression block. Configure the block to transmit frames to the development computer.
7. Add the Image Receive block to the portion of the model that runs on the development computer. Configure the block to receive frames from the target computer.
8. Add the JPEG Decompression block to the development computer portion. Connect this block to the output of the Image Receive block.
9. Add the To Video Display block from the Computer Vision System Toolbox to the development computer portion.
10. Build and download the target computer portion of the model to the target computer.
11. Run the development computer portion of the model on the development computer.

USB Video Display on Target Computer

The following workflow acquires and displays video frames using a USB Video Class (UVC) compliant webcam. To view the images on the target computer, use a Video Display block.

1. Enable USB general support on the target computer. See Speedgoat target computer documentation.
2. Acquire a USB Video Class (UVC) compliant webcam, and then connect it to the target computer USB port.
3. To query the available camera configurations, add the USB Video Device List block.
4. Add the image input block From USB Video Device to the portion of the model that runs on the target computer. Configure the block as required.

   Record the setting of the **Image signal** parameter.
5. Add the Video Display block. Set its **Image signal** parameter to match the corresponding setting in the From USB Video Device block.
6. Build and download the target computer portion of the model to the target computer.
7. Execute the model on the target computer.

Serial Camera Configuration

To enter a serial command for the camera, enter it without quotation marks in the **Camera configuration serial command** text box.

Enter multiple commands as one line without white space.

Each camera manufacturer has a serial command set documented in the manual for each camera or family of cameras. Each command set is different from any other. For example, suppose that you want to put a camera from the given series into triggered mode and to set the shutter time to 1 ms. Use these settings and serial setup commands.

Camera Series	Mask Settings	Serial Setup
CIS VCC G21	9600 baud 8 bits No parity	`000000W004001\r` `000000W002005\r`
Pulnix TM1400	9600 baud 8 bits No parity	`:SA5\r`
Sony XCL	38,400 baud 8 bits No parity	`SHUTTER 7\r` `TRG_MODE 1\r`

Cohu 7800	9600 baud
	8 bits
	No parity
		`\002\0377cE2,10`*chk*`\003`
		`\002\0377cM2`*chk*`\003`
		`\002\0377T0,1`*chk*`\003`
		chk is a checksum for the given portion of the message.

For your camera, find the required command codes in the manufacturer documentation for that specific camera. If those codes are not in the manual, contact the camera manufacturer.

Some cameras return a success or failure value in response to the serial command. The image input block dialog box includes a check box to allow the software to display such values. Some responses include non-printable control characters, displayed as C escape sequences.

Escape Sequence	Control Character	Definition
`\r`	`CR`	Carriage Return
`\002`	`STX`	Start of Text
`\003`	`ETX`	End of Text
`\006`	`ACK`	Acknowledge
`\025`	`NACK`	Negative Acknowledge

Some camera manufacturers offer Windows utilities to send configuration commands to their cameras. Using such a utility, you can configure the camera on Windows and save the settings on the camera. You can then connect the camera to the target computer and use it.

Other camera manufacturers require a special serial cable that connects the serial communication line of the camera to a separate serial connection. In this case, you cannot initialize the camera using settings in the **Camera configuration serial command** section. Instead, you connect the DB9 to a serial port on the development computer and use manufacturer software to configure the camera before use.

Video Blocks

From USB Video Device

From USB Video Device block

Library

Simulink Real-Time Library for USB Camera

Description

The From USB Video Device block enables you to acquire real-time video frames or still images from a USB Video Class (UVC) webcam. You attach the webcam to a USB port on the target computer. After you acquire the image, you can:

- Display the output on the target computer monitor using a Video Display block.
- Stream captured frames to the development computer display (for example, using the To Video Display block from Computer Vision System Toolbox).
- Analyze the image signals on the development computer.
- Compress or decompress the input signal with the JPEG Compression or JPEG Decompression blocks.

When you add this block, also add the USB Video Device List block to help configure the webcam.

The **Image signal** setting determines the **Image signal** setting for blocks receiving this signal, such as the Video Display block.

Note When you execute a model containing a From USB Video Device block on a single-core target computer, insufficient time is sometimes available to process frames received

through the USB port. Under these conditions, the block can drop frames. If the block is dropping frames, specify a larger frame interval, lengthen the sample time, or use a multicore target computer.

Block Parameters

Configuration

Select a configuration that you specified in the USB Video Device List block. When you click the **Reload Device List** button on the USB Video Device List block, this configuration list is updated.

Port address (-1 for any)

Specify the port to which the webcam is attached. Enter `-1` for any USB port.

Image width

Enter the width of the image input from the USB port, in pixels.

Image height

Enter the height of the image input from the USB port, in pixels.

Frame interval

Select the sample time between frame transfers:

- `1/60`
- `1/30`
- `1/25`
- `1/20`
- `1/15`
- `1/10`
- `1/7.5`
- `1/5`

Frame format

Select whether the incoming frames are to be compressed:

- `Uncompressed`

  Do not compress frames.

- `MJPEG`

  Compress frames using Motion JPEG format. Each frame is individually compressed as a JPEG image. Selecting this option disables the **Color format** and **Image signal** parameters.

Color format

Select the color format for the incoming frames:

- `RGB24 (8:8:8)`

  Output frames using RGB24 color encoding.
- `YCbCr (4:2:2)`

  Output frames using YCbCr color encoding.

Image signal

- `One multidimensional signal`

  One signal where each dimension contains color information. Selecting this option creates one port, Image.
- `Separate color signals`

  Multiple color signals where each signal contains the information for one color. Selecting this option creates the following ports, depending upon the colorspace.

  - RGB: ports R, G, B
  - YCbCr: ports Y, Cb, Cr

Show trigger input

Select this check box to display an input port, Trigger, for the block.

Show length output

Select this check box to display an output port, Length, for the block.

See Also

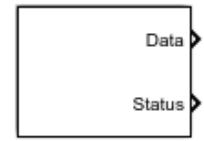
Topics

“Serial Camera Configuration” on page 22-5

Introduced in R2011a

Image Receive

Receive video image
Library: Video / Video Utilities



Description

Specify the source computer that the Image Receive block receives a video image from by using the IPv4 address-and-port pair.

Ports

Output Arguments

`Data` — Video data received by block
[byte]

Byte vector containing video data received by the block.

`Status` — Status of frame capture
`1` | `0`

Outputs `1` if the block received a full fixed-length frame. Otherwise, outputs `0`.

Dependency

This output is available when the **Allow variable length packets** check box is not selected.

`Length` — Number of bytes received
numeric

Outputs the number of bytes that the block received.

Dependency

This output replaces the **Status** output when you select the **Allow variable length packets** check box.

Parameters

`IP address to receive from (0.0.0.0 for accepting all)` — Source IP address

'0.0.0.0' (default) | 'xx.xx.xx.xx'

Enter a valid IPv4 address as a dotted decimal character vector for the source address, for example, 10.10.10.3. You can also use a MATLAB expression that returns a valid IPv4 address as a character vector.

The default address, 0.0.0.0, enables the block to accept frames from any accessible computer. If you set this parameter to a specific IP address, packets arriving from only that IP address are received.

The **IP port to receive from** parameter specifies the port for the source.

`IP port to receive from` — Source port

numeric

Specify the port of the computer from which to receive the video frames.

The **IP address to receive from** parameter specifies the IP address for the source.

`Output port width (number of bytes)` — Number of bytes that the block can propagate

50 * 1024 (default) | numeric

Number of bytes that the block can output in one sample time.

Dependency

- **Allow variable length packets** is selected:
  - Data size > **Output port width** — Block ignores the packet.
  - Data size ≤ **Output port width** — Block outputs packet data through the `Data` port and packet length through the `Length` port. Use packet length to consume data.

- **Allow variable length packets** is not selected:
  - Data size ≠ **Output port width** — Block ignores the packet.
  - Data size = **Output port width** — Block outputs packet data through the `Data` port. The `Status` port is set to `1`.

`Allow variable length packets` — Receive variable-length frames
`0` (default) | `1`

Select this check box to enable the reception of variable-length frames. For example, use this option for compressed frames because the length of each frame varies.

Dependency

Selecting this parameter replaces the default Status output port with the Length output port.

`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.

See Also

Image Transmit

Topics

“Serial Camera Configuration” on page 22-5

Introduced in R2011a

Image Transmit

Transmit video frame
Library: Video / Video Utilities

Description

The Image Transmit block sends the video frame from one computer to another computer. The computers can be two target computers or one development computer and one target computer. Specify the destination computer by IPv4 address-and-port pair.

Ports

Input

`Data` — Video data to transmit

[byte]

Byte vector containing video data for the block to transmit.

`Length` — Number of bytes of video data

numeric

Number of bytes to transmit.

Dependency

This input is available when you select the **Allow variable length packets** check box.

Parameters

`IP address sent to (255.255.255.255 for broadcast)` — Computer IP address
'255.255.255.255' (default) | 'xx.xx.xx.xx'

Specify the IPv4 address of the computer to which you send the video frames. To broadcast the video frames to all listening computers, enter `255.255.255.255`. The **Remote IP port to send to** parameter specifies the port for the destination.

`Remote IP port to send to` — Computer port
numeric

Specify the computer port to which you send the video frames. The **IP address sent to (255.255.255.255 for broadcast)** parameter specifies the IP address for the destination.

`Use the following local IP port (-1 for automatic port assignment)` — Local computer port
numeric

Specify the computer port from which to send the video frames.

To assign automatically a port for the computer, enter `-1`.

`Allow variable length packets` — Transmit variable-length frames
`0` (default) | `1`

Select this check box to enable the transmission of variable-length frames. For example, use this option for compressed frames because the length of each frame varies.

If you select this check box, the `Data` port sends the actual data. The `Length` input port sends the number of bytes being transmitted. If the port size is less than `Length`, the block sends up to the port size.

If this check box is not selected (default), the block sends only fixed-length packets.

Dependency

Selecting this parameter displays the Length output port.

`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.

See Also

Image Receive

Topics

“Serial Camera Configuration” on page 22-5

Introduced in R2011a

JPEG Compression

JPEG Compression block

Library

Simulink Real-Time Library for Video Utilities

Description

The JPEG Compression block compresses the video frame received by the target computer.

Block Inputs and Outputs

Inputs

Name	Description
Image	Video frame to compress

Outputs

Name	Description
Data	Vector of `byte` containing video data compressed by block.
Length	Number of bytes of compressed data. Visible when the **Show output image length** check box is set.

Block Parameters

Compression quality (-1:default)

Enter a value between `0` and `100` to specify how much to compress the incoming video frame. The lower the value, the less the compression quality.

Enter `-1` to use the default compression quality for the video frame.

Input colorspace

One of the following:

- `Grayscale`

  Compress the video frame using a grayscale color space scheme.
- `YCbCr 4;4:4`

  Compress the video frame using the YCbCr color space scheme.
- `RGB`

  Compress the video frame using the red, blue, green (RGB) color space scheme.

Image signal

- `One multidimensional signal`

  One signal where each dimension contains color information. Selecting this option creates one port, Image.
- `Separate color signals`

Multiple color signals where each signal contains the information for one color. Selecting this option creates the following ports, depending upon the colorspace.

- Grayscale: port Image
- RGB: ports R, G, B
- YCbCr: ports Y, Cb, Cr

Max output image size (bytes)

Enter the maximum output size for the compressed video frame, in bytes. Use format *`height`* * *`width`*.

Show output image length

Select this check box to output the video frame length. Selecting this check box displays the Length port.

See Also

JPEG Decompression

Introduced in R2011a

JPEG Decompression

JPEG Decompression block

Library

Simulink Real-Time Library for Video Utilities

Description

The JPEG Decompression block decompresses the video frame received by the target computer.

Block Inputs and Outputs

Inputs

Name	Description
Data	Vector of `byte` containing video data for the block to decompress.
Trigger	Trigger for video decompression, active high. Visible when the **Show trigger input** check box is set.

Outputs

Name	Description
Image	Decompressed video frame

Block Parameters

Image width

Specify the width, in bytes, of the video frame to be decompressed.

Image height

Specify the height, in bytes, of the video frame to be decompressed.

Output colorspace

One of the following:

- `Grayscale`

  Compress the video frame using a grayscale color space scheme.
- `YCbCr 4;4:4`

  Compress the video frame using the YCbCr color space scheme.
- `RGB`

  Compress the video frame using the red, blue, green (RGB) color space scheme.

Image signal

- `One multidimensional signal`

  One signal where each dimension contains color information. Selecting this option creates one port, Image.
- `Separate color signals`

  Multiple color signals where each signal contains the information for one color. Selecting this option creates the following ports, depending upon the colorspace.

  - Grayscale: port Image
  - RGB: ports R, G, B

- YCbCr: ports Y, Cb, Cr

Show trigger input

Select this check box to display an input port, Trigger, for the block.

See Also

JPEG Compression

Introduced in R2011a

USB Video Device List

USB Video Device List block

Library

Simulink Real-Time Library for USB Camera

Description

When you connect a USB Video Class (UVC) webcam to the target computer, the USB Video Device List block probes the device and displays the manufacturer information of the webcam. Based on this information, the block configures its parameters with supported options. You can choose the configuration parameters required by your USB webcam, and then name the configuration for future use.

When you add the From USB Video Device block to your model, add the USB Video Device List block. You need the USB Video Device List block to configure the webcam.

Block Parameters

Manufacturer

Select the manufacturer for the installed webcam.

Format

Select an image format that the connected webcam supports, for example, `MJPEG`.

Resolution

Select a supported resolution for the image.

Interval

Select the sample time for the video frame.

Configurations

Use this parameter to store and name a particular configuration. A configuration consists of the settings that you select for a particular webcam.

After you select the options in the other parameters:

1. In the edit field, enter a name for the configuration.
2. Click the **Add** button to add the configuration.

To remove the configuration, click the **Remove** button.

Reload Device List

Click this button to refresh the list of webcam information. Clicking this button also updates the mask and parameters for the From USB Video Device block.

See Also

Image Receive

Topics

“Serial Camera Configuration” on page 22-5

Introduced in R2011a

Video Display

Video Display block

Library

Simulink Real-Time Library for Video Utilities

Description

When you add the Video Display block to your model, you can display an RGB video signal on the target computer. The signal can come from a USB Video webcam or a constant block.

The **Image signal** setting must match the **Image signal** setting for the camera output block.

There can be no more than two Video Display blocks in a model. The combined number of Video Display blocks and target scopes cannot exceed nine.

Block Parameters

Image signal

- `One multidimensional signal`

  One signal where each dimension contains color information. Selecting this option creates one port.
- `Separate color signals`

Multiple color signals where each signal contains the information for one color. Selecting this option creates three ports.

Image colorspace

Can only be RGB.

See Also

“Target Scope Usage”

Introduced in R2014b

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 using XCP in master mode to replace (bypass) a subsystem of the ECU controller. The bypass model applies stimulus to the subsystem output signals and acquires signal response from the ECU controller.

To support XCP master mode, the Simulink Real-Time software provides the XCP sublibrary. You can:

- Parse A2L (ASAP2 database) files.
- Synchronize one or more slave or ECU devices.
- Initialize an XCP slave server running in an ECU.
- Apply stimulus data.
- Acquire real-time measurement data when specific events occur.

To create models to run in master mode:

- Provide an A2L (ASAP2) format file that contains signal and parameter access information for the slave ECUs and for the XCP-specific network elements.
- Provide an XCP Configuration block to load the A2L data into the XCP database.
- Provide one XCP CAN Transport Layer or XCP UDP Transport Layer block for each XCP 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 Transport Layer | XCP Configuration | XCP Data Acquisition | XCP Data Stimulation | XCP UDP Transport Layer

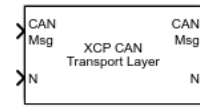
More About

- “CAN”
- “Real-Time UDP”
- “Third-Party Calibration Support”

XCP Blocks

XCP CAN Transport Layer

Generate and consume XCP messages that are transported by CAN hardware
Library: XCP



Description

The XCP CAN Transport Layer block handles CAN messages that your model transmits or receives with 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 Configuration | XCP Data Acquisition | XCP Data Stimulation

External Websites

www.asam.net

Introduced in R2014a

XCP Configuration

Configure XCP slave connection

Library

XCP Communication

Description

The XCP Configuration block uses the parameters specified in the A2L file and the ASAP2 database to establish XCP slave connection.

Specify the A2L file to use in your XCP Configuration before you acquire or stimulate data. Use one XCP Configuration to configure one slave for data acquisition or stimulation. If you add Data Acquisition and Data Stimulation blocks, your model checks to see if there is a corresponding XCP Configuration block and will prompt you to add one.

Other Supported Features

The XCP communication blocks support the use of Simulink Accelerator and Rapid Accelerator mode. Using this feature, you can speed up the execution of Simulink models. For more information on this feature, see the Simulink documentation.

The XCP communication blocks also support code generation with limited deployment capabilities. Code generation requires the Microsoft C++ compiler.

Parameters

Config name

Specify a unique name for your XCP session.

A2L File

Click **Browse** to select an A2L file for your XCP session.

Enable seed/key security

Select this option if your slave requires a secure key to establish connection. You need to select a file that contains the seed/key definition to enable the security.

File (*.DLL)

This field is enabled if you select **Enable seed/key security**. Click **Browse** to select the file that contains seed and key security algorithm used to unlock an XCP slave module.

Output connection status

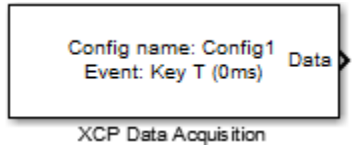
Select this option to display the status of the connection to the slave module. Selecting this option adds a new output port.

See Also

Introduced in R2013a

XCP Data Acquisition

Acquire selected measurements from configured slave



Library

XCP Communication

Description

The XCP Data Acquisition block acquires data from the configured slave based on the selected measurements. 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 Data Acquisition block to select your event and measurements for the configured slave. The block displays the selected measurements as output ports.

Note A model with XCP Data Acquisition blocks does not disconnect from the XCP slave when the simulation ends. The model continues to acquire measurements until the data transmission from the XCP slave is terminated.

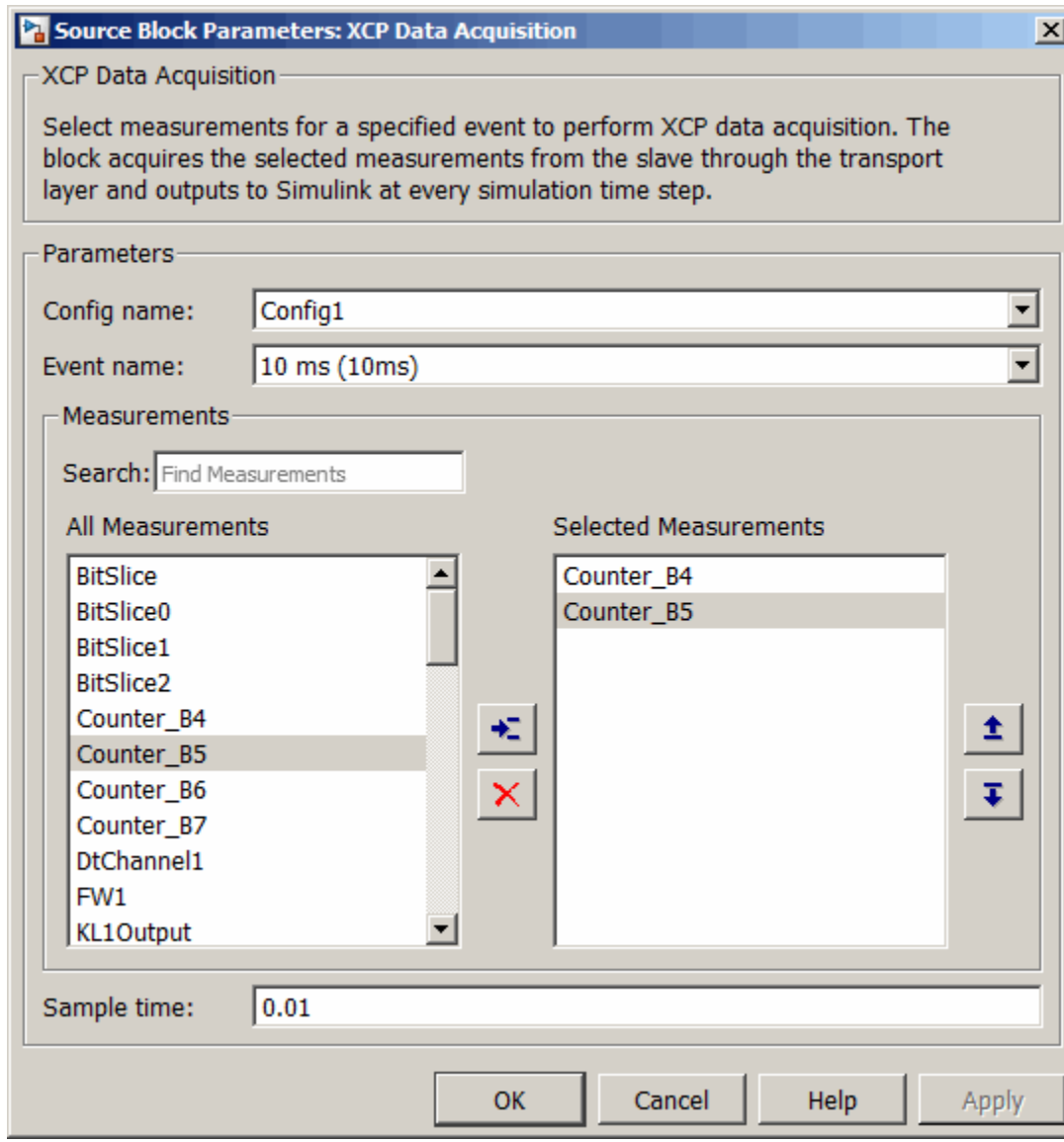
Other Supported Features

The XCP communication blocks support the use of Simulink Accelerator and Rapid Accelerator mode. Using this feature, you can speed up the execution of Simulink models. For more information on this feature, see the Simulink documentation.

The XCP communication blocks also support code generation with limited deployment capabilities. Code generation requires the Microsoft C++ compiler.

Dialog Box

Use the Block Parameters dialog box to select your data acquisition parameters.



Parameters

Config name

Select the name of XCP configuration you want to use. The list displays all available names specified in the available XCP Configuration blocks in the model. Selecting a configuration displays events and measurements available in this configuration's A2L file.

Note You can acquire measurements for only one event using an XCP Data Acquisition block. Use one block each for each event whose measurements you want to acquire.

Event name

Select an event from the available list of events. The XCP Configuration block uses the specified A2L file to populate the events list.

Measurements

Search

Type the name of the measurement you want to use. The All Measurements lists displays a list of all matching terms. Click the x

to clear your search.

All Measurements

This list displays all measurements available for the selected event. Select the measurement you want to use and click the add button,  to add it to the selected measurements. Hold the `Ctrl` key on your keyboard to select multiple measurements.

Selected Measurements

This list displays selected measurements. To remove a measurement from this list, select the measurement and click the remove button,  .

Toggle buttons

Use the toggle buttons  to reorder the selected measurements.

Sample time

Specify the sampling time of the block during simulation, which is the simulation time as described by the Simulink documentation. This value defines the frequency at which the XCP Data Acquisition block runs during simulation. If the block is inside a triggered subsystem or 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 (in seconds).

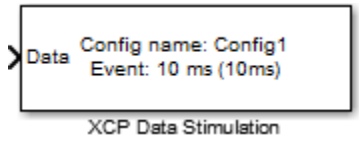
See Also

XCP Configuration

Introduced in R2013a

XCP Data Stimulation

Perform data stimulation on selected measurements



Library

XCP Communication

Description

The XCP Data Stimulation block sends data to the selected slave 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 Data Stimulation block to select your event and measurements on the configured slave. The block displays the selected measurements as input ports.

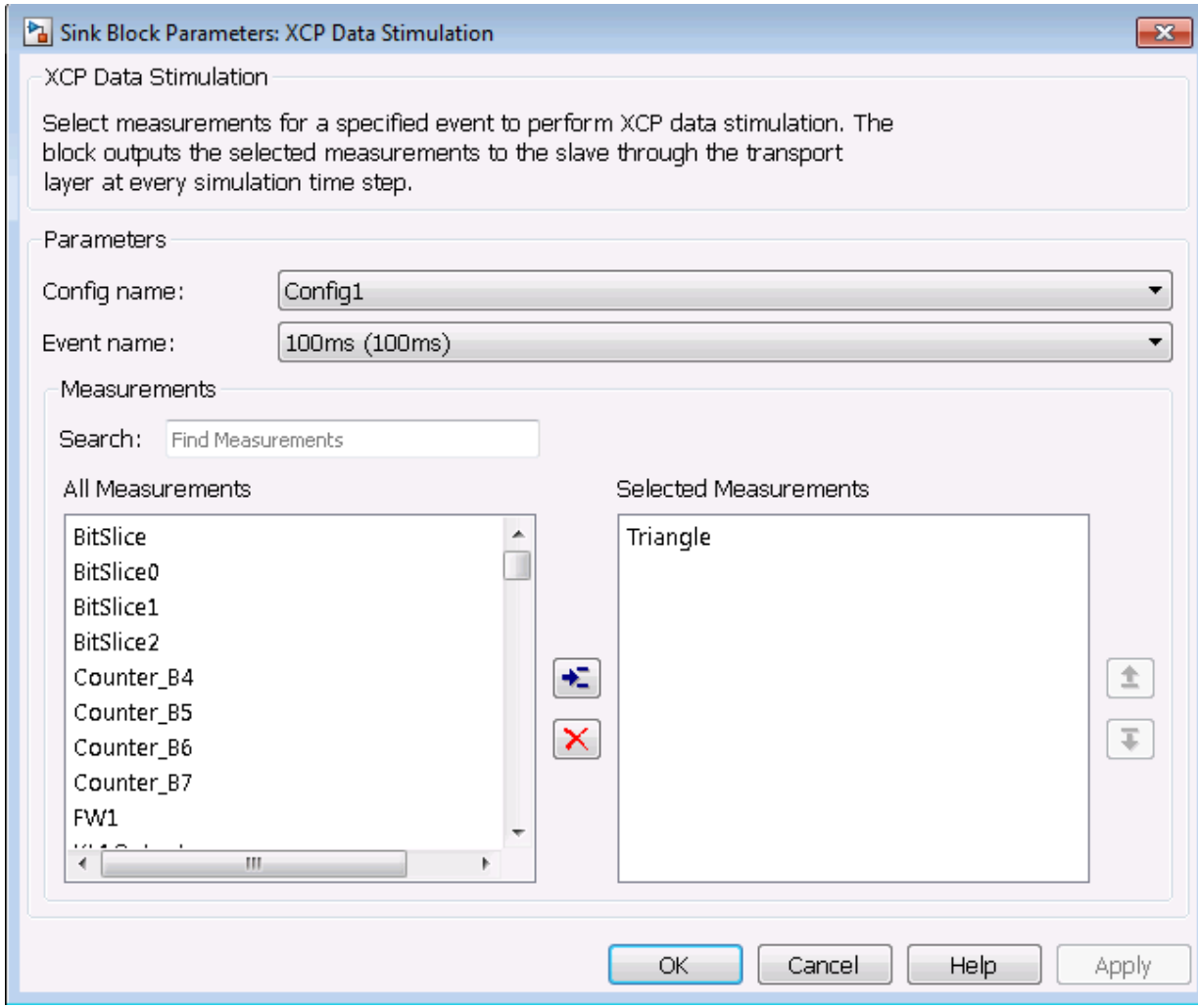
Other Supported Features

The XCP communication blocks support the use of Simulink Accelerator and Rapid Accelerator mode. Using this feature, you can speed up the execution of Simulink models. For more information on this feature, see the Simulink documentation.

The XCP communication blocks also support code generation with limited deployment capabilities. Code generation requires the Microsoft C++ compiler.

Dialog Box

Use the Block Parameters dialog box to select your data stimulation parameters.



Parameters

Config name

Select the name of XCP configuration you want to use. The list displays all available names specified in the available XCP Configuration blocks in the model. Selecting a

configuration displays events and measurements available in this configuration's A2L file.

Note You can stimulate measurements for only one event using an XCP Data Stimulation block. Use one block each for each event whose measurements you want to stimulate.

Event name

Select an event from the available list of events. The XCP Configuration block uses the specified A2L file to populate the events list.

Measurements

Search

Type the name of the measurement you want to use. The All Measurements lists displays a list of all matching terms. Click the x

to clear your search.

All Measurements

This list displays all measurements available for the selected event. Select the measurement 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.

Selected Measurements

This list displays selected measurements. To remove a measurement from this list, select the measurement and click the remove button,  .

Toggle buttons

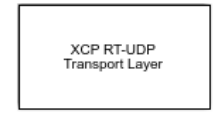
Use the toggle buttons  to reorder the selected measurements.

See Also

Introduced in R2013a

XCP UDP Transport Layer

Send and receive XCP messages over real-time UDP
Library: XCP



Description

The XCP UDP Transport Layer block uses the specified Ethernet device to send and receive XCP messages. Specify the Ethernet device using the PCI bus, slot, and function numbers.

The combination of **Local IP Address** and **Subnet mask** must be unique across all Ethernet cards in the target computer, including the card for communicating between the development and target computers. Distinguish cards by specifying a different subnet for each. The subnet is the IP address masked by the subnet mask.

Parameters

`Local IP Address` — IP address for the Ethernet interface
`x.x.x.x`

Enter the IP address for the dedicated Ethernet board.

The addresses `0.0.0.0` and `255.255.255.255` are invalid local IP addresses.

`Local port (0 for automatic assignment)` — Port address for the Ethernet interface
`0` (default) | integer

If **Local port** is `0`, the block chooses a port address from those available.

`Subnet mask` — Subnet mask for interface
`255.255.255.0` (default) | *`x.x.x.x`*

Mask that designates a logical subdivision of a network.

`Gateway` — IP address for gateway interface
`0.0.0.0` (default) | *`x.x.x.x`*

The gateway must be within the network.

To indicate that a gateway is not being used, enter `0.0.0.0` (the default). The address `255.255.255.255` is an invalid gateway IP address.

`PCI bus` — PCI bus number of Ethernet card
`0` (default) | integer

Enter the PCI bus number for the Ethernet card.

`PCI slot` — PCI slot number of Ethernet card
`0` (default) | integer

Enter the PCI slot number for the Ethernet card.

`PCI function` — PCI function number of Ethernet card
`0` (default) | integer

Enter the PCI function number for the Ethernet card.

`Sample time` — Sample time of block
`.01` (default) | numeric

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

`MaxNumMessages` — Maximum number of XCP messages
`1` (default) | integer

Maximum number of XCP messages sent

See Also

XCP Configuration | XCP Data Acquisition | XCP Data Stimulation

External Websites

www.asam.net

Introduced in R2014a

Speedgoat

Speedgoat Support

Speedgoat Target Computers and Support

Speedgoat target computers are real-time computers fitted with a set of I/O hardware, Simulink programmable 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 for use in confined areas. Provides extended operating temperature. Supports up to 14 I/O modules.
- Baseline — Small, rugged, and fanless real-time system. For mobile, in-vehicle, and classroom use, and for use in confined areas. Special pricing for academia available. Provides extended operating temperature. Supports up to 7 I/O modules
- Audio — Real-time system optimized for audio applications, such as hearing aids and car acoustics.

When you install the Speedgoat block library, the installer sets up help for the blocks in the MATLAB Help browser. To view the block library documentation, open the Help browser and navigate to the home page. At the bottom right of the home page, under **Supplemental Software**, click **Simulink Real-Time - Speedgoat Library**. The help opens in the current window.

To install your Speedgoat library, navigate on the Internet to `www.speedgoat.com/login`, the Speedgoat Customer Portal. Follow the directions to download and install your library.

You can find Speedgoat real-time target machine configuration documentation online here:

`www.speedgoat.com/help`

You can find Speedgoat real-time target machine product information online here:

`www.speedgoat.com/products`

Speedgoat I/O Hardware

Speedgoat provides a wide range of I/O hardware with ready-to-use configurations that include I/O emulation products typically used with hardware-in-the-loop (HIL) simulations. Speedgoat I/O connectivity includes support for:

- Analog I/O: A/D, D/A, single or differential, with or without isolation, 16–24 bit, both voltage and current
- 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
  - I^2C master and slave
  - Cam and crank decoder and simulator (measurement and simulation)
  - UART (RS-485/RS-422)
  - Aurora 64B/66B master and slave
- 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

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, Modbus TCP, Modbus RTU, POWERLINK
- 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 and UVC-compliant USB video cameras (webcams)
- Aurora 64B/66B multigigabit links for FPGA

See Also

External Websites

- www.speedgoat.com/help
- www.speedgoat.com/products
- www.speedgoat.com

UEI, Asynchronous Events

Asynchronous Events

Asynchronous Event Support

In this section...

Adding an Asynchronous Event

The Simulink Real-Time software includes support for asynchronous events in response to an interrupt from I/O boards. In response to these interrupts, the CPU can suspend normal execution and jump to another section of code called an Interrupt Service routine (ISR).

When developing an Simulink Real-Time model, you can model an Interrupt Server Routine (ISR) by using a Function-Call Subsystem. Also, add an IRQ Source block connected to the Function-Call Subsystem block. This subsystem is then executed when an interrupt occurs and the CPU is ready to accept it.

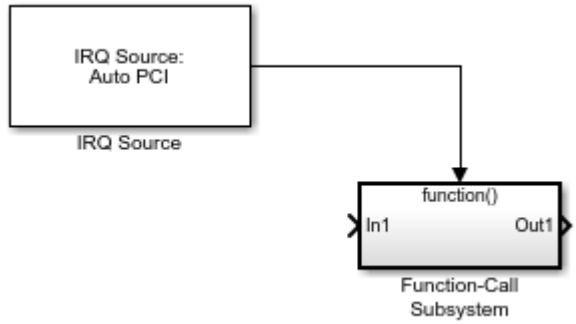
After you install an I/O board with interrupt support into your target computer, you can add Simulink Real-Time asynchronous blocks to your Simulink model.

1. In the MATLAB Command Window, type

   ```
   slrtlib
   ```

   The Simulink Real-Time Library opens.
2. Double-click the Asynchronous Event group block.

   The Library: slrtlib/Asynchronous Event window opens.
3. Drag the Simulink Real-Time IRQ block into your Simulink model and connect the output to this block to the input of a Function-Call Subsystem. For more information on Function-Call subsystems, see the Simulink and Simulink Coder documentation.



In the setup shown above, the CPU executes the contents of the Function Call-Subsystem whenever IRQ 5 occurs.

4 Double-click the IRQ Source block.

The Block Parameters: IRQ Source dialog box opens.

5 To determine and use the IRQ that the BIOS assigned to the board, from the **IRQ line number** list, select `Auto (PCI only)`.

Alternatively, select one of the values 3–15 for this number. To determine the available IRQ line numbers on the target computer, use the function `SimulinkRealTime.target.getPCIInfo`.

6 From the **I/O board generating the interrupt** drop-down list, select an interrupt board.

7 In the **PCI slot (-1: autosearch) or ISA base address** field, enter the PCI slot number or enter `-1` to let the Simulink Real-Time software determine the number.

8 Click **OK**.

For more information about the IRQ Source block, see Async IRQ Source.

To transfer data from your ISR, add an Async Transition block or Async Read/Write block to your Simulink model. See Async Rate Transition, Async Buffer Write and Read, and “Asynchronous Interrupt Examples” on page 27-4.

If you are using a CAN field bus with interrupts, see “Asynchronous Interrupt Examples” on page 27-4.

Asynchronous Interrupt Examples

The Simulink Real-Time software provides several example models. If you installed the MATLAB software in the default location, these models are located here:

```
C:\MATLAB\toolbox\rtw\targets\xpc\xpcdemos
```

To access these models, in the MATLAB Command Window, type the name of the model. Each model contains annotations documenting its purpose, and serves as an example of how to use these blocks.

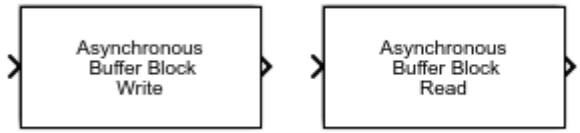
- `xpcasynctrans` — Model using an external TTL signal to trigger and interrupt on the parallel port. Data exchange between an asynchronous task and a rate monotonic task using an Async Rate Transition block.
- `xpccanintpc104` — Model using interrupt driven CAN I/O communication with the CAN-AC2-104 board.
- `xpccanintpci` — Model using interrupt driven CAN I/O communication with the CAN-AC2-PCI board.

Asynchronous Event: Blocks

This topic describes the Target Management library and Displays and Logging library blocks:

Async Buffer Write and Read

Async Buffer Write and Read blocks



Library

Simulink Real-Time Library for Asynchronous Event

Description

These blocks provide double buffering of data between the ISR and the model which executes rate-monotonically in real time. Use these blocks in pairs with an Async Buffer Write Block leading into an Async Buffer Read block. The Async Buffer Write Block has to be part of the ISR, and the Async Buffer Read block is outside the ISR.

Unlike the rate transition block, Async Buffer Write and Read blocks do not copy data from one buffer to another. Instead, the software disables interrupts and swaps buffer pointers. This method disables interrupts for a shorter time than the rate transition block and protects against data corruption caused by overwriting partially copied buffers.

Block Parameters

Sample time

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

See Also

Topics

“Asynchronous Event Support” on page 27-2

Introduced before R2006a

Async IRQ Source

Async IRQ Source block

Library

Simulink Real-Time Library for Asynchronous Event

Description

The IRQ Source block configures the Simulink and Simulink Real-Time software to treat a particular Function-Call Subsystem as an Interrupt Service Routine (ISR). This block is actually a virtual block and does not exist at model execution time. However, the model initialization code sets up the CPU to execute the ISR when the specified interrupt occurs.

Block Parameters

IRQ line number

Select `Auto (PCI only)` to enable the Simulink Real-Time software to automatically determine the IRQ that the BIOS assigned to the board and use it.

Alternatively, select the IRQ line number you are using for this block. This depends on the characteristics of your I/O module. You may need to query the PCI bus in the target computer to find what IRQ the PCI bus assigned to your I/O module. Use the function `SimulinkRealTime.target.getPCIInfo`.

Valid IRQ numbers are between 3 and 15.

I/O board generating the interrupt

For many I/O boards, you need to set up the board to generate the interrupt. You might also need to set up board specific features at the beginning and/or end of an ISR. Select the board you intend to use from the drop-down list.

PCI slot (-1: autosearch) or ISA base address

- If PCI:

  If only one board of this type is in the target computer, enter `-1` to automatically locate the board.

  If two or more boards of this type are in the target computer, enter the bus number and the PCI slot number of the board associated with this driver block. Use the format [BusNumber, SlotNumber]. To determine the bus number and the PCI slot number, type:

  ```
  tg = slrt;
  getPCIInfo(tg, 'installed')
  ```

- If ISA, enter the base address.

See Also

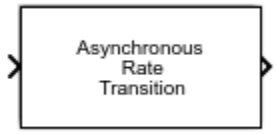
Topics

“Asynchronous Event Support” on page 27-2

Introduced before R2006a

Async Rate Transition

Async Rate Transition block



Library

Simulink Real-Time Library for Asynchronous Event

Description

Use the Asynchronous Rate Transition block to double buffer data between the function call subsystem and the rest of the model, which executes rate-monotonically in real time.

Normally, the interrupt service routine writes to the first buffer. When the next model step executes, the first buffer is copied to the second buffer and its value is used for model calculations.

If a second interrupt occurs while the buffer is being copied, data is corrupted. The CPU copies part of the data from the first buffer. When the second interrupt occurs, it writes over the entire first buffer. When the CPU returns from the second interrupt, it continues the copy operation from the first buffer, which contains data written during the second interrupt.

To prevent data corruption, use Async Buffer Write and Read blocks.

Block Parameters

Sample time

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

See Also

Topics

“Asynchronous Event Support” on page 27-2

Introduced before R2006a

Utility Drivers, Target Management, Displays and Logging

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. Others are available as sublibraries of the I/O function they support.

Bit Packing

Construct data frames

Library

Simulink Real-Time Library for Utilities

Description

This 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.

Block Parameters

Bit Patterns

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, and therefore the data types of the input ports are inherited from the driving blocks.

Output port (packed) data type

From the list, select an output port (packed) data type.

- `double`
- `single`
- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean`

Output port (packed) dimensions

Specify the dimensions the output port (packed). Enter this as a vector. Specify the size of the port using a format compatible with the MATLAB `size` command.

See Also

Bit Unpacking

Introduced in R2006a

Bit Unpacking

Deconstruct data frames

Library

Simulink Real-Time Library for Utilities

Description

This block is used to extract 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.

Block Parameters

Bit Patterns

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.

Input port (packed) data types

From the list, select an input port (packed) data type.

- `double`
- `single`
- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean`

Input port (packed) dimension

Specify the dimensions of the input port (packed). Enter this as a vector. Specify the size of the port using a format compatible with the MATLAB `size` command.

Output port (unpacked) data types (cell array)

The output ports (packed) can be of 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 can be

- `double`
- `single`
- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean`

Output port (unpacked) dimension (cell array)

Specify the dimensions of each output port (unpacked). Enter this as a cell array of vector sizes.

Sign extend

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.

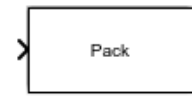
See Also

Bit Packing

Introduced in R2006a

Byte Packing

Construct data frames
Library: 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.

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` | `uint8` | `uint16` | `uint32` | `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` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `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.

`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.

`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.

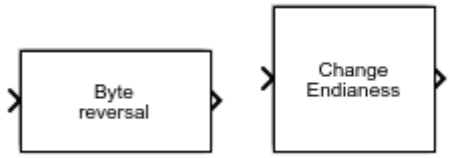
See Also

Byte Unpacking

Introduced in R2006a

Byte Reversal/Change Endianess

Reverse little-endian data for big-endian processor



Library

Simulink Real-Time Library for 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.

Block Parameters for Change Endianess

Number of input ports

The number of input ports adjusts automatically to follow this parameter, and the number of outputs is equal to the number of inputs.

Machine word length

Select one of the following machine word lengths to which to convert the data:

- Byte
- Word
- Double Word

The following is the Byte Reversal block.

Byte Reversal Block Parameters

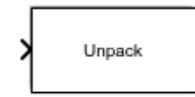
Number of inputs

The number of input ports adjusts automatically to follow this parameter, and the number of outputs is equal to the number of inputs.

Introduced in R2006a

Byte Unpacking

Deconstruct data frames
Library: Utilities



Description

This 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.

For example, suppose that you are unpacking a `uint8` vector signal into three signals. The signals have the following 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` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `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` | `uint8` | `uint16` | `uint32` | `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.

`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.

`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

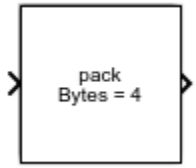
See Also

Byte Packing

Introduced in R2006a

Shared Memory Pack

Shared memory pack



Library

Simulink Real-Time Library for 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.

Block Parameters

Partition struct

Enter the name of the predefined shared memory partition structure.

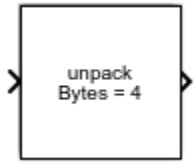
See Also

Shared Memory Unpack

Introduced in R2006a

Shared Memory Unpack

Shared memory unpacking



Library

Simulink Real-Time Library for 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.

Block Parameters

Partition struct

Enter the name of the predefined shared memory partition structure. The block unpacks the double word array vector into this structure.

See Also

Shared Memory Pack

Introduced in R2006a

30

Target Management, Display, and Logging Blocks

CPU Temperature

Return current CPU temperature in Celsius
Library: Target Management / Target Information

Description

This block outputs the CPU temperature. You can monitor this value and halt real-time execution when the temperature reaches a value that depends on the temperature range of the target computer.

Ports

Output

`Port_1` — CPU temperature
scalar

Outputs the CPU temperature in Celsius with granularity of 1 °C.

Data Types: `double`

Parameters

`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.

See Also

Introduced in R2017a

Current Available Stack Size

Current Available Stack Size returns free stack available

Library

Simulink Real-Time Library for Execution Parameters

Description

This block outputs the number of bytes of stack memory currently available to the real-time application thread.

See Also

Minimum Available Stack Size

Introduced in R2014a

Elapsed Time

Read target computer time
Library: Target Management / Target Information

Description

The Elapsed Time block outputs in an internal format the elapsed time since the last restart of the target computer.

To compute the difference in nanoseconds between two vector time values, pass both time values to the Time Stamp Delta block. To convert a single time value to nanoseconds, pass one time value to a Time Stamp Delta block and ground the other input.

Ports

Input

`D` — Sort block order
scalar

Dynamically typed, for use in establishing the block execution order. The block does not use the port value.

Data Types: `double`

Output

`T` — Target computer time
[00000000 FFFFFFFF]

The time value is in an internal format. To convert it to nanoseconds, use the Time Stamp Delta block.

Data Types: `[uint32 uint32]`

Parameters

`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.

See Also

Time Stamp Delta

Introduced in R2017b

Enable Profiler

Start and stop profiler on target computer
Library: Displays and Logging

Description

A rising edge on `Start` starts the profiler. A rising edge on `Stop` stops the profiler. A rising edge on both ports does nothing.

Ports

Input

`Start` — Starts the profiler

`0 | 1`

When the `Start` input changes from `0` to `1`, the block starts the profiler.

The profiler starts collecting data after the resources required to collect the data become available in the background. 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 `1`GB. The profiler stops by itself when it reaches this limit.

Data Types: `Boolean`

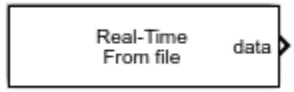
See Also

`Profiler Data` | `SimulinkRealTime.target.getProfilerData` | `SimulinkRealTime.target.resetProfiler` | `SimulinkRealTime.target.startProfiler` | `SimulinkRealTime.target.stopProfiler`

Introduced in R2017b

From File

Read data from file on target computer



Library

Simulink Real-Time Library for Target Management

Description

The From File block reads data from a file on the target computer hard disk and outputs that data in chunks every sample time. As the Simulink Real-Time kernel on the target computer reads the file data, it writes that data into a software buffer whose size is user-defined. The From File block then reads the data from this buffer and propagates it to the block outputs for use by the real-time application. For example, use the From File block to drive a model with externally acquired data (data from a file).

The From File block distributes the data as a sequence of bytes. To use these data bytes as input to a model, convert the data into one or more signals. To do so, use the Byte Unpacking block. This block outputs data in various Simulink data types. For example, assume that data in your file represents a single precision scalar and a double precision vector of width 3. To convert data of this type, set up the block to output every sample time:

```
28 bytes (1 * sizeof('single') + 3 * sizeof('double'))
```

This data can then be converted into signal values by the Byte Unpacking block.

See the following topics:

- “File Format” on page 30-10 — Describes the target computer source file format
- “Block Parameters” on page 30-11 — Describes the block parameters for the From File block

File Format

Before you use a target computer file as the source for the From File block, format the data in the file. The file format is a concatenation of the different data elements for one time step, followed by the next time step, and so on.

For example, assume that your file contains the data from the preceding example. Assign a variable to each component, for example,

- `a` — single precision value
- `b` — double precision vector of 3

Assume, also, that there are `N` time steps worth of data. The array dimension for `a` and `b` are then

- `size(a)` — `[1, N]`
- `size(b)` — `[3, N]`

In sequence, write out the data like the following to create the file.

```
a(1, 1)    4 bytes
b(:, 1)   24 bytes
a(1, 2)    4 bytes
b(:, 2)   24 bytes
...
...
a(1, N)    4 bytes
b(:, N)   24 bytes
```

If you already have the data as MATLAB variables, use the `SimulinkRealTime.utils.bytes2file` function to create the file on the development computer. This function has the following syntax:

```
SimulinkRealTime.utils.bytes2file(filename, var1, ... varn)
```

where

- `filename` — Specify the name of the data file from which the From File block distributes data
- `var1, ... varn` — Specify the column of data to be output to the model.

You can then use `SimulinkRealTime.copyFileToTarget` to download the file to the target computer.

Block Parameters

Filename

Enter the name of the target computer file that contains the data.

Output port width

Enter the size, in bytes, of the data to be distributed each sample time.

Buffer size

Enter the size of the software FIFO, in bytes. The Simulink Real-Time kernel fills this FIFO with the data to be input to the model. The From File block empties this FIFO as it inputs the data to the model.

This parameter should ideally be

- Much larger than **Output port width**
- At least several time the disk read size

  Increasing this parameter value helps prevent the real-time application from emptying the buffer faster than the background task can fill it. This can happen if you have multitasking models or conditionally executed subsystems, which can cause temporary increases in task execution time and leave less time for the background task to fill the buffer.

Disk read size

Enter the number of bytes to read to fill the buffer.

To understand this parameter, assume the following default values:

- **Buffer size** is 2000
- **Disk read size** is 512
- **Output port width** is 8

This means that the data buffer is of size 2000.

This buffer is initially full. Each time the block executes, eight bytes are output to the model, and the number of bytes in the buffer decreases by eight. Each time the

number of free bytes in the buffer goes to 512 or higher, the Simulink Real-Time kernel attempts to read 512 bytes from the Simulink Real-Time data file to fill the buffer.

Setting this parameter to another value, for example 1024, causes the From File block to wait until 1024 bytes are free before attempting the next read.

For efficiency, set this value to a multiple of 512 (a disk sector is 512 bytes).

When reaching EOF

Select the behavior of the block for when you run the real-time application beyond when you have data in the file. Select

- `Hold last output` — Stops reading and stops the output at the last value
- `Seek to beginning` — Returns to the beginning of the file and starts reading the data (this option results in periodic data)

Sample time

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

Show IsValid port

Select the **Show IsValid port** check box to make the port IsValid visible in the model. Port IsValid outputs `1` if the file read succeeds and `0` if it fails.

Introduced before R2006a

From Target

Read data from target computer

Library

Simulink Real-Time Library for Displays and Logging

Description

This block behaves like a source. Its output is connected to the input of a display device.

You can use the function `SimulinkRealTime.utils.createInstrumentationModel` to create an instrumentation model using the From Target and To Target blocks.

The From Target block runs as a non-real-time Simulink block on the development computer. It is asynchronous to the real-time application running on the target computer. If the real-time application has a sample time slower than the non-real-time model, the From Target block can query the target computer more than once. The target computer returns the same value in this case. Conversely, it is possible for the From Target block to miss some sample times between two successively returned values.

Note The use of From Target blocks requires a connection between the development and target computers. Without a connection, operations such as opening a model or copying these blocks take longer than normal.

Some notes on the From Target block behavior:

- To highlight a signal line that a From Target block refers to, double-click the From Target block.

- If the From Target block has not yet been configured, double-clicking the From Target block does nothing.
- To edit the From Target block parameters, right-click the block and select **Mask Parameters**.

Block Parameters

Target application name

The function `SimulinkRealTime.utils.createInstrumentationModel` automatically enters a name entry for this parameter. It is the same name as the Simulink model that the Simulink Real-Time software uses to build the real-time application.

Signal name (block name)

The function `SimulinkRealTime.utils.createInstrumentationModel` automatically enters a name entry for this parameter. For multiple blocks, the function creates a From Target block for each block. Using this method of specifying signals returns signal values one per time step.

You can also manually enter a cell array of signals for this parameter. Using this method of specifying signals returns the values of a vector of signals (up to 1000) as fast as it can acquire them. The signal values may not be at the same time step and the signal values are more likely to be spaced closely together.

Observer sample time

The function `SimulinkRealTime.utils.createInstrumentationModel` automatically enters the sample time for the Simulink block with this signal. It can be equal to the model base sample time or a multiple of the base sample time.

Use default target PC

Selecting this option directs Simulink Coder to build and download the real-time application to the default target computer. This assumes that you configured a default target computer through the Simulink Real-Time Explorer (see “PCI Bus Ethernet Setup” or “USB-to-Ethernet Setup” if you have not). By default, this check box is selected.

Specify target name

If you deselect the **Use default target PC** check box, this field is displayed. Enter the name of the configured target computer.

See Also

`SimulinkRealTime.utils.createInstrumentationModel`

Topics

“Creating a Custom Graphical Interface”

Introduced in R2014a

Get Overload Counter

Get Overload Counter returns number of CPU overloads

Library

Simulink Real-Time Library for Execution Parameters

Description

This block returns the CPU overload count. To display the value, connect the block output to a real-time Scope block. To achieve your required refresh rate, adjust the Simulink Real-Time Scope block parameter **Number of samples** to a small number, such as `10`.

For multirate models in multitasking mode, Get Overload Counter returns the number of overloads of the base rate task.

Block Parameters

Sample time

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

See Also

Set Overload Counter

“CPU Overload Options”

Introduced in R2014a

Minimum Available Stack Size

Get the smallest amount of free stack available

Library

Simulink Real-Time Library for Execution Parameters

Description

This block outputs the number of bytes that have not been used in the stack since the thread was created.

Note The block traverses the entire stack to find unused bytes. Use this block only for diagnostic purposes.

See Also

Current Available Stack Size

Introduced in R2014a

Scope

Real-time Scope block
Library: Displays and Logging

Description

The real-time Scope block acquires data in chunks of size **Number of samples** from the real-time application that is executing on the target computer.

You can configure real-time scope blocks for three types: `Target`, `Host`, and `File`. The target scope displays data on the target computer screen. The host scope transmits data to the development computer for processing and display. The file scope writes data to a file on the target computer.

The block dialog box changes depending on the setting for parameter **Scope type**. By default, the block dialog box displays the parameters for `Target` scopes.

In some situations, an output signal of a block is not observable by the Scope block. You can make the signal observable by adding test points, by adding unit Gain blocks, or by turning off the **Signal storage reuse** or **Block reduction** configuration parameters. For more information, see “Troubleshoot Signals Not Accessible by Name”.

The real-time application can generate data faster than the kernel can process it. Previous data can be overwritten, causing gaps. If gaps occur in the data, consider increasing the value of the **Decimation** property of the scope.

Ports

Input

`Signal` — Input signal that scope displays

numeric

Time-varying numeric value, which can be of any type that Simulink Real-Time supports.

`Trigger signal` — Trigger signal to scope

numeric

Time-varying numeric value, which can be of any type that Simulink Real-Time supports.

Dependency

This input becomes visible when you set **Trigger mode** to `Signal triggering` and set the **Add signal port to connect a signal trigger source** parameter.

Parameters

- “Common and Host Scope Parameters” on page 30-20
- “Target Scope Parameters” on page 30-24
- “File Scope Parameters” on page 30-26

Common and Host Scope Parameters

Host scopes require only the common scope parameters.

`Scope number` — Unique number identifying scope

1 (default) | numerical

Contains a unique number to identify the scope that is displayed. This number is incremented each time you add a Simulink Real-Time Scope block.

This number identifies the Simulink Real-Time Scope block and the scope display on the development or target computer.

`Scope type` — Location of scope output
`Target` (default) | `Host` | `File`

- `Target` — Output appears on the target computer screen
- `Host` — Output goes to the development computer. Usually, you display it with a host scope display in Simulink Real-Time Explorer.
- `File` — Output goes to a file on the target computer. You can download the file to the development computer for display or postprocessing.

`Start scope when application starts` — Starts scope with real-time application
`'on'` (default) | `'off'`

Select this check box to start a scope when you download and start the real-time application. After it starts, the scope waits for a trigger. With a target scope, the scope window opens automatically. With a host scope, you can open a host scope viewer window from Simulink Real-Time Explorer.

`Number of samples` — Number of values per data package
250 (default) | integer

Enter the number of values to be acquired in a data package. The minimum number is `3` samples.

`Number of pre/post samples` — Number of samples to save or skip
`0` (default) | integer

Specify a value less than `0` to save this number of samples before a trigger event. Specify a value greater than `0` to skip this number of samples after the trigger event before data acquisition begins.

`Decimation` — Sample time interval at which to collect data
`1` (default) | unsigned integer

Enter a value to collect data at each sample time (`1`) or to collect data at less than every sample time (`2` or greater).

`Trigger mode` — Define trigger event
`FreeRun` (default) | `Software triggering` | `Signal triggering` | `Scope triggering`

When a real-time scope is triggered, it acquires up to **Number of samples** of data from the real-time application that is executing on the target computer.

- `FreeRun` — The scope acquires data continuously without waiting for a trigger.
- `Software triggering` — The scope triggers in response to a user action, such as clicking the Trigger button () in Simulink Real-Time Explorer.
- `Signal triggering` — The scope triggers in response to a signal level crossing.
  - In the **Trigger signal** box, enter the index of a signal previously added to the scope.

    (Alternatively) Click the **Add signal port to connect a signal trigger source** check box, then connect an arbitrary trigger signal to the port Trigger signal. If the **Add signal port to connect a signal trigger source** check box is selected, parameter **Trigger signal** does not apply.
  - In the **Trigger level** box, enter a value for the signal to cross before triggering.
  - From the **Trigger slope** list, select one of `Either`, `Rising`, or `Falling`.
- `Scope triggering` — The scope triggers in response to the triggering of another scope.
  - In the **Trigger scope number** box, enter the scope number of a Scope block. If you use this trigger mode, you must also add a second Scope block to your Simulink model.
  - If you want the scope to trigger on a specific sample of the other scope, enter a value in the text box **Sample to trigger on (-1 for end of acquisition)**. The default value of `0` indicates that the triggering scope starts at the same time as the triggered (current) scope.

Dependency

- `Signal triggering` — The scope block adds the **Trigger signal**, **Add signal port to connect signal trigger source**, **Trigger level**, and **Trigger slope** parameters.
- `Scope triggering` — The scope block adds the **Trigger scope number** and **Sample to trigger on (-1 for end of acquisition)** parameters.

`Trigger signal` — Index of signal on which to trigger scope

1 (default) | integer

Enter the signal index. To find the index number for a signal, in the Command Window, type:

```
tg.ShowSignals = 'on'
```

.

Dependency

This parameter does not apply if the **Add signal port to connect a signal trigger source** check box is selected.

This parameter becomes visible when you set **Trigger mode** to **Signal triggering**.

`Add signal port to connect a signal trigger source` — Adds trigger signal port

`'off'` (default) | `'on'`

Adds a port to the block to which you can connect a trigger signal. If you do not select this parameter, the `Signal` port is the trigger port.

Dependency

This parameter becomes visible when you set **Trigger mode** to `Signal triggering`.

When you select the **Add signal port to connect a signal trigger source** parameter, output `Trigger signal` becomes visible.

`Trigger level` — Value that triggers scope

`0.0` (default) | numerical

The scope triggers when the value on the trigger signal passes through value **Trigger level** in the direction given by **Trigger slope**.

Dependency

This parameter becomes visible when you set **Trigger mode** to `Signal triggering`.

`Trigger slope` — Direction of value change that triggers scope

`Either` (default) | `Rising` | `Falling`

The scope triggers when the value on the trigger signal passes through value **Trigger level** in the direction given by **Trigger slope**.

Dependency

This parameter becomes visible when you set **Trigger mode** to `Signal triggering`.

`Trigger scope number` — ID number of scope on which to trigger
integer

Enter the scope ID. To find the ID number for a scope, double-click the scope block or, in the Command Window, type:

```
tg.Scopes
```

Dependency

This parameter becomes visible when you set **Trigger mode** to `Scope triggering`.

`Sample to trigger on (-1 for end of acquisition)` — Offset into scope acquisition at which to trigger
`0` (default) | integer

Number of samples into the trigger scope acquisition on which to trigger this scope. If the value is `-1`, trigger at the end of acquisition.

Dependency

This parameter becomes visible when you set **Trigger mode** to `Scope triggering`.

Target Scope Parameters

Target scopes require the common scope parameters and also the following parameters.

`Scope mode` — Display mode for target scope
`Graphical redraw` (default) | `Numerical` | `Graphical rolling` | `Graphical sliding`

- `Numerical` — Displays the data numerically. The scope acquires **Number of samples** values before updating the output.
- `Graphical redraw` — Displays a cycle of data continuously without scrolling (refreshing the entire plot). The scope acquires **Number of samples** values before redrawing the graph.
- `Graphical rolling` — Displays running data continuously scrolling from left to right across the scope (similar behavior to oscilloscopes).
- `Graphical sliding` — The legacy value `'sliding'` will be removed in a future release. It behaves like value `rolling`.

Dependency

If the scope mode is `Numerical`, the scope block adds a **Numerical format** text box to the dialog box, set by default to `%15.6f`.

`Numerical format` — Define the display format for the data

`'%15.6f'` (default) | `'[LabelN] [%width.precisiontype] [LabelX]'`

Use this box to define the display format for the data.

- `LabelN` (optional) — Signal label. You can use a different label for each signal or the same label for each signal.
- `width` (optional) — Minimum number of characters to offset from the left of the screen or label.
- `precision` (optional) — Maximum number of decimal points for the signal value. For a whole integer signal value, enter `0` for the precision value.
- `type` — Data type for the signal format, one of:

Type	Description
`%e` or `%E`	Exponential format using `e` or `E`
`%f`	Floating point
`%g`	Signed value printed in `f` or `e` format depending on which is smaller
`%G`	Signed value printed in `f` or `E` format depending on which is smaller

- `LabelX` (optional) — Second label for the signal. You can use a different label for each signal or the same label for each signal.

You can have multiple **Numerical format** entries, separated by a comma. You can enter as many format entries as you have signals for the scope. The entries apply to the signals in order. If the format contains fewer label entries than signals, the default format (`'%15.6f'`) applies to the remaining signals. If the format contains more entries than signals, the unmatched entries are ignored.

Delimit each entry with a comma and surround the entire character vector with a pair of quotes:

```
'Start1 %15.6f end1,Start2 %15.6f end2'
```

The default format is `'%15.6f'`, a floating point format without a label.

`Grid` — Displays grid lines on the scope
`'on'` (default) | `'off'`

Select this check box to display grid lines on the scope. This parameter is only applicable for target scopes and scope modes of type `Graphical redraw` and `Graphical rolling`.

`Y-Axis limits` — Define upper and lower limits of Y-axis
`[0,0]` (default) | `[numeric, numeric]`

Enter a row vector with two elements where the first element is the lower limit of the *y*-axis and the second element is the upper limit. If you enter `0` for both elements, then the scaling is set to auto. This parameter only applies to target scopes that were set to the scope modes `Graphical redraw` or `Graphical rolling`.

File Scope Parameters

File scopes require the common scope parameters and also the following parameters.

`Filename` — Name of file on target computer
`C:\data.dat` (default) | text

Enter a name for the file to contain the signal data. By default, the target computer writes the signal data to a file named `C:\data.dat`.

If you select the **Dynamic file name enabled** and **AutoRestart** check boxes, configure **Filename** to increment dynamically. Use a base file name, an underscore (_), and a `< >` specifier. Within the specifier, enter one to eight `%` symbols. Each symbol `%` represents a decimal location in the file name. The specifier can appear anywhere in the file name. For example, the following value for **Filename**, `C:\work\file_<%%%>.dat` creates file names with the following pattern:

```
file_001.dat
file_002.dat
file_003.dat
```

The last file name of this series is `file_999.dat`. If the function is still logging data when the last file name reaches its maximum size, the function overwrites the first file name in the series.

A fully qualified file name can have a maximum of 260 characters: The file part can have at most 12 characters: eight for the file name, one for the period, and at most three for the file extension. A file name longer than eight characters is truncated to six characters followed by `'~1'`.

`Mode` — File access table update policy
`Lazy` (default) | `Commit`

Both `Lazy` and `Commit` mode open a file, write signal data to the file, then close that file at the end of the session. The difference is in when the block updates the file access table (FAT) entry for the file.

- `Lazy` — The block updates the FAT entry only when the file is closed and not during each file write operation. This mode is faster than `Commit` mode. However, if the system crashes before the file is closed, the file system does not know the actual file size (the file contents, however, are intact).
- `Commit` — The block updates the FAT entry for the file with each file write operation. This mode is slower than `Lazy` mode, but the file system maintains the actual file size.

You cannot read a file that was written during real-time execution until execution has completed.

`WriteSize` — Size, in bytes, of data chunks that the block writes
`512` (default) | unsigned integer

This parameter specifies that a memory buffer, of length **Number of samples**, writes data to the file in **WriteSize** chunks. By default, this parameter is `512` bytes, which is the typical disk sector size. Using a block size that is the same as the disk sector size provides better performance.

`AutoRestart` — Restart capture after acquisition
`'off'` (default) | `'on'`

The **AutoRestart** setting works with the **Number of samples** parameter.

- Autorestart is on — When the scope triggers, the scope starts collecting data into a memory buffer. A background task examines the buffer and writes data to the disk continuously, appending new data to the end of the file. When the scope reaches the number of samples that you specified, it starts collecting data again, overwriting the memory buffer. If the background task cannot keep pace with data collection, data can be lost.

- Autorestart is off — When the scope triggers, the scope starts collecting data into a memory buffer. It stops when it has collected the number of samples that you specified. A background task examines the buffer and writes data to the disk continuously, appending the new data to the end of the file.

Dependency

Selecting this parameter makes visible the **Dynamic file name enabled** and **Max file size in bytes (multiple of WriteSize)** parameters.

`Dynamic file name enabled` — Dynamically create multiple log files

`'off'` (default) | `'on'`

Select this check box to enable the ability dynamically to create multiple log files for file scopes.

To enable this parameter, select the **AutoRestart** check box. When you enable **Dynamic file name enabled**, configure **Filename** to create incrementally numbered file names for the multiple log files. Failure to do so causes an error when you try to start the scope.

You can enable the creation of up to 99999999 files (`<%%%%%%%%>.dat`). The length of a file name, including the specifier, cannot exceed eight characters.

Dependency

This parameter becomes visible when you select the **AutoRestart** parameter.

`Max file size in bytes (multiple of WriteSize)` — Maximum size of output file

`536870912` (default) | integer

When the log file reaches **Max file size in bytes (multiple of WriteSize)** in size, the software creates the next numbered file name in the series. It continues logging data to that file, up until the highest log file number you have specified. If the software cannot create additional log files, it overwrites the first log file.

Dependency

This parameter becomes visible when you select the **AutoRestart** parameter.

See Also

Topics

“Signal Tracing Basics”
“Simulink Real-Time Scope Usage”
“Target Scope Usage”
“Host Scope Usage”
“File Scope Usage”
“Troubleshoot Signals Not Accessible by Name”
“File System Basics”
“Configure Real-Time Target Scope Blocks”
“Configure Real-Time Host Scope Blocks”
“Configure Real-Time File Scope Blocks”

Introduced in R2014a

Set Overload Counter

Set current CPU overload count

Library

Simulink Real-Time Library for Execution Parameters

Description

This block enables you to adjust the CPU overload count.

Block Parameters

Sample time

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

See Also

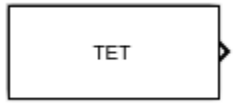
Get Overload Counter

“CPU Overload Options”

Introduced in R2014a

Task Execution Time

Task execution time (TET), in seconds



Library

Simulink Real-Time Library for Execution Parameters

Description

This block outputs the task execution time (TET) in seconds.

To visualize the TET while your real-time application is running, connect the output of this block to a Simulink Real-Time Scope block.

Task execution time (TET) measures how long it takes the kernel to run for one base-rate time step. For a multirate model, use the profiler to find out what the execution time is for each rate.

Block Parameters

Sample time

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

Introduced in R2014a

Time Stamp Delta

Time stamp delta
Library: Target Management / Target Information

Description

This block takes as input two `uint32` vectors of length 2. Two separate Elapsed Time blocks can supply each vector. The output is a scalar double that contains the difference between the two timestamps, in nanoseconds.

To compute the difference in nanoseconds between two vector time values, pass both time values to the Time Stamp Delta block. To convert a single time value to nanoseconds, pass one time value to a Time Stamp Delta block and ground the other input.

Ports

Input

`t1` — First target computer time
[00000000 FFFFFFFF]

The time value is in an internal format.

Data Types: `[uint32 uint32]`

`t2` — Second target computer time
[00000000 FFFFFFFF]

The time value is in an internal format.

Data Types: `[uint32 uint32]`

Output Arguments

`Delta` — Difference between time values
double

Difference in nanoseconds between `t1` and `t2`.

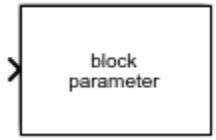
See Also

Elapsed Time

Introduced in R2006a

To Target

Send data to target computer



Library

Simulink Real-Time Library for Displays and Logging

Description

This block behaves as a sink. The main purpose of this block is to write a new value to a specific parameter on the real-time application.

You can use the function `SimulinkRealTime.utils.createInstrumentationModel` to create an instrumentation model using the To Target and From Target blocks.

The To Target block runs as a non-real-time Simulink block on the development computer. It is asynchronous to the real-time model running on the target computer. If the To Target block receives continuously changing input, two parameter updates can be sent to the target computer before the next sample time of the real-time application. In this case, the real-time application only uses the last parameter value received. Conversely, it is also possible for one or more sample times to elapse on the target computer before the To Target block sends the next parameter value.

In either case, the To Target block only sends parameter values to the target computer when there is a change (for example, the input of the To Target block changes).

Note The use of To Target blocks requires a connection between the development and target computers. Without a connection, operations such as opening a model or copying these blocks take longer than normal.

Some notes on the To Target block behavior:

- To highlight the Simulink model block referenced by a To Target block, double-click the block.
- If the To Target block has not yet been configured, double-clicking the block does nothing.
- To edit the To Target block parameters, right-click the block and select **Mask Parameters**.

Block Parameters

Target application name

The function `SimulinkRealTime.utils.createInstrumentationModel` automatically enters a name entry for this parameter. It is the same name as the Simulink model that Simulink Real-Time uses to build the real-time application.

Path to block in target application

The function `SimulinkRealTime.utils.createInstrumentationModel` automatically enters an entry for this parameter and uses it to access the block identifier.

Parameter name

The function `SimulinkRealTime.utils.createInstrumentationModel` automatically determines the entry for this parameter and enters it. Note that the parameter name might not match the label name for that parameter in the Block Parameters dialog box. For example, the label name for a gain block is `Constant value`, but the parameter name is **Value**.

Use default target PC

Selecting this option directs Simulink Coder to build and download the real-time application to the default target computer. This assumes that you configured a default target computer through the Simulink Real-Time Explorer (see “PCI Bus Ethernet Setup” or “USB-to-Ethernet Setup” if you have not). By default, this check box is selected.

Specify target name

If you deselect the **Use default target PC** check box, this field is displayed. Enter the name of the configured target computer.

See Also

`SimulinkRealTime.utils.createInstrumentationModel`

Topics

“Creating a Custom Graphical Interface”

Introduced in R2014a