Simulink® Real-Time™

User's Guide

MATLAB®&SIMULINK®

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

Revision History

September 1999	First printing	New for Version 1 (Release 11.1)
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)
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)

Contents

Model Architectures

Third-Party Calibration Support

2

Incorporating Fortran S-Functions

3

Real-Time Application Setup

Real-Time Application Environment

4

Signals and Parameters

5

Execution Modes

6

Real-Time Application Execution

Execution Using Graphical User Interface Models

7

Execution Using the Target Computer Command Line

8

Execution Using MATLAB Scripts

11

Real-Time Applications and Scopes in the MATLAB Interface

Logging Signal Data with File System Objects

12

Deploy the MATLAB Application as a Standalone Executable

13

Automated Test by Using Simulink Test

Troubleshooting

Getting Started with Troubleshooting

Confidence Test Failures

Development Computer Configuration

17

Target Computer Configuration

18

Ethernet and Serial Links

19

Target Computer Start Process

20

Modeling

21

Model Compilation

22

Real-Time Application Performance

27

Simulink Real-Time Support

28

Model Architectures

FPGA Models

Speedgoat FPGA Support

Simulink Real-Time and HDL Coder™ software enable you to implement Simulink algorithms and configure I/O functionality on Speedgoat field programmable gate array (FPGA) boards.

For a description of the development workflow for FPGA I/O boards, see "FPGA Programming and Configuration" on page 1-6. You do not use these blocks outside of HDL Coder HDL Workflow Advisor.

The Speedgoat FPGA I/O board block descriptions are for informational purposes only. The HDL Coder HDL Workflow Advisor uses these blocks to generate a Simulink Real-Time interface subsystem. The subsystem mask controls the block parameters. Do not edit the parameters directly.

Speedgoat I/O FPGA boards are sold as part of Speedgoat target computer systems. See `www.mathworks.com/products/simulink-real-time/supported/hardware-drivers.html`.

Simulink Real-Time supports the following Speedgoat (`www.speedgoat.ch`) FPGA I/O boards.

`Speedgoat IO301`	The `Speedgoat IO301` is a field-programmable gate array (FPGA) board that provides 64 bidirectional TTL I/O lines. This board is based on a Xilinx® Virtex-II chip with 6912 logic cells.
`Speedgoat IO302`	The `Speedgoat IO302` is a field-programmable gate array (FPGA) board that provides 32 bidirectional RS-422 I/O lines. This board is based on a Xilinx Virtex-II chip with 6912 logic cells.
`Speedgoat IO303`	The `Speedgoat IO303` is a field-programmable gate array (FPGA) board that provides 16 bidirectional TTL and 24 bidirectional RS-422 I/O lines. This board is based on a Xilinx Virtex-II chip with 6912 logic cells.

Speedgoat IO311   The `Speedgoat IO311` is a field-programmable gate array (FPGA) board that provides 64 bidirectional TTL I/O lines. This board is based on a Xilinx Virtex-II chip with 24192 logic cells.

Speedgoat IO312   The `Speedgoat IO312` is a field-programmable gate array (FPGA) board that provides 32 bidirectional RS-422 I/O lines. This board is based on a Xilinx Virtex-II chip with 24192 logic cells.

Speedgoat IO313   The `Speedgoat IO313` is a field-programmable gate array (FPGA) board that provides 16 TTL and 24 RS-422 I/O lines. This board is based on a Xilinx Virtex-II chip with 24192 logic cells.

Speedgoat IO314   The `Speedgoat IO314` is a field-programmable gate array (FPGA) board that provides 32 bidirectional LVDS I/O lines. This board is based on a Xilinx Virtex-II chip with 24192 logic cells.

Speedgoat IO321   The `Speedgoat IO321` is a field-programmable gate array (FPGA) board that provides 64 bidirectional LVCMOS or 32 bidirectional LVDS (four are input only) I/O lines and two 16-bit 105 MHz analog input channels. This board is based on a Xilinx Virtex-4 chip with 41472 logic cells.

The `Speedgoat IO321` is the base board. The `Speedgoat IO321-5` is the `Speedgoat IO321` plus the AXM-A30 high-speed A/D port subassembly.

`Speedgoat IO331`	The `Speedgoat IO331` is a field-programmable gate array (FPGA) board that provides 64 bidirectional LVCMOS or 32 bidirectional LVDS I/O lines. This board is based on a Xilinx Spartan® 6 chip with 147333 logic cells.

The `Speedgoat IO331` is the base board. The `Speedgoat IO331-6` is the AXM-A75 A/D converter, an add-on to the `Speedgoat IO331`. |

To work with FPGAs in the Simulink Real-Time environment, you must:

- Install HDL Coder and Xilinx ISE. For the specific ISE version required, see the Speedgoat board documentation. For more information, see "Tool Setup" in the HDL Coder documentation.
- Install the Speedgoat FPGA I/O board in the Speedgoat target machine.
- Be familiar with FPGA technology. In particular, you must know the clock frequency and the I/O connector pin and channel configuration of your FPGA board.
- Have experience using data type conversion and designing Simulink fixed-point algorithms.

To generate HDL code for your FPGA target, you do not need to have HDL programming experience.

The Simulink Real-Time product provides the following FPGA examples.

Related Examples
- "FPGA Programming and Configuration" on page 1-6
- "Tool Setup"
- "Servo Control with the Speedgoat IO301 FPGA Board"
- "Digital I/O with the Speedgoat IO303 FPGA Board"
- "PLL-Based Interrupt Generation from FPGA Input"

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

- www.speedgoat.ch

FPGA Programming and Configuration

To implement Simulink algorithms on a Speedgoat FPGA I/O board, you use "HDL Workflow Advisor" to specify an FPGA board and its I/O interface, synthesize the Simulink algorithm for FPGA programming, and generate a Simulink Real-Time interface subsystem model. The interface subsystem model contains blocks to program the FPGA and communicate with the FPGA I/O board during real-time application execution. You add the generated subsystem to your Simulink Real-Time domain model.

The workflow looks like this figure.
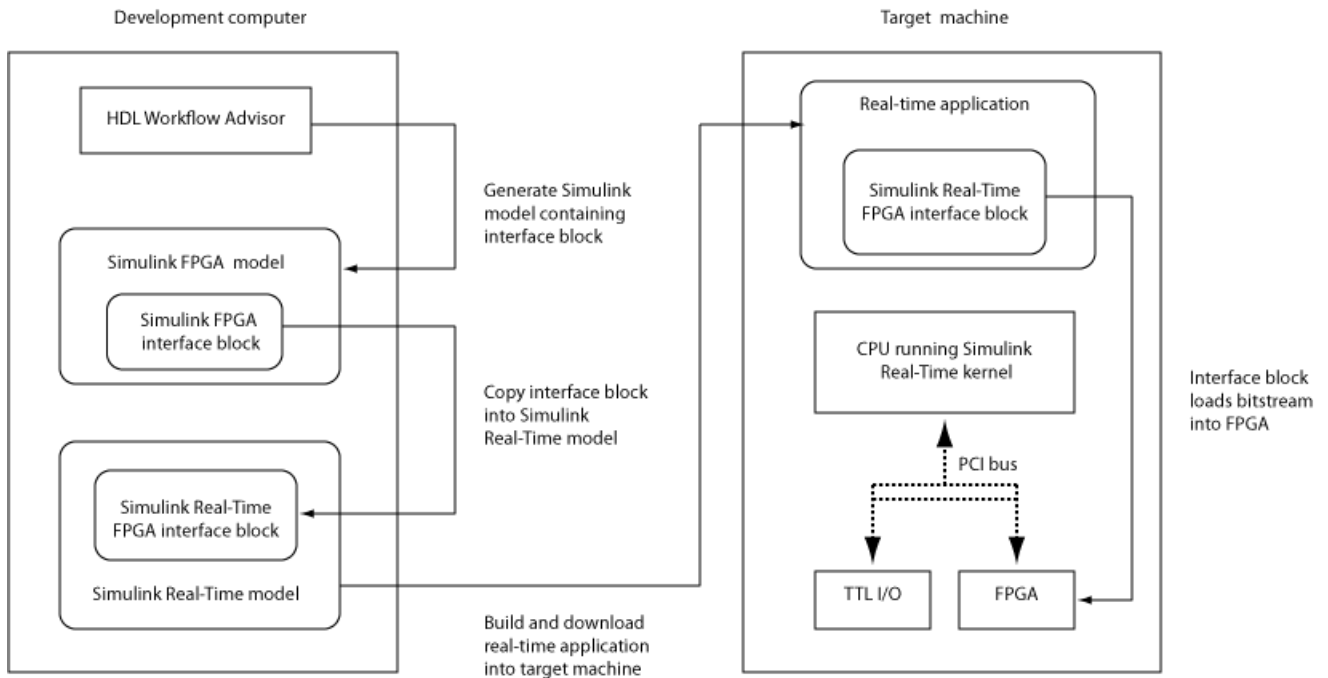
This procedure uses example "Servo Control with the Speedgoat IO301 FPGA Board". Before you begin, you must first complete an "FPGA Subsystem Plan" on page 1-22.

In this section...
"Simulink Domain Model" on page 1-7

Simulink Domain Model

The Simulink FPGA domain model contains a subsystem (algorithm) to be programmed onto the FPGA chip. Using this model, you can test your FPGA algorithm in a simulation environment before you download the algorithm to an FPGA board.

1   Create a Simulink model to contain the algorithm that you want to load onto the FPGA.

2   Place the algorithm to be programmed on the FPGA inside a `Subsystem` block. The model can include other blocks and subsystems for testing. However, one subsystem must contain the FPGA algorithm.

3   Set or confirm the subsystem inport and outport names and data types.

    The HDL Coder HDL Workflow Advisor uses these settings for routing and mapping algorithm signals to I/O connector channels. See "FPGA Subsystem Plan" on page 1-22.

4   Save the model.

This model is your FPGA domain model. It represents the simulation sample rate of the clock on your FPGA board. For example, the `Speedgoat IO301` has an onboard 33MHz clock. One second of simulation equals 33e6 iterations of the model.

For an example of an FPGA domain model, see `dxpcSGIO301servo_fpga`. The ServoSystem subsystem contains the FPGA algorithm.
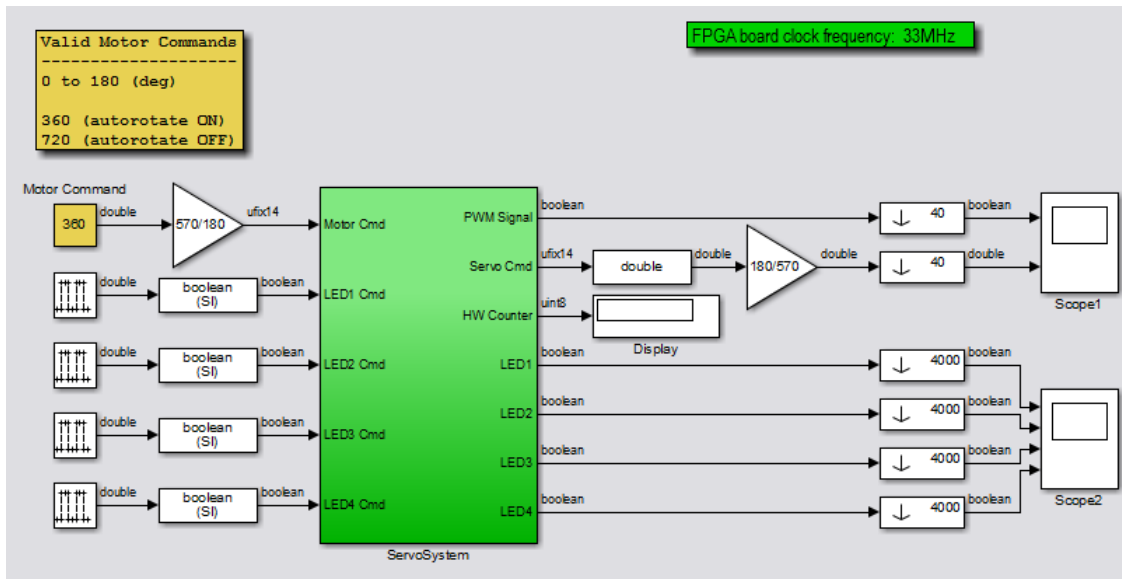
FPGA Target Configuration

This procedure uses the `dxpcSGIO301servo_fpga` example. You must have already created an FPGA subsystem (algorithm) in an FPGA domain model and developed an FPGA subsystem plan. See "FPGA Subsystem Plan" on page 1-22.

1 Open the FPGA domain model `dxpcSGIO301servo_fpga`.

2 In the FPGA model, right-click the FPGA subsystem (ServoSystem). From the context menu, select **HDL Code** > **HDL Workflow Advisor**.

   The **HDL Workflow Advisor** dialog box displays a number of tasks for the subsystem. You need to address only a subset of the tasks.

3 Expand the **Set Target** folder and select task **1.1 Set Target Device and Synthesis Tool**.

4 Set **Target Workflow** to `Simulink Real-Time FPGA I/O`.

5 From the **Target platform** list, select the Speedgoat FPGA I/O board installed in your Speedgoat target machine.

   For the `dxpcSGIO301servo_fpga` example, this is `Speedgoat IO301`.

6 From the **Synthesis tool** list, select `Xilinx ISE`.

7 Click **Run This Task**.
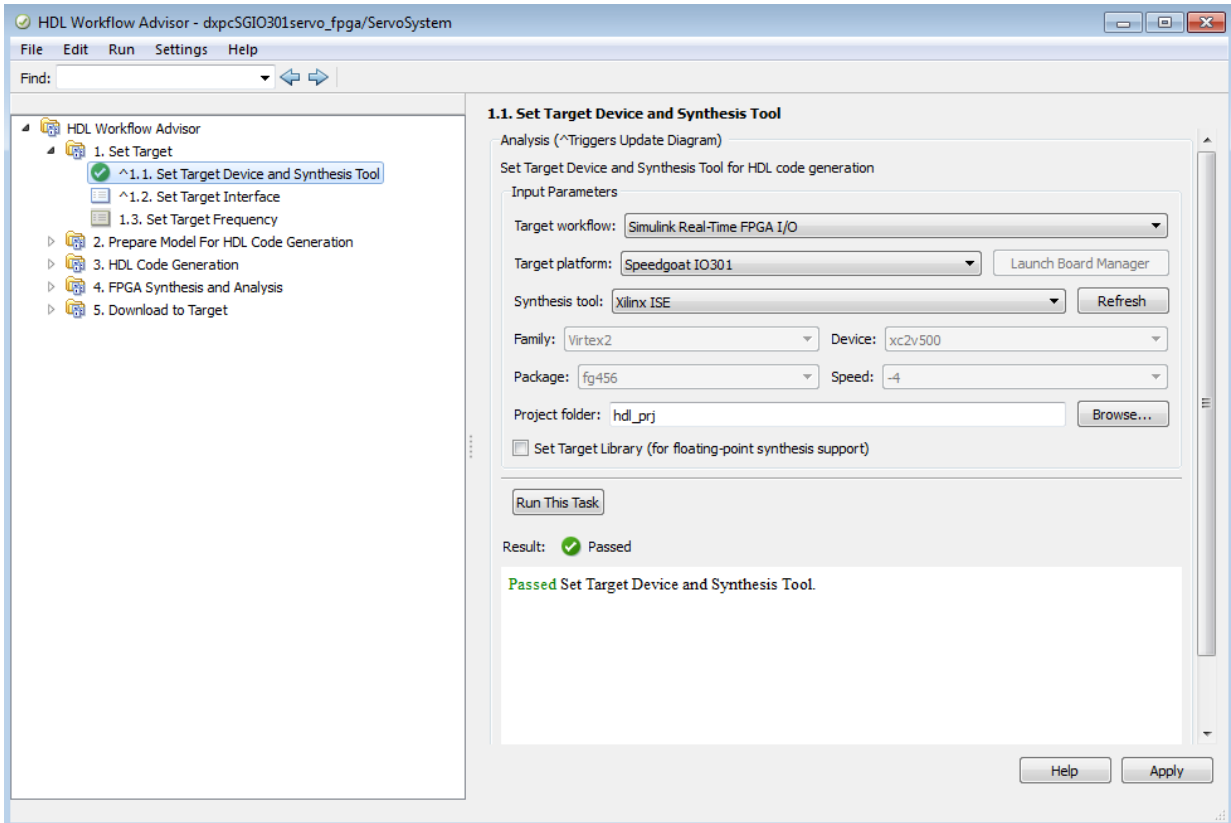
FPGA Target Interface Configuration

This procedure uses the `dxpcSGIO301servo_fpga` example. You must have already developed an FPGA subsystem plan and configured the FPGA target. See "FPGA Subsystem Plan" on page 1-22.

1 In the **Set Target** folder, select task **1.2 Set Target Interface**.

2 In the **Processor/FPGA synchronization** box, select `Free running`.

For information about FPGA synchronization modes, see "FPGA Synchronization Modes" on page 1-26.

3   For signals between the Speedgoat target machine and the FPGA (**Port Type** `Inport`) — In the **Target Platform Interfaces** column, select `PCI Interface`.

In the **Bit Range/Address/FPGA Pin** column, use the automatically generated values. Do not enter PCI address values.

4   For signals from the FPGA to I/O lines (**Port Type** `Outport`)— In the **Target Platform Interfaces** column, select the required I/O channel type (for example, `TTL I/O Channel [0:63]`).

In the **Bit Range/Address/FPGA Pin** column, enter the channel value for each signal.

5   After specifying interfaces for the required signals, click **Run This Task**.
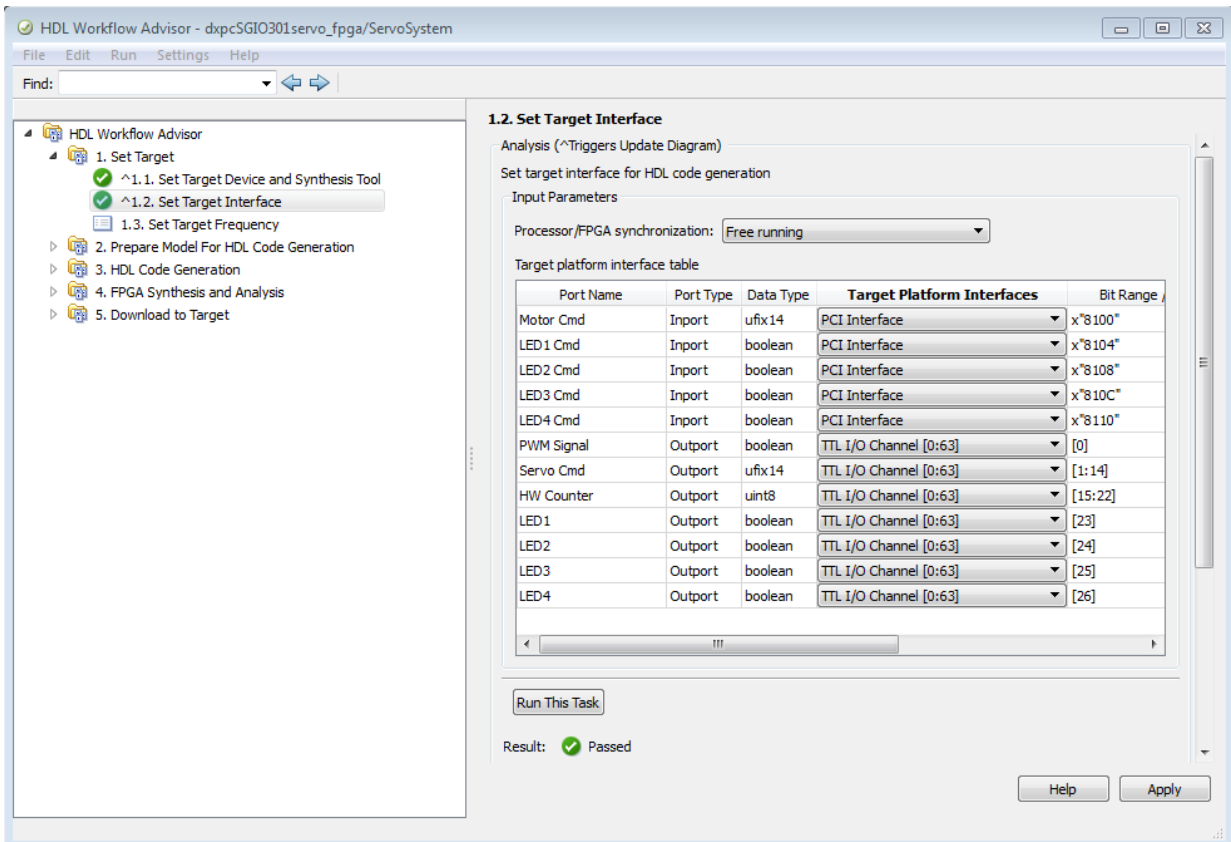
For more information about mapping Speedgoat FPGA I/O pins in HDL Coder HDL Workflow Advisor, see "Set the Target Interface for Speedgoat Boards".

FPGA Target Frequency Configuration

This optional procedure uses the `dxpcSGIO301servo_fpga` example. You must have already developed an FPGA subsystem plan and configured the FPGA target interface. See "FPGA Subsystem Plan" on page 1-22.

1 In the **Set Target** folder, select task **1.3 Set Target Frequency** (optional).

   The **Set Target Frequency** pane contains fields showing the FPGA input clock frequency (fixed) and the FPGA system clock frequency. The FPGA system clock frequency defaults to the FPGA input clock frequency.

2 To specify a different system clock frequency (for example, 50 MHz), type the new value in the field **FPGA system clock frequency (MHz)**. For the permitted range for the system clock rate, see the Speedgoat board characteristics table.

   The system may set a value different from the one you specified. For more information, see "FPGA Clock Frequency" on page 1-24.

3 Click **Run This Task**.
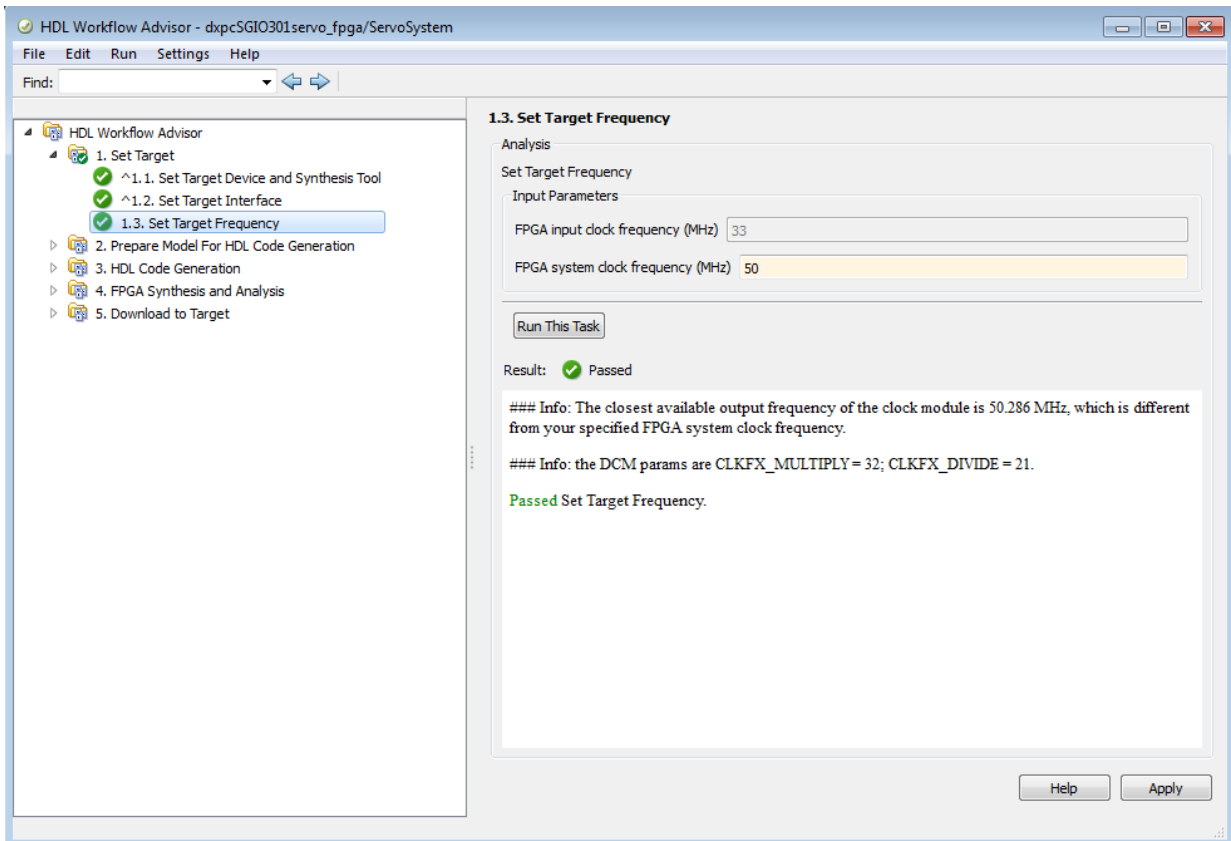
Simulink Real-Time Interface Subsystem Generation

This procedure uses the `dxpcSGIO301servo_fpga` example. You must have already configured the FPGA target interface and the required target frequency. If you have specified vector inports or outports, you must have already selected the **Scalarize vector ports** check box. This check box is on the **Coding style** tab of node **Global Settings**, under node **HDL Code Generation** in the Configuration Parameters dialog box.

1   Expand the **Download to Target** folder, and right-click task **5.2 Generate Simulink Real-Time Interface**.

2   In this pane, click **Run To Selected Task**.

This action:

- Runs the remaining tasks.

- Creates the FPGA bitstream file in the `hdlsrc` folder. The Simulink Real-Time interface subsystem references this bitstream file during the build and download process.

- Generates a model named `gm_`*`fpgamodelname`*`_slrt`, which contains the Simulink Real-Time interface subsystem.

Here is an example of the HDL Coder HDL Workflow Advisor after this action.
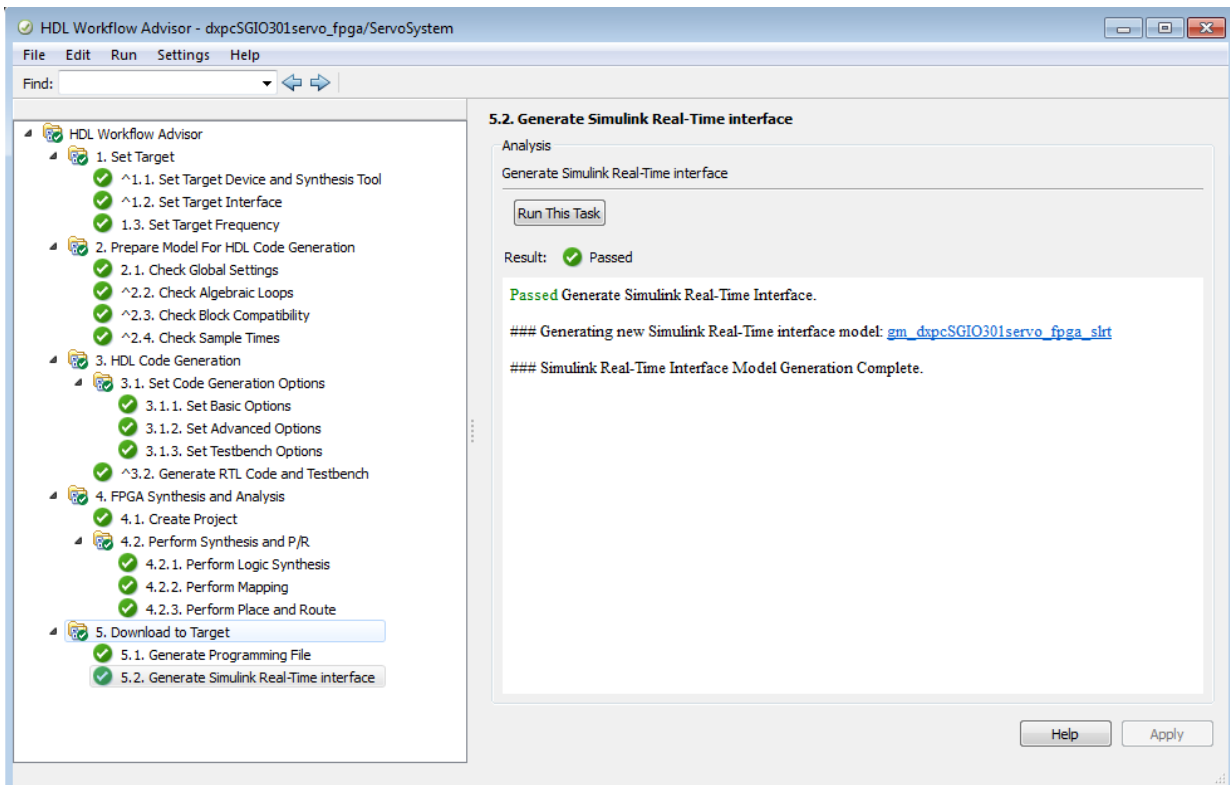
The generated interface subsystem looks like this figure.
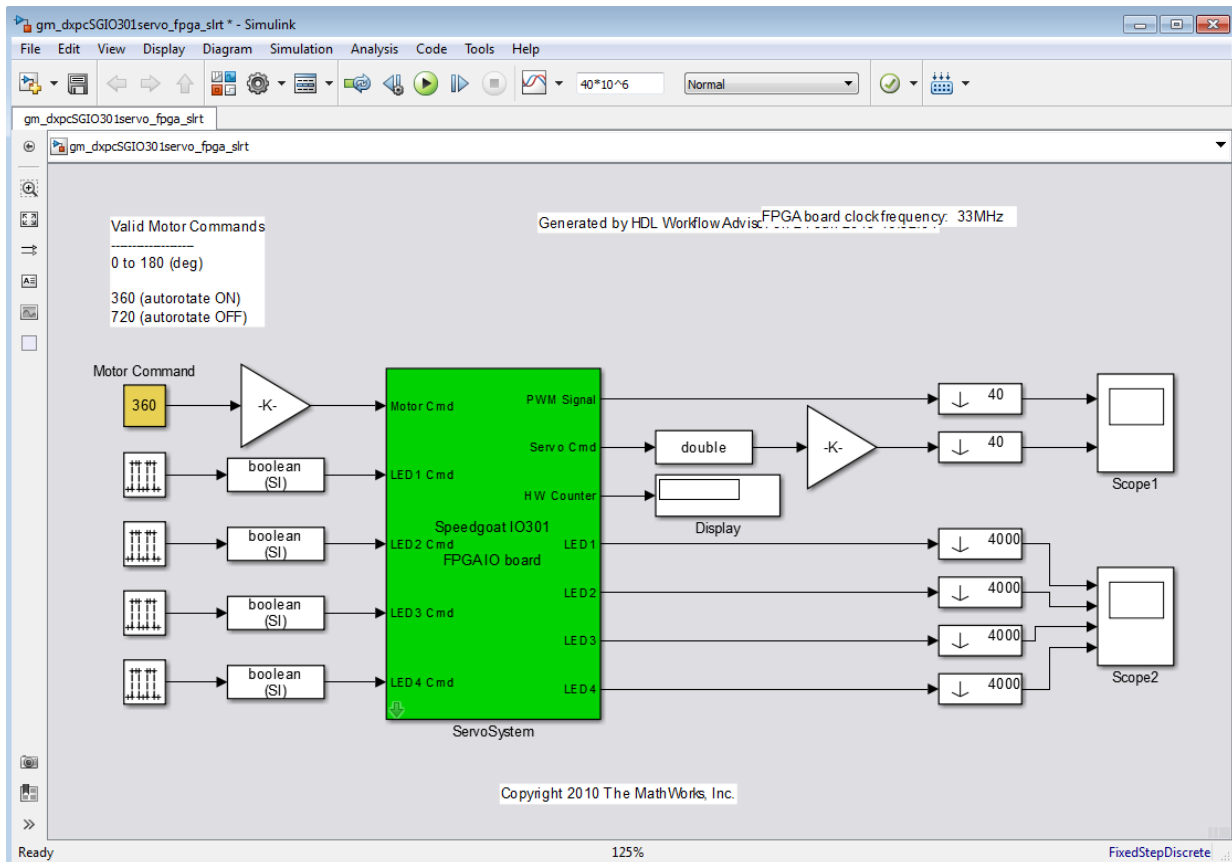
This generated model contains a masked subsystem with the same name as the subsystem in the Simulink FPGA domain model. Although the appearance is similar, this subsystem does not contain the Simulink algorithm. Instead, the algorithm is implemented in an FPGA bitstream. You reference and load this algorithm into the FPGA from this subsystem.

Simulink Real-Time Domain Model

Using the Simulink Real-Time software, you can transform a Simulink or Stateflow® domain model into a Simulink Real-Time domain model and execute it on a Speedgoat target machine for real-time testing applications. After creating a Speedgoat FPGA

domain model and the Simulink Real-Time interface subsystem using HDL Coder HDL Workflow Advisor, you can include the FPGA board in your Simulink Real-Time domain model by inserting the interface subsystem.

1   Create a Simulink Real-Time domain model with the functionality that you want to simulate in conjunction with the FPGA algorithm.

    Leave the inports and outports of the FPGA subsystem disconnected.

2   Save the model.

The Simulink Real-Time domain model looks like this figure. See example model dxpcSGIO301servo_xpc.
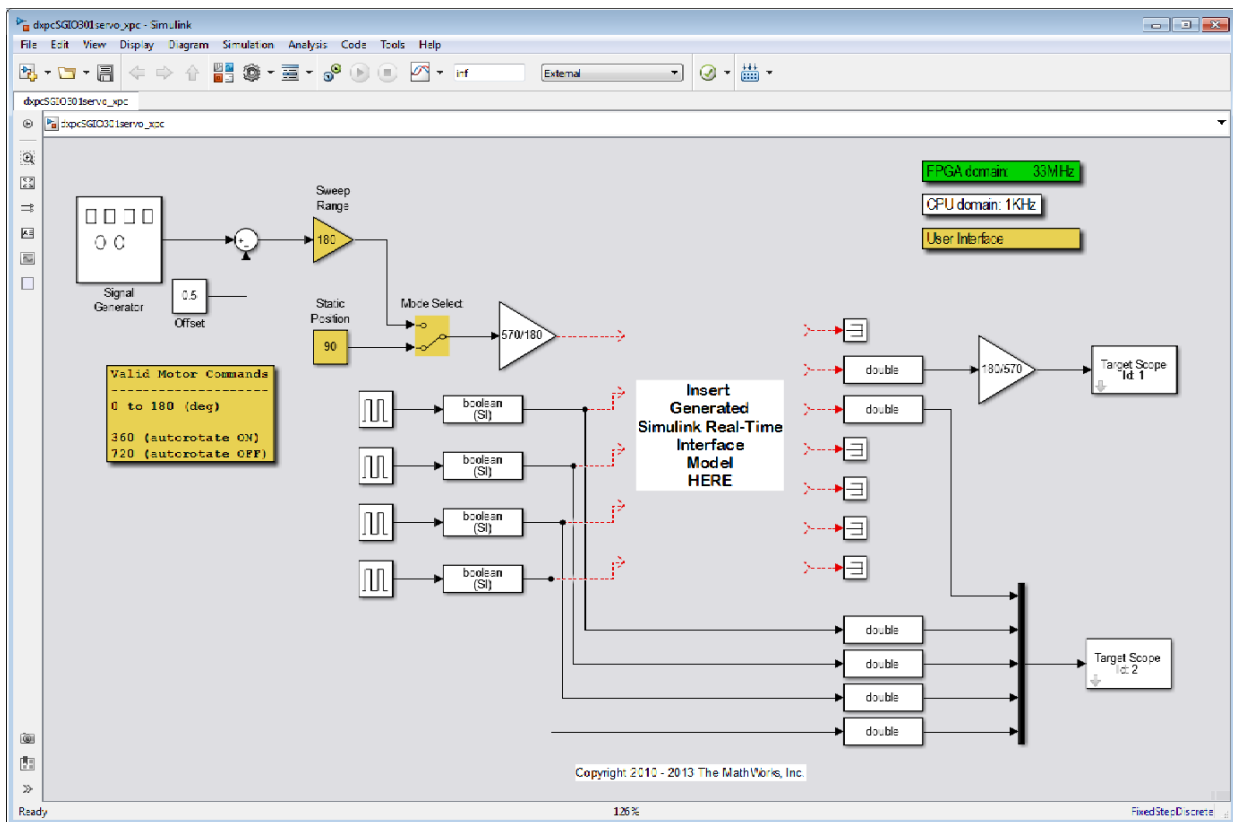
Simulink Real-Time Interface Subsystem Integration

Before doing this procedure, you must have already generated a Simulink Real-Time interface subsystem with the HDL Coder software.

You need to set three parameters in the Simulink Real-Time interface subsystem mask:

- Device index
- PCI slot
- Sample time

In addition, you must evaluate the communication timeout requirements for your model. The default communication timeout of 5 seconds may not be long enough to download and program a large FPGA, such as the `Speedgoat IO331`.

1 In the Simulink editor, open gm_*fpgamodelname*_slrt.

2 Copy and paste the this subsystem, Simulink Real-Time interface subsystem, into the Simulink Real-Time domain model.

3 Save or discard gm_*fpgamodelname*_slrt. You can recreate it as required using the HDL Coder HDL Workflow Advisor.

4 In the Simulink Real-Time domain model, connect signals to the inports and outports of the Simulink Real-Time interface subsystem.

5 Set the block parameters according to the FPGA I/O boards in your Speedgoat target machine.

   - If you have a single FPGA I/O board, leave the device index and PCI slot at the default values. You can set the sample time or leave it at −1 for inheritance.
   - If you have multiple FPGA I/O boards, give each board a unique device index.
   - If you have two or more boards of the same type (for example, two `Speedgoat IO301` boards), specify the PCI slot ([bus, slot]) for each board. Get this information with the `SimulinkRealTime.target.getPCIInfo` function.

6 If you need a larger communication timeout, in the Configuration Parameters dialog box for the model, expand the **Code Generation** and **Simulink Real-Time Options** nodes, clear the **Use default communication timeout check** box, and then enter a new value in the **Specify the communication timeout in seconds** box.

7 Save the model.

The updated Simulink Real-Time domain model looks like this figure. See example model `dxpcSGIO301servo_xpc_wiss`.
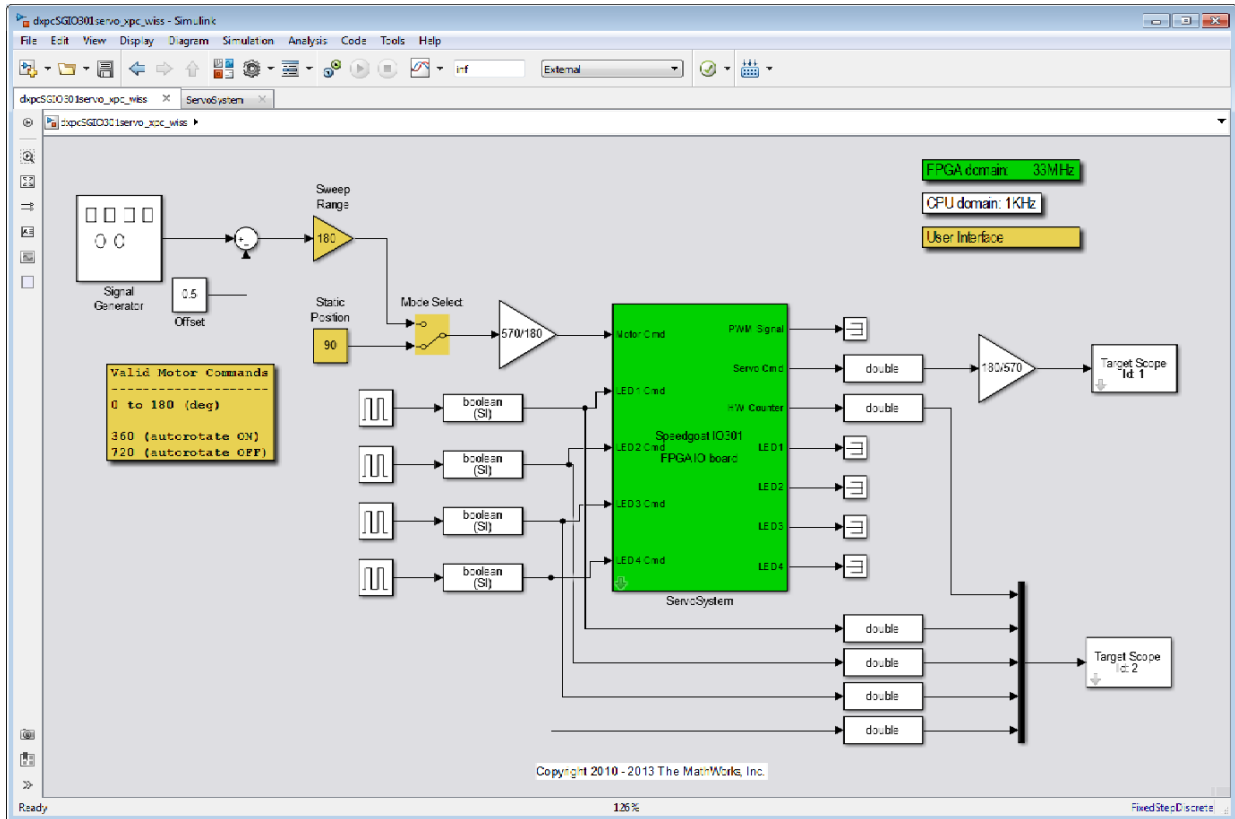
Real-Time Application Execution

To do this procedure, you must have already created a Simulink Real-Time domain model that includes a Simulink Real-Time interface subsystem generated from the HDL Coder HDL Workflow Advisor.

1   Configure the Speedgoat target machine and connect it to the development computer.

2 Build and download the Simulink Real-Time model. The Simulink Real-Time model loads onto the Speedgoat target machine and the FPGA algorithm bitstream loads onto the FPGA.

3 If you are using I/O lines (channels), confirm that you have connected the lines to your external hardware under test.

The start and stop of the Simulink Real-Time model controls the start and stop of the FPGA algorithm. The FPGA algorithm executes at the clock frequency of the FPGA I/O board, while the real-time application executes in accordance with the model sample time.

Related Examples

- "Servo Control with the Speedgoat IO301 FPGA Board"

More About

- "HDL Workflow Advisor"
- "FPGA Subsystem Plan" on page 1-22

Interrupt Configuration

Simulink Real-Time software schedules the real-time application using either the internal timer of the Speedgoat target machine (default) or an interrupt from an I/O board. You can use your Speedgoat FPGA board to generate an interrupt, which allows you to:

- Schedule execution of the real-time application based on this interrupt (synchronous execution). For this method, you must generate the interrupt periodically.
- Execute a designated subsystem in your real-time application (asynchronous execution).

To use FPGA-based interrupts, set up and configure the FPGA domain and Simulink Real-Time domain models.
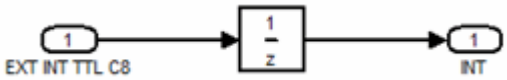
In this section...
"FPGA Domain Model" on page 1-19
"Simulink Real-Time Domain Model" on page 1-20

FPGA Domain Model

In the FPGA domain subsystem, create the interrupt source for the real-time application in one of the following ways.

Source	Description
Internal	A clock you create using Simulink blocks to create input signals. This clock is a binary pulse train of zeros and ones (transition from 0 to 1 and vice versa). The clock generates an interrupt on a rising edge. The following is an example of an internally generated interrupt source from Simulink blocks. Connect the internally generated interrupt source to an outport labelled INT.

Source	Description
External	A clock signal that comes from a device outside the Speedgoat target machine. You use a digital input pin to connect to this signal. The following is an example of an externally generated interrupt source that comes from TTL channel 8. Delay this source by one FPGA clock cycle and connect to an outport labeled INT.

In both cases, wire the interrupt source to an outport in the FPGA subsystem. Assign the outport as `Interrupt from FPGA` in the HDL Coder HDL Workflow Advisor task 1.2 **Set Target Interface**.

You are now ready to set up interrupt support in the Simulink Real-Time domain model. See "Simulink Real-Time Domain Model" on page 1-20.

Simulink Real-Time Domain Model

For overview information, see "FPGA Domain Model" on page 1-19.

Configure the model Simulink Real-Time domain model to set up interrupt support:

1 Open the Simulink Real-Time domain model.
2 In the Simulink editor, select **Simulation** > **Model Configuration Parameters**.
3 Navigate to node **Simulink Real-Time Options**, under node **Code Generation**.
4 From the **Real-time interrupt source** list, select one of the following:

   - `Auto (PCI only)`
   - The IRQ assigned to your FPGA board

5 From the **I/O board generating the interrupt** parameter, select your FPGA board, for example, `Speedgoat_IO301`.
6 Add the Simulink Real-Time interface subsystem to the model (see "Simulink Real-Time Interface Subsystem Integration" on page 1-16).
7 Build and download the real-time application to the Speedgoat target machine.
8 When you start the real-time application, simulation updates occur when the application receives an interrupt from the FPGA I/O board.

Related Examples

- "PLL-Based Interrupt Generation from FPGA Input"

FPGA Subsystem Plan

Before you work with the HDL Coder HDL Workflow Advisor, plan how to prepare the FPGA subsystem for HDL code generation and FPGA synthesis.

In this section...
"Target Device" on page 1-22
"FPGA Synchronization Mode" on page 1-22
"FPGA Inports and Outports" on page 1-22
"FPGA Clock Frequency" on page 1-24
"FPGA Deployment" on page 1-24

Target Device

You must decide which FPGA to target for code generation. The example procedure uses the Simulink Real-Time FPGA workflow and the `Speedgoat IO301` FPGA IO board for target platform. These choices require that you use the Xilinx ISE synthesis tool.

For information about other target devices, see "Supported Third-Party Tools and Hardware".

FPGA Synchronization Mode

To select the processor/FPGA synchronization mode, you must decide which of the FPGA synchronization modes to use:

- `Free running`
- `Coprocessing — blocking`
- `Coprocessing — nonblocking with delay`.

For more information, see "FPGA Synchronization Modes" on page 1-26.

FPGA Inports and Outports

Inports and outports may transmit signal data between the Speedgoat target machine and the FPGA over the PCI bus or map to I/O channels for communicating with external

devices. For connector pin and I/O channel assignments of your supported FPGA I/O board, see the board reference page for your board.

In addition to the **Port Name** and **Port Type** (`Inport` or `Outport`), to specify the I/O interface, see:

- **Data Type**—Encodes such attributes as width and sign. Data types must map consistently to their corresponding I/O pins. An inport of type `Boolean` requires 1 bit, one of type `uint32` requires 32 bits, and so on. For example, you cannot connect an inport of type `uint32` to an FPGA I/O interface of type `TTL I/O channel [0:7]`; it requires `TTL I/O channel [0:31]`.

- **Target Platform Interfaces**—Encodes the I/O channels on the FPGA as well as their functional type. For a single-ended interface (`TTL`, `LVCMOS`), one channel maps to one connector pin. For a differential interface (`RS422`, `LVDS`), one channel maps to two connector pins. To discover the mapping for a particular pin, see the pin connector map provided with the board description.

  I/O channels may also map to a predefined specification or role (`PCI Interface`, `Interrupt from FPGA`).

  For information on using FPGA interrupts, see "Interrupt Configuration" on page 1-19.

- **Bit Range/Address/FPGA Pin**—Encodes the pins on the target platform to which the inports and outports are assigned, along with the channel number used by the port. For specification `PCI Interface`, **Bit Range/Address/FPGA Pin** encodes the PCI address used by the port.

If vector inports or outports are required, specify a vector port:

- `Inport` — Add a mux outside the subsystem that connects to a demux inside the subsystem.
- `Outport` – Add a mux inside the subsystem that connects to a demux outside the subsystem.
- `Inport` and `Outport` – Configure the port dimension to be greater than 1.

Workflow Advisor automatically inserts a strobe to achieve a simultaneous update of vector elements.

If you have specified vector inports or outports, before generating code, you must select the **Scalarize vector ports** check box. This check box is on the **Coding style** tab

of node **Global Settings**, under node **HDL Code Generation** in the Configuration Parameters dialog box.

FPGA Clock Frequency

The FPGA system clock frequency defaults to the fixed FPGA input clock frequency. The fixed FPGA input clock frequency is shown in the **FPGA input clock frequency (MHz)** box. You can specify another frequency in this box. If the FPGA clock circuits cannot generate the specified value exactly, HDL Coder HDL Workflow Advisor generates the closest match. The closest match, F_{system}, is based on the following formula:

$$F_{system} = F_{input} * ClkFxMultiply / ClkFxDivide$$

F_{input} is the fixed FPGA input clock frequency. `ClkFxMultiply` and `ClkFxDivide` are integers.

FPGA Deployment

When HDL Coder HDL Workflow Advisor generates the programmed FPGA subsystem, it writes an SLX file (`gm_`*`mdlname`*`.slx`) and a C file (*`blkorrefmdlname`*`_topio`*`speedgoat#`*`.c`) into the model folder. The SLX file contains the FPGA subsystem. The C file contains the bitstream.

For example, assume that model `fpga_model.slx` contains a `Subsystem` block named `fpga_subsystem`, and that you configure the FPGA target platform for the model as `Speedgoat IO301`. Then HDL Coder HDL Workflow Advisor generates the following files:

```
gm_fpga_model.slx
fpga_subsystem_topio301.c
```

When you build your domain model with the integrated subsystem, the model compiler reads the C file and inserts its contents into the compiled output. The compiler assumes that the SLX file and the C file are in the same folder. If you deploy the model to another location on the disk, copy the SLX file and the C file to the new location.

More About

- "Supported Third-Party Tools and Hardware"

- "FPGA Synchronization Modes" on page 1-26
- "Interrupt Configuration" on page 1-19

FPGA Synchronization Modes

In Simulink Real-Time, an FPGA operates in three synchronization modes:

- `Free running`
- `Coprocessing — blocking`
- `Coprocessing — nonblocking with delay`

- `Free running` (default) — The CPU and the FPGA each run nonsynchronized, continuously, and in parallel. When you want the CPU to run continuously without interrupts, select this mode. For example, you could select this mode when the model is processing continuous PWM output.

  The Speedgoat target machine CPU strobes data out of the FPGA, reads the results from the FPGA outputs, writes data to the FPGA inputs, and strobes the data into the FPGA.
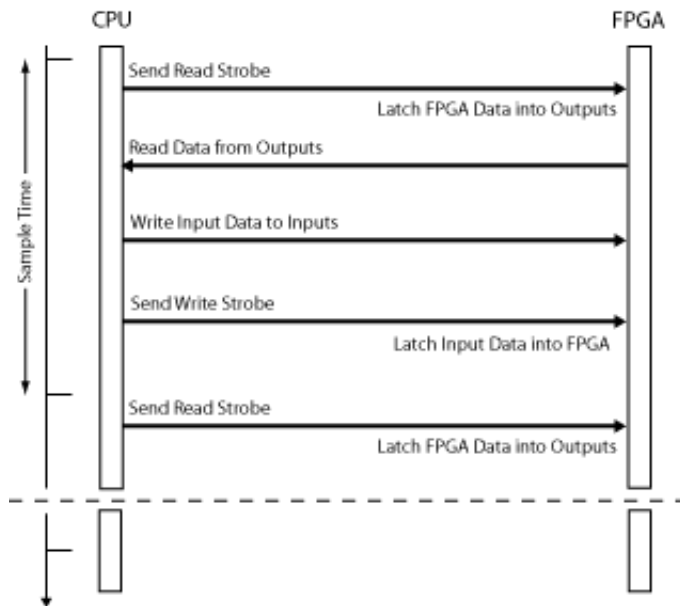
- `Coprocessing — blocking` — The CPU and the FPGA run synchronized and in tandem. When the FPGA execution time is short compared to the Speedgoat target machine sample time, and you want the FPGA to complete before the model continues, select this coprocessor mode.

The CPU writes data to the FPGA inputs, strobes the data into the FPGA, waits for the FPGA to finish executing, and reads the result out of the FPGA outputs.
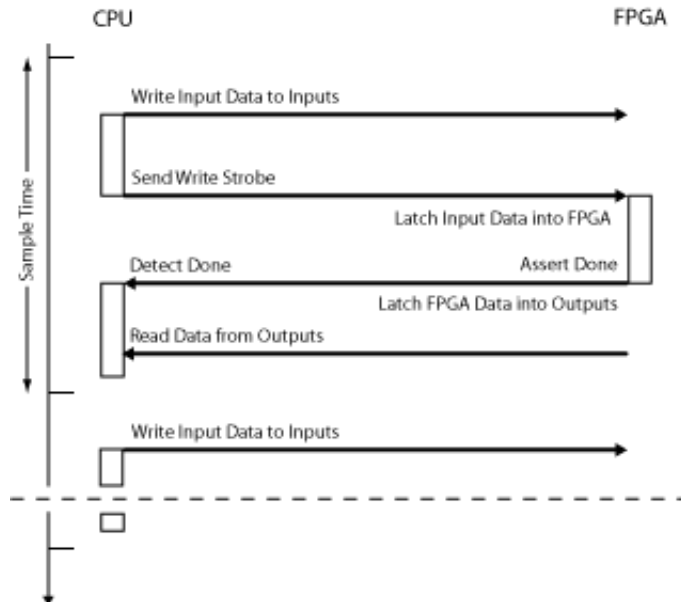
- `Coprocessing — nonblocking with delay` — The CPU and the FPGA run synchronized and in tandem. When the FPGA execution time is long compared to the Speedgoat target machine sample time, select this coprocessor mode. For example, you could select this mode to manage multiple FPGAs effectively in parallel.

  The CPU waits for the FPGA to finish executing, reads the data from the previous time step, writes new data to the FPGA inputs, and strobes the data into the FPGA.
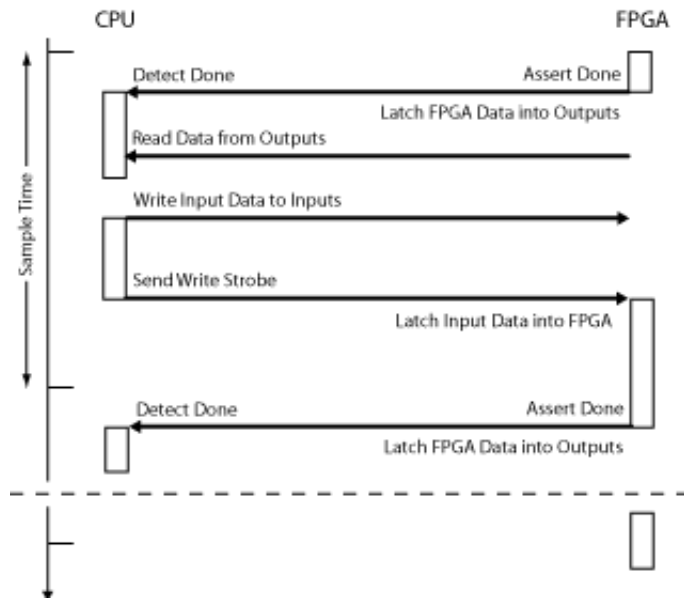
Third-Party Calibration Support

- "Calibrate Real-Time Application" on page 2-2
- "Prepare ASAP2 Data Description File" on page 2-4
- "Calibrate Parameters by Using Vector CANape" on page 2-10
- "Calibrate Parameters by Using ETAS Inca" on page 2-13

Calibrate Real-Time Application

Simulink Real-Time supports interaction with third-party calibration tools such as Vector CANape (`www.vector.com`) and ETAS Inca (`www.etas.com`). Use these tools for:

- Parameter display and tuning
- Calibration data saving, restoring, and swapping by page
- Signal value streaming

These tools run in XCP master mode. Simulink Real-Time emulates an electronic control unit (ECU) operating in XCP slave mode. To enable a real-time application to work with the third-party software:

- Configure the third-party software to communicate with the real-time application as an ECU.
- Generate a real-time application that provides data that is compliant with the software.
- Provide a standard TCP/IP physical layer between the development and target computers.

  Simulink Real-Time supports third-party calibration software only through TCP/IP.
- Generate A2L (ASAP2) files that the software can load into its database. The generated file contains signal and parameter access information for the real-time application.

Related Examples

- "Prepare ASAP2 Data Description File" on page 2-4
- "Calibrate Parameters by Using Vector CANape" on page 2-10
- "Calibrate Parameters by Using ETAS Inca" on page 2-13

More About

- "XCP Master Mode"

External Websites

- www.vector.com

- www.etas.com

Prepare ASAP2 Data Description File

This example shows how to configure a Simulink Real-Time model so that the build generates an ASAP2 (A2L) data description file for the real-time application. The real-time application models a damped oscillator that feeds into 1-D and 2-D lookup tables, which invert and rescale the input waveform.
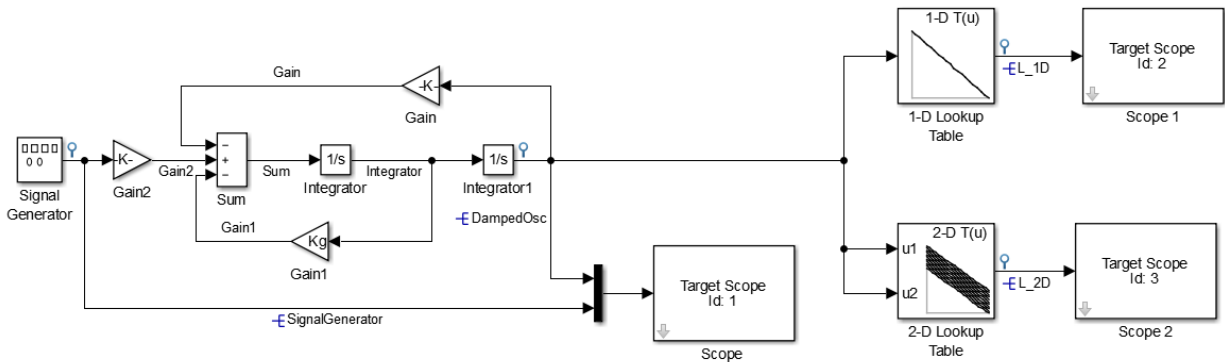
Model xpcosc
Simulink Real-Time example model

Copyright 1999-2016 The MathWorks, Inc.

The goal of calibration is reducing the ringing in signals DampedOsc, L_1D, and L_2D.

In this section...

In this section...
"Generate Data Description File" on page 2-8

Initial Setup

1 To load MATLAB® workspace variables for the example model, click ex_slrt_cal_osc_data.mat.

The MATLAB workspace variables have the following functions:

- Kg — Parameter object for the Gain1 block
- DampedOsc, SignalGenerator, L_1D, L_2D — Signal objects for output signals
- ydata, zdata — 1-D and 2-D lookup tables respectively
- xbreak1, xbreak2, ybreak — Indexes into lookup tables

2 To open the example model, click ex_slrt_cal_osc.

Set Up Parameters

Set up parameter tuning by using Simulink parameter objects.

1 In ex_slrt_cal_osc, on the toolbar, click the **Model Explorer** button ▦.

2 Select **Base Workspace** in the **Model Hierarchy** pane.

3 Verify that the Kg parameter object exists and has these properties:

- **Value** — 400
- **Data type** — double
- **Storage class** — ExportedGlobal

4 If the parameter object does not exist, add it. On the toolbar, click the **Add Simulink Parameter** button ▦.

5 Open ex_slrt_cal_osc/Gain1.

6 Verify that you have set the **Gain** value to the parameter object Kg.

Set Up Signals

As a best practice, set up signal viewing by using Simulink signal objects.

1 In `ex_slrt_cal_osc`, on the toolbar, click the **Model Explorer** button ▦.

2 Select **Base Workspace** in the **Model Hierarchy** pane.

3 Verify that the `DampedOsc` signal object exists and has these properties:

   - **Minimum** — `-10`
   - **Maximum** — `10`
   - **Data type** — `double`
   - **Storage class** — `ExportedGlobal`.

4 Verify that the `SignalGenerator` signal object exists and has these properties:

   - **Minimum** — `-10`
   - **Maximum** — `10`
   - **Data type** — `double`
   - **Storage class** — `ExportedGlobal`.

5 Verify that the `L_1D` signal object exists and has these properties:

   - **Minimum** — `-15`
   - **Maximum** — `15`
   - **Data type** — `double`
   - **Storage class** — `ExportedGlobal`.

6 Verify that the `L_2D` signal object exists and has these properties:

   - **Minimum** — `-15`
   - **Maximum** — `15`
   - **Data type** — `double`
   - **Storage class** — `ExportedGlobal`.

7 If a signal does not exist, add it. On the toolbar, click the **Add Simulink Signal** button ▦.

8 For each signal, open its Properties dialog box.

9 Verify that you selected the **Signal name must resolve to Simulink signal object** and the **Test point** check boxes.

Set Up Lookup Tables

The example model contains 1-D and 2-D lookup tables.

1 Open the block parameters for the `1-D Lookup Table` block.

2 In the **Table and Breakpoints** pane, verify the following settings:

   - **Number of table dimensions** — 1
   - **Table data** — `ydata`
   - **Breakpoints specification** — `Explicit values`
   - **Breakpoints 1** — `xbreak1`

3 Open the block parameters for the `2-D Lookup Table` block.

4 In the **Table and Breakpoints** pane, verify the following settings:

   - **Number of table dimensions** — 2
   - **Table data** — `zdata`
   - **Breakpoints specification** — `Explicit values`
   - **Breakpoints 1** — `xbreak2`
   - **Breakpoints 2** — `ybreak`

To view the contents of the lookup tables, click **Edit table and breakpoints**, and then click **Plot** > **Mesh**.

Generate Data Description File

1 Open **Simulation** > **Model Configuration Parameters**.

2 In the left pane, click the **Simulink Real-Time Options** node.

3 In the **Miscellaneous options** area, select the **Generate INCA/CANape extensions** check box.

This option enables real-time applications to generate a ASAP2 (A2L) data description file. You can then use third-party calibration software.

4 Build the model.

The build produces a file named `ex_slrt_cal_osc_slrt.a2l` in the working folder.

5   When the build is complete, on the target computer monitor, look for the following
message.

```
XCP Server set up, waiting for connection
```

This message indicates that you have built the real-time application without
producing an error. You can now connect to the target with a third-party calibration
tool.

See Also

"Generate INCA/CANape extensions" | `n-D Lookup Table`

Related Examples

- "Calibrate Parameters by Using Vector CANape" on page 2-10
- "Calibrate Parameters by Using ETAS Inca" on page 2-13

External Websites

- www.vector.com
- www.etas.com

Calibrate Parameters by Using Vector CANape

This example shows how to view signals and tune parameters by using Vector CANape. You must have already completed the steps in "Prepare ASAP2 Data Description File" on page 2-4.

You also must be familiar with the Vector CANape user interface. For information about the user interface, see the vendor documentation (`www.vector.com`).

In this section...
"Prepare Project" on page 2-10
"Prepare Device" on page 2-10
"Configure Signals and Parameters" on page 2-11
"Perform Signal Measurement and Parameter Calibration" on page 2-11
"Limitations" on page 2-11

Prepare Project

1 Build and download real-time application `ex_slrt_cal_osc`.

2 Open Vector CANape.

3 Create a Vector CANape project with project name `ex_slrt_cal_osc`.

Accept the default folder.

Prepare Device

1 From `ex_slrt_cal_osc_slrt.a2l` in your build folder, create an XCP device named `ex_slrt_cal_osc_slrt`.

Do not configure dataset management.

2 Select your local PC Ethernet adapter as the Ethernet channel

3 Accept the remaining defaults.

4 Upload data from the device.

Configure Signals and Parameters

1 Open device `ex_slrt_cal_osc_slrt`, and then open `ex_slrt_cal_osc_slrt.a2l`.

2 Add signals `DampedOsc`, `SignalGenerator`, `L_1D`, and `L_2D` to separate display windows.

3 To make the waveform easier to evaluate, set the time and *y*-axis scaling.

   For example, try the following settings for `DampedOsc`:

   - *y*-axis min home value — `-25`
   - *y*-axis max home value — `25`
   - Min home time-axis value — `0s`
   - Max home time-axis value — `0.1s`
   - Time duration — `0.1s`

4 Open the measurement list.

5 To set the required sample rate for a signal, open the measurement properties for the signal. Select the required sample rate from the measurement mode list.

   The default rate is the base sample rate.

6 Add a graphic control on parameter `Kg`.

Perform Signal Measurement and Parameter Calibration

1 Start the Vector CANape measurement.

2 In Simulink Real-Time Explorer, start the real-time application.

   The signal windows show the four waveforms, corresponding to the displays on the target computer screen.

3 To shorten the ring time on `DampedOsc`, `L_1D`, and `L_2D`, set parameter `Kg` to, for example, `800`.

4 As required, toggle between calibration RAM active and inactive.

Limitations

For Vector CANape, the Simulink Real-Time software does not support:

- Starting and stopping the real-time application by using Vector CANape commands.

  To start and stop the real-time application on the target computer, use the Simulink Real-Time start and stop commands, for example `start(tg)`, `stop(tg)`.
- Vector CANape flash programming.
- Multiple simultaneous Vector CANape connections to a single target computer.

Event mode data acquisition has the following limitations:

- Every piece of data that the Simulink Real-Time software adds to the event list slows down the real-time application. The amount of data that you can observe depends on the model sample time and the speed of the target computer. It is possible to overload the target computer CPU to where data integrity is reduced.
- You can trace only signals and scalar parameters. You cannot trace vector parameters.

Related Examples

- "Prepare ASAP2 Data Description File" on page 2-4

External Websites

- www.vector.com

Calibrate Parameters by Using ETAS Inca

This example shows how to view signals and tune parameters by using ETAS Inca. You must have already completed the steps in "Prepare ASAP2 Data Description File" on page 2-4.

You also must be familiar with the ETAS Inca user interface. For information about the user interface, see the vendor documentation (`www.etas.com`).

In this section...
"Prepare Database" on page 2-13
"Prepare Project" on page 2-13
"Prepare Workspace" on page 2-13
"Prepare Experiment" on page 2-14
"Configure Signals and Parameters" on page 2-14
"Perform Signal Measurement and Parameter Calibration" on page 2-14
"Limitations" on page 2-15

Prepare Database

1 Build and download real-time application `ex_slrt_cal_osc`.
2 Open ETAS Inca.
3 Add an ETAS Inca database with folder named `SLRTDatabase`.
4 Add subfolders named `Experiment`, `Project`, and `Workspace`.

Prepare Project

1 Under folder `Project`, add an ECU project.
2 When prompted, select A2L file `ex_slrt_cal_osc_slrt.a2l` in your build folder. Ignore the prompt for a HEX file.

   If you change and rebuild the real-time application, delete the ECU project and recreate it with the new A2L file.

Prepare Workspace

1 Under folder `Workspace`, add workspace `ex_slrt_cal_osc_wksp`.

2   Add project `ex_slrt_cal_osc_slrt` to workspace `ex_slrt_cal_osc_wksp`.

3   When prompted, add an Ethernet system XCP device to the workspace.

4   Configure the XCP device and initialize it. Auto configure the ETAS network.

5   To upload data from the device hardware, use enhanced operations on memory pages.

     Data is uploaded from the real-time application on the target computer.

Prepare Experiment

1   Under folder `Experiment`, add experiment `ex_slrt_cal_osc_exp`.

2   Add experiment `ex_slrt_cal_osc_exp` to workspace `ex_slrt_cal_osc_wksp`.

Configure Signals and Parameters

1   Start experiment `ex_slrt_cal_osc_exp`.

2   To create graphic controls for the variables, add variables `Kg`, `DampedOsc`, `SignalGenerator`, `L_1D`, `L_2D`, and `zdata`.

3   Add YT oscilloscopes for `DampedOsc`, `SignalGenerator`, `L_1D`, `L_2D`.

4   For each signal, set the rate to the base sample rate of the real-time application (250 µs).

Perform Signal Measurement and Parameter Calibration

1   Start the ETAS Inca measurement.

2   In Simulink Real-Time Explorer, start the real-time application.

     The signal windows show the four waveforms, corresponding to the displays on the target computer screen.

3   To shorten the ring time on `DampedOsc`, `L_1D`, and `L_2D`, set parameter `Kg` to, for example, `800`.

4   As required, toggle between reference page and working page.

5   To freeze the parameter set on the target computer, use the freeze working data command.

     To save the working data on the development computer, use the save working data command.

Limitations

For ETAS Inca, the Simulink Real-Time software does not support:

- Starting and stopping the real-time application by using ETAS Inca commands.

  To start and stop the real-time application on the target computer, use the Simulink Real-Time start and stop commands, for example `start(tg)`, `stop(tg)`.
- ETAS Inca flash programming.
- Multiple simultaneous ETAS Inca connections to a single target computer.

Event mode data acquisition has the following limitations:

- Every piece of data that the Simulink Real-Time software adds to the event list slows down the real-time application. The amount of data that you can observe depends on the model sample time and the speed of the target computer. It is possible to overload the target computer CPU to where data integrity is reduced.
- You can trace only signals and scalar parameters. You cannot trace vector parameters.

Related Examples
- "Prepare ASAP2 Data Description File" on page 2-4

External Websites
- www.etas.com

Incorporating Fortran S-Functions

Fortran S-Functions

The Simulink Real-Time product supports Fortran in Simulink models using S-functions. For more details, see "Create Level-2 Fortran S-Functions" and "Port Legacy Code".

In this section...
"Prerequisites" on page 3-2
"Simulink Demos Folder" on page 3-2
"Steps to Incorporate Fortran" on page 3-3

Prerequisites

You must have Simulink Real-Time Version 1.3 or later to use Fortran for real-time applications. The Simulink Real-Time product supports the Fortran compiler(s) listed here:

www.mathworks.com/support/compilers/current_release

Simulink Demos Folder

The Simulink `demos` folder contains a tutorial and description on how to incorporate Fortran code into a Simulink model using S-functions. To access the tutorial and description,

1   In the MATLAB Command Window, type

    demos

    A list of MATLAB products appears on the left side of the MATLAB Online Help window.

2   From the left side of the window, select **Simulink** > **Demos** > **Modeling Features**.

    A list of Simulink examples appears.

3   Click **Custom Code and Hand Coded Blocks using the S-function API**.

    The associated Simulink examples page opens.

4   Click **Open this model**.

S-function examples are displayed.

5 Double-click the `Fortran S-functions` block.

Fortran S-functions and associated templates appear.

Steps to Incorporate Fortran

This topic lists the general steps to incorporate Fortran code into a real-time application. Detailed commands follow in the accompanying examples.

1 Using the Fortran compiler, compile the Fortran code (subroutines (`*.f`)). You will need to specify particular compiler options.

2 Write a Simulink C-MEX wrapper S-function. This wrapper S-function calls one or more of the Fortran subroutines in the compiled Fortran object code from step 1.

3 Use the `mex` function to compile this C-MEX S-function using a Microsoft® Visual C++® compiler. Define several Fortran run-time libraries to be linked in.

This step creates the Simulink S-function MEX-file.

4 Run a simulation C-MEX file with the Simulink software to validate the compiled Fortran code and wrapper S-function.

5 Copy the relevant Fortran run-time libraries to the real-time application build folder.

6 Define the Fortran libraries, and the Fortran object files from step 1, in the Simulink Coder™ dialog box of the Simulink model. You must define these libraries and files as additional components to be linked in when the real-time application link takes place.

7 Initiate the Simulink Real-Time specific Simulink Coder build procedure for the example model. Simulink Coder builds and downloads Simulink Real-Time onto the target computer.

Fortran `Atmosphere` Model

This example uses the example Atmosphere model that comes with the Simulink product. The following procedures require you to know how to write Fortran code according to Simulink and Simulink Real-Time software requirements.

Before you start, create a Simulink Real-Time Simulink model for the `Atmosphere` model. See "Creating a Fortran `Atmosphere` Model" on page 3-4.

In this section...
"Creating a Fortran `Atmosphere` Model" on page 3-4
"Compiling Fortran Files" on page 3-5
"Creating a C-MEX Wrapper S-Function" on page 3-6
"Compiling and Linking the Wrapper S-Function" on page 3-10
"Validating the Fortran Code and Wrapper S-Function" on page 3-11
"Preparing the Model for the Real-Time Application Build" on page 3-12
"Building and Running the Real-Time Application" on page 3-13

Creating a Fortran `Atmosphere` Model

To create a Simulink Real-Time `Atmosphere` model in Fortran, you need to add a Simulink Real-Time Scope block to the `sfcndemo_atmos` model. Perform this procedure if you do not already have a Simulink Real-Time `Atmosphere` model for Fortran.

1   From the MATLAB window, change folder to the working folder, for example, `xpc_fortran_test`.

2   Type

     sfcndemo_atmos

     The `sfcndemo_atmos` model is displayed.

3   Add a Simulink Real-Time Scope block of type `Target`.

4   Connect this Scope block to the `Tamb, K` signal.

     The model `sfcndemo_atmos` should look like the figure shown.
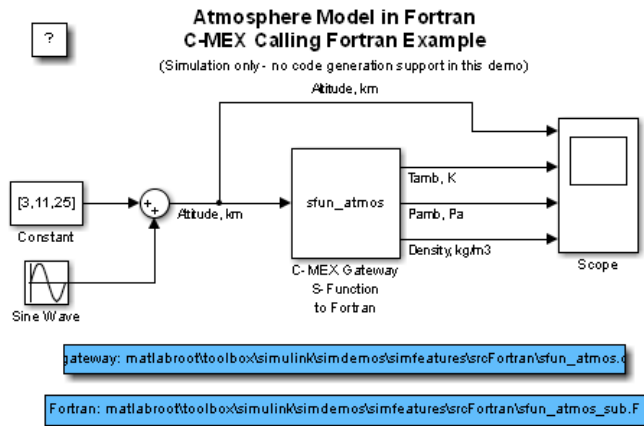
5   Double-click the target Scope block.

6   From the **Scope mode** parameter, choose Graphical rolling.

7   For the **Number of samples** parameter, enter 240.

8   Click **Apply**, then **OK**.

9   Double-click the Sine Wave block.

10 For the **Sample time** parameter, enter 0.05.

11 Click **OK**.

12 From the **File** menu, click **Save as**. Browse to your current working folder,
    for example, xpc_fortran_test. Enter a filename. For example, enter
    fortran_atmos_xpc and then click **Save**.

Your next task is to compile Fortran code. See "Compiling Fortran Files" on page
3-5.

Compiling Fortran Files

To access the files for this example, in the Command Window, type:

```
cd(fullfile(matlabroot,'toolbox','simulink','simdemos',...
        'simfeatures','srcFortran'));
```

1 In the Command Window, copy the file **sfun_atmos_sub.F** into your Fortran working folder, for example, `xpc_fortran_test`. This is sample Fortran code that implements a subroutine for the `Atmosphere` model.

2 From *Fortran_compiler_dir*`\lib\ia64`, copy the following files to the working folder:

   - `libifcore.lib`
   - `libifcoremd.lib`
   - `ifconsol.lib`
   - `libifportmd.lib`
   - `libifport.lib`
   - `libmmd.lib`
   - `libm.lib`
   - `libirc.lib`
   - `libmmt.lib`
   - `libifcoremt.lib`
   - `svml_disp.lib`

3 From a DOS prompt, change folder to the working folder and create the object file. For example:

```
ifort /fpp /Qprec /c /nologo /MT /fixed /iface:cref -Ox sfun_atmos_sub.F
```

Your next task is to create a wrapper S-function. See "Creating a C-MEX Wrapper S-Function" on page 3-6.

Creating a C-MEX Wrapper S-Function

This topic describes how to create a C-MEX wrapper S-function for the Fortran code in **sfun_atmos_sub.f**. This function is a level 2 S-function. It incorporates existing Fortran code into a Simulink S-function block and lets you execute Fortran code from the Simulink software. Before you start:

- Compile your Fortran code. See "Compiling Fortran Files" on page 3-5.

- Become familiar with the guidelines and calling conventions for Simulink Fortran level 2 S-functions (see "Create Level-2 Fortran S-Functions").

- Implement the required callback functions using standard functions to access the fields of the S-function's simulation data structure, `SimStruct` (see "Templates for C S-Functions").

The following procedure outlines the steps to create a C-MEX wrapper S-function to work with `sfun_atmos_sub.f`. It uses the template file `sfuntmpl_gate_fortran.c`.

Note: This topic describes how to create a level 2 Fortran S-function for the `fortran_atmos_xpc` model. This file is also provided in `sfun_atmos.c`.

---

1 Copy the file `sfuntmpl_gate_fortran.c` to your working folder.

This is your C-MEX file for calling into your Fortran subroutine. It works with a simple Fortran subroutine.

2 With a text editor of your choice, open `sfuntmpl_gate_fortran.c`.

3 Inspect the file. This is a self-documenting file.

This file contains placeholders for standard Fortran level 2 S-functions, such as the S-function name specification and Simulink callback methods.

4 In the `#define S_FUNCTION_NAME` definition, add the name of your S-function. For example, edit the definition line to look like

```
#define S_FUNCTION_NAME sfun_atmos
```

5 In the file, read the commented documentation for fixed-step and variable-step fixed algorithm support.

6 Delete or comment out the code for fixed-step and variable-step fixed-algorithm support. You do not need these definitions for this example.

7 Find the line that begins `extern void nameofsub_`. Specify the function prototype for the Fortran subroutine. For the `sfun_atmos_sub.obj` executable, the Fortran subroutine is `atmos_`. Replace

```
extern void nameofsub_(float *sampleArgs, float *sampleOutput);
```

with

```
extern void atmos_(float *falt, float *fsigma, float *fdelta, float *ftheta);
```

Enter a `#if defined`/`#endif` statement like the following for Windows® compilers.

```
#ifdef _WIN64
#define atmos_ atmos
#endif
```

8   Add a `typedef` to specify the parameters for the block. For example,

```
typedef enum {TO_IDX=O, PO_IDX, RO_IDX, NUM_SPARAMS } paramIndices;

#define TO(S) (ssGetSFcnParam(S, TO_IDX))
#define PO(S) (ssGetSFcnParam(S, PO_IDX))
#define RO(S) (ssGetSFcnParam(S, RO_IDX))
```

9   Use the `mdlInitializeSizes` callback to specify the number of inputs, outputs, states, parameters, and other characteristics of the S-function. S-function callback methods use SimStruct functions to store and retrieve information about an S-function. Be sure to specify the `temperature`, `pressure`, and `density` parameters. For example,

```
static void mdlInitializeSizes(SimStruct *S)
{
  ssSetNumSFcnParams(S,NUM_SPARAMS);    /* expected number */
#if defined(MATLAB_MEX_FILE)
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) goto EXIT_POINT;
#endif

    {
        int iParam = O;
        int nParam = ssGetNumSFcnParams(S);

        for ( iParam = O; iParam < nParam; iParam++ )
        {
            ssSetSFcnParamTunable( S, iParam, SS_PRM_SIM_ONLY_TUNABLE );
        }
    }

  ssSetNumContStates( S, O );
  ssSetNumDiscStates( S, O );
  ssSetNumInputPorts(S, 1);
  ssSetInputPortWidth(S, 0, 3);
  ssSetInputPortDirectFeedThrough(S, 0, 1);
  ssSetInputPortRequiredContiguous(S, 0, 1);
  ssSetNumOutputPorts(S, 3);
  ssSetOutputPortWidth(S, O, 3); /* temperature */
  ssSetOutputPortWidth(S, 1, 3); /* pressure     */
  ssSetOutputPortWidth(S, 2, 3); /* density      */

#if defined(MATLAB_MEX_FILE)
EXIT_POINT:
#endif
    return;
}
```

10 Use the `mdlInitializeSampleTimes` callback to specify the sample rates at which this S-function operates.

```
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, O, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, O, 0.0);
    ssSetModelReferenceSampleTimeDefaultInheritance(S);
}
```

11 Use the `mdlOutputs` callback to compute the signals that this block emits.

```
static void mdlOutputs(SimStruct *S, int_T tid)
{
  double *alt = (double *) ssGetInputPortSignal(S,O);
  double *T   = (double *) ssGetOutputPortRealSignal(S,0);
  double *P   = (double *) ssGetOutputPortRealSignal(S,1);
  double *rho = (double *) ssGetOutputPortRealSignal(S,2);
  int    w    = ssGetInputPortWidth(S,0);
  int    k;
  float  falt, fsigma, fdelta, ftheta;

  for (k=0; k<w; k++) {

    /* set the input value */
    falt  = (float) alt[k];

    /* call the Fortran routine using pass-by-reference */
    atmos_(&falt, &fsigma, &fdelta, &ftheta);

    /* format the outputs using the reference parameters */
    T[k]   = mxGetScalar(TO(S)) * (double) ftheta;
    P[k]   = mxGetScalar(PO(S)) * (double) fdelta;
    rho[k] = mxGetScalar(RO(S)) * (double) fsigma;
  }
}
```

12 Use the `mdlTerminate` callback to perform the actions required at termination of the simulation. Even if you do not have require such operations, you must include a stub for this callback.

```
static void mdlTerminate(SimStruct *S)
{
}
```

13 In the file, read the commented documentation for the following callbacks:

- mdlInitalizeConditions — Initializes the state vectors of this S-function.
- mdlStart — Initializes the state vectors of this S-function. This function is called once at the start of the model execution.
- mdlUpdate — Updates the states of a block.

These are optional callbacks that you can define for later projects. You do not need to specify these callbacks for this example.

14 Delete or comment out the code for these callbacks.

15 Save the file under another name. For example, save this file as sfun_atmos.c. Do not overwrite the template file.

16 Copy the file sfun_atmos.c into your Fortran working folder, for example, xpc_fortran_test.

Your next task is to compile and link the wrapper S-function. See "Compiling and Linking the Wrapper S-Function" on page 3-10.

Compiling and Linking the Wrapper S-Function

This topic describes how to create (compile and link) a C-MEX S-function from the sfun_atmos.c file. Before you start, copy the following files into the working folder, xpc_fortran_test. (You should have copied these files when you performed the steps in "Compiling Fortran Files" on page 3-5.)

- libifcore.lib
- libifcoremd.lib
- ifconsol.lib
- libifportmd.lib
- libifport.lib
- libmmd.lib
- libm.lib
- libirc.lib
- libmmt.lib
- libifcoremt.lib
- svml_disp.lib

Use the `mex` command with a C/C++ compiler such as Microsoft Visual C++ Version 6.0.

This topic assumes that you have created a C-MEX wrapper S-function. See "Creating a C-MEX Wrapper S-Function" on page 3-6.

Invoking the `mex` command requires you to compile the wrapper C file `sfun_atmos.c`. Be sure to link in the following:

- Compiled Fortran code: `sfun_atmos_sub.obj`
- Fortran run-time libraries to resolve external function references and provide the Fortran run-time environment

When you are ready, `mex` the code. For example

```
mex -v LINKFLAGS="$LINKFLAGS /NODEFAULTLIB:libcmt.lib libifcoremd.lib
ifconsol.lib libifportmd.lib libmmd.lib libirc.lib svml_disp.lib" sfun_atmos.c
sfun_atmos_sub.obj
```

Note: The command and all its parameters must be on one line.

This command compiles and links the `sfun_atmos_sub.c` file. It creates the `sfun_atmos.mex` file in the same folder.

Your next task is to validate the Fortran code and wrapper S-function. See "Validating the Fortran Code and Wrapper S-Function" on page 3-11.

Validating the Fortran Code and Wrapper S-Function

Validate the generated C-MEX S-function, `sfun_atmos.mex`. Bind the C-MEX S-function to an S-function block found in the Simulink block library. You can mask the S-function block like other S-function blocks to give it a specific dialog box.

This topic assumes that you have compiled and linked a wrapper S-function. See "Compiling and Linking the Wrapper S-Function" on page 3-10.

The `Atmosphere` model example has a Simulink model associated with it.

1 In the MATLAB window, type

   `fortran_atmos_xpc`

This opens the Simulink model associated with the Atmosphere model. This model includes an S-function block bound to sfun_atmos.mex.

2   Select **Simulation** > **Run** to simulate the model.

3   Examine the behavior of the Atmosphere model by looking at the signals traced by the Scope block.

Your next task is to prepare the model to build a real-time application. See "Preparing the Model for the Real-Time Application Build" on page 3-12.

Preparing the Model for the Real-Time Application Build

Before you build the Atmosphere model for Simulink Real-Time, define the following build dependencies:

- The build procedure has access to sfun_atmos.sub.obj for the link stage.
- The build procedure has access to the Fortran run-time libraries (see "Compiling and Linking the Wrapper S-Function" on page 3-10) for the link stage.

This topic assumes that you have validated the Fortran code and wrapper S-function (see "Validating the Fortran Code and Wrapper S-Function" on page 3-11).

1   In the MATLAB window, type

```
fortran_atmos_xpc
```

This opens the Simulink model associated with the Atmosphere model.

2   In the Simulink model, click **Simulation** > **Model Configuration Parameters**.

The Configuration Parameters dialog box appears.

3   In the left pane, click the **Code Generation** node.

The Code Generation pane opens.

4   In the **Target selection** section, click the **Browse** button at the **System target file** list.

5   Click slrt.tlc.

6   In the **Make command** field, replace make_rtw with one for the Fortran compiler.

```
make_rtw S_FUNCTIONS_LIB="..\sfun_atmos_sub.obj ..\libifcoremt.lib ..\libmmt.lib
..\ifconsol.lib ..\libifport.lib ..\libirc.lib ..\svml_disp.lib"
```

Note: The command and all its parameters must be on one line.

---

7   Click **Apply**.

8   Click **OK**.

9   From the **File** menu, click **Save**.

This command requires that the real-time application build folder be the current folder (one level below the working folder, `xpc_fortran_test`). Because of this, all additional dependency designations must start with `..\`.

Specify all Fortran object files if your model (S-Function blocks) depends on more than one file. For this example, you specify the run-time libraries only once.

Your next task is to build and run the real-time application. See "Building and Running the Real-Time Application" on page 3-13 .

Building and Running the Real-Time Application

This topic assumes that you have prepared the model to build a Simulink Real-Time application. See "Preparing the Model for the Real-Time Application Build" on page 3-12.

Build and run the real-time application as usual. Be sure that you have defined Microsoft Visual C++ as the Simulink Real-Time C compiler using `slrtsetCC`.

After the build procedure succeeds, Simulink Real-Time automatically downloads the real-time application to the target computer. The `Atmosphere` model already contains a Simulink Real-Time Scope block. This allows you to verify the behavior of the model. You will be able to compare the signals shown on the target display with the signals obtained earlier by the Simulink simulation run (see "Validating the Fortran Code and Wrapper S-Function" on page 3-11).

Real-Time Application Setup

4

Real-Time Application Environment

Simulink Real-Time Explorer Basic Operations

Simulink Real-Time Explorer is a graphical user interface for the Simulink Real-Time product. It runs on your development computer and provides a single point of contact for most interactions.

Note: Do not use Simulink external mode while Simulink Real-Time Explorer is running. Use only one interface or the other.

Through Simulink Real-Time Explorer, you can perform basic operations, such as:

- Add and configure target computers for the Simulink Real-Time software, up to 64 target computers.
- Create boot DVDs or CDs, removable drives, and network boot images for particular target computers.
- Connect the target computers for your Simulink Real-Time system to the development computer.
- Download a prebuilt real-time application, or DLM, to a target computer.
- Start and stop the real-time application that has been downloaded to the target.
- Add and remove host, target, or file scopes associated with the downloaded real-time application.
- Monitor signals.
- Add or remove signals associated with Simulink Real-Time scopes.
- Start and stop scopes.
- Adjust parameter values for the signals while the real-time application is running.
- Export a Simulink Real-Time Explorer configuration as a separate executable.

To start Simulink Real-Time Explorer:

- In the Simulink Editor, click **Tools** > **Simulink Real-Time**.
- In the Command Window, type `slrtexplr`.

There are four major panes in Simulink Real-Time Explorer:

- **Targets** pane — The top-left **Targets** pane lists the targets in your Simulink Real-Time hierarchy. Under each target are nodes representing the properties and (if accessible) the file system of the target.

- **Applications** pane — The bottom left **Applications** pane lists the real-time applications running on the targets. Under each application are nodes representing the properties, signal and parameter groupings, and (if available) the model hierarchy of the application.

- **Scopes** pane — The top right **Scopes** pane lists the scopes defined on the active real-time applications, whether predefined or dynamically created.

- **Output** pane — The bottom center **Output** pane receives status messages from Simulink Real-Time Explorer.

The top center pane displays under separate tabs information associated with nodes selected in one of the other panes.

Default Target Computers

When you start Simulink Real-Time Explorer for the first time, it opens a default node, `TargetPC1`. You can configure this node for a target computer, then connect the node to the target computer. If you later build a real-time application from a Simulink model, the Simulink Real-Time software builds and downloads that application the default target computer.

You can add other target computer nodes and designate one of them as the default target computer instead of the first one. To set a target computer node as the default, right-click that node and select **Set As Default Target** from the context-sensitive menu. The default target computer node is boldface.

If you delete a default target computer node, the target computer node preceding it becomes the default target computer node. The last target computer node becomes the default target computer node and cannot be deleted.

If you want to use the Simulink Real-Time command-line interface to work with the target computer, you must indicate which target computer the command is interacting with. If you do not identify a particular target computer, the Simulink Real-Time software uses the default target computer.

The `SimulinkRealTime` target computer environment, manages collective and individual target computer environments. See "Command-Line Setup" on page 4-6.

When you call `SimulinkRealTime.getTargetSettings` without arguments (for example, `env = SimulinkRealTime.getTargetSettings`), the constructor gets the target environment settings for the default target computer.

When you instantiate the target object constructor `slrt` without arguments (for example, `tg = slrt`), the constructor uses the link properties of the default target computer to communicate with the target computer.

Command-Line C Compiler Configuration

To configure the development computer for the C compiler using MATLAB language, use this procedure.

The command `mex -setup` sets the default compiler for Simulink Real-Time builds, provided the MEX compiler is a supported Microsoft compiler. Use `slrtsetCC -setup` only if you need to specify different compilers for MEX and Simulink Real-Time.

1 Install a supported C compiler on the development computer.

For more about the Simulink Real-Time C compiler requirements, see `www.mathworks.com/support/compilers/current_release`.

2 In the Command Window, type:

`slrtsetCC setup`

The function queries the development computer for C compilers that the Simulink Real-Time environment supports. It returns output like the following:

```
Select your compiler for Simulink Real-Time.

[1] Microsoft Visual C++ Compilers 2008 Professional Edition (SP1) in
    c:\Program Files (x86)\Microsoft Visual Studio 9.0
[2] Microsoft Visual C++ Compilers 2010 Professional in
    C:\Program Files (x86)\Microsoft Visual Studio 10.0

[0] None

Compiler:
```

3 At the `Compiler` prompt, enter the number for the compiler that you want to use. For example, `2`.

The function verifies that you have selected the required compiler:

```
Verify your selection:

Compiler: Microsoft Visual C++ Compilers 2010 Professional
Location: C:\Program Files (x86)\Microsoft Visual Studio 10.0

Are these correct [y]/n?
```

4 Type `y` or press **Enter** to verify the selection.

Command-Line Setup

Use the following procedures to configure single- and multiple-target systems.

You must have installed and configured a C compiler and verified the target computer BIOS settings. If not, see:

- "Command-Line C Compiler Configuration" on page 4-5.
- "BIOS Settings"

Note: RS-232 communication type has been removed. Configure TCP/IP communication instead.

---

1 "Command-Line PCI Bus Ethernet Setup" on page 4-7 or "Command-Line USB-to-Ethernet Setup" on page 4-10

2 "Command-Line Target Computer Settings" on page 4-21

3 "Command-Line Target Boot Methods" on page 4-24

The next task is "Run Confidence Test on Configuration".

Command-Line PCI Bus Ethernet Setup

If your target computer has a PCI bus, use an Ethernet card for the PCI bus. The PCI bus has a faster data transfer rate than the other bus types.

In this section...
"PCI Bus Ethernet Protocol Hardware" on page 4-7
"Command-Line PCI Bus Ethernet Settings" on page 4-8

PCI Bus Ethernet Protocol Hardware

Ethernet link
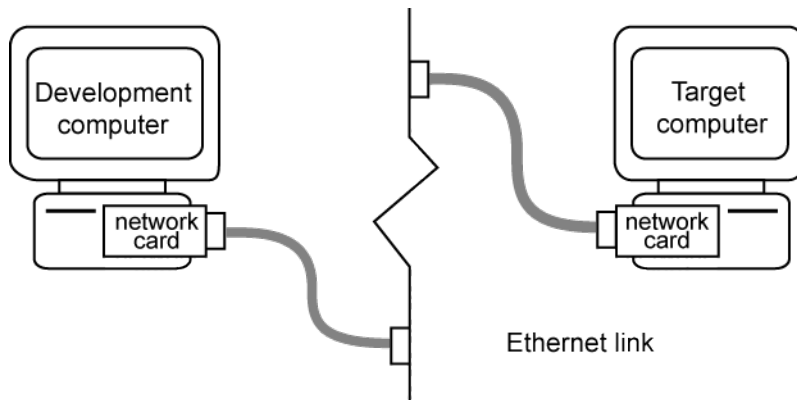
To install PCI bus Ethernet protocol interface hardware:

1 Acquire a supported PCI bus Ethernet card.

If you want to start the target computer from the network, the Ethernet adapter must be compatible with the Preboot eXecution Environment (PXE) specification.

2 Turn off your target computer.

3 If the target computer already has an unsupported Ethernet card, remove the card.

4 Plug the supported Ethernet card into a free PCI bus slot.

5 Assign a static IP address to the target computer Ethernet card.

Although the target computer Ethernet card must have a static IP address, the development computer network adapter card can have a Dynamic Host Configuration Protocol (DHCP) address and can be accessed from the network. You

must configure the DHCP server to reserve static IP addresses to prevent these addresses from being assigned to other systems.

6 Connect your target computer Ethernet card to your LAN using an unshielded twisted-pair (UTP) cable.

You can directly connect your computers using a crossover UTP cable with RJ45 connectors. Both computers must have static IP addresses. If the development computer has a second network adapter card, that card can have a DHCP address.

Command-Line PCI Bus Ethernet Settings

After you install the PCI bus Ethernet card, before you can build and download a real-time application, you must specify the environment properties for the development and target computers.

Before you start, ask your system administrator for the following information for your target computer:

- IP address
- Subnet mask address
- Port number (optional)
- Gateway (optional)

Use the following procedure for target `TargetPC1`:

1 At the MATLAB prompt, create an environment object for this target computer and make it the default target:

```
env = SimulinkRealTime.addTarget('TargetPC1');
setAsDefaultTarget(env);
```

All other settings will be made to this object.

2 Set the IP address for your target computer. For example:

```
env.TcpIpTargetAddress = '10.10.10.15';
```

3 Set the subnet mask address of your LAN. For example:

```
env.TcpIpSubNetMask = '255.255.255.0';
```

4 Set the TCP/IP port (optional) to a value higher than `'20000'` and less than `'65536'`. For example:

```
env.TcpIpTargetPort = '22222';
```

This property is set by default to `'22222'`, a value higher than the reserved area (`telnet`, `ftp`, and so on).

5 Set the TCP/IP gateway (optional) to the gateway required to access the target computer. For example:

```
env.TcpIpGateway = '255.255.255.255';
```

This property is set by default to `'255.255.255.255'`, which means that you do not use a gateway to connect to your target computer. If you connect your computers with a crossover cable, leave this property as `'255.255.255.255'`.

If you communicate with the target computer from within your LAN, you might not need to change this setting. If you communicate from a development computer located in a LAN different from your target computer (especially via the Internet), you must define a gateway and enter its IP address in this box.

6 Set the bus type to `'PCI'`.

```
env.TcpIpTargetBusType = 'PCI';
```

7 Set the target driver to one of `'3C90x'`, `'I8254x'`, `'I82559'`, `'NS83815'`, `'R8139'`, `'R8168'`, `'Rhine'`, `'RTLANCE'`, or `'Auto'` (the default).

```
env.TcpIpTargetDriver = 'Auto';
```

For target driver `'Auto'`, the software determines the target computer TCP/IP driver from the card installed on the target computer. If no supported Ethernet card is installed in your target computer, the software returns an error.

8 If the target computer has multiple Ethernet cards, follow the procedure in "Command-Line Ethernet Card Selection by Index" on page 4-19.

Repeat this procedure as required for each target computer.

The next task is "Command-Line Target Computer Settings" on page 4-21.

Command-Line USB-to-Ethernet Setup

If the target computer has a USB 2.0 port but no supported PCI Ethernet card, use a USB-to-Ethernet adapter.

In this section...
"USB-to-Ethernet Protocol Hardware" on page 4-10
"Command-Line USB-to-Ethernet Settings" on page 4-11

USB-to-Ethernet Protocol Hardware
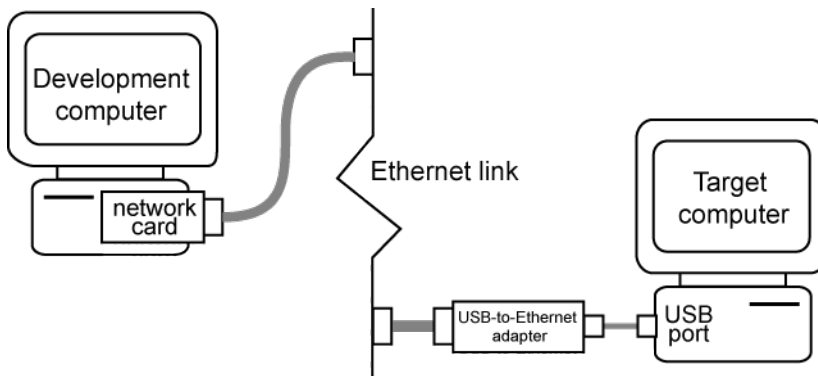
To install USB-to-Ethernet protocol interface hardware:

1 Acquire a supported USB-to-Ethernet adapter.

   If you want to start the target computer from the network, the Ethernet adapter must be compatible with the Preboot eXecution Environment (PXE) specification.

2 Turn off your target computer.

3 Plug an Ethernet-to-USB adapter into the USB port on the target.

4 Connect the Ethernet-to-USB adapter to your LAN using an unshielded twisted-pair (UTP) cable.

5 Assign a static IP address to the target computer USB-to-Ethernet adapter.

   Although the target computer Ethernet adapter must have a static IP address, the development computer network adapter can have a Dynamic Host Configuration

Protocol (DHCP) address and can be accessed from the network. You must configure the DHCP server to reserve static IP addresses to prevent these addresses from being assigned to other systems.

6 Connect your target computer Ethernet adapter to your LAN using an unshielded twisted-pair (UTP) cable.

You can directly connect your computers using a crossover UTP cable with RJ45 connectors. Both computers must have static IP addresses. If the development computer has a second network adapter, that adapter can have a DHCP address.

Do not connect the development computer USB port to the target computer USB port using a USB cable. A USB-to-Ethernet adapter plugged into the target computer USB port behaves like an Ethernet card installed on the target computer.

Command-Line USB-to-Ethernet Settings

After you install the USB-to-Ethernet adapter, before you can build and download a real-time application, you must specify the environment properties for the development and target computers.

Before you start, ask your system administrator for the following information for your target computer:

- IP address
- Subnet mask address
- Port number (optional)
- Gateway (optional)

Use the following procedure for target `TargetPC1`:

1 At the MATLAB prompt, create an environment object for this target computer and make it the default target:

```
env = SimulinkRealTime.addTarget('TargetPC1');
setAsDefaultTarget(env);
```

All other settings will be made to this object.

2 Set the IP address for your target computer. For example:

```
env.TcpIpTargetAddress = '10.10.10.15';
```

3 Set the subnet mask address of your LAN. For example:

```
env.TcpIpSubNetMask = '255.255.255.0';
```

4 Set the TCP/IP port (optional) to a value higher than `'20000'` and less than `'65536'`. For example:

```
env.TcpIpTargetPort = '22222';
```

This property is set by default to `'22222'`, a value higher than the reserved area (`telnet`, `ftp`, and so on).

5 Set the TCP/IP gateway (optional) to the gateway required to access the target computer. For example:

```
env.TcpIpGateway = '255.255.255.255';
```

This property is set by default to `'255.255.255.255'`, which means that you do not use a gateway to connect to your target computer. If you connect your computers with a crossover cable, leave this property as `'255.255.255.255'`.

If you communicate with the target computer from within your LAN, you might not need to change this setting. If you communicate from a development computer located in a LAN different from your target computer (especially via the Internet), you must define a gateway and enter its IP address in this box.

6 Set the bus type to `'USB'`.

```
env.TcpIpTargetBusType = 'USB';
```

7 Set the target driver to one of `'USBAX772'`, `'USBAX172'`, or `'Auto'`.

```
env.TcpIpTargetDriver = 'Auto';
```

If the target driver is `'Auto'`, the software sets the driver to `'USBAX772'`, the driver most commonly used.

Repeat this procedure as required for each target computer.

The next task is "Command-Line Target Computer Settings" on page 4-21.

Command-Line ISA Bus Ethernet Setup

Your target computer might not have an available PCI bus slot or USB 2.0 port. In these cases, use an Ethernet card for an ISA bus.

Note: Support for using ISA bus Ethernet cards to communicate between the development and target computers will be removed in a future release. Use PCI bus or USB bus Ethernet cards instead.

In this section...
"ISA Bus Ethernet Protocol Hardware" on page 4-13
"Command-Line ISA Bus Ethernet Settings" on page 4-14

ISA Bus Ethernet Protocol Hardware

To install ISA bus Ethernet protocol interface hardware:

1   Acquire a supported ISA bus Ethernet card.

    If you want to start the target computer from the network, the Ethernet adapter must be compatible with the Preboot eXecution Environment (PXE) specification.

2   Turn off your target computer.

3   On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers or switches on the card. Write down these settings, because you must enter them in Simulink Real-Time Explorer.

    Set the IRQ line to 5 and the I/O-port base address to around 0x300. If one of these settings leads to a conflict in your target computer, select another IRQ or I/O-port base address.

    If your ISA bus card does not contain jumpers to set the IRQ line and the base address, after installation use the utility on the installation disk supplied with your card to manually assign the IRQ line and base address.

    If you use an Ethernet card for an ISA bus within a target computer that has a PCI bus, after installation you must reserve the chosen IRQ line number for the

Ethernet card in the PCI BIOS. To set up the PCI BIOS, refer to your BIOS setup documentation.

Do not configure the card as a PnP-ISA device.

4 If the target computer already has an unsupported Ethernet card, remove the card. Plug the compatible network card into a free ISA bus slot.

5 Assign a static IP address to the target computer Ethernet card.

Although the target computer Ethernet card must have a static IP address, the development computer network adapter card can have a Dynamic Host Configuration Protocol (DHCP) address and can be accessed from the network. You must configure the DHCP server to reserve static IP addresses to prevent these addresses from being assigned to other systems.

6 Connect your target computer Ethernet card to your LAN using an unshielded twisted-pair (UTP) cable.

You can directly connect your computers using a crossover UTP cable with RJ45 connectors. Both computers must have static IP addresses. If the development computer has a second network adapter card, that card can have a DHCP address.

Command-Line ISA Bus Ethernet Settings

After you install the ISA bus Ethernet card, before you can build and download a real-time application, you must specify the environment properties for the development and target computers.

Before you start, ask your system administrator for the following information for your target computer:

- IP address
- Subnet mask address
- Port number (optional)
- Gateway (optional)

Use the following procedure for target `TargetPC1`:

1 At the MATLAB prompt, create an environment object for this target computer and make it the default target:

```
env = SimulinkRealTime.addTarget('TargetPC1');
```

```
setAsDefaultTarget(env);
```

All other settings will be made to this object.

2 Set the IP address for your target computer. For example:

```
env.TcpIpTargetAddress = '10.10.10.15';
```

3 Set the subnet mask address of your LAN. For example:

```
env.TcpIpSubNetMask = '255.255.255.0';
```

4 Set the TCP/IP port (optional) to a value higher than `'20000'` and less than `'65536'`. For example:

```
env.TcpIpTargetPort = '22222';
```

This property is set by default to `'22222'`, a value higher than the reserved area (`telnet`, `ftp`, and so on).

5 Set the TCP/IP gateway (optional) to the gateway required to access the target computer. For example:

```
env.TcpIpGateway = '255.255.255.255';
```

This property is set by default to `'255.255.255.255'`, which means that you do not use a gateway to connect to your target computer. If you connect your computers with a crossover cable, leave this property as `'255.255.255.255'`.

If you communicate with the target computer from within your LAN, you might not need to change this setting. If you communicate from a development computer located in a LAN different from your target computer (especially via the Internet), you must define a gateway and enter its IP address in this box.

6 Set the bus type to `'ISA'`.

```
env.TcpIpTargetBusType = 'ISA';
```

7 Set the target driver to one of `'NE2000'` or `'SMC91C9X'`. For example:

```
env.TcpIpTargetDriver = 'NE2000';
```

Target driver `'Auto'` is not supported for bus type `'ISA'`.

8 Set the I/O-port base address and IRQ to values that correspond with the jumper settings or ROM settings on your ISA bus Ethernet card. For example:

```
env.TcpIpTargetISAMemPort = '0x300';
```

```
env.TcpIpTargetISAIRQ = '5';
```

Repeat this procedure as required for each target computer.

Ethernet Card Selection by Index

If the target computer has multiple Ethernet cards, you must specify which card to use for the Ethernet link. Use the following procedure to discover the Ethernet index of the PCI cards on the target computer and to specify which card to use.

Note: For this procedure, you must be able to burn CDs on your development computer and use `Network` boot mode for routine target operations.

Use the following procedure for target `TargetPC1`:

1 At the MATLAB prompt, get the environment object for this target computer and make it the default target:

```
env = SimulinkRealTime.getTargetSettings('TargetPC1');
setAsDefaultTarget(env);
```

All other settings will be made to this object.

2 At the MATLAB prompt, type:

```
env.ShowHardware = 'on';
```

3 At the MATLAB prompt, type: `slrtexplr`.

4 In the **Targets** pane, expand the target computer node.

5 In the toolbar, click the Target Properties button 🔧.

6 Select **Host-to-Target communication** and set **Target driver** to `Auto`. If you set **Target driver** to a specific driver, such as `INTEL_I82559`, the kernel displays only information about boards that use that driver.

7 Select **Target settings** and clear the **Graphics mode** check box. This setting causes the kernel to print text only.

8 Select **Boot configuration** and set **Boot mode** to `CD`.

9 Click **Create boot disk** and follow the prompts to create a new boot disk.

10 Insert the new boot disk and restart the target computer from the target computer boot switch.

After the start is complete, the target monitor displays information about the Ethernet cards in the target computer, for example :

```
index: 0, driver: RTLANCE, Bus: 16, Slot: 7, Func: 0
index: 1, driver: R8139, Bus: 16, Slot: 8, Func: 0
index: 2, driver: I82559, Bus: 16, Slot: 9, Func: 0
```

You might need to change the boot order from the target computer BIOS to allow starting from your disk. After the kernel starts with ShowHardware 'on', the development computer cannot communicate with the target computer.

11 Note the index of the Ethernet card that you want to use for the Ethernet link, for example, 2.

12 At the MATLAB prompt, type:

```
env.ShowHardware = 'off';
env.EthernetIndex = '#';
```

is the index of the Ethernet card, for example, 2.

13 Select **Target settings** and select the **Graphics mode** check box.

14 Set **Boot mode** to Network.

15 Click **Create boot disk**.

16 Remove the boot disk from the target computer drive and start the target computer from the target computer boot switch.

The kernel selects the specified Ethernet card as the target computer card, instead of the default card with index number 0.

Repeat this procedure as required for each target computer.

Command-Line Ethernet Card Selection by Index

If you are using multiple target computers that have multiple Ethernet cards, you must specify which card to use for the Ethernet link. Use the following procedure to discover the Ethernet index of the PCI cards on a specific target and specify which card to use.

Note: For this procedure, you must be able to burn CDs on your development computer and use network boot mode for routine target operations.

Use the following procedure for target `TargetPC1`:

1 At the MATLAB prompt, get the environment object for this target computer and make it the default target:

```
env = SimulinkRealTime.getTargetSettings('TargetPC1');
setAsDefaultTarget(env);
```

All other settings will be made to this object.

2 At the MATLAB prompt, type:

```
env.ShowHardware = 'on';
```

3 Set the Ethernet driver to the default:

```
env.TcpIpTargetDriver = 'Auto';
```

If `TcpIpTargetDriver` is set to a specific driver, such as `'I82559'`, the kernel displays only information about boards that use that driver.

4 Set the boot method to CD/DVD boot:

```
env.TargetBoot='CDBoot';
```

5 Set the target monitor to print text only:

```
env.TargetScope = 'Disabled' ;
```

6 Type `SimulinkRealTime.createBootImage`.

The Simulink Real-Time software displays the following message and creates the CD/DVD boot image.

```
Current boot mode: CDBoot
```

```
CD boot image is successfully created

Insert an empty CD/DVD. Available drives:
  [1]  d:\
  [0]  Cancel Burn
```

7 Insert the new boot disk and restart the target computer from the target computer boot switch.

After the start is complete, the target monitor displays information about the Ethernet cards in the target computer, for example:

```
index: 0, driver: RTLANCE, Bus: 16, Slot: 7, Func: 0
index: 1, driver: R8139, Bus: 16, Slot: 8, Func: 0
index: 2, driver: I82559, Bus: 16, Slot: 9, Func: 0
```

You might need to change the boot order from the target computer BIOS to allow starting from your disk. After the kernel starts with ShowHardware 'on', the development computer cannot communicate with the target computer.

8 Note the index of the Ethernet card you want to use for the Ethernet link, for example, 2.

9 At the MATLAB prompt, type:

```
env.ShowHardware = 'off';
env.EthernetIndex = '#';
```

is the index of the Ethernet card, for example, 2.

10 Set the boot method back to network boot:

```
env.TargetBoot= 'NetworkBoot';
```

11 Set the target monitor to graphics mode:

```
env.TargetScope = 'Enabled' ;
```

12 Type SimulinkRealTime.createBootImage.

13 Start the target computer from the target computer boot switch.

The kernel selects the specified Ethernet card as the target computer card, instead of the default card with index number 0.

Repeat this procedure as required for each target computer.

Command-Line Target Computer Settings

To run a Simulink Real-Time model on a target computer, you must configure the target settings to match the capabilities of the target computer.

Use the following procedure for target `TargetPC1`:

1   At the MATLAB prompt, get the environment object for this target computer and make it the default target:

```
env = SimulinkRealTime.getTargetSettings('TargetPC1');
setAsDefaultTarget(env);
```

All other settings will be made to this object.

2   Assign the following target computer settings as required:

- **Target scope display**

  - `env.TargetScope='Enabled'` (the default) — Use if you want to display information, such as a target scope, in graphic format.
  - `env.TargetScope='Disabled'` — Use if you want to display information as text.

  To use the full features of a target scope, install a keyboard on the target computer.

- **USB support**

  - `env.USBSupport='on'` (the default) — Use if you want to use a USB port on the target computer; for example, to connect a USB mouse.
  - `env.USBSupport='off'` — Otherwise.

- **Secondary IDE support**

  - `env.SecondaryIDE='on'` — Use only if you want to use the disks connected to a secondary IDE controller.
  - `env.SecondaryIDE='off'` (the default) — Otherwise.

- **Multicore support**

  `env.MulticoreSupport='on'` (the default) — Use if your target computer has multicore processors that you want to take advantage of.

env.MulticoreSupport='off' — Otherwise.

· **Non-Pentium support**

env.NonPentiumSupport='on' — Use if your target computer has a 386 or 486 compatible processor.

env.NonPentiumSupport='off' (the default) — Otherwise.

If your target computer has a Pentium or higher compatible processor, setting this value to 'on' slows the performance of your target computer.

· **Target RAM size**

env.TargetRAMSizeMB='Auto' (the default) — Use if you want the real-time application to read the target computer BIOS and determine the amount of memory installed in the target computer.

env.TargetRAMSizeMB='xxx' — Use if the real-time application cannot read the BIOS. You must assign the amount of memory, in megabytes, installed in the target computer.

The Target RAM size parameter defines the total amount of installed RAM in the target computer available for the kernel, real-time application, data logging, and other functions that use the heap.

The Simulink Real-Time kernel supports a maximum of 4 GB of target computer memory.

· **Maximum model size**

env.MaxModelSize='1MB' (the default) — Use if the real-time application requires at most this much memory on the target computer.

env.MaxModelSize='4MB' — For a medium-sized model.

env.MaxModelSize='16MB' — For a large-sized model.

Setting **Maximum model size** takes effect for env.TargetBoot='StandAlone' only.

Memory not used by the real-time application is used by the kernel and by the heap for data logging. Selecting too high a value leaves less memory for data

logging. Selecting too low a value does not reserve enough memory for the real-time application and creates an error. You can approximate the size of the real-time application by the size of the DLM file produced by the build process.

Repeat this procedure as required for each target computer.

The next task is "Command-Line Target Boot Methods" on page 4-24.

Command-Line Target Boot Methods

You can start your target computer with the Simulink Real-Time kernel using one of several methods.

Speedgoat systems come with `DOS Loader` software preinstalled. You can set up the `DOS Loader` boot method on your development computer or configure another boot method. See your Speedgoat system documentation or follow the link from "Speedgoat Real-Time Target Machines" for further information.

1   Before creating a boot kernel, perform "Command-Line Kernel Creation Prechecks" on page 4-25.

2   Select one of the following methods:

   • "Command-Line Network Boot Method" on page 4-26
   • "Command-Line CD/DVD Boot Method" on page 4-28
   • "Command-Line DOS Loader Boot Method" on page 4-29
   • "Command-Line Removable Disk Boot Method" on page 4-31
   • "Command-Line Standalone Boot Method" on page 4-33

3   For boot methods other than `StandAlone`, perform "Run Confidence Test on Configuration".

   For boot method `StandAlone`, create a model-specific confidence test, restart the target computer, and run that confidence test. The default confidence test is not intended for standalone execution.

Command-Line Kernel Creation Prechecks

Before creating the target boot kernel, configure your Simulink Real-Time system. At a minimum, do the following:

1 Check the physical connections between the development computer and the target computer. If you are using TCP/IP, these are Ethernet connections that may pass through a LAN.

2 Check your target computer BIOS settings (see "BIOS Settings").

3 Check that you have write permission for your current working folder.

4 At the MATLAB prompt, get the environment object for this target computer and make it the default target:

```
env = SimulinkRealTime.getTargetSettings('TargetPC1');
setAsDefaultTarget(env);
```

The contents of environment object `env` are printed in the Command Window.

5 Check the link settings. See "Command-Line PCI Bus Ethernet Setup" on page 4-7 or "Command-Line USB-to-Ethernet Setup" on page 4-10.

6 Check that `TargetBoot` is set to the required value.

Repeat this procedure as required for each target computer.

Command-Line Network Boot Method

After you have configured the target computer environment parameters, you can use a dedicated Ethernet network to load and run the Simulink Real-Time kernel. You do not need a boot CD or removable boot drive.

There are the following limitations:

- Do not use the network boot method on a corporate or nondedicated network. Doing so might interfere with dynamic host configuration protocol (DHCP) servers and cause problems with the network.
- Your Ethernet card must be compatible with the Preboot eXecution Environment (PXE) specification.
- If `Stand Alone` mode is enabled, you cannot start the target computer across the network.

Before you start, establish the required Ethernet connection between development and target computers using the procedures in "Command-Line PCI Bus Ethernet Setup" on page 4-7 or "Command-Line USB-to-Ethernet Setup" on page 4-10.

Use the following procedure for target `TargetPC1`:

1 At the MATLAB prompt, get the environment object for this target computer and make it the default target:

```
env = SimulinkRealTime.getTargetSettings('TargetPC1');
setAsDefaultTarget(env);
```

The contents of environment object `env` are printed in the Command Window. Some properties may already have the required values.

2 Set network boot method:

```
env.TargetBoot='NetworkBoot'
```

3 Set a TCP/IP address. Verify that the subnet of this IP address is the same as the development computer. Otherwise your network boot fails. For example, type:

```
env.TcpIpTargetAddress='10.10.10.11'
```

4 Set the target computer MAC address (in hexadecimal). For example, type:

```
env.TargetMACAddress='01:23:45:67:89:ab'
```

5 In the Command Window, type:

```
SimulinkRealTime.createBootImage
```

The following message appears:

```
Current boot mode: NetworkBoot
Synchronizing network boot table....ok
Starting network boot server....ok
Creating batch file (slrtnetboot.bat)....ok
Network boot image created successfully
```

The software creates and starts a network boot server process on the development computer. You start the target computer using this process.

A minimized icon () representing the network boot server process appears on the bottom right of the development computer system tray.

Repeat this procedure as required for each target computer.

The next task is "Run Confidence Test on Configuration".

Command-Line CD/DVD Boot Method

After you have configured the target computer environment parameters, you can use a target boot CD or DVD to load and run the Simulink Real-Time kernel. This topic describes using the MATLAB command line to create a boot CD or DVD for a single target computer system.

Use the following procedure for target `TargetPC1`:

1 At the MATLAB prompt, get the environment object for this target computer and make it the default target:

```
env = SimulinkRealTime.getTargetSettings('TargetPC1');
setAsDefaultTarget(env);
```

All other settings will be made to this object.

2 Set the CD boot method:

```
env.TargetBoot='CDBoot'
```

3 In the Command Window, type `SimulinkRealTime.createBootImage`.

The Simulink Real-Time software displays the following message and creates the CD/DVD boot image.

```
Current boot mode: CDBoot
CD boot image is successfully created

Insert an empty CD/DVD. Available drives:
  [1]  d:\
  [0]  Cancel Burn
```

4 Insert the empty CD or DVD in the development computer.

5 Type 1 and then press **Enter**.

6 When the write operation has finished, remove the CD or DVD from the drive.

7 Insert the bootable CD/DVD into your target computer drive and restart the target computer.

Repeat this procedure as required for each target computer.

The next task is "Run Confidence Test on Configuration".

Command-Line DOS Loader Boot Method

DOS Loader mode allows you to start the Simulink Real-Time kernel on a target computer from a fixed or removable device with DOS boot capability, such as a hard disk or flash memory. After starting the target computer, you can download your real-time application from the development computer over a serial or Ethernet link between the development and target computers.

Note: To run in DOS Loader mode, the target computer boot device must provide a minimal DOS environment complying with certain restrictions. For details, see:

- "Create a DOS System Disk"
- "DOS Loader Mode Restrictions"

Use the following procedure for target `TargetPC1`:

1 At the MATLAB prompt, get the environment object for this target computer and make it the default target:

```
env = SimulinkRealTime.getTargetSettings('TargetPC1');
setAsDefaultTarget(env);
```

All other settings will be made to this object.

2 Set the DOS Loader boot method:

```
env.TargetBoot = 'DOSLoader';
```

3 Set `DOSLoaderLocation` to the folder where you want to create the DOS Loader boot files. This location can be a local folder on the development computer or a removable storage device that you use to start the target computer. By default, the folder is the current working folder.

```
env.DOSLoaderLocation = 'D:\';
```

4 In the Command Window, type `SimulinkRealTime.createBootImage`.

The Simulink Real-Time software displays the following message:

```
Current boot mode: DOSLoader
Simulink Real-Time DOS Loader files are successfully created
```

This operation creates the following boot files in the specified location:

```
autoexec.bat
xpcboot.com
*.rtb
```

5 If you create boot files on a local hard disk, copy these files to a floppy disk, CD/DVD, or other removable storage media.

6 Transfer the boot files to your target computer or insert the removable media containing the boot files into the target computer drive or USB port.

7 Verify that `autoexec.bat` file is on the DOS boot path (typically the root folder).

8 Select the required boot device in the BIOS of the target computer.

9 Start the target computer.

When the target computer starts, it loads DOS, which executes the `autoexec.bat` file. This file starts the Simulink Real-Time kernel (*.rtb). The target computer then awaits commands from the development computer.

Repeat this procedure as required for each target computer.

The next task is "Run Confidence Test on Configuration".

Command-Line Removable Disk Boot Method

After you have configured the target computer environment parameters, you can use a target boot floppy disk, removable drive, or USB flash drive to load and run the Simulink Real-Time kernel. This topic describes using the MATLAB command line to create a removable boot disk.

If you are creating a removable boot drive from a USB flash drive, before performing this procedure, create a bootable partition on the drive. See "Create a Bootable Partition".

Use the following procedure for target `TargetPC1`:

1 At the MATLAB prompt, get the environment object for this target computer and make it the default target:

```
env = SimulinkRealTime.getTargetSettings('TargetPC1');
setAsDefaultTarget(env);
```

All other settings will be made to this object.

2 In the output of the `setAsDefaultTarget` command, verify that property `TargetBoot` is `BootFloppy`.

If required, update property `TargetBoot`, for instance by using the command `env.TargetBoot='BootFloppy'`.

3 If you are creating a removable boot disk from a USB drive, insert the USB drive in the development computer USB port and wait for it to be recognized.

4 In the Command Window, type `SimulinkRealTime.createBootImage`.

The Simulink Real-Time software creates the CD/DVD boot image and displays the following message:

```
Current boot mode: BootFloppy
Insert a formatted floppy disk into your host PC's
disk drive and press a key to continue
```

5 If required, insert an empty removable disk in the development computer drive and then press a key.

6 When the write operation has finished, remove the removable disk from the drive or USB port.

7 Insert the removable boot disk into your target computer drive or USB port and restart the target computer.

Repeat this procedure as required for each target computer.

The next task is "Run Confidence Test on Configuration".

Command-Line Standalone Boot Method

Using the MATLAB command line, you can configure the Simulink Real-Time software to run as a standalone real-time application. For information on **Boot mode** `Stand Alone`, see "Standalone Mode".

To run in `Stand Alone` mode, the target computer and its DOS environment must meet specific requirements and restrictions.

In this section...
"Target Computer Requirements" on page 4-33
"DOS Environment Restrictions" on page 4-34
"Command-Line Standalone Settings" on page 4-34
"Real-Time Application Build" on page 4-35
"Real-Time Application Transfer" on page 4-36
"Real-Time Application Boot Configuration" on page 4-37

Target Computer Requirements

To run in `Stand Alone` mode, the target computer must meet specific requirements.

In `Stand Alone` mode, the real-time application is combined with the kernel on a hard drive or flash memory. The kernel and real-time application start together on the target computer. The development computer does not need to be connected to the target computer, but it must be able to start from DOS.

For each target computer, verify that the system meets the following requirements:

- The target computer hard drive must be a serial ATA (SATA) or parallel ATA (PATA)/ Integrated Device Electronics (IDE) drive.
- The target computer must not cable select the hard drive.
- The target computer BIOS must be able to detect the hard drive.
- The hard drive must be formatted with one of these file systems:
  - FAT-12
  - FAT-16

- FAT-32
- You must install a supported version of DOS on the target computer hard drive. If required, create a standard DOS boot device from a CD ROM, 3.5–inch floppy drive, flash drive, or hard drive. See "Create a DOS System Disk".
- You must configure the Ethernet link. This communication transfers the kernel and real-time application files built on the development computer to the target computer.

DOS Environment Restrictions

To use `Stand Alone` mode, the target computer DOS environment must comply with the following restrictions:

- The CPU must execute in real mode.
- While loaded in memory, the DOS partition must not overlap the address range of a real-time application.

To satisfy these restrictions:

- Avoid additional memory managers, such as `emm386` or `qemm`.
- Avoid utilities that attempt to load in high memory (for example, `himem.sys`).
- Avoid including memory manager entries in the `autoexec.bat` file.
- Avoid using a `config.sys` file.

There are additional restrictions on target and scope settings:

- In the Simulink Real-Time Explorer **Target settings** pane, you can use only **Model size** values `1 MB` and `4 MB`. Setting **Model size** is enabled for **Boot mode** `Stand Alone` only.
- If you want to use real-time `Scope` blocks to display or record output:
  - You must use only `Target` or `File` type blocks.
  - You must select the **Start scope when application starts** check box in the block parameters dialog box.

Command-Line Standalone Settings

Use the command line to set the kernel environment properties. When you are done, you can create a standalone kernel combined with your real-time application.

For **Boot mode** Stand Alone, you do not create a Simulink Real-Time boot disk or network boot image. Instead, you copy files created from the build process to the target computer hard drive.

Use the following procedure for target TargetPC1:

1   At the MATLAB prompt, get the environment object for this target computer and make it the default target:

```
env = SimulinkRealTime.getTargetSettings('TargetPC1');
setAsDefaultTarget(env);
```

All other settings will be made to this object.

2   Set network boot method:

```
env.TargetBoot='StandAlone';
```

Repeat this procedure as required for each target computer.

Real-Time Application Build

After you set Simulink Real-Time boot mode to Stand Alone, you can use Simulink Real-Time, Simulink Coder, and a C/C++ compiler in Stand Alone mode to build a combined kernel and real-time application with utility files.

1   In the Command Window, type the name of your Simulink model, for example, ex_slrt_rt_osc. Simulink Editor opens displaying the model.

2   From the **Code** menu, click **C/C++ Code** > **Build Model**.

The Simulink Coder and Simulink Real-Time software create a folder, ex_slrt_rt_osc_slrt_emb, containing the following files:

- autoexec.bat — Contains Simulink Real-Time-specific code that calls the xpcboot.com executable to start the Simulink Real-Time kernel (the *.rtb file).

- xpcboot.com — Loads and executes the *.rtb file. This static file is part of the Simulink Real-Time software.

- ex_slrt_rt_osc.rtb — Contains the Simulink Real-Time kernel and the real-time application. This image also contains applicable options, such as serial or Ethernet communications and the COM port or IP address of the target computer.

You do not need a `config.sys` file to start the kernel using `autoexec.bat` and `xpcboot.com`.

The **Model size** parameter gives an upper bound on the size of the compiled real-time application (DLM). It is only enabled in `Stand Alone` mode. You can use only **Model size** values `1 MB` and `4 MB`. If the DLM is larger than **Model size**, the build software generates an error.

Repeat this procedure as required for each real-time application.

Real-Time Application Transfer

After building the kernel and real-time application on a development computer, transfer the files to a target computer using the Simulink Real-Time FTP functions:

1 Start the target computer from a target computer boot disk.

2 In the Command Window, change to the folder that contains the kernel and real-time application files.

3 In the Command Window, create a Simulink Real-Time FTP object:

```
fsys = SimulinkRealTime.fileSystem
```

4 Use the FTP object to create the folder `C:\xpcfiles` on the target computer:

```
mkdir(fsys, 'xpcfiles')
```

5 Move to the `xpcfiles` folder:

```
cd(fsys, 'xpcfiles')
```

6 Copy these files to the `xpcfiles` folder:

```
tg = slrt;
SimulinkRealTime.copyFileToTarget(tg,'autoexec.bat')
SimulinkRealTime.copyFileToTarget(tg,'xpcboot.com')
SimulinkRealTime.copyFileToTarget(tg,'ex_slrt_rt_osc.rtb')
```

7 Remove the boot disk from the target computer.

Repeat this procedure as required for each target computer.

Real-Time Application Boot Configuration

After transferring the files to a target computer, configure the target computer to run the real-time application automatically upon start.

1 Restart the target computer and bring up the DOS prompt.

If the target computer that you want to start in `Stand Alone` mode was previously started from the network boot image, select the `Stand Alone` check box to disable the network boot capability.

2 At the DOS prompt, save a copy of the target computer `C:\autoexec.bat` file to a backup file, such as `C:\autoexec_back.wrk`.

3 Edit the target computer file `C:\autoexec.bat` to include the following lines:

```
cd C:\xpcfiles
autoexec
```

Adding these commands to `C:\autoexec.bat` directs the system to execute the `autoexec.bat` file located in `C:\xpcfiles`.

4 Restart the target computer.

The target computer executes the following sequence of calls:

a `C:\autoexec.bat`
b `C:\xpcfiles\autoexec.bat`
c `C:\xpcfiles\xpcboot.com`
d `C:\xpcfiles\<application>.rtb`

Repeat this procedure as required for each target computer.

Note: Do not confuse `C:\xpcfiles\autoexec.bat` with `C:\autoexec.bat`. `C:\xpcfiles\autoexec.bat` includes the command `xpcboot.com`, which starts the Simulink Real-Time kernel and the standalone real-time application. `C:\autoexec.bat` includes the files you want the system to execute when the system starts.

Continue by restarting the target computer and testing your real-time application in `Stand Alone` mode. The confidence test is not intended for standalone execution. You must create and execute your own confidence test for `Stand Alone` mode.

5

Signals and Parameters

Changing parameters in your real-time application while it is running, viewing the resulting signal data, and checking the results, are important prototyping tasks. The Simulink Real-Time software includes command-line and graphical user interfaces to complete these tasks.

Signal Monitoring Basics

Signal monitoring acquires real-time signal data without time information during real-time application execution. There is minimal additional load on the real-time tasks. Use signal monitoring to acquire signal data without creating scopes that run on the target computer.

In addition to signal monitoring, Simulink Real-Time enables you to monitor Stateflow states as test points through the Simulink Real-Time Explorer and MATLAB command-line interfaces. You designate data or a state in a Stateflow diagram as a test point, making it observable during execution. You can work with Stateflow states as you do with Simulink Real-Time signals, such as monitoring or plotting Stateflow states.

When you monitor signals from referenced models, first set the test point for the signal in the referenced model.

Note:

- Simulink Real-Time Explorer works with multidimensional signals in column-major format.

- Some signals are not observable.

You can monitor signals using Simulink Real-Time Explorer and MATLAB language. You can monitor Stateflow states using Simulink Real-Time Explorer, MATLAB language, and Simulink external mode.

More About

- "Nonobservable Signals" on page 5-136
- "Simulink Real-Time Scope Usage" on page 5-22
- "Target Scope Usage" on page 5-24
- "Host Scope Usage" on page 5-29
- "File Scope Usage" on page 5-77

Monitor Signals Using Simulink Real-Time Explorer

This procedure uses the model xpcosc as an example. You must have already completed the following setup:

1   Built and downloaded the real-time application to the target computer using Simulink (📅 on the toolbar).

2   Run Simulink Real-Time Explorer (**Tools** > **Simulink Real-Time**).

3   Connected to the target computer in the **Targets** pane (🖥 on the toolbar).

4   Set property **Stop time** to inf in the **Applications** pane (⚙ on the toolbar).

To monitor a signal:

1   In Simulink Real-Time Explorer, expand the **Model Hierarchy** node under the real-time application node.

2   To view the signals in the real-time application, select the model node. On the toolbar, click the **View Signals** button ⊢. The Signals workspace opens.

3   To view the value of a signal, in the Signals workspace, select the **Monitor** check box for the signal. For instance, select the check boxes for **Signal Generator** and **Integrator1**. The signal values are shown in the **Monitoring Value** column.

4   To start execution, click the real-time application. On the toolbar, click the **Start** button ▶.

5   To stop execution, click the real-time application. On the toolbar, click the **Stop** button ■.

The Application Parameters and Signals workspaces look like this figure.

To make both workspaces visible at the same time, drag one workspace tab down until the ⬩ icon appears in the middle of the dialog box. Continue to drag until the cursor reaches the required quadrant, and then release the mouse button.

To save your Simulink Real-Time Explorer layout, click **File** > **Save Layout**. In a later session, you can click **File** > **Restore Layout** to restore your layout.

Related Examples

·   "Create Signal Groups Using Simulink Real-Time Explorer" on page 5-51

·   "Signal Group Monitoring Formats" on page 5-12

More About

·   "Nonobservable Signals" on page 5-136

Monitor Signals Using MATLAB Language

This procedure uses the model `ex_slrt_rt_osc` as an example. You must have already completed the setup in "Prepare Real-Time Application Using MATLAB Language".

Note:

- Signal access by signal index will be removed in a future release. Access signals by signal name instead.

- The Simulink Real-Time software lists referenced model signals with their full block path. For example, `ex_slrt_rt_osc/childmodel/gain`.

---

1 To get a list of signals, type:

```
tg.ShowSignals = 'on'

Target: TargetPC1
   Connected            = Yes
   Application          = xpcosc
.
.
.
   Scopes               = 1
   NumSignals           = 7
   ShowSignals          = on
   Signals              =
     INDEX  VALUE     Type     BLOCK NAME      LABEL
     0      0.000000  DOUBLE   Gain
     1      0.000000  DOUBLE   Gain1
     2      0.000000  DOUBLE   Gain2
     3      0.000000  DOUBLE   Integrator
     4      0.000000  DOUBLE   Integrator1
     5      0.000000  DOUBLE   Signal Generator
     6      0.000000  DOUBLE   Sum
.
.
.
```

If your signal has a unique label, its label is displayed in the `Label` column. If the label is not unique, the command returns an error.

2 To get the value of a signal, use the `getsignal` method. In the Command Window, type:

```
getsignal(tg,'Integrator1')
ans =

   -3.8771
```

Related Examples

- "Configure Target Scopes Using MATLAB Language" on page 5-60

More About

- "Nonobservable Signals" on page 5-136

Configure Stateflow States as Test Points

This procedure uses the model `old_sf_car` as an example. It describes one way to set Stateflow states as test points for monitoring.

1 In the Command Window, type `old_sf_car`.

2 In the Simulink Editor, click **Simulation** > **Model Configuration Parameters**.

3 In the Configuration Parameters dialog box, click the **Code Generation** node.

4 To build a basic real-time application, in the **Target selection** section of the **Code Generation** pane, click **Browse** at the **System target file** list. Click `slrt.tlc`, and then click **OK**.

5 In the `old_sf_car` model, double-click the `shift_logic` chart.
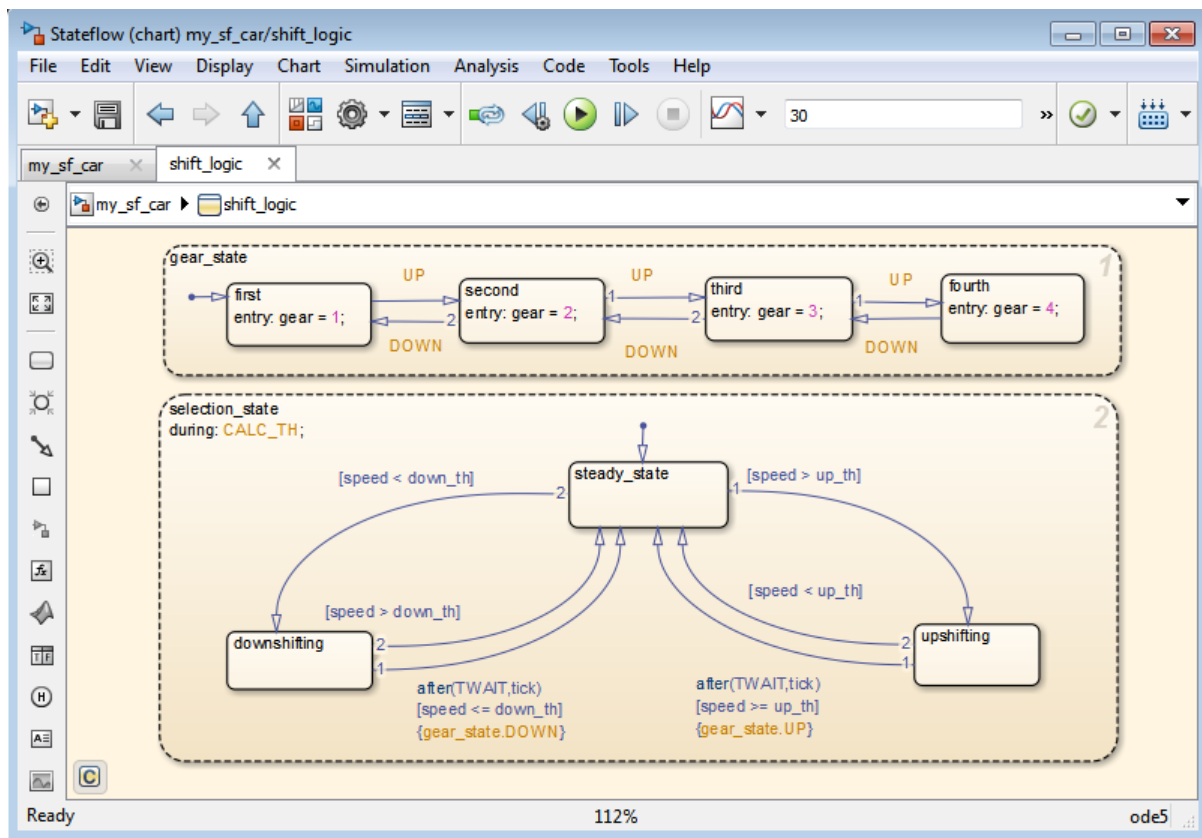
6 In the `shift_logic` chart, click **Tools** > **Model Explorer**.

7 In the Model Explorer, expand **old_sf_car**, and then **shift_logic**.

8 Expand **gear_state**, and then select **first**.

9 In the **State first** pane **Logging** tab, select the **Test point** check box. This selection creates a test point for the `first` state.

10 Click **Apply**.

11 Repeat steps 8–10 for **gear_state** values **second**, **third**, and **fourth**.

12 Build and download the `old_sf_car` real-time application to the target computer.

You can now view the states with Simulink Real-Time Explorer or the MATLAB interface.

Related Examples

- "Monitor Stateflow States Using Simulink Real-Time Explorer" on page 5-10
- "Monitor Stateflow States Using MATLAB Language" on page 5-13
- "Animate Stateflow Charts Using Simulink External Mode" on page 5-14

More About

- "Nonobservable Signals" on page 5-136

Monitor Stateflow States Using Simulink Real-Time Explorer

You must have already and built and downloaded the real-time application to carry out this procedure.

This procedure uses the model old_sf_car as an example. You must have already completed the following setup:

1   Set Stateflow states as test points.

2   Built and downloaded the real-time application to the target computer using Simulink (on the toolbar).

3   Run Simulink Real-Time Explorer (**Tools** > **Simulink Real-Time**).

4   Connected to the target computer in the **Targets** pane (on the toolbar).

To monitor a test point:

1   In the **Applications** pane, expand the real-time application and the **Model Hierarchy** node.

2   To view the test point, select shift_logic and click the **View Signals** button on the toolbar.

    The Signals workspace opens. The test point gear_state.first appears like other signals in the Signals workspace.

3   In the Signals workspace, select the **Monitor** check box for gear_state.first, gear_state.second, gear_state.third, and gear_state.fourth. The values of the signals are shown in the **Monitoring Value** column.

4   To start execution, click the real-time application. On the toolbar, click the **Start** button .

5   To stop execution, click the real-time application. On the toolbar, click the **Stop** button .

    The Application Parameters and Signals workspaces look like this figure.

To make both workspaces visible at the same time, drag one workspace tab down until the icon appears in the middle of the dialog box. Continue to drag until the cursor reaches the required quadrant, and then release the mouse button.

To save your Simulink Real-Time Explorer layout, click **File** > **Save Layout**. In a later session, you can click **File** > **Restore Layout** to restore your layout.

Related Examples

- "Create Signal Groups Using Simulink Real-Time Explorer" on page 5-51

More About

- "Signal Group Monitoring Formats" on page 5-12

Signal Group Monitoring Formats

When monitoring a signal group using Simulink Real-Time Explorer, you can change the output format of the group by selecting one of the **Format** options. The monitoring formats are an extension of the options used in C sprint format strings.

Data Type	Digits	Meaning	Example
F	1–7	Decimal float, with from one to seven digits to the right of the decimal point	Decimal 31.5415 with format F7 is 31.5415000.
E	1–7	Scientific notation, with from one to seven digits to the right of the decimal point	Decimal 31.5415 with format E7 is 3.1541500E1.
G	1–7	The shorter of F and E, with from one to seven digits to the right of the decimal point	Decimal 31.5415 with format G7 is 31.5415000.
H	1–7	Hexadecimal integer, one to seven hexadecimal digits wide	Decimal 315 with format H7 is 0x000013B.
B	1–7	Binary integer, one to seven binary digits wide	Decimal 31 with format B7 is 0011111.

Related Examples

·   "Create Signal Groups Using Simulink Real-Time Explorer" on page 5-51

Monitor Stateflow States Using MATLAB Language

You must have already set Stateflow states as test points. If you have not, see "Configure Stateflow States as Test Points" on page 5-8.

1 To get a list of signals in the Command Window, type:

```
tg = slrt
```

2 To display the signals in the real-time application, type:

```
tg.ShowSignals = 'on'
```

The latter causes the Command Window to display a list of the target object properties for the available signals.

For Stateflow states that you have set as test points, the state appears in the `BLOCK NAME` column. For example, if you set a test point for the `first` state of `gear_state` in the `shift_logic` chart of the `old_sf_car` model, the state of interest is `first`. In the list of signals in the MATLAB interface, this state appears as follows:

```
shift_logic:gear_state.first
```

`shift_logic` is the path to the Stateflow chart. `gear_state.first` is the path to the specific state.

More About

·   "Nonobservable Signals" on page 5-136

Animate Stateflow Charts Using Simulink External Mode

The Simulink Real-Time software supports the animation of Stateflow charts in your model to provide visual verification that your chart behaves as expected.

You must be familiar with the use of Stateflow animation. For more information on Stateflow animation, see "Animate Stateflow Charts" in the Stateflow documentation.

1 In the Simulink Editor, select **Simulation** > **Mode** > **External**.

2 Select **Code** > **External Mode Control Panel**.

3 Select **Signal & Triggering**.

4 In the Trigger section of the External Signal & Triggering window:

- Set **Mode** to normal.
- In the **Duration** box, enter 5.
- Select the **Arm when connecting to target** check box.

5 Click **Apply**.

6 Select **Simulation** > **Model Configuration Parameters**.

7 Navigate to the **Simulink Real-Time Options** node.

8 Select the **Enable Stateflow animation** check box.

9 Click **Apply**.

10 Build and download the model to the target computer.

11 On the toolbar, click the **Connect To Target** button .

The current Simulink model parameters are downloaded from the development computer to the real-time application.

12 To start the simulation, click the **Run** button on the toolbar.

The simulation begins to run. You can observe the animation by opening the Stateflow Editor for your model.

13 To stop the simulation, click the **Stop** button on the toolbar.

Note: Enabling the animation of Stateflow charts also displays additional Stateflow information. The Stateflow software requires this information to animate charts. You can disregard this information.

More About

Signal Tracing Basics

Signal tracing acquires signal and time data from a real-time application. While the real-time application is running, you can visualize the data on the target computer using a target scope. You can also upload the data to the development computer and display it using a host scope.

You trace signals using target and host scopes and view them using Simulink Real-Time Explorer, Simulink external mode, MATLAB language, and a web browser interface.

Simulink Real-Time Explorer can display multidimensional signals in column-major format.

Some signals are not observable.

More About

- "Nonobservable Signals" on page 5-136
- "Simulink Real-Time Scope Usage" on page 5-22
- "Target Scope Usage" on page 5-24
- "Host Scope Usage" on page 5-29
- "File Scope Usage" on page 5-77

Configure Real-Time Target Scope Blocks

Simulink Real-Time includes a specialized `Scope` block that you can configure to display signal and time data on the target computer monitor. Add a `Scope` block to the model, select **Scope type** `Target`, and configure the other parameters as described in the following procedure.

- Do not confuse Simulink Real-Time Scope blocks with standard Simulink Scope blocks.

- For more on using Simulink Real-Time Scope blocks, see "Simulink Real-Time Scope Usage" on page 5-22.

- For more on using target scopes, see "Target Scope Usage" on page 5-24.

This procedure starts from the example model `ex_slrt_rt_osc`.

1 In the Command Window, type `ex_slrt_rt_osc`.

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

2 Double-click the block labeled `Scope`.

The Block Parameters: `Scope` dialog box opens. By default, the target scope dialog box is displayed.

3 In the **Scope number** box, a unique number is displayed that identifies the scope. 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 screen on the development or target computers.

4 From the **Scope type** list, select `Target` if it is not already selected. The updated dialog box is displayed.

5 To start the scope automatically when the real-time application executes, select the **Start scope when application starts** check box. The target scope opens automatically on the target computer monitor.

In `Stand Alone` mode, this setting is mandatory, because the development computer is not available to issue a command to start scopes.

6 From the **Scope mode** list, select `Numerical`, `Graphical redraw`, `Graphical sliding`, or `Graphical rolling`.

If you have a scope type of `Target` and a scope mode of `Numerical`, the scope block dialog box adds a **Numerical format** box. You can define the display format for the data. If you choose not to complete the **Numerical format** box, the Simulink Real-Time software displays the signal using the default format of `%15.6f`. This format is a floating-point format without a label.

7 If you select scope mode `Numerical`, in the **Numerical format** box, type a label and associated numeric format type in which to display signals. By default, the entry format is floating-point without a label, `%15.6f`. The **Numerical format** box takes entries of the format:

`'[LabelN] [%width.precision][type] [LabelX]'`

- `LabelN` is the label for the signal. You can use a different label for each signal or the same label for each signal. This argument is optional.

- `width` is the minimum number of characters to offset from the left of the screen or label. This argument is optional.

- `precision` is the maximum number of decimal places for the signal value. This argument is optional.

- `type` is the data type for the signal format. You can use one or more of the following types.

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` is a second label for the signal. You can use a different label for each signal or the same label for each signal. This argument is optional.

Enclose the contents of the **Numerical format** text box in single quotation marks.

For example:

`'Foo %15.2f end'`

For a whole integer signal value, enter 0 for the precision value. For example:

```
'Foo1 %15.0f end'
```

For a line with multiple entries, delimit each entry with a command and enclose the entire string in single quotation marks. For example:

```
'Foo2 %15.6f end,Foo3 %15.6f end2'
```

You can have multiple **Numerical format** entries, separated by a comma. If you enter one entry, that entry applies to each signal (scalar expansion). If you enter N label entries for N+Ksignals, the first N-1 entries apply to the first N-1 signals. The Nth entry is scalar expanded for the remaining K+1 signals. If you have two entries and one signal, the software ignores the second label entry and applies the first entry. You can enter as many format entries as you have signals for the scope. The format string has a maximum length of 100 characters, including spaces, for each signal.

8   To display grid lines on the scope, select the **Grid** check box. This parameter is applicable only for target scopes with scope modes of type Graphical redraw, Graphical sliding, or Graphical rolling.

9   In the **Y-Axis limits** box, enter a row vector with two elements. 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, scaling is set to auto. This parameter is applicable only for target scopes with scope modes of type Graphical redraw, Graphical sliding, or Graphical rolling.

10 In the **Number of samples** box, enter the number of values to be acquired in a data package.

- If you select a **Scope mode** of Graphical redraw, the display redraws the graph every Number of samples.

- If you select a **Scope mode** of Numerical, the block updates the output every Number of samples.

- If you select a **Trigger mode** other than FreeRun, this parameter can specify the Number of samples to be acquired before the next trigger event.

11 In the **Number of pre/post samples** box, enter the number of samples to save or skip. To save N samples before a trigger event, specify the value -N. To skip N samples after a trigger event, specify the value N. The default is 0.

12 In the **Decimation** box, enter a value to indicate when data is collected. The value `1` means that data is collected at each sample time. A value of `2` or greater means that data is collected at less than every sample time.

13 From the **Trigger mode** list, select `FreeRun`.

- If you select `FreeRun` or `Software Triggering`, you do not need to enter additional parameters.

- If you select `Signal Triggering`, then enter the following additional parameters, as required:

  - In the **Trigger signal** box, enter the index of a signal previously added to the scope.

    This parameter does not apply if the **Add signal port to connect a signal trigger source** check box is selected.

  - (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.

  - 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`.

- If you select `Scope Triggering`, then enter the following additional parameters, as required:

  - In the **Trigger scope number** box, enter the scope number of a Scope block. If you use this trigger mode, add a second Scope block to your Simulink model.

  - To trigger one scope on a specific sample of another scope, enter a value in **Sample to trigger on (-1 for end of acquisition)**. The default value of `0` indicates that the triggering scope and the triggered scope start simultaneously.

    For more information on this field, see "Trigger One Scope with Another Scope" on page 11-24.

The target scope dialog box looks like this figure.
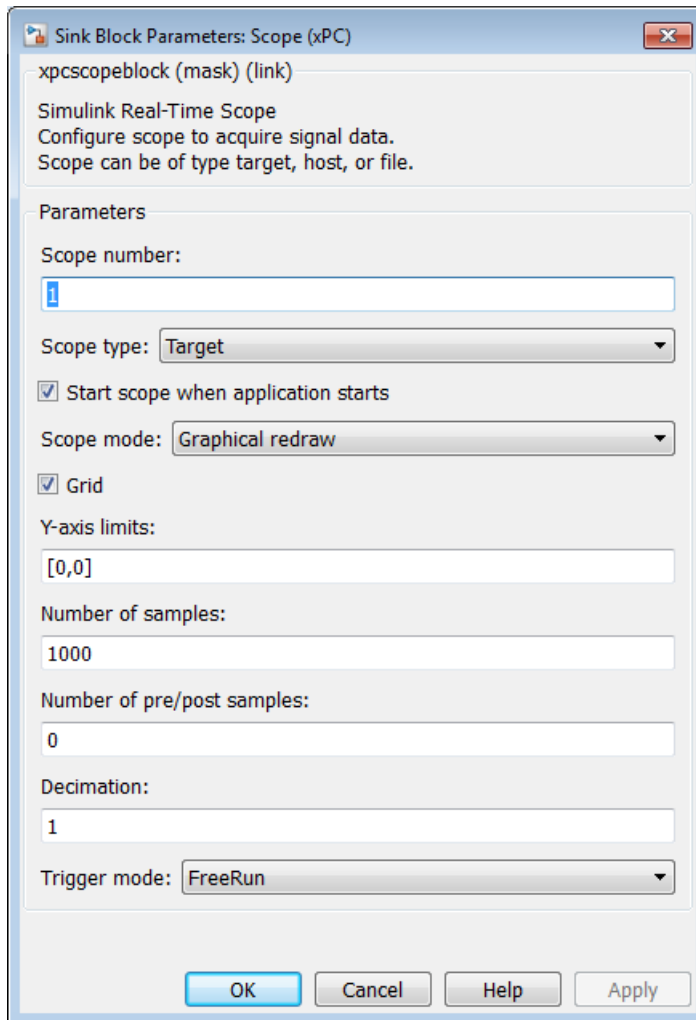
14 Click **OK**.

15 From the **File** menu, click **Save As**. Save the model as `ex_slrt_target_osc`.

Simulink Real-Time Scope Usage

For usage issues common to all Simulink Real-Time `Scope` types:

- To monitor an output signal from a `Constant` block by connecting it to a Simulink Real-Time `Scope` block, add a test point for the `Constant` block output signal.

- You can add a Simulink Real-Time `Scope` block only to the topmost model, not to a referenced model. To log signals from referenced models, use Simulink Real-Time Explorer scopes or Simulink Real-Time language scope objects.

- When you build and download the real-time application, the Simulink Real-Time kernel creates a scope representing the real-time `Scope` block. You can change the `Scope` parameters after building the real-time application or while it is running. To change the parameters, assign the scope to a MATLAB variable using the target object method `SimulinkRealTime.target.getscope`. You can use `SimulinkRealTime.target.getscope` to remove a scope created during the build and download process. The Simulink Real-Time kernel recreates the scope when you restart the real-time application.

- If the output of a `Mux` block is connected to the input of a Simulink Real-Time Scope block, the signal is not observable. To observe the signal, add a unity gain block (a `Gain` block with a gain of `1.0`) between the `Mux` block and the Simulink Real-Time `Scope` block.

- You can pass vector signals into a Simulink Real-Time `Scope` block. The real-time application interprets the vector as a series of individual signals. However, you cannot pass a matrix signal into a `Scope` block. Doing so results in a build error. To display a matrix signal, pass it to a `Reshape` block and pass the resulting vector into the `Scope` block.

- 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.

See Also
`Gain` | `Mux` | `Reshape` | `SimulinkRealTime.target.getscope`

More About
- "Nonobservable Signals" on page 5-136
- "Target Scope Usage" on page 5-24

Target Scope Usage

For usage issues unique to target scopes:

- There can be no more than nine target scopes in a model, whether created by using a real-time `Scope` block or using the run-time interface. Each target scope can contain up to 10 signals.

- The combined number of target scopes and `Video Display` blocks in the model cannot exceed nine.

- With one graphical target scope active on the target computer, the graphical and numerical formats are displayed. With more than one target scope active, only the format that the **Scope mode** parameter specifies is displayed.

- For a target scope, logged data (`sc.Data` and `sc.Time`) is not accessible over the command-line interface on the development computer. Logged data is accessible only when the scope object status (`sc.Status`) is set to `Finished`. When the scope completes one data cycle (time to collect the number of samples), the scope engine restarts the scope instead of setting `sc.Status` to `Finished`.

  If you create a scope object, for example, `sc = getscope(tg,1)` for a target scope, you cannot get the logged data by typing `sc.Data`. Instead, you get an error message:

  ```
  Scope # 1 is of type 'Target'! Property Data

      is not accessible.
  ```

  To view data on the development computer while the data is being displayed on the target computer, define a second scope object with type `host`. Then synchronize the acquisitions of the two scope objects by setting `TriggerMode` for the second scope to `'Scope'`.

- To display the target scope image in a display window on the development computer screen, use `SimulinkRealTime.target.viewTargetScreen`.

  To save the target scope image to a file, right-click in the display window and then click **Save as image**.

See Also
`SimulinkRealTime.target.getscope` |
`SimulinkRealTime.target.viewTargetScreen` | Video Display

Related Examples

- "Configure Real-Time Target Scope Blocks" on page 5-17
- "Create Target Scopes Using Simulink Real-Time Explorer" on page 5-30

More About

- "Simulink Real-Time Scope Usage" on page 5-22

Configure Real-Time Host Scope Blocks

Simulink Real-Time includes a specialized real-time `Scope` block that you can configure to display signal and time data on the development computer monitor. Add a `Scope` block to the model, select **Scope type** `Host`, and configure the other parameters as described in the following procedure.

*   Do not confuse Simulink Real-Time Scope blocks with standard Simulink Scope blocks.
*   For more on using Simulink Real-Time Scope blocks, see "Simulink Real-Time Scope Usage" on page 5-22.
*   For more on host scopes, see "Host Scope Usage" on page 5-29.

This procedure starts from the example model `ex_slrt_rt_osc`.

1   In the Command Window, type `ex_slrt_rt_osc`.

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

2   Double-click the block labeled **Scope**.

    The Block Parameters: `Scope` dialog box opens. By default, the target scope dialog box is displayed.

3   In the **Scope number** box, a unique number is displayed that identifies the scope. This number is incremented each time that you add a Simulink Real-Time scope.

    This number identifies the Simulink Real-Time Scope block and the scope screen on the development or target computers.

4   From the **Scope type** list, select `Host`. The updated dialog box is displayed.

5   To start the scope automatically when the real-time application executes, select the **Start scope when application starts** check box. You can then open a host scope viewer from Simulink Real-Time Explorer.

    In `Stand Alone` mode, this setting is mandatory because the development computer is not available to issue a command to start scopes.

6   In the **Number of samples** box, enter the number of values to be acquired in a data package.

7   In the **Number of pre/post samples** box, enter the number of samples to save or skip. To save `N` samples before a trigger event, specify the value `-N`. To skip `N` samples after a trigger event, specify the value `N`. The default is `0`.

8 In the **Decimation** box, enter a value to indicate when data is collected. The value `1` means that data is collected at each sample time. A value of `2` or greater means that data is collected at less than every sample time.

9 From the **Trigger mode** list, select `FreeRun`.

- If you select `FreeRun` or `Software Triggering`, you do not need to enter additional parameters.

- If you select `Signal Triggering`, then enter the following additional parameters, as required:

  - In the **Trigger signal** box, enter the index of a signal previously added to the scope.

    This parameter does not apply if the **Add signal port to connect a signal trigger source** check box is selected.

  - (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.

  - 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`.

- If you select `Scope Triggering`, then enter the following additional parameters, as required:

  - In the **Trigger scope number** box, enter the scope number of a Scope block. If you use this trigger mode, add a second Scope block to your Simulink model.

  - To trigger one scope on a specific sample of another scope, enter a value in **Sample to trigger on (-1 for end of acquisition)**. The default value of `0` indicates that the triggering scope and the triggered scope start simultaneously.

    For more information on this field, see "Trigger One Scope with Another Scope" on page 11-24.

The host scope dialog box looks like this figure.
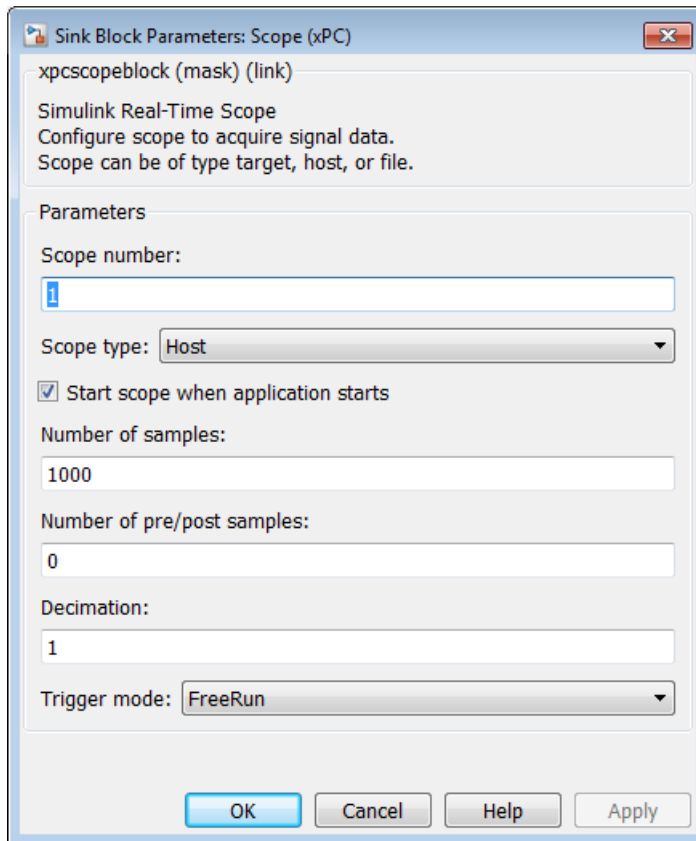
10 Click **OK**.

11 From the **File** menu, click **Save As**. Save the model as `ex_slrt_host_osc`.

Host Scope Usage

For usage issues unique to host scopes:

- Simulink Real-Time supports as many host scopes as the target computer resources can support. Each host scope can contain as many signals as the target computer resources can support.

- Use host scopes to log signal data triggered by an event while your real-time application is running. The host scope acquires the first N samples into a buffer. You can retrieve this buffer into the scope object property `sc.Data`. The scope then stops and waits for you to restart the scope manually.

  The number of samples N to log after triggering an event is equal to the value that you entered in the **Number of samples** parameter.

  Select the type of trigger event in the Block Parameters: `Scope` dialog box by setting **Trigger Mode** to `Signal Triggering`, `Software Triggering`, or `Scope Triggering`.

Related Examples

More About

Create Target Scopes Using Simulink Real-Time Explorer

You can create a virtual target scope on the target computer using Simulink Real-Time Explorer. These scopes have the full capabilities of the `Scope` block in `Target` mode, but do not persist past the current execution.

Note: For information on using target scope blocks, see "Configure Real-Time Target Scope Blocks" on page 5-17 and "Target Scope Usage" on page 5-24.

This procedure uses the model `xpcosc` as an example. You must have already completed the following setup:

1   Built and downloaded the real-time application to the target computer using Simulink (on the toolbar).

2   Run Simulink Real-Time Explorer (**Tools** > **Simulink Real-Time**).

3   Connected to the target computer in the **Targets** pane (on the toolbar).

4   Set property **Stop time** to `inf` in the **Applications** pane (on the toolbar).

To configure a virtual target scope:

1   In the **Scopes** pane, expand the **xpcosc** node.

2   To add a target scope, select **Target Scopes** and then click the **Add Scope** button on the toolbar.

The new scope appears under node **Target Scopes**, for example **Scope 1**.

3   Select **Scope 1** and then click the **Properties** button on the toolbar.

4   In the **Scope Properties** workspace, click **Signals**. You add signals from the **Applications** Signals workspace.

5   In the **Applications** pane, expand the real-time application node and then node **Model Hierarchy**.

6   Select the model node and then click the **View Signals** button on the toolbar.

The Signals workspace opens, showing a table of signals with properties and actions.

7   In the Signals workspace, to add signal `Signal Generator` to **Scope1**, drag signal `Signal Generator` to the **Scope1** properties workspace.

8 Add signal `Integrator1` to **Scope 1** in the same way.

The dialog box looks like this figure.
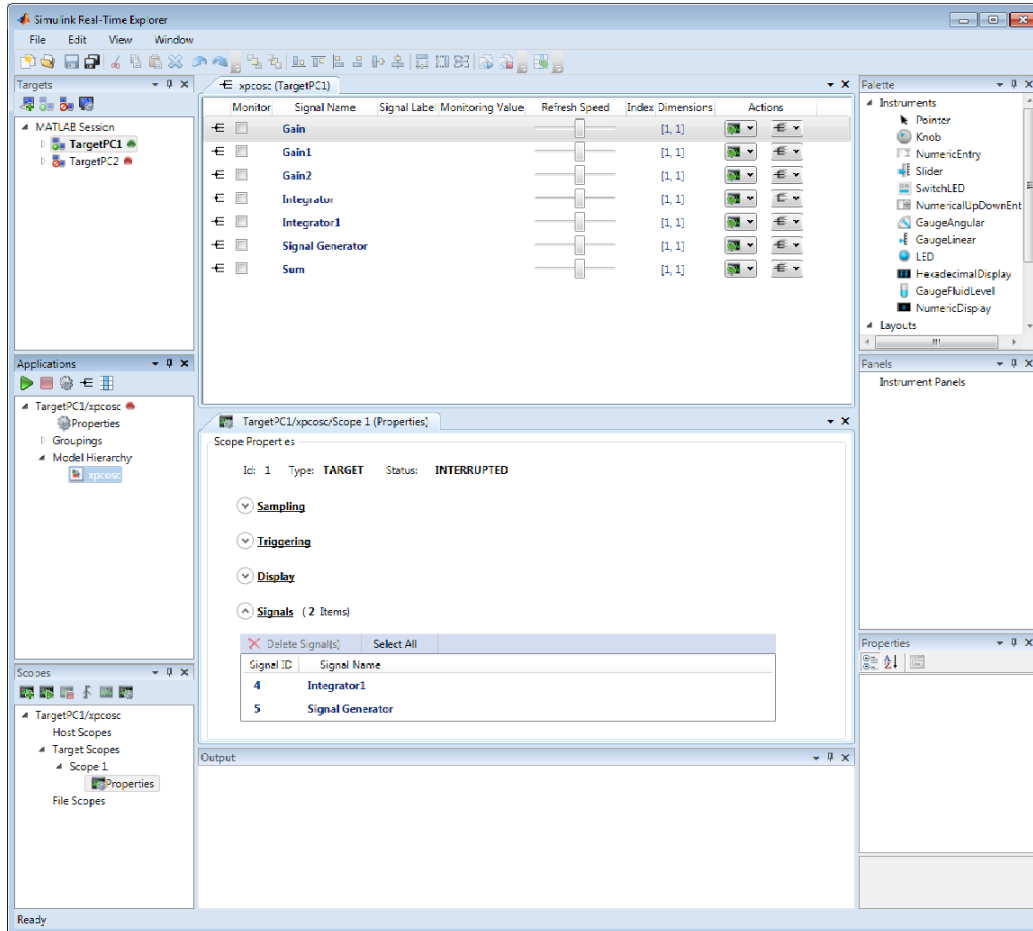
9 To start execution, click the real-time application and then click the **Start** button ▶ on the toolbar.

The application starts running. No output appears on the target computer monitor.

10 To start **Scope 1**, click **Scope 1** in the **Scopes** pane and then click the **Start Scope** button on the toolbar.

Output for signals `Signal Generator` and `Integrator1` appears on the target computer monitor.

11 To stop **Scope 1**, click **Scope 1** in the **Scopes** pane and then click the **Stop Scope** button on the toolbar.

The signals shown on the target computer stop updating while the real-time application continues running. The target computer monitor displays a message like this message:

```
Scope: 1, set to state 'interrupted'
```

12 To stop execution, click the real-time application and then click the **Stop** button on the toolbar.

The real-time application on the target computer stops running, and the target computer displays messages like these messages:

```
minimal TET: 0.0000006 at time 0.001250
```

```
maximal TET: 0.0000013 at time 75.405500
```

The target computer screen looks like this figure.
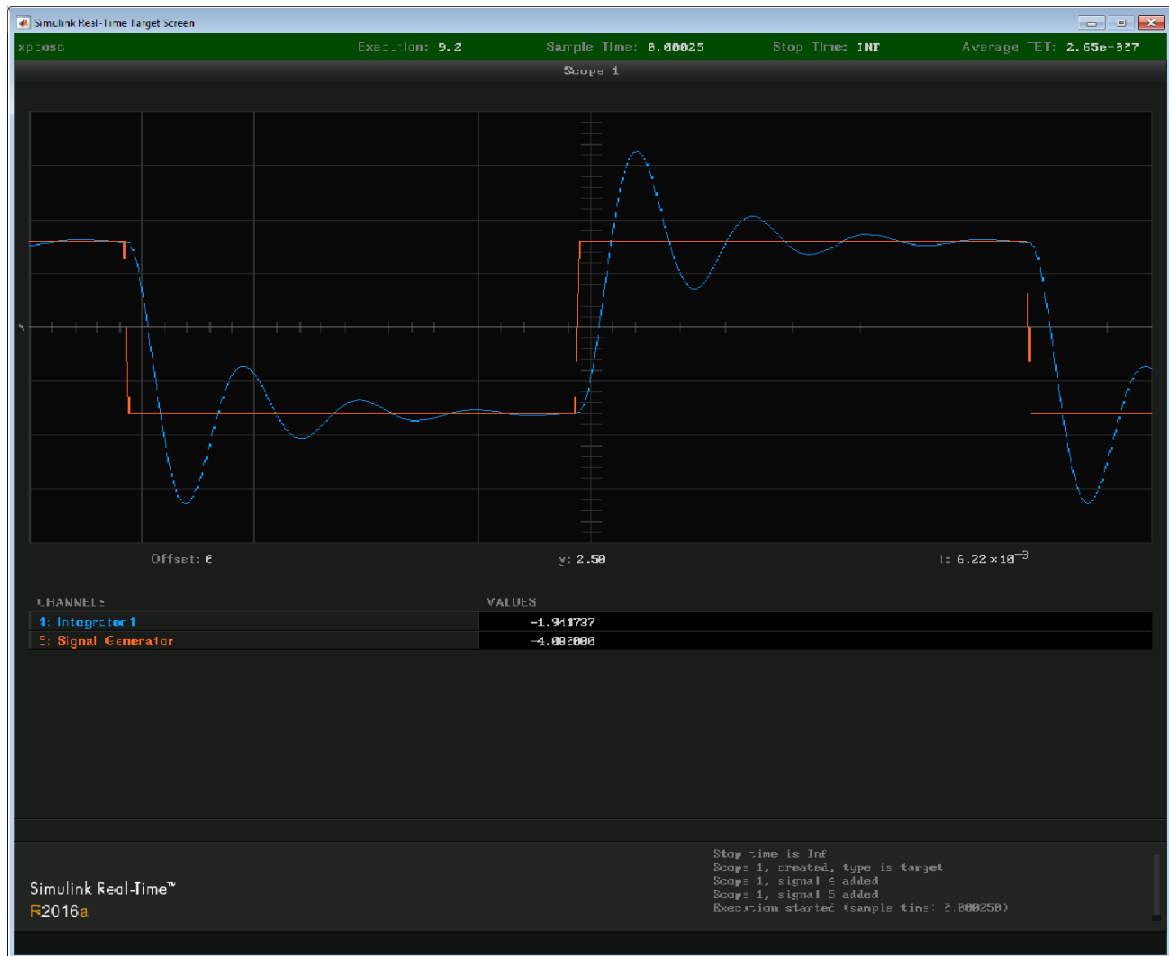
You can create a virtual target scope from the scope types list by clicking **Add Scope** next to scope type **Target Scopes**. You can add or remove signals from a virtual target scope while the scope is either stopped or running.

To make both workspaces visible at the same time, drag one workspace tab down until the ⊞ icon appears in the middle of the dialog box. Continue to drag until the cursor reaches the required quadrant, and then release the mouse button.

To save your Simulink Real-Time Explorer layout, click **File** > **Save Layout**. In a later session, you can click **File** > **Restore Layout** to restore your layout.

See Also

`SimulinkRealTime.target.viewTargetScreen`

Related Examples

- "Create Signal Groups Using Simulink Real-Time Explorer" on page 5-51
- "Configure Target Scopes Using Simulink Real-Time Explorer" on page 5-47
- "Configure Scope Sampling Using Simulink Real-Time Explorer" on page 5-35
- "Trigger Scopes Interactively Using Simulink Real-Time Explorer" on page 5-38
- "Trigger Scopes Noninteractively Using Simulink Real-Time Explorer" on page 5-41

Configure Scope Sampling Using Simulink Real-Time Explorer

You can customize sampling for Simulink Real-Time scopes to facilitate data access to the running model. You can configure sampling whether you added a Scope block to the model or added the scope at run time.

This procedure uses the model xpcosc as an example. You must have already completed the procedure in "Create Target Scopes Using Simulink Real-Time Explorer" on page 5-30. Target execution and scopes must be stopped.

1   Select **Scope 1** and open the Properties pane (on the **Scopes** toolbar).

2   In the **Scope 1** properties pane, click **Sampling**.

3   In the **Number of Samples** box, enter the number of values to be acquired in a data package, here 250.

    If you select a **Display mode** of Graphical redraw, the display redraws the graph every Number of Samples.

    If you select a **Display mode** of Numerical, the block updates the output every Number of Samples.

    If you select a **Trigger Mode** other than FreeRun, this parameter can specify the number of samples to be acquired before the next trigger event.

4   In the **Decimation** box, enter 10 to indicate that data must be collected at every 10th sample time. The default is 1, to collect data at every sample time.

5   In the **Number of pre/post samples** box, enter the number of samples to save or skip. To save N samples before a trigger event, specify the value -N. To skip N samples after a trigger event, specify the value N. The default is 0.

    The dialog box looks like this figure.
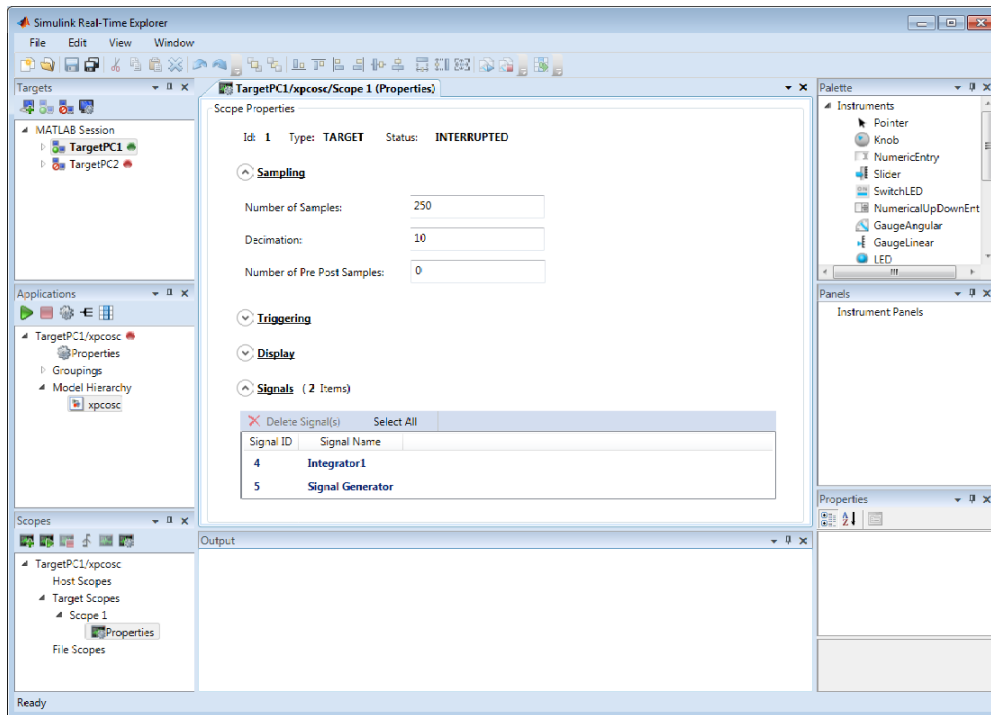
6   Start execution (▶ on the **Applications** toolbar).

7   Start **Scope 1** (on the toolbar).

Output for signals `Signal Generator` and `Integrator1` appears on the target computer monitor.
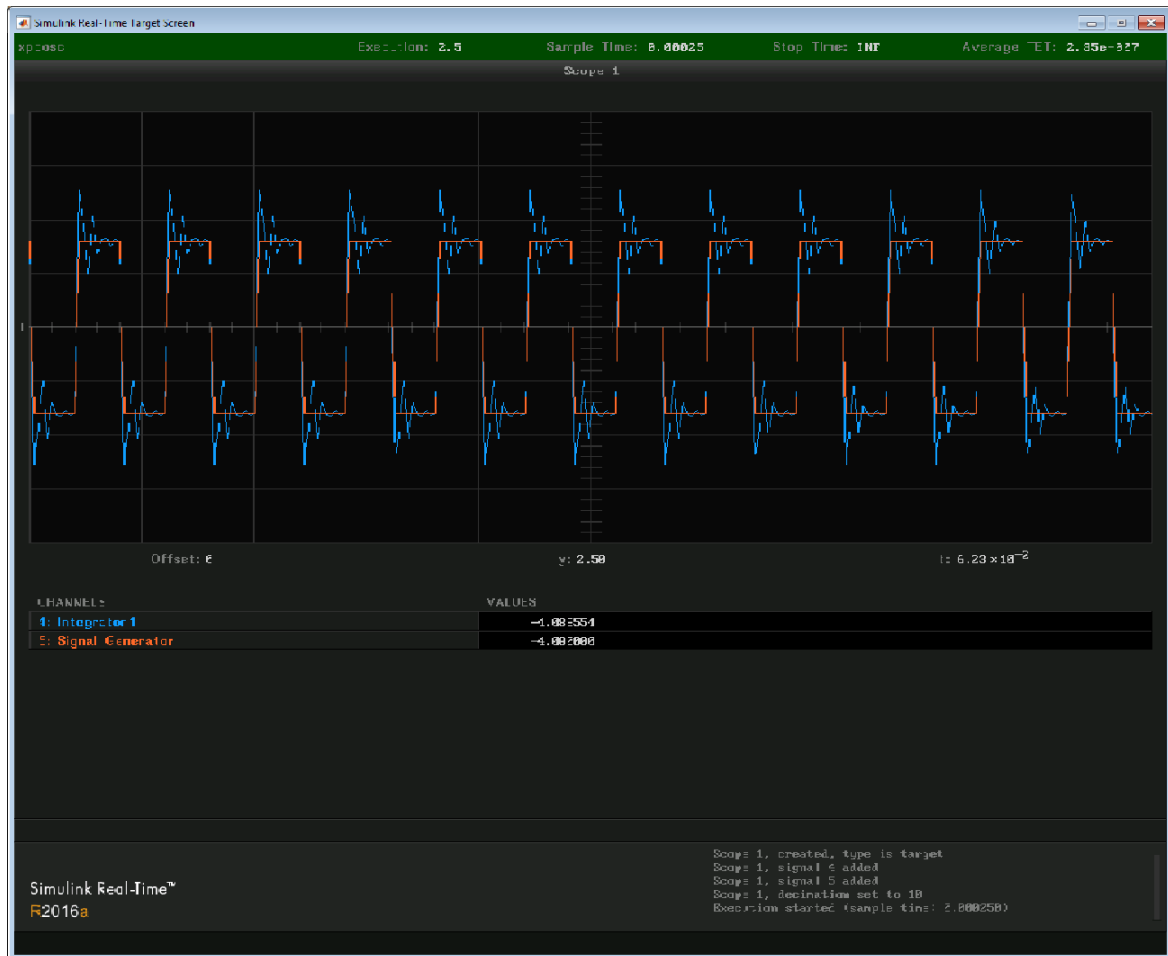
8   Stop **Scope 1** (on the toolbar).

9   Stop execution (on the **Applications** toolbar).

Related Examples

- "Trigger Scopes Interactively Using Simulink Real-Time Explorer" on page 5-38
- "Trigger Scopes Noninteractively Using Simulink Real-Time Explorer" on page 5-41

Trigger Scopes Interactively Using Simulink Real-Time Explorer

To facilitate your interaction with the running model, you can customize scope triggering for Simulink Real-Time scopes. You can configure triggering whether you created the scope by adding a Scope block to the model or by adding the scope at run time.

The following procedures use the model xpcosc as an example. You must have already completed the procedure in "Create Target Scopes Using Simulink Real-Time Explorer" on page 5-30. Target execution and scopes must be stopped.

In this section...
"Freerun Triggering" on page 5-38
"Software Triggering" on page 5-38

Freerun Triggering

In **Trigger Mode** Freerun, the scope triggers automatically when it is started. It displays data until it is stopped. By default, **Trigger Mode** is set to Freerun.

1   Start execution (on the **Applications** toolbar).

2   Select **Scope 1** and open the Properties pane (on the **Scopes** toolbar).

3   In the **Scope 1** pane, click **Triggering**.

4   Select **Trigger Mode** Freerun.

5   Start and stop **Scope 1** (and on the toolbar).

    Signal data is displayed on the target computer monitor when the scope starts and stops when the scope stops.

6   Stop execution (on the **Applications** toolbar).

Software Triggering

In **Trigger Mode** Software, the scope triggers when you select **Scope 1** and then click the **Trigger** button on the toolbar.

1 Start execution (▶ on the **Applications** toolbar).

2 Select **Trigger Mode** `Software`.

3 Start **Scope 1** (⬛ on the toolbar).

The **Trigger** button 🔽 is enabled on the toolbar.

4 Click the **Trigger** 🔽 button on the **Scopes** toolbar.

The current signal data is displayed on the target computer monitor when you click the button.

5 Stop **Scope 1** (⬛ on the toolbar).

The dialog box looks like this figure.
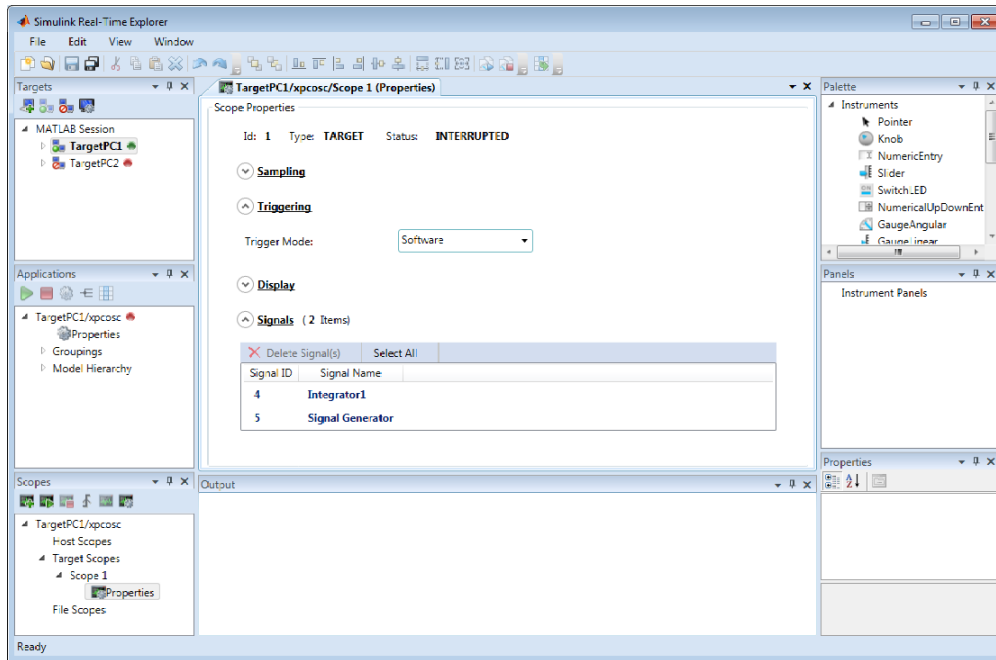
The target monitor looks like this figure.
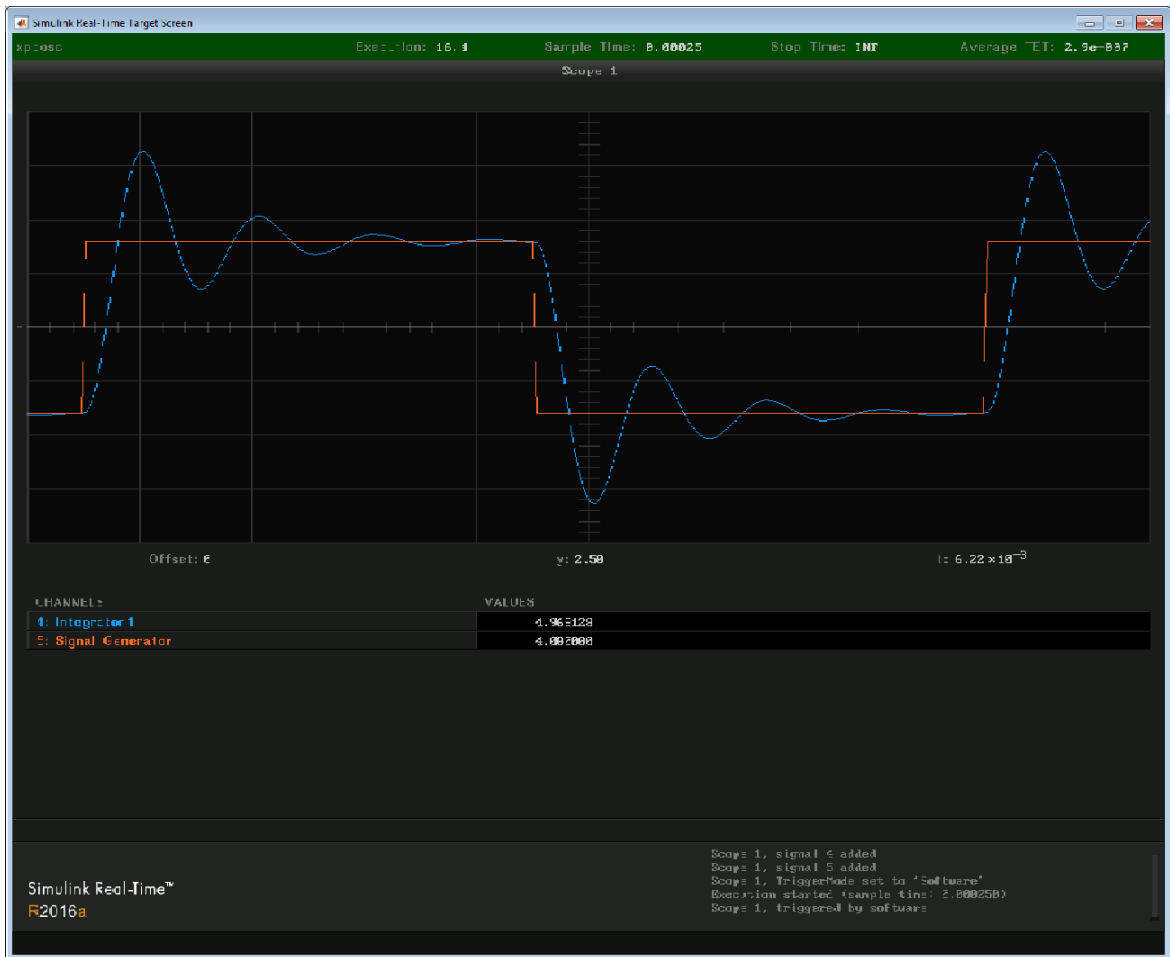
6 Stop execution (on the **Applications** toolbar).

Trigger Scopes Noninteractively Using Simulink Real-Time Explorer

To facilitate your control of the running model, you can customize scope triggering for Simulink Real-Time scopes. You can configure triggering whether you added a `Scope` block to the model or added the scope at run time.

The following procedures use the model `xpcosc` as an example. You must have already completed the procedure in "Create Target Scopes Using Simulink Real-Time Explorer" on page 5-30. Target execution and scopes must be stopped.

In this section...
"Signal Triggering" on page 5-41
"Scope Triggering" on page 5-43

Signal Triggering

In **Trigger Mode** `Signal`, the scope triggers when a signal rises or falls through a specified level.

1   Start execution (▶ on the **Applications** toolbar).

2   Select **Scope 1** and open the Properties pane (🖼 on the **Scopes** toolbar).

3   In the **Scope 1** pane, click **Triggering**.

4   Select **Trigger Mode** `Signal`.

Settings **Trigger Signal**, **Trigger Slope**, and **Trigger Level** appear.

5   Type the number displayed on the target computer screen for `Signal Generator` (here, 5) in the **Trigger Signal** text box.

6   Set **Trigger Slope** to `Rising`.

7   Leave **Trigger Level** as 0, indicating that the signal crosses 0 before **Scope 1** triggers.
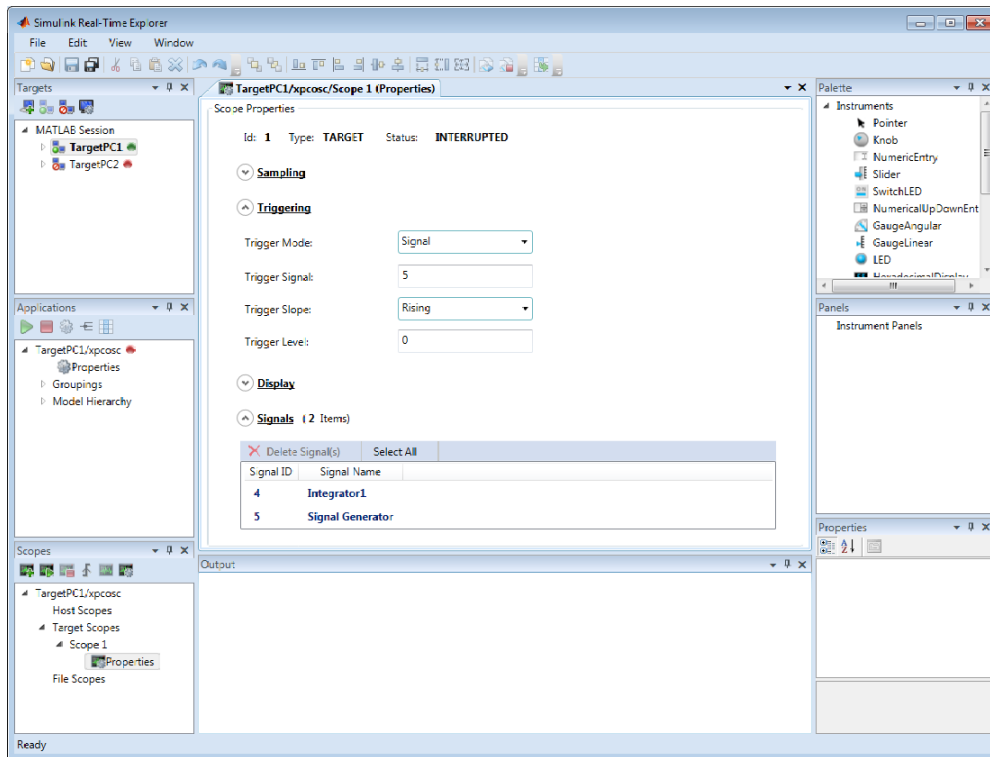
8 Start **Scope 1** (on the toolbar).

Signal data is displayed on the target computer monitor, with the rising pulse of `Signal Generator` just beyond the left side.
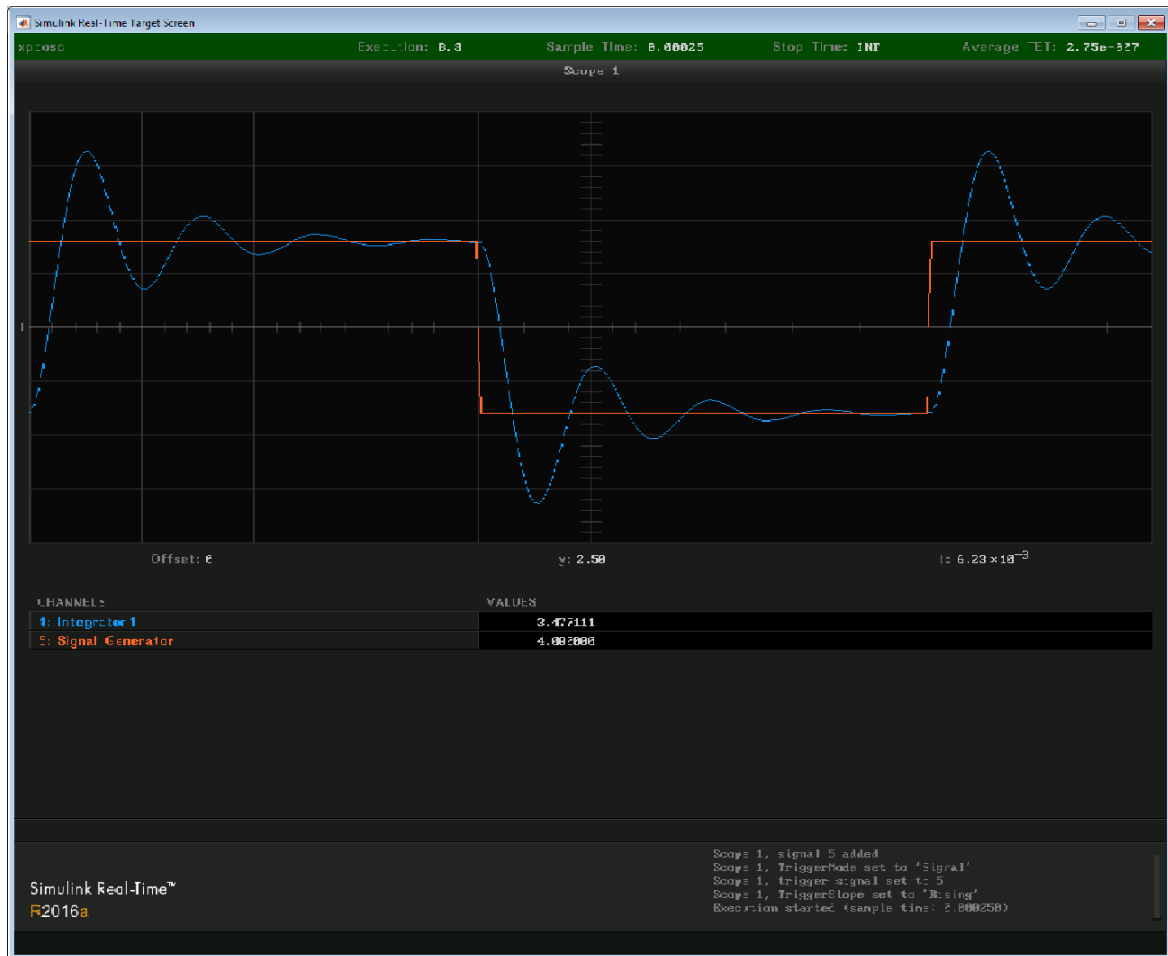
9 Stop **Scope 1** (![icon] on the toolbar).

10 Stop execution (![icon] on the **Applications** toolbar).

Scope Triggering

In **Trigger Mode** `Scope`, the scope triggers when another scope triggers. In this example, Scope 2 triggers when signal-triggered Scope 1 triggers.

1 Start execution (▶ on the **Applications** toolbar).

2 Add scope **Scope 2** (⊞ on the **Scopes** toolbar).

3 Open the Signals pane (⊢ on the **Applications** toolbar).

4 Add signal `Integrator` to **Scope 2** in the **Signals** pane.

5 In the **Scope 2** pane, click **Triggering**.

6 Select **Trigger Mode** `Scope`.

   Settings **Trigger scope** and **Trigger scope sample** appear.

7 Set **Trigger scope** to `1`. Press **Enter**. **Scope 2** then triggers when **Scope 1** triggers.

8 Leave **Trigger scope sample** set to `0`. **Scope 2** triggers on the same sample as **Scope 1**.
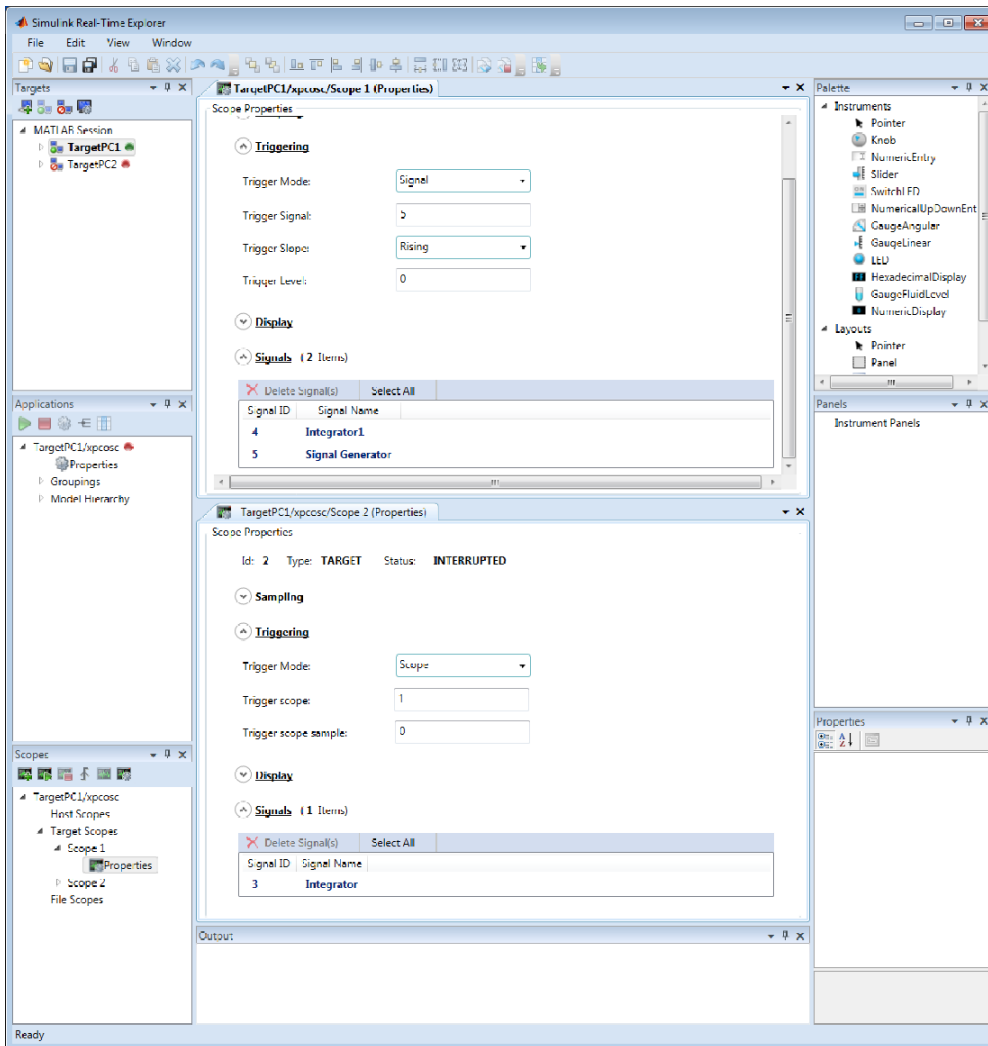
9 Explicitly start both **Scope 1** and **Scope 2** (on the toolbar).

Scope 1 and **Scope 2** display signal data on the target computer monitor.
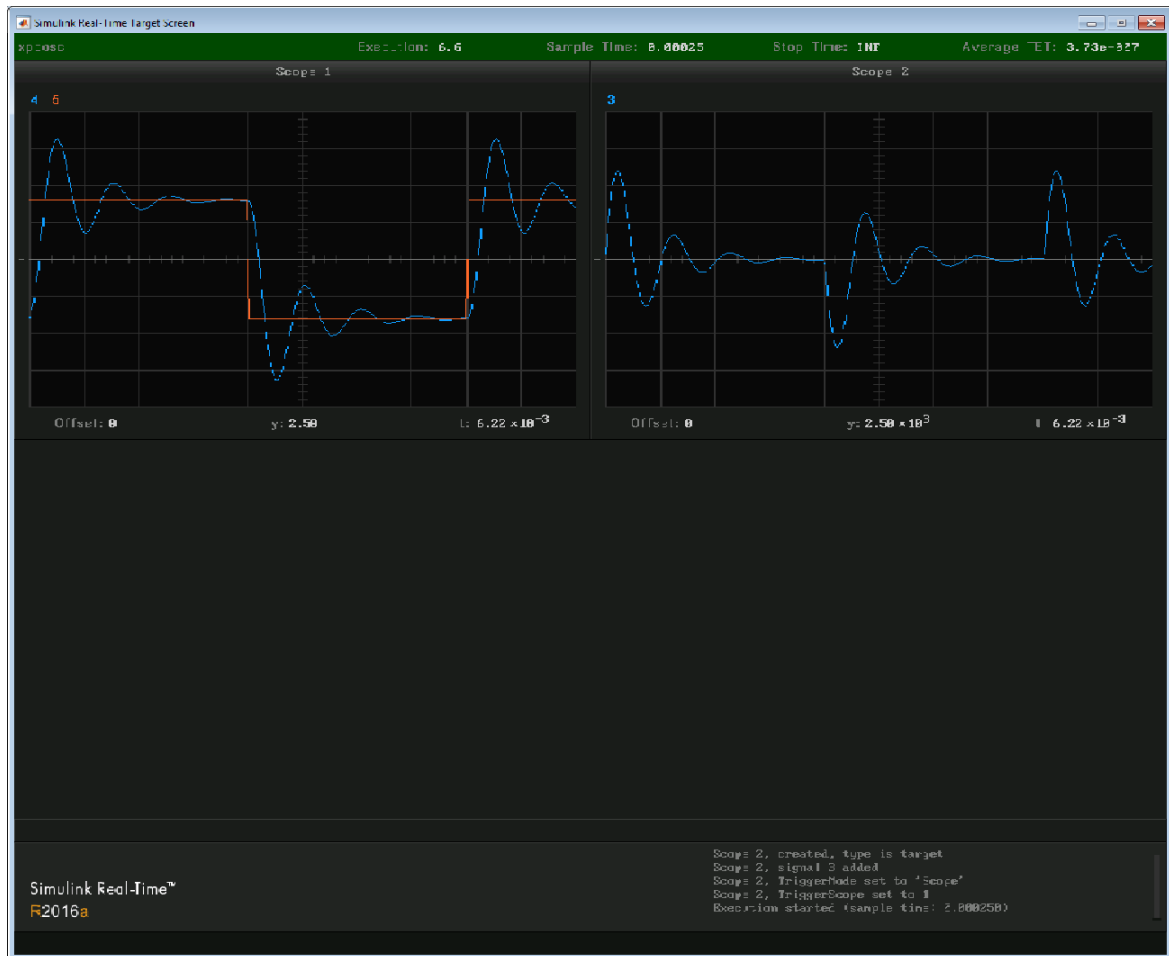
10 Explicitly stop both **Scope 1** and **Scope 2** (🖳 on the toolbar).

11 Stop execution (🔴 on the **Applications** toolbar).

Configure Target Scopes Using Simulink Real-Time Explorer

To facilitate your view of the signal data, you can configure the target scope display. You can configure the display whether you added a `Scope` block to the model or added the scope at run time.

This procedure uses the model `xpcosc` as an example. You must have already completed the procedure in "Create Target Scopes Using Simulink Real-Time Explorer" on page 5-30. Target execution and scopes must be stopped.

1   Start execution (▶ on the **Applications** toolbar).

2   Select **Scope 1** and open the Properties pane (⬛ on the **Scopes** toolbar).

3   In the **Scope 1** pane, click **Display**.

4   Select **Display mode** `Redraw` and then click in the **Y-Limits** box.

    This value is the default. It causes the scope display to redraw when it has acquired as many samples as specified in **Number of Samples**.

5   Start **Scope 1** (⬛ on the toolbar).

    Signal data is displayed on the target computer monitor, appearing to move to the left.

6   Enter `[0,10]` in the **Y-Limits** box and then press **Enter**. The default setting is `[0,0]`, which automatically scales the output according to the signal values.

    The display changes to show only values at and above the zero line.

7   Clear the **Grid (On/Off)** check box. By default, the box is selected.
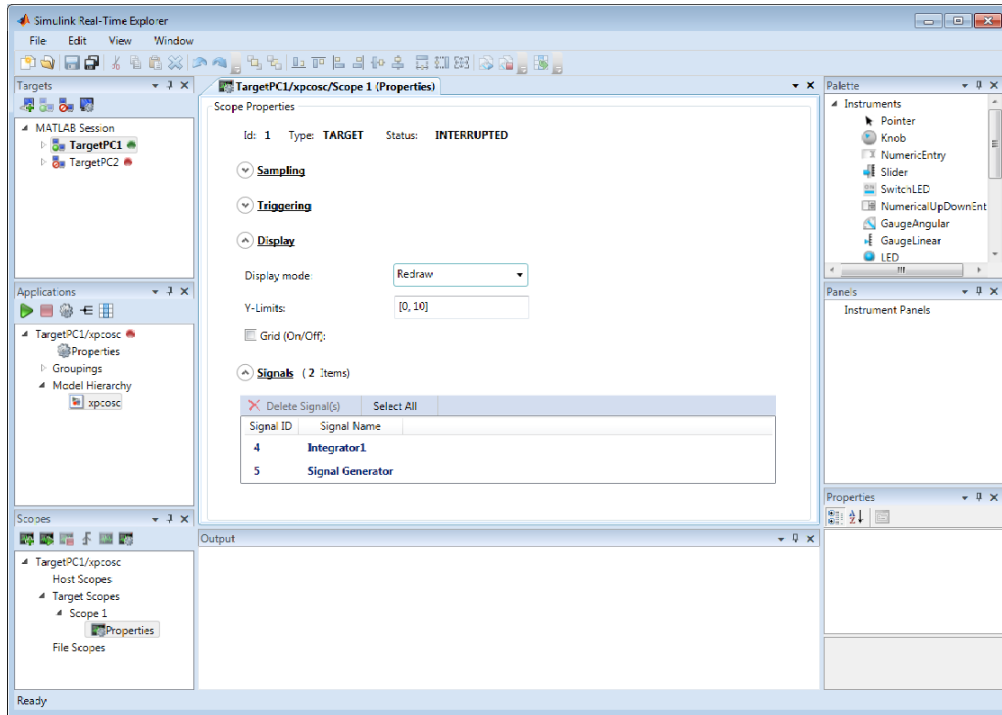
The target computer monitor looks like this figure.
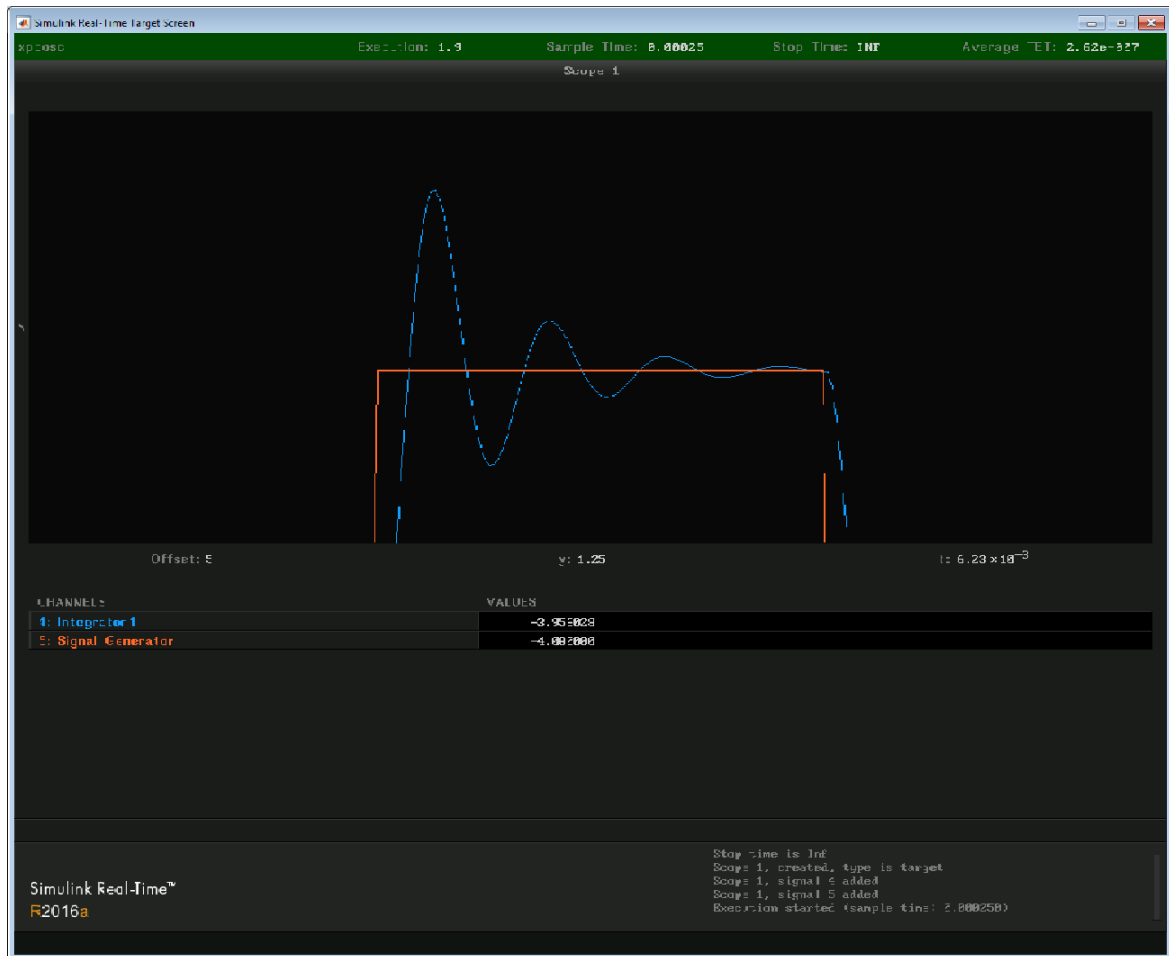
8 Select **Display mode** `Numerical` and then click in the **Y-Limits** box.

The grid and axes disappear. The target computer monitor displays the signals, color coded, in the default format of `%15.6f` (a floating-point format without a label).

9 Select **Display mode** `Rolling` and then click in the **Y-Limits** box.

The display changes to a display that continuously moves a window along the signal stream. New data enters the display from the right and then moves toward the left.

10 Select **Display mode** `Sliding` and then click in the **Y-Limits** box. In this mode, the scope refreshes continuously. New data overwrites the display from the left toward the right.

11 Stop **Scope 1** (on the toolbar).

12 Stop execution (on the **Applications** toolbar).

See Also

`SimulinkRealTime.target.viewTargetScreen`

Create Signal Groups Using Simulink Real-Time Explorer

When testing a complex model with many signals, you frequently must select signals for tracing or monitoring from multiple parts and levels of the model hierarchy. You can make this task easier by using Simulink Real-Time Explorer to create a signal group and save it to disk.

This procedure uses the model xpcosc as an example. You must have already completed the following setup:

1. Built and downloaded the real-time application to the target computer using Simulink (🛠 on the toolbar).

2. Run Simulink Real-Time Explorer (**Tools** > **Simulink Real-Time**).

3. Connected to the target computer in the **Targets** pane (🔵 on the toolbar).

To create a signal group:

1. In the **Applications** pane, expand the real-time application node and right-click node **Groupings**.

2. Click **New Signal Group**.

    The Add New Signal Group Item dialog box appears.

3. In the Add New Signal Group Item dialog box, enter a name in the **Name** text box, for example **signalgroup1.sig**. In the **Location** text box, enter a folder for the group file.

4. Click **OK**.

    A new signal group appears, along with its Signal Group workspace.

5. In the **Applications** pane, expand the real-time application node and then expand node **Model Hierarchy**.

6. Select the model node and then click the **View Signals** button 🗏 on the toolbar.

    The Signals workspace opens, showing a table of signals with properties and actions.

7. In the Signals workspace, to add signal Signal Generator to **signalgroup1.sig**, drag signal Signal Generator to the **signalgroup1.sig** properties workspace.

8. Add signal Integrator1 to **signalgroup1.sig** in the same way.

9. Press **Enter**, and then click the Save button 💾 on the toolbar.

When you are monitoring a signal group, you can change the output format of the group by selecting one of the options in the **Format** column. See"Signal Group Monitoring Formats" on page 5-12.
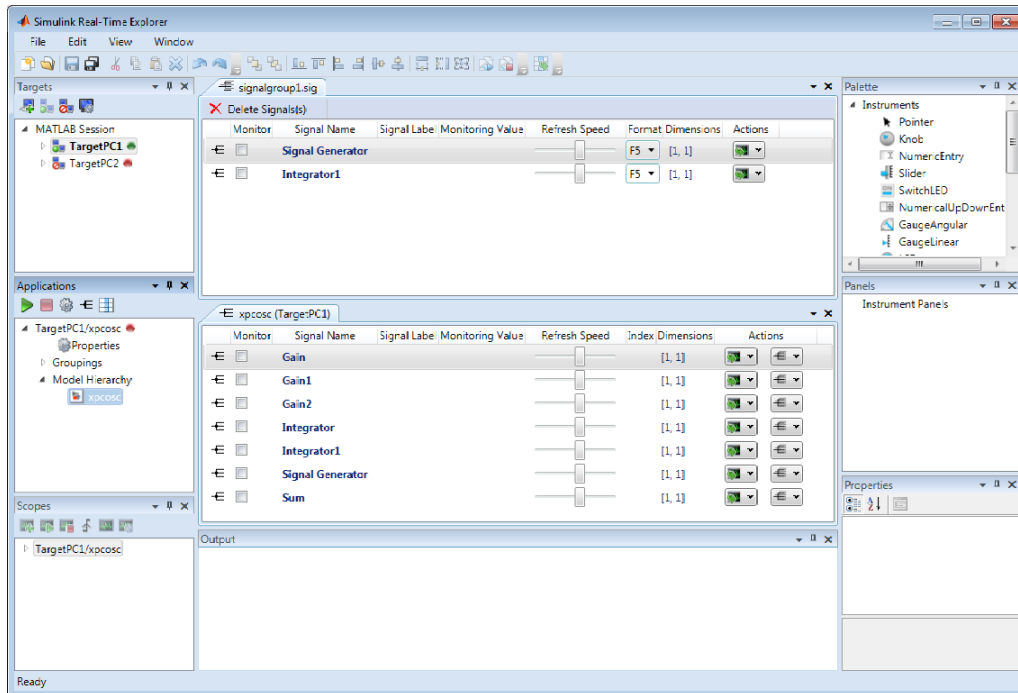
Signals are defined within a particular real-time application. To open a signal group from the **File** > **Open** > **Group** menu, you must first select an application.

To remove signals from the signal group, select the signal items in the group list and click **Delete Signals**.

To remove the signal group, navigate to the signal group under **Groupings** > **Signals**, right-click the signal group, and click **Remove**.

To make both workspaces visible at the same time, drag one workspace tab down until the icon appears in the middle of the dialog box. Continue to drag until the cursor reaches the required quadrant, and then release the mouse button.

To save your Simulink Real-Time Explorer layout, click **File** > **Save Layout**. In a later session, you can click **File** > **Restore Layout** to restore your layout.

Related Examples

- "Monitor Signals Using Simulink Real-Time Explorer" on page 5-4
- "Create Target Scopes Using Simulink Real-Time Explorer" on page 5-30
- "Create Host Scopes Using Simulink Real-Time Explorer" on page 5-54
- "Create File Scopes Using Simulink Real-Time Explorer" on page 5-80

Create Host Scopes Using Simulink Real-Time Explorer

You can create a virtual host scope on the target computer using Simulink Real-Time Explorer. These scopes have the full capabilities of the `Scope` block in `Host` mode, but do not persist past the current execution.

Note: For information on using host scope blocks, see "Configure Real-Time Host Scope Blocks" on page 5-26 and "Host Scope Usage" on page 5-29.

This procedure uses the model `xpcosc` as an example. You must have already completed the following setup:

1 Built and downloaded the real-time application to the target computer using Simulink (📅 on the toolbar).

2 Run Simulink Real-Time Explorer (**Tools** > **Simulink Real-Time**).

3 Connected to the target computer in the **Targets** pane (🔵 on the toolbar).

4 Set property **Stop time** to `inf` in the **Applications** pane (⚙ on the toolbar).

To configure a virtual host scope:

1 In the **Scopes** pane, expand the **xpcosc** node.

2 To add a host scope, select **Host Scopes** and then click the **Add Scope** button 🔲 on the toolbar.

   Under the **Host Scopes** node, the new scope is displayed, for example **Scope 1**.

3 Expand **Scope 1** and then click the **Properties** button 🔲 on the toolbar.

4 In the **Scope Properties** pane, click **Signals**. Add signals from the **Applications** Signals workspace.

5 In the **Applications** pane, expand the real-time application node and then node **Model Hierarchy**.

6 Select the model node and then click the **View Signals** button ⊨ on the toolbar.

7 In the Signals workspace, to add signal `Signal Generator` to **Scope1**, drag signal `Signal Generator` to the **Scope1** properties workspace.

   Add signal `Integrator1` to **Scope 1** in the same way.
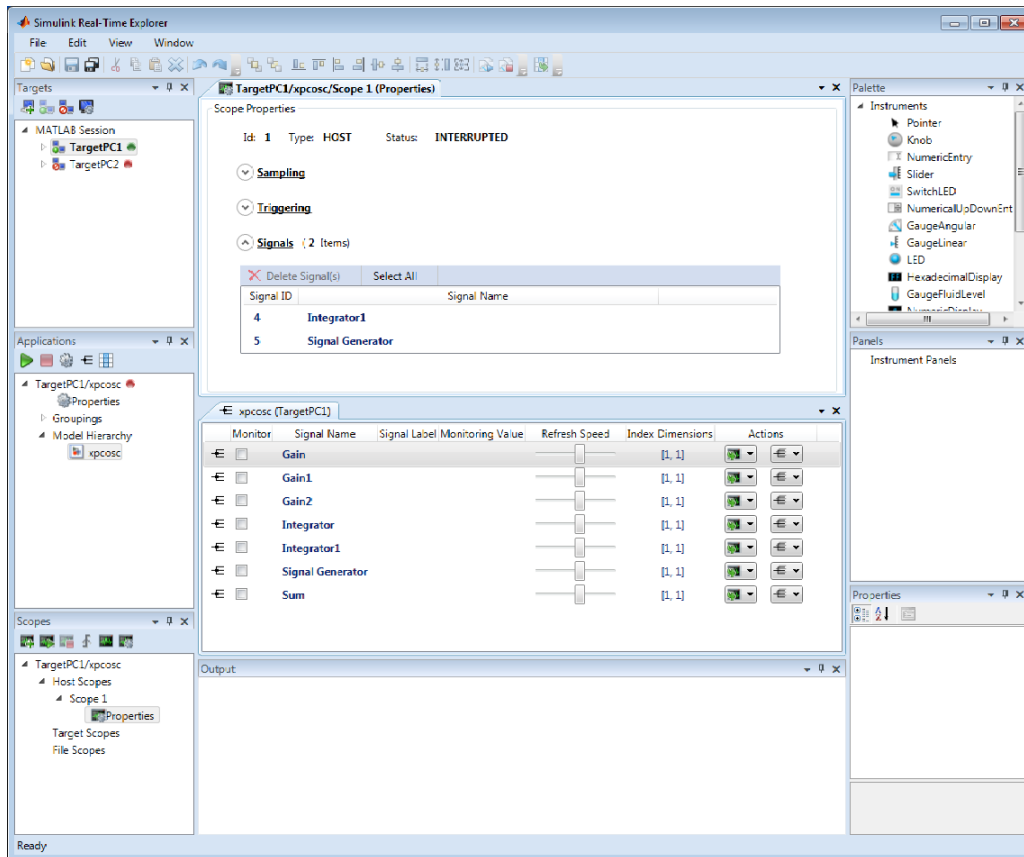
8 To view the host scope, select **Scope 1** and then click the **View Scope** button on the toolbar.

The host scope viewer opens as a separate tab. The signals you add to the scope appear at the top right of the viewer.

9 To start execution, click the real-time application and then click the **Start** button on the toolbar.

The real-time application starts running. No output appears on the host scope viewer.

10 To start **Scope 1**, click **Scope 1** in the **Scopes** pane and then click the **Start Scope** button [image] on the toolbar.

Output for signals `Signal Generator` and `Integrator1` appears on the host scope viewer.
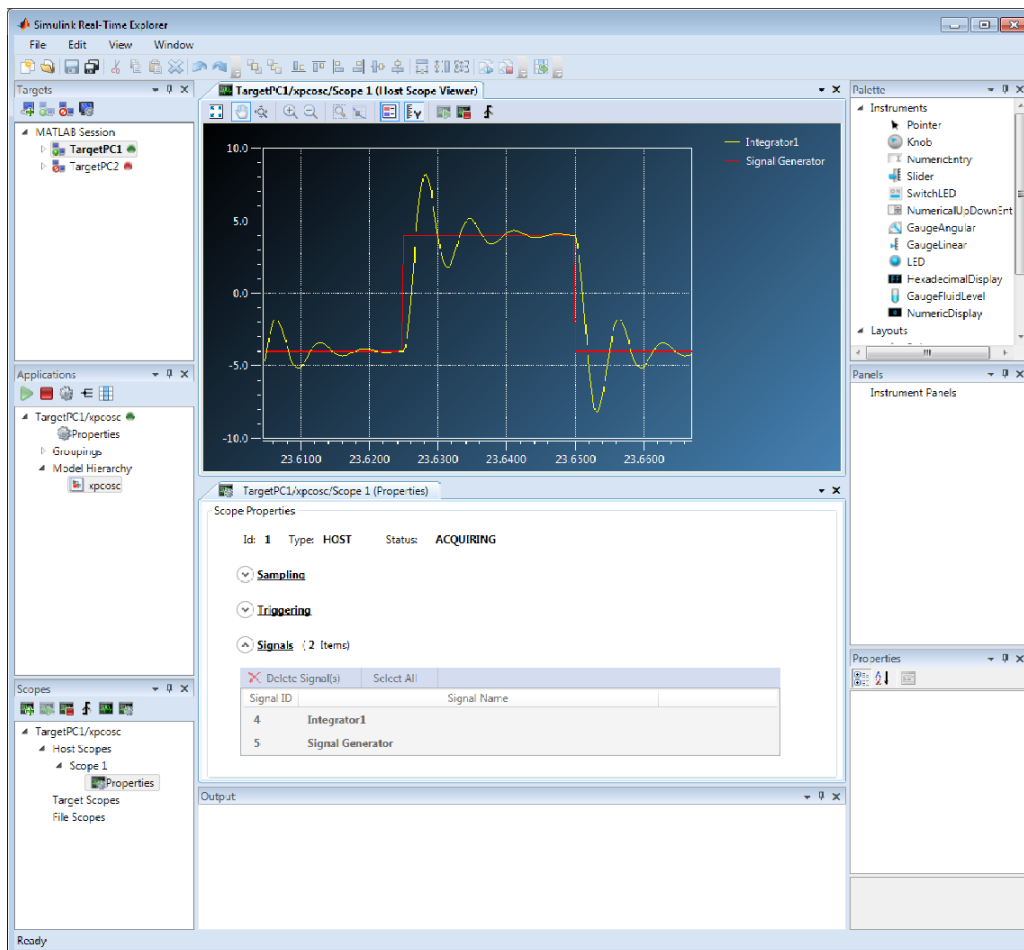
11 To stop **Scope 1**, click **Scope 1** in the **Scopes** pane and then click the **Stop Scope** button [image] on the toolbar.

The signals shown on the target computer stop updating while the real-time application continues running. The target computer monitor displays a message like this message:

```
Scope: 1, set to state 'interrupted'
```

12 To stop execution, click the real-time application and then click the **Stop** button ▉ on the toolbar.

The real-time application on the target computer stops running, and the target computer displays messages like this message:

```
minimal TET: 0.0000006 at time 0.001250
```

```
maximal TET: 0.0000013 at time 75.405500
```

You can create a virtual host scope from the scope types list by clicking **Add Scope** next to scope type **Host Scopes**.

To make both workspaces visible at the same time, drag one workspace tab down until the ✛ icon appears in the middle of the dialog box. Continue to drag until the cursor reaches the required quadrant, and then release the mouse button.

To save your Simulink Real-Time Explorer layout, click **File** > **Save Layout**. In a later session, you can click **File** > **Restore Layout** to restore your layout.

Related Examples

Configure the Host Scope Viewer

You can customize the viewer for each host scope to facilitate your interaction with the running model.

This procedure uses the model xpcosc as an example. You must have already completed the procedure in "Create Host Scopes Using Simulink Real-Time Explorer" on page 5-54. Target execution and scopes must be stopped.

1   In the Signals workspace, to add signal Integrator to host scope **Scope1**, drag signal Integrator to the Host Scope Viewer display.

2   Start execution (▶ on the **Applications** toolbar).

3   To start **Scope 1**, click the **Start** button 🖥 on the Host Scope Viewer toolbar.

4   To trigger **Scope 1**, click the **Trigger** button ⬆ on the Host Scope Viewer toolbar.

    To trigger a capture interactively using the **Trigger** button ⬆ , first set the scope **Trigger Mode** to Software or Scope.

5   In the Simulink Real-Time Host Scope Viewer, right-click anywhere in the axis area of the viewer and then click **Edit**.

    The Host Scope Viewer display parameter buttons become enabled on the toolbar.

6   Adjust the Host Scope Viewer display using:

    - **Auto Scale** 🔳 — To scale the display to accommodate the top and bottom of the Y-axis.

    - **Axes Scroll** ✋ — To move the content up and down and right and left relative to the axes. The axes scroll as required.

    - **Axes Zoom** 🔍 — To stretch and compress the X-axis and Y-axis.

    - **Zoom In** 🔍 — To zoom in on the current center of the display.

    - **Zoom Out** 🔍 — To zoom out from the current center of the display.

    - **Zoom Box** 🔍 — To select an area of interest in the display. When you release the mouse button, the display zooms in upon the selected area.

- **Data Cursor** ▣ — To display data values using a set of cross-hairs in the display.

  Data is displayed as the pair `x-value,y-value`, indicating the value at that point on the display. You can drag the center of the cross hairs and observe the value at each point.

- **Legends** ▣ — To toggle display of the signal names.

7   To stop **Scope 1**, click the **Stop** button ▣ on the Host Scope Viewer toolbar.

8   Stop execution (▣ on the **Applications** toolbar).

Related Examples

- "Trigger Scopes Interactively Using Simulink Real-Time Explorer" on page 5-38

Configure Target Scopes Using MATLAB Language

Creating a scope object allows you to select and view signals using Simulink Real-Time functions instead of the Simulink Real-Time user interface.

This procedure uses the Simulink model xpcosc as an example. To do this procedure, you must have already built the real-time application forxpcosc and downloaded it to the default target computer. It describes how to trace signals with target scopes.

1 Start running your real-time application. Type:

```
tg = slrt;
start(tg)
```

2 To get a list of signals, type:

```
tg.ShowSignals = 'on'
```

The Command Window displays a list of the target object properties for the available signals. For example, the signals for the model xpcosc are:

```
Target: TargetPC1
   Connected          = Yes
   Application        = xpcosc
.
.
.
   Scopes             = 1
   NumSignals         = 7
   ShowSignals        = on
   Signals            =
     INDEX  VALUE     Type   BLOCK NAME      LABEL
     0      0.000000  DOUBLE Gain
     1      0.000000  DOUBLE Gain1
     2      0.000000  DOUBLE Gain2
     3      0.000000  DOUBLE Integrator
     4      0.000000  DOUBLE Integrator1
     5      0.000000  DOUBLE Signal Generator
     6      0.000000  DOUBLE Sum
.
.
.
```

3 Create a scope to be displayed on the target computer. For example, to create a scope with an identifier of 1 and a scope object name of sc1, type:

```
sc1 = addscope(tg, 'target', 1)

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId             = 1
   Status              = Interrupted
   Type                = Target
   NumSamples          = 250
   NumPrePostSamples   = 0
   Decimation          = 1
   TriggerMode         = FreeRun
   TriggerSignal       = -1
   TriggerLevel        = 0.000000
   TriggerSlope        = Either
   TriggerScope        = 1
   TriggerSample       = 0
   DisplayMode         = Redraw (Graphical)
   YLimit              = Auto
   Grid                = on
   Signals             = no Signals defined
```

4 Add signals to the scope object. For example, to add `Integrator1` and `Signal Generator`, type:

```
addsignal(sc1,[4,5])

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId             = 1
   Status              = Interrupted
   Type                = Target
.
.
.
   Grid                = on
   Signals             = 4  : Integrator1
                         5  : Signal Generator
```

The target computer displays the following messages:

```
Scope: 1, signal 4 added

Scope: 1, signal 5 added
```

After you add signals to a scope object, the signal values are not shown on the target display until you start the scope.

5 Start the scope. For example, to start the scope sc1, type:

```
start(sc1)
```

The target display plots the signals after collecting each data package. During this time, you can observe the behavior of the signals while the scope is running.

6 Stop the scope. Type:

```
stop(sc1)
```

The signals shown on the target computer stop updating while the real-time application continues running. The target computer displays the following message:

```
Scope: 1, set to state 'interrupted'
```

7 Stop the real-time application. In the Command Window, type:

```
stop(tg)
```

Related Examples

- "Monitor Signals Using MATLAB Language" on page 5-6

Trace Signals Using Simulink External Mode

You can use Simulink external mode to establish a communication channel between your Simulink block diagram and your real-time application. The block diagram becomes a user interface to your real-time application. Simulink scopes can display signal data from the real-time application, including from models referenced inside a top model. You can control which signals to upload through the External Signal & Triggering dialog box (see "Select Signals to Upload" and "Control External Mode Operations").

Note: Do not use Simulink external mode while Simulink Real-Time Explorer is running. Use only one interface or the other.

This procedure uses the model `xpcosc` as an example. `xpcosc` contains a Simulink Scope block.

1 In the Command Window, type `xpcosc`.

2 In the Simulink Editor, from the **Code** menu, select **External Mode Control Panel**.

3 In the External Mode Control Panel dialog box, click the **Signal & Triggering** button.

4 In the External Signal & Triggering dialog box, set the **Source** parameter to `manual`.

5 Set the **Mode** parameter to `normal`. In this mode, the scope acquires data continuously.

6 Select the **Arm when connecting to target** check box.

7 In the **Delay** box, enter `0`.

8 In the **Duration** box, enter the number of samples for which external mode is to log data, for example `1000`.

The External Signal & Triggering dialog box looks like this figure.
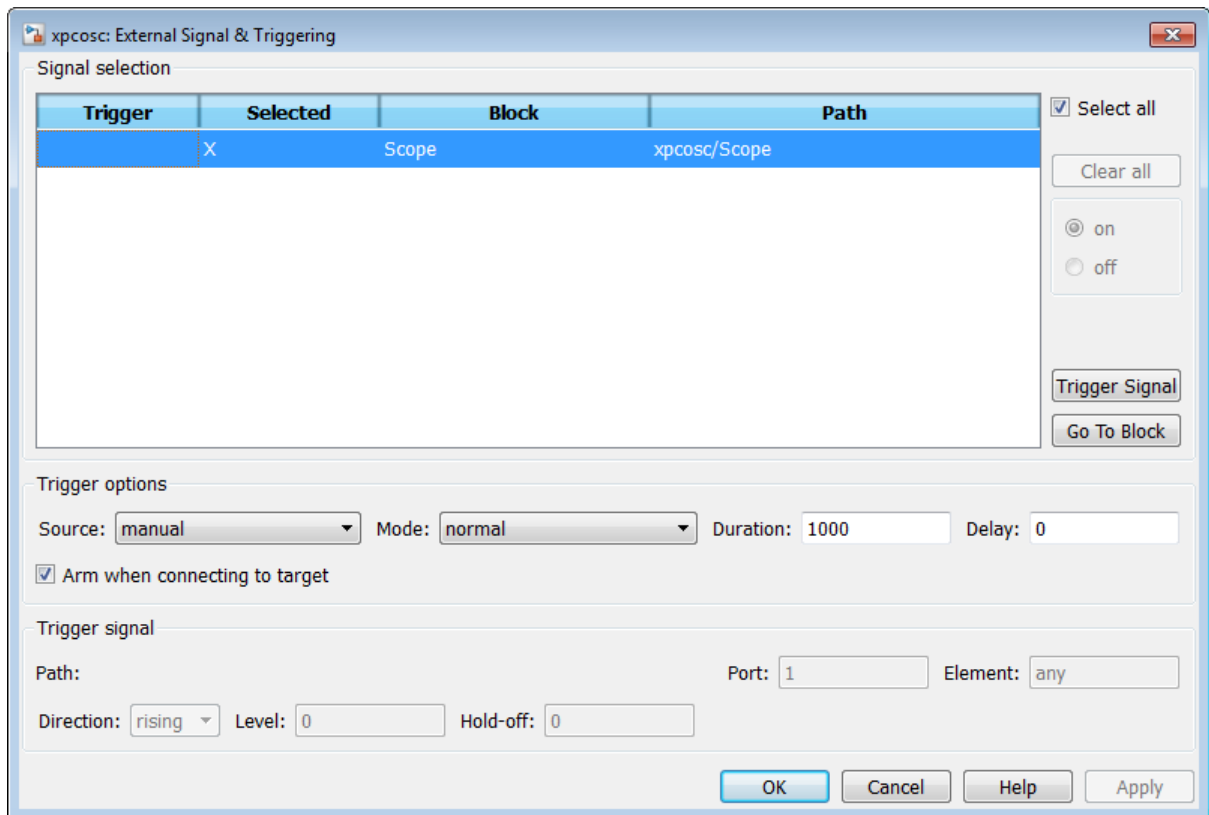
9   Click **Apply**, and then **Close**.

10 In the External Mode Control Panel dialog box, click **OK**.

11 In the Simulink toolbar, increase the simulation stop time to, for example, 50.

12 From the **File** menu, select **Save As** and enter a file name. For example, enter ex_slrt_ext_osc, and then click **OK**.

13 In the Simulink Editor, click **Simulation** > **Mode** > **External**. A check mark appears next to the menu item **External**, indicating that Simulink external mode is activated.

14 If a scope window is not displayed for the Scope block, double-click the Scope block.

15 Build and download the model to the target computer.

16 On the toolbar, click the **Connect To Target** button.

The current Simulink model parameters are downloaded from the development computer to the real-time application.

17 To start the simulation, click the **Run** button ▶ on the toolbar.

The real-time application begins running on the target computer. The Scope window displays plotted data.
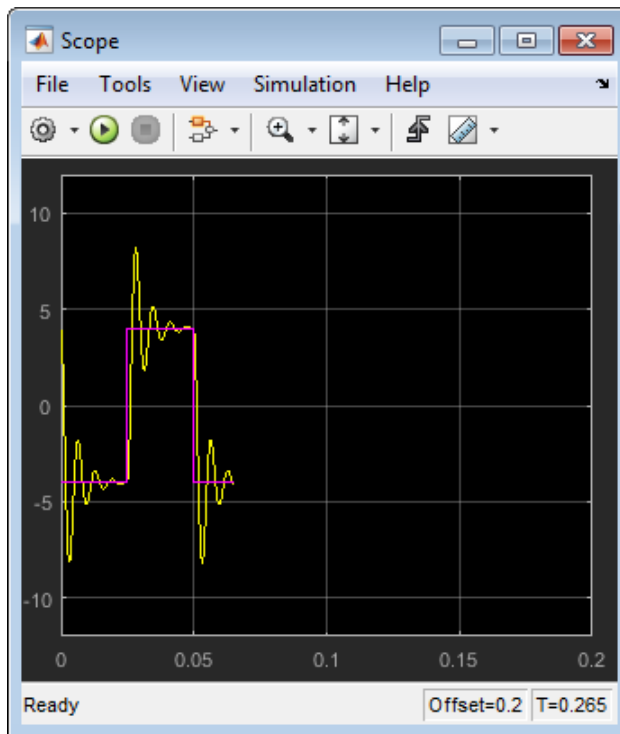
18 To stop the simulation, click the **Stop** button ■ on the toolbar.

Inspect Simulink Real-Time Signals with Simulation Data Inspector

Use Simulink external mode to establish a communication channel between your Simulink block diagram and your real-time application. You can use Simulation Data Inspector (SDI) to stream signal data from the real-time application from models referenced at arbitrary levels within a model hierarchy. Control which signals to display by selecting them in the model.

Note:

- Do not use Simulink external mode while Simulink Real-Time Explorer is running. Use only one interface or the other.

- The real-time application sometimes generates data faster than the kernel can transmit it to the development computer, causing gaps in the output. If gaps occur, reduce the number of signals being inspected or decrease the sample rate.

This procedure uses the model xpcosc as an example. Make sure that you have started the target computer and established communication between the development and target computers.

1 Open xpcosc.

2 On the toolbar, increase the simulation stop time to, for example, 10 seconds.

3 Click **Simulation** > **Mode** > **External**.

4 In the model, select the signals Signal Generator and Integrator1.

5 On the toolbar, click the arrow next to the **Simulation Data Inspector** button ![icon], and then select **Stream Selected Signals to Data Inspector**.

   A faint **Simulation Data Inspector** icon ![icon] appears next to each signal.

6 Build the model and download it to the target computer.

7 Click the **Connect To Target** button ![icon].

   The **Simulation Data Inspector** button glows, indicating that Simulation Data Inspector has data available for viewing.

8 Click the glowing **Simulation Data Inspector** button ![icon].

9 In Simulation Data Inspector, select the signals `Integrator1:1` and
   `SignalGenerator:1`.

10 To start the simulation, click the **Run** button ▶.

   The real-time application begins running on the target computer. Simulation Data
   Inspector displays plotted data.
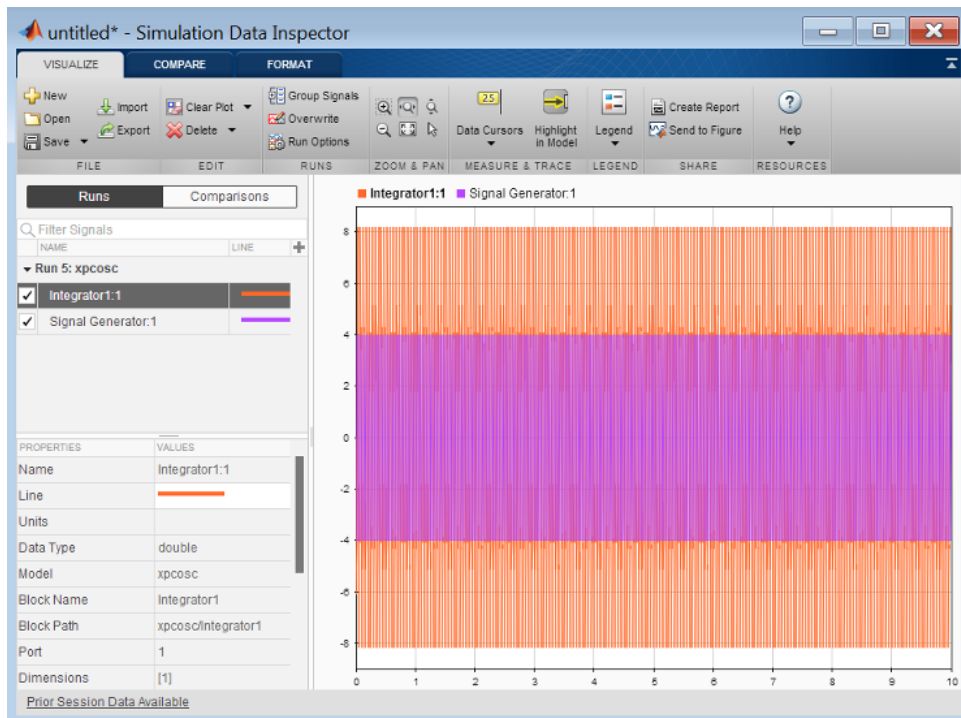
11 To stop the simulation, click the **Stop** button ■.

12 After the simulation, you can use the toolbar buttons to explore the data. For
   example, to view the simulation between seconds 1 and 2, in Simulation Data
   Inspector, click the **Zoom in Time** button. Drag the cursor over the range from 1
   to 2.
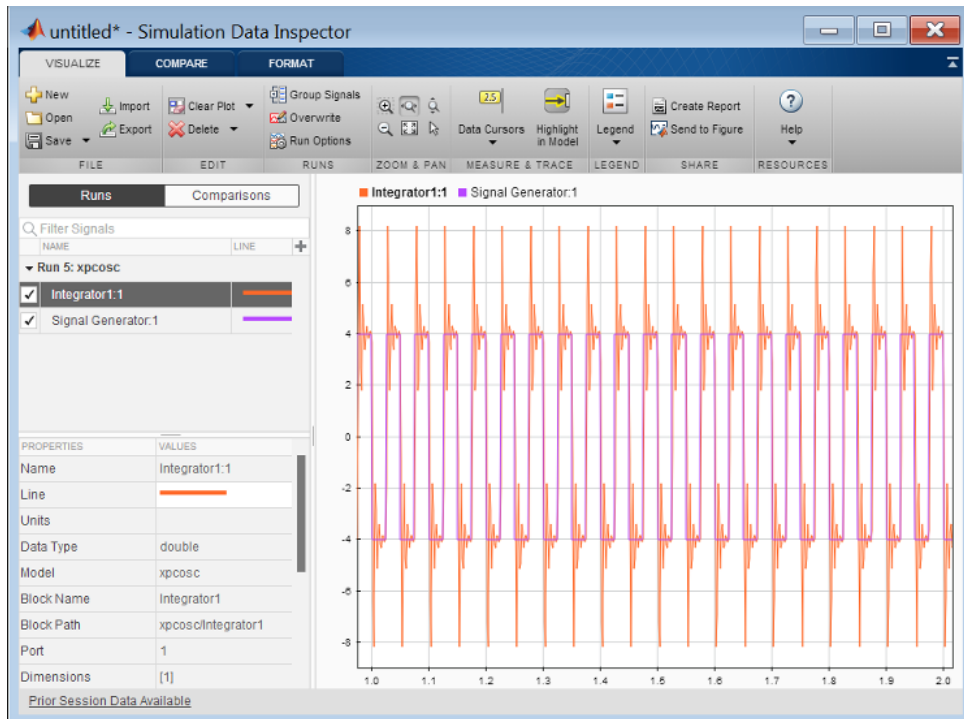
13 To save the Simulation Data Inspector session as a `.mat` file, click **Save**.

More About

· "Inspect Signal Data with Simulation Data Inspector"

External Mode Usage

- When setting up signal triggering (Source set to signal), explicitly specify the element number of the signal in the **Trigger signal:Element** box. If the signal is a scalar, enter a value of 1. If the signal is a wide signal, enter a value from 1 to 10. When uploading Simulink Real-Time signals to Simulink scopes, do not enter Last or Any in this box.

- The **Direction:Holdoff** value does not affect the Simulink Real-Time signal uploading feature.

- Attempting to upload information from buses and virtual signals inside a reference model generates a warning.

Signal Logging Basics

Signal logging acquires signal data during a real-time run and stores it on the target computer. After you stop the real-time application, you transfer the data from target computer to development computer for analysis. Signal logging is also known as real-time data streaming to the target computer. You can plot and analyze the data, and later save it to a disk on the development computer.

Simulink Real-Time signal logging samples at the base sample time. If you have a model with multiple sample rates, add Simulink Real-Time scopes to the model to sample signals at the required sample rates.

- The Simulink Real-Time software does not support logging data with decimation.
- Simulink Real-Time Explorer works with multidimensional signals in column-major format.
- Some signals are not observable.

You can log signals using the following methods:

- Outports in the model
- File scopes in the model
- Virtual file scopes created using Simulink Real-Time Explorer
- Virtual file scopes created using MATLAB language

Related Examples

- "Configure File Scopes Using Simulink Real-Time Explorer" on page 5-84
- "Log Signal Data with Outport Block and Simulink Real-Time Explorer" on page 5-92
- "Log Signal Data with Outport Block and MATLAB Language" on page 5-98

More About

- "Nonobservable Signals" on page 5-136
- "Simulink Real-Time Scope Usage" on page 5-22
- "Target Scope Usage" on page 5-24
- "Host Scope Usage" on page 5-29

-

Configure Real-Time File Scope Blocks

Simulink Real-Time includes a specialized `Scope` block that you can configure to save signal and time data to a file on the target computer hard drive, flash drive, or removable drive. Add a `Scope` block to the model, select **Scope type** `File`, and then configure the other parameters as described in the following procedure.

Do not confuse Simulink Real-Time Scope blocks with standard Simulink Scope blocks.

This procedure starts from the example model `ex_slrt_rt_osc`.

1 In the Command Window, type `ex_slrt_rt_osc`.

2 In the Simulink Editor, double-click the block labeled `Scope`.

   The Block Parameters: `Scope` dialog box opens. By default, the target scope dialog box is displayed.

3 In the **Scope number** box, a unique number is displayed that identifies the scope. This number is incremented each time you add a Simulink Real-Time scope.

   This number identifies the Simulink Real-Time Scope block and the scope screen on the development or target computer.

4 From the **Scope type** list, select `File`. The updated dialog box opens.

5 **Caution** Before starting the scope, copy previously acquired data to the development computer. When the file scope starts, the software overwrites previously acquired data in files of the specified name or name pattern. A partially overwritten file or a file opened but left unwritten loses its original contents.

   To start the scope automatically when the real-time application executes, select the **Start scope when application starts** check box.

   In `Stand Alone` mode, this setting is mandatory because the development computer is not available to issue a command to start scopes.

6 In the **Number of samples** box, enter the number of values to be acquired in a data package.

   The **Number of samples** parameter works with the autorestart setting.

   • Autorestart is on — When the scope triggers, the scope collects 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 up with data collection, data can be lost.

- Autorestart is off — When the scope triggers, the scope collects data into a memory buffer up to the number of samples that you specified, and then the scope stops. A background task examines the buffer and writes data to the disk continuously, appending the new data to the end of the file.

7   In the **Number of pre/post samples** box, enter the number of samples to save or skip. To save `N` samples before a trigger event, specify the value `-N`. To skip `N` samples after a trigger event, specify the value `N`. The default is `0`.

8   In the **Decimation** box, enter a value to indicate that data is collected at each sample time (`1`) or at less than every sample time (`2` or greater).

9   From the **Trigger mode** list, select `FreeRun`, `Software Triggering`, `Signal Triggering`, or `Scope Triggering`.

- If you select `FreeRun` or `Software Triggering`, you do not need to enter additional parameters.

- If you select `Signal Triggering`, then enter the following additional parameters, as required:

  - In the **Trigger signal** box, enter the index of a signal previously added to the scope.

    This parameter does not apply if the **Add signal port to connect a signal trigger source** check box is selected.

  - (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.

  - 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`.

- If you select `Scope Triggering`, then enter the following additional parameters, as required:

  - In the **Trigger scope number** box, enter the scope number of a Scope block. If you use this trigger mode, add a second Scope block to your Simulink model.

- To trigger one scope on a specific sample of another scope, enter a value in **Sample to trigger on (-1 for end of acquisition)**. The default value of 0 indicates that the triggering scope and the triggered scope start simultaneously.

10 In the **Filename** box, enter a name for the file to contain the signal data.

By default, the target computer writes the signal data to `C:\data.dat`.

A fully qualified file name can have a maximum of 260 characters. The file part can have 12 characters — a maximum of eight characters for the file name, one character for the period, and a maximum of three characters for the file extension. If the file name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

11 From the **Mode** list, select either `Lazy` or `Commit`.

With the `Commit` mode, each file write operation simultaneously updates the FAT entry for the file. The file system maintains the actual file size after each write. With the `Lazy` mode, the FAT entry is updated only when the file is closed.

If your system stops responding, you lose WriteSize bytes of data.

12 In the **WriteSize** box, enter the block size, in bytes, of the data chunks. This parameter specifies that a memory buffer of length Number of samples is written to the file in chunks of size WriteSize. By default, this parameter is 512 bytes. Using a block size that is the same as the disk sector size improves performance.

If your system stops responding, you lose **WriteSize** bytes of data.

13 Select the **AutoRestart** check box to have the file scope collect data up to **Number of samples** and then start over again reading new data. Clear the **AutoRestart** check box to have the file scope collect data up to **Number of samples** and then stop.

If the named signal data file exists when the file scope starts, the Simulink Real-Time software overwrites the old data with the new signal data.

Setting this check box enables the following parameters: **Dynamic file name enabled** and **Max file size in bytes (multiple of WriteSize)**.

The file scope dialog box looks like this figure.
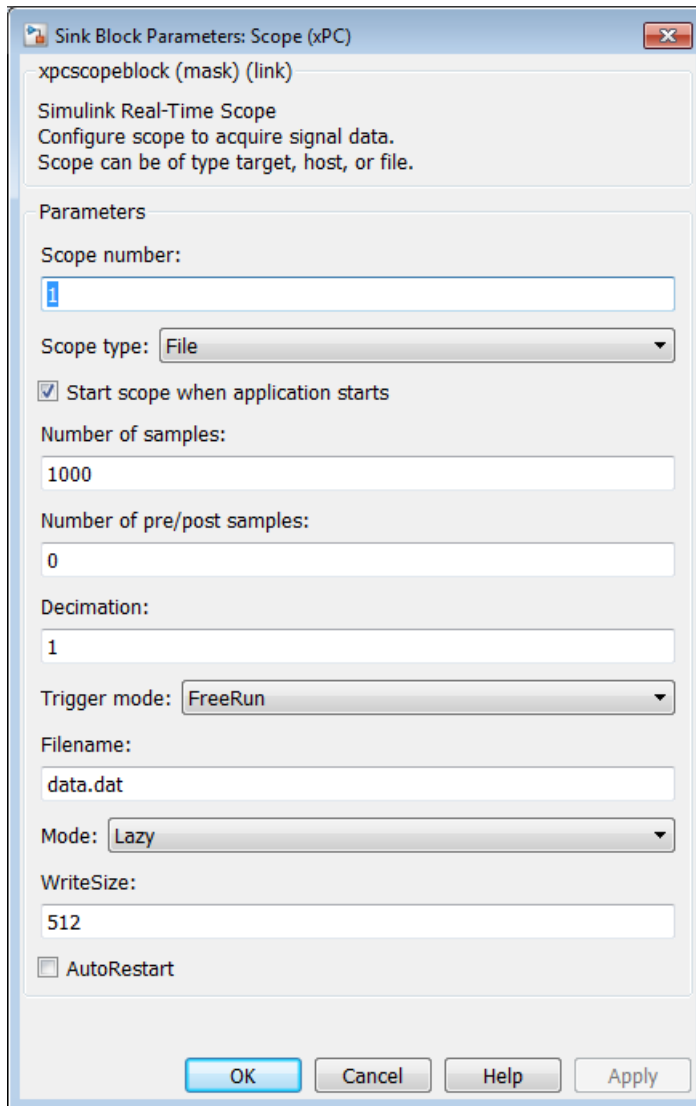
14 Click **OK**.

15 From the **File** menu, click **Save As**. Save the model as `ex_slrt_file_osc`.

See Also

Scope

Related Examples

- "Trigger One Scope with Another Scope" on page 11-24

More About

- "Simulink Real-Time Scope Usage" on page 5-22
- "File Scope Usage" on page 5-77

File Scope Usage

For usage issues unique to file scopes:

- Simulink Real-Time supports eight file scopes. Each file scope can contain as many signals as the target computer resources can support.

- At most, eight files can be open on the target computer at the same time.

- The largest single file that you can create is 4 GB.

- A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.

- A fully qualified file name can have a maximum of 260 characters. The file part can have 12 characters — a maximum of eight characters for the file name, one character for the period, and a maximum of three characters for the file extension. If the file name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

- If you enter just the file name, the file appears in folder `C:\`. To put the file in a folder, create the folder separately using the target computer command line or the `SimulinkRealTime.fileSystem.mkdir` command.

- You can configure the scope to generate multiple, dynamically named files in one session.

- Before starting the scope, copy previously acquired data to the development computer. When the file scope starts, the software overwrites previously acquired data in files of the specified name or name pattern. A partially overwritten file or a file opened but left unwritten loses its original contents.

- You cannot read a file that was written during real-time execution until execution has completed.

- After real-time execution, the file scope software generates a signal data file on the target computer, even if it is running in `Stand Alone` mode. To access the contents of the signal data file that a file scope creates, use the `SimulinkRealTime.fileSystem` object from a development computer Command Window. To view or examine the signal data, use the `SimulinkRealTime.utils.getFileScopeData` utility and the `plot` function. Saving signal data to files lets you recover signal data from a previous run in the event of system failure.

- The signal data file can quickly increase in size. To gauge the growth rate for the file, examine the file size between runs. If the signal data file grows beyond the available space on the disk, the signal data is corrupted.

- The file scope acquires data and writes it to the file named in the **FileName** parameter. The scope writes data samples into a memory buffer of size given by the **Number of Samples** parameter. It copies data from the memory buffer into the file in blocks of size given by the **WriteSize** parameter.

  The **Number of samples** parameter works with the autorestart setting.

  - Autorestart is on — When the scope triggers, the scope collects 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 up with data collection, data can be lost.

  - Autorestart is off — When the scope triggers, the scope collects data into a memory buffer up to the number of samples that you specified, and then the scope stops. A background task examines the buffer and writes data to the disk continuously, appending the new data to the end of the file.

- When real-time execution stops without an error, both the `Lazy` and `Commit` settings of the **Mode** box have the same result. Both settings cause the model to open a file, write signal data to the file, and close that file at the end of the session. The differences are in when the software updates the FAT entry for the file.

  - In `Commit` mode, the FAT entry and the actual file size are updated during each file write operation.

  - In `Lazy` mode, the FAT entry and the actual file size are updated only when the file is closed and not during each file write operation.

  `Lazy` mode is faster than `Commit` mode. However, if the target computer enters an error state, the system can stop responding before the file is closed. In `Lazy` mode, the actual file size can be lost, even though the file was written. You can lose an amount of data equivalent to the setting of the **WriteSize** parameter.

- Select the type of trigger event in the Block Parameters: `Scope` dialog box by setting **Trigger Mode** to `Signal Triggering`, `Software Triggering`, or `Scope Triggering`.

  The number of samples N to log after triggering an event is equal to the value that you entered in the **Number of Samples** parameter.

See Also

Using File System Objects | `SimulinkRealTime.fileSystem.mkdir` | `SimulinkRealTime.utils.getFileScopeData`

Related Examples

- "Configure Real-Time File Scope Blocks" on page 5-72
- "Create File Scopes Using Simulink Real-Time Explorer" on page 5-80
- "Log Signal Data into Multiple Files" on page 5-88

More About

- "Simulink Real-Time Scope Usage" on page 5-22
- "Using SimulinkRealTime.fileSystem Objects" on page 12-4

Create File Scopes Using Simulink Real-Time Explorer

You can create a virtual file scope on the target computer using Simulink Real-Time Explorer. These scopes have the full capabilities of the Scope block in File mode, but do not persist past the current execution.

Note: For information on using file scope blocks, see "Configure Real-Time File Scope Blocks" on page 5-72 and "File Scope Usage" on page 5-77.

This procedure uses the model xpcosc as an example. You must have already completed the following setup:

1   Built and downloaded the real-time application to the target computer using Simulink (⊞ on the toolbar).

2   Run Simulink Real-Time Explorer (**Tools** > **Simulink Real-Time**).

3   Connected to the target computer in the **Targets** pane (⚬ on the toolbar).

4   Set property **Stop time** to inf in the **Applications** pane (⚙ on the toolbar).

To configure a virtual file scope:

1   In the **Scopes** pane, expand the **xpcosc** node.

2   To add a file scope, select **File Scopes**, and then click the **Add Scope** button ⊕ on the toolbar.

3   Expand **Scope 1**, and then click the **Properties** button ⚙ on the toolbar.

4   In the **Scope Properties** pane, click **Signals**.

    Add signals from the **Applications** Signals workspace.

5   In the **Applications** pane, expand both the real-time application node and the node **Model Hierarchy**.

6   Select the model node and then click the **View Signals** button ⋿ on the toolbar.

7   In the Signals workspace, to add signal Signal Generator to **Scope1**, drag signal Signal Generator to the **Scope1** properties workspace.

8   Add signal Integrator1 to **Scope 1** in the same way.

9 In the **Scope Properties** pane, click **File**.

10 Enter a name in the **File name** text box, for example `scope1.dat`.

11 Select the **AutoRestart** check box to have the file scope collect data up to **Number of samples** and then start over again reading new data.

12 Leave the **Dynamic File Mode** check box cleared.

13 To start execution, click the real-time application and then click the **Start** button ▶ on the toolbar.

14 **Caution** Before starting the scope, copy previously acquired data to the development computer. When the file scope starts, the software overwrites previously acquired data in files of the specified name or name pattern. A partially overwritten file or a file opened but left unwritten loses its original contents.

To start **Scope 1**, click **Scope 1** in the **Scopes** pane, and then click the **Start Scope** button 🔳 on the toolbar.

15 To stop **Scope 1**, click **Scope 1** in the **Scopes** pane, and then click the **Stop Scope** button 🔳 on the toolbar.

For file scopes, before adding or removing signals, stop the scope first.

16 To stop execution, click the real-time application, and then click the **Stop** button 🔴 on the toolbar.

17 To view the file that you generated, in the **Targets** pane, expand the target computer and then double-click **File System**.

18 Select `C:\`. The dialog box looks like this figure.
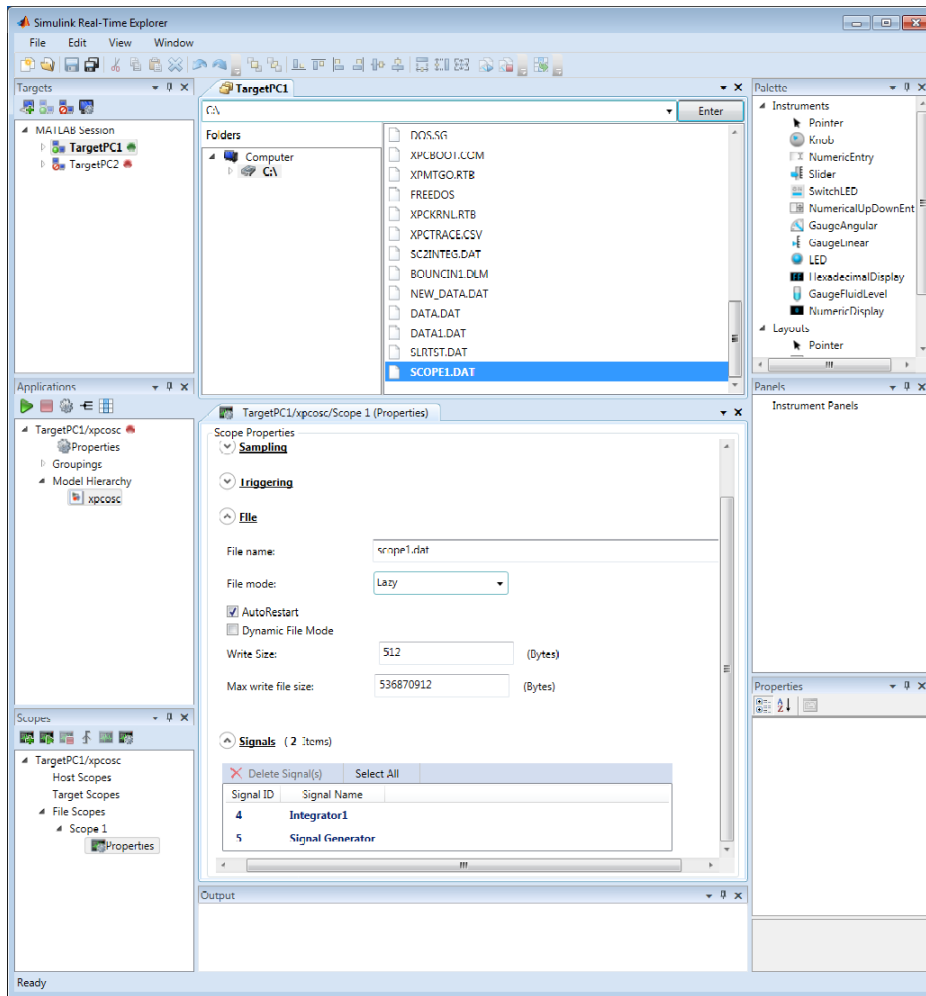
19 To retrieve the file from the target computer, select the file in the target computer **File System** pane. Drag it to the MATLAB **Current Folder** pane or to a Windows Explorer window.

You can create a virtual file scope from the list of scope types by clicking **Add Scope** next to scope type **File Scopes**.

To rename file `SCOPE1.DAT`, right-click the file name, select **Rename**, type the new name in the text box, and then click **Enter**.

To delete file `SCOPE1.DAT`, right click the file name and select **Delete**.

To make both workspaces visible at the same time, drag one workspace tab down until the ⊹ icon appears in the middle of the dialog box. Continue to drag until the cursor reaches the required quadrant, and then release the mouse button.

To save your Simulink Real-Time Explorer layout, click **File** > **Save Layout**. In a later session, you can click **File** > **Restore Layout** to restore your layout.

Related Examples

Configure File Scopes Using Simulink Real-Time Explorer

You can configure your file scopes to facilitate data logging. You can configure a file scope whether you added a `Scope` block to your model or added the scope at run time.

This procedure uses the model `xpcosc` as an example. You must have already completed the procedure in "Create File Scopes Using Simulink Real-Time Explorer" on page 5-80. Target execution and scopes must be stopped.

1 Select **Scope 1**, and then open the Properties pane (on the **Scopes** toolbar).

2 In the **Scope 1** Properties pane, click **File**.

3 Enter a name in the **File name** text box, for example `scope2.dat`.

   File names on the target computer are limited to eight characters in length, not counting the file extension. If the name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

   If you enter just the file name, the file appears in folder `C:\`. To put the file in a folder, create the folder separately using the target computer command line or MATLAB language.

   A fully qualified file name can have a maximum of 260 characters. The file part can have 12 characters — a maximum of eight characters for the file name, one character for the period, and a maximum of three characters for the file extension. If the file name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

   If a file with this name exists when you start the file scope, the file scope overwrites the old data with the new data.

4 Select **File mode** `Commit`.

   The default **File mode** is `Lazy`. When real-time execution stops without an error, both the `Lazy` and `Commit` settings of the **Mode** box have the same result. Both settings cause the model to open a file, write signal data to the file, and close that file at the end of the session. The differences are in when the software updates the FAT entry for the file.

   • In `Commit` mode, the FAT entry and the actual file size are updated during each file write operation.

- In `Lazy` mode, the FAT entry and the actual file size are updated only when the file is closed and not during each file write operation.

Lazy mode is faster than `Commit` mode. However, if the target computer enters an error state, the system can stop responding before the file is closed. In `Lazy` mode, the actual file size can be lost, even though the file was written. You can lose an amount of data equivalent to the setting of the **WriteSize** parameter.

5   Select the **AutoRestart** check box to have the file scope collect data up to **Number of samples** and then start over again reading new data.

6   Leave the **Dynamic File Mode** check box cleared.

7   Leave **Write Size** set to the default value of `512`.

Using a block size that is the same as the disk sector size improves performance.

8   Leave **Max write file size** set to the default value, which is a multiple of **Write Size**.

9   Start execution (▶ on the **Applications** toolbar).

10 **Caution** Before starting the scope, copy previously acquired data to the development computer. When the file scope starts, the software overwrites previously acquired data in files of the specified name or name pattern. A partially overwritten file or a file opened but left unwritten loses its original contents.

Start **Scope 1**(⬛ on the **Scopes** toolbar). Let it run for up to a minute.

11 Stop **Scope 1** (⬛ on the **Scopes** toolbar).

12 Stop execution (⬛ on the **Applications** toolbar).
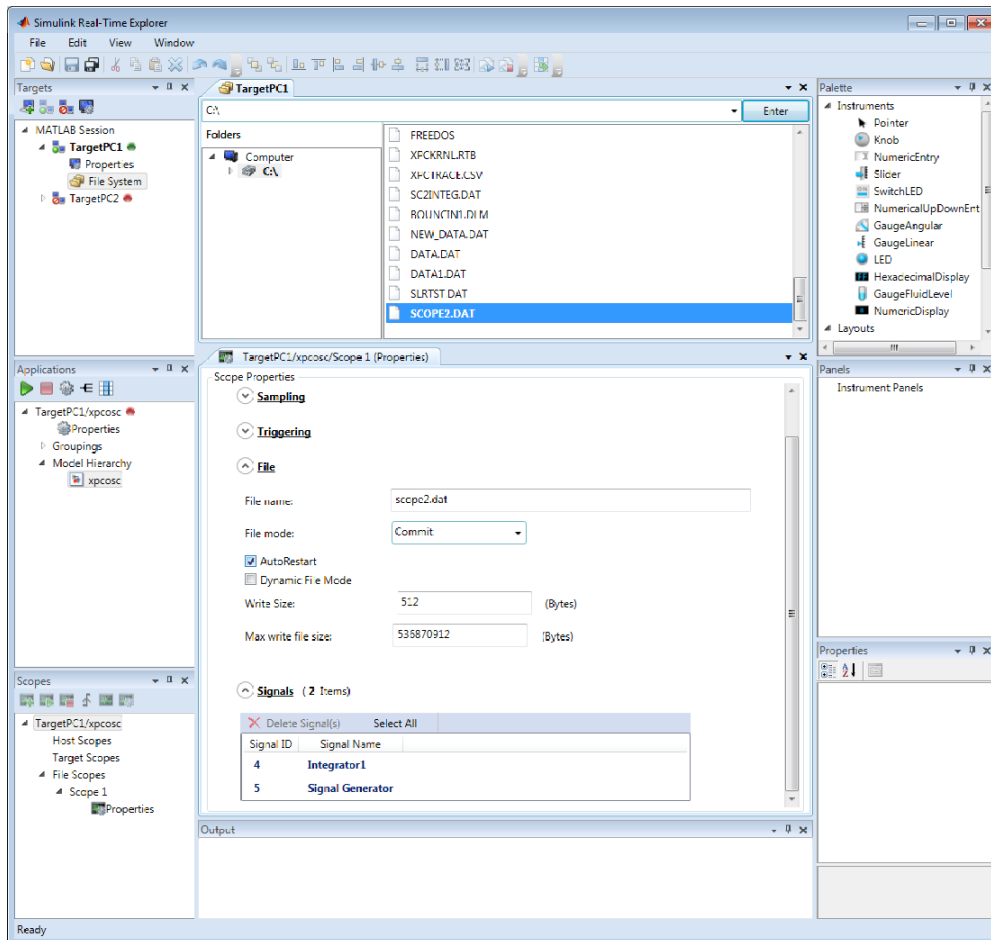
13 To retrieve the file from the target computer, select the file in the target computer **File System** pane and drag it to the MATLAB **Current Folder** pane or to a Windows Explorer window.

To rename file `SCOPE2.DAT`, right-click the file name, select **Rename**, type the new name in the text box, and then click **Enter**.

To delete file `SCOPE2.DAT`, right-click the file name and select **Delete**.

See Also
`SimulinkRealTime.fileSystem.mkdir`

Related Examples

Log Signal Data into Multiple Files

You can acquire signal data to store in multiple, dynamically named files on the target computer. You can then examine one file while the scope continues to acquire data to store in other files. To acquire data for multiple files, add a file scope to the real-time application, and then configure that scope to log signal data to multiple files.

Using model xpcosc, complete the setup tasks in "Create File Scopes Using Simulink Real-Time Explorer" on page 5-80.

1   In Simulink Real-Time Explorer, in the **Scopes** pane, expand the **xpcosc** node.

2   Select **File Scopes** and expand node **File Scopes**.

3   Expand **Scope 1** and then click the **Properties** button on the toolbar.

4   In the **Scope Properties** pane, click **File**.

5   Select the **AutoRestart** check box.

   When you select the **AutoRestart** box, the file scope collects data up to **Number of samples** and then starts over again reading new data. Setting **AutoRestart** enables the following parameters: **Dynamic file name enabled** and **Max file size in bytes (multiple of WriteSize)**.

6   Select the **Dynamic File Mode** check box.

7   To enable the file scope to create multiple log files based on the same name, in the **File name** box, enter a name like scope1_<%>.dat.

   This sequence directs the software to create up to 9 log files, scope1_1.dat to scope1_9.dat, on the target computer file system.

   You can configure the file scope to create up to 99999999 files (<%%%%%%%%>.dat). The length of a file name, including the specifier, cannot exceed eight characters.

8   In the **Max write file size** box, enter a value to limit the size of the signal log files. This value must be a multiple of the **Write Size** value. For example, if the write size is 512, enter 4096 to limit each log file size to 4096 bytes.

9   To start execution, click the real-time application and then click the **Start** button on the toolbar.

10 **Caution** Before starting the scope, copy previously acquired data to the development computer. When the file scope starts, the software overwrites previously acquired

data in files of the specified name or name pattern. A partially overwritten file or a file opened but left unwritten loses its original contents.

To start **Scope 1**, click **Scope 1** in the **Scopes** pane and then click the **Start Scope** button on the toolbar.

Let **Scope 1** run for up to a minute.

11 To stop **Scope 1**, click **Scope 1** in the **Scopes** pane and then click the **Stop Scope** button on the toolbar.

12 To stop execution, click the real-time application and then click the **Stop** button on the toolbar.

13 To view the files that you generated, in the **Targets** pane, expand the target computer, and then double-click **File System**.

14 Select C:\. The dialog box looks like this figure.
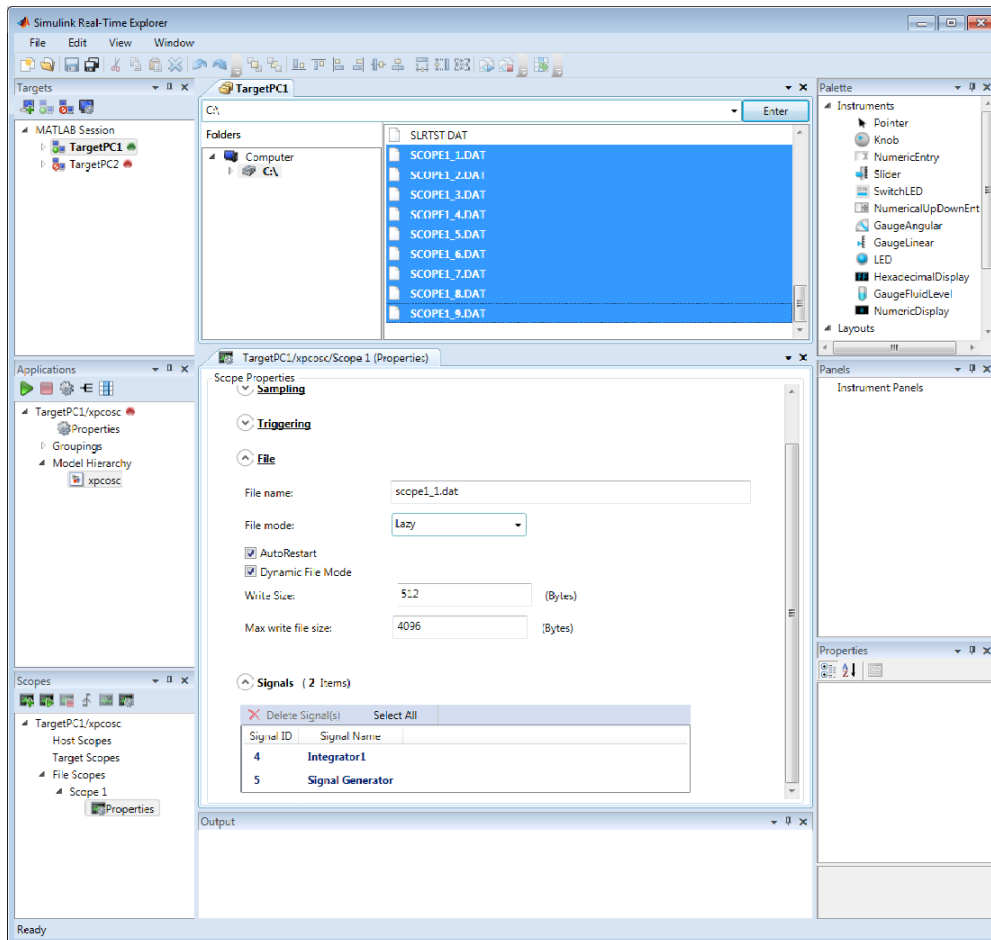
The software creates a log file named SCOPE1_1.DAT and writes data to that file. When the size of the first file reaches 4096 bytes (**Max write file size**), the software closes the first file and creates the second file, SCOPE1_2.DAT. When the size of the second file reaches 4096 bytes, the software creates the third file, the fourth file, and so on.

If the real-time application continues to collect data after the software closes
SCOPE1_9.DAT, the software reopens SCOPE1_1.DAT, SCOPE1_2.DAT, and so on,
overwriting the existing contents.

15 Drag each file from the target computer **File System** pane to the MATLAB **Current
Folder** pane or to a Windows Explorer window.

See Also

Using File System Objects

Related Examples

- "Using SimulinkRealTime.fileSystem Objects" on page 12-4

More About

- "File Scope Usage" on page 5-77

Log Signal Data with Outport Block and Simulink Real-Time Explorer

To use Simulink Real-Time Explorer for signal logging, add an `Outport` block to your Simulink model. Activate logging on the **Data Import/Export** pane in the Configuration Parameters dialog box.

To access the data log that the real-time application creates when it is running on the target computer, use Real-Time Application Properties.

The example begins with the `ex_slrt_rt_osc` model. Here is the final configured model, `ex_slrt_outport_osc`.
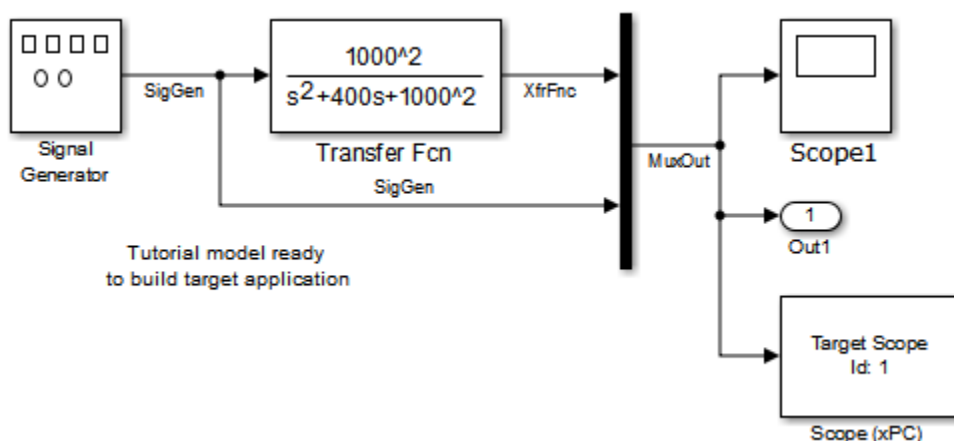
The logged outputs are the signals connected to Simulink `Outport` blocks. The model has one `Outport` block, with index 1. This `Outport` block shows the signals leaving the block labeled `Mux`.

In this section...
"Data Logs" on page 5-93
"Configure the Model for Data Logging" on page 5-93
"Log the Data" on page 5-94
"Download and Plot the Data" on page 5-94

Data Logs

Simulink Real-Time stores logged data in four data logs that you can access on the development computer by using Real-Time Application Properties. In the following list, `tg` is the name of the `SimulinkRealTime.target` object that you use to communicate with the target computer.

- `tg.TimeLog` — Time or T-vector, specified as a vector of double. To turn on, in the **Data Import/Export** pane, set the **Time** model parameter.
- `tg.OutputLog` — Output or Y-vector, specified as a matrix. To turn on, in the **Data Import/Export** pane, set the **Output** model parameter.
- `tg.TETLog` — Task execution time vector, specified as a vector of double. To turn on, in the **Simulink Real-Time Options** pane, set the **Log Task Execution Time** model parameter.
- `tg.StateLog` — State or X-vector, specified as a matrix. To turn on, in the **Data Import/Export** pane, set the **State** model parameter.

Turn on logging for only the data that you are interested in.

Each `Outport` block has an associated column vector in `tg.OutputLog`. You can access the data that corresponds to a particular `Outport` block by specifying the column vector for that block. For example, to access the data that corresponds to `Outport 2`, use `tg.outputlog(:,2)`.

To download part of the logs, use the target object method `SimulinkRealTime.target.getlog`.

Note:

- The data logging variables `tout`, `xout`, `yout`, and `logsout` are available only when you use Simulink to simulate the model in non-real-time.

- You cannot use Simulink Data Inspector to create a data log on the target computer. You can log only signals that are connected to an `Outport` block.

Configure the Model for Data Logging

1   Click **Simulation** > **Model Configuration Parameters**.

2   To allow Simulink to log signals, in the **Data Import/Export** pane, verify that
the **Time** and **Output** check boxes are selected. These check boxes are selected by
default.

3   To plot the task execution time, in the **Code Generation** > **Simulink Real-Time
Options** pane, verify that the **Log Task Execution Time** check box is selected.
This check box is selected by default.

4   To create a buffer for the signals that you are logging, set **Signal logging buffer
size in doubles** to the required value.

The default value of 100000 units is large enough for this model.

5   From the **File** menu, click **Save as**. Enter ex_slrt_outport_osc, and then click
Save.

6   Click **OK**.

Log the Data

1   In the Simulink Editor, on the toolbar, click the **Build Model** button 📅.

2   Run Simulink Real-Time Explorer (**Tools** > **Simulink Real-Time**).

3   To connect to the target computer in the **Targets** pane, click the **Connect** button
🔗 on the toolbar.

4   To start execution, click the real-time application, and then on the toolbar, click the
Start button ▶.

5   To stop execution, click the real-time application, and then on the toolbar, click the
Stop button ⏹.

Download and Plot the Data

1   Download and plot the logged times and output values from the target computer. In
the Command Window, type:

```
tg = slrt;
timelog = tg.TimeLog;
outputlog = tg.OutputLog;
plot(timelog, outputlog)
```

2 Download and plot the task execution times for the target computer. In the Command Window, type:

```
tetlog = tg.TETLog;
plot(timelog, tetlog)
```

The plot shown is the result of a real-time execution.

3 In the Command Window, type:

```
tg.AvgTET
```

```
ans =
```

```
  5.7528e-006
```

The percentage of CPU performance is the average TET divided by the sample time.

For a single-rate model, task execution time (TET) measures how long it takes the kernel to update the model state, propagate the new model state to the outputs, and log the

model state during a sample interval. For a multirate model, TET measures how long it takes the kernel to perform those tasks for the base-rate portion only. You can plot the TET to show under what circumstances subsystems are executed and the additional CPU time required for those executions.

See Also

Real-Time Application Properties | `SimulinkRealTime.target.getlog`

Related Examples

- "Simulate Simulink Model Using MATLAB Language"
- "Log Signal Data with Outport Block and MATLAB Language" on page 5-98

More About

- "Signal Logging Buffer Size" on page 5-105

Log Signal Data with Outport Block and MATLAB Language

To use MATLAB language for signal logging, add an Outport block to your Simulink model. Activate logging by using MATLAB commands.

To access the data log that the real-time application creates when it is running on the target computer, use Real-Time Application Properties.

The example begins with the ex_slrt_rt_osc model. Here is the final configured model, ex_slrt_outport_osc.
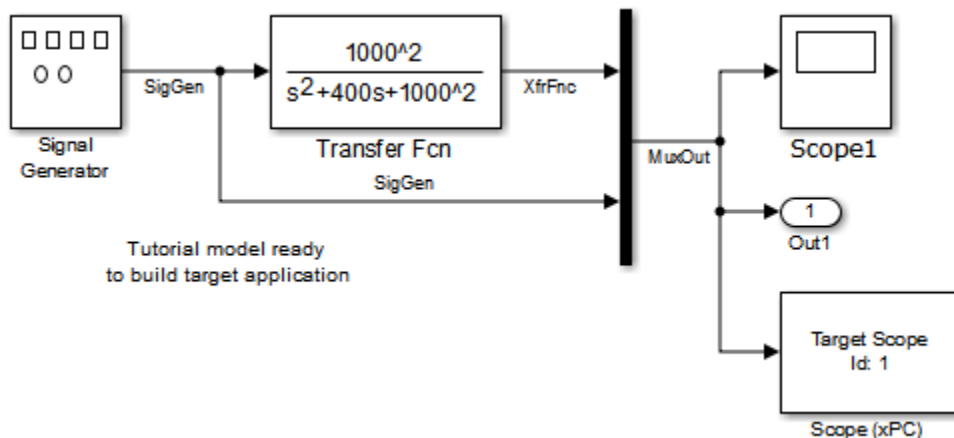
The logged outputs are the signals connected to Simulink Outport blocks. The model has one Outport block, with index 1. This Outport block shows the signals leaving the block labeled Mux.

In this section...
"Data Logs" on page 5-99
"Configure the Model for Data Logging" on page 5-99
"Log the Data" on page 5-100
"Download and Plot the Data" on page 5-101

Data Logs

Simulink Real-Time stores logged data in four data logs that you can access on the development computer by using Real-Time Application Properties. In the following list, `tg` is the name of the `SimulinkRealTime.target` object that you use to communicate with the target computer.

- `tg.TimeLog` — Time or T-vector, specified as a vector of double. To turn on, set the `SaveTime` model parameter.
- `tg.OutputLog` — Output or Y-vector, specified as a matrix. To turn on, set the `SaveOutput` model parameter.
- `tg.TETLog` — Task execution time vector, specified as a vector of double. To turn on, set the `RL32LogTETModifier` model parameter.
- `tg.StateLog` — State or X-vector, specified as a matrix. To turn on, set the `SaveState` model parameter.

Turn on logging for only the data that you are interested in.

Each `Outport` block has an associated column vector in `tg.OutputLog`. You can access the data that corresponds to a particular `Outport` block by specifying the column vector for that block. For example, to access the data that corresponds to `Outport 2`, use `tg.outputlog(:,2)`.

To download part of the logs, use the target object method `SimulinkRealTime.target.getlog`.

Note:

- The data logging variables `tout`, `xout`, `yout`, and `logsout` are available only when you use Simulink to simulate the model in non-real-time.

- You cannot use Simulink Data Inspector to create a data log on the target computer. You can log only signals that are connected to an `Outport` block.

Configure the Model for Data Logging

1 Open model `ex_slrt_rt_osc`.

```
mdl = 'ex_slrt_rt_osc';
open_system(mdl);
```

2 Verify that signal data and task execution time logging are set to `'on'`.

```
get_param(mdl,'SaveTime')

ans =

on

get_param(mdl,'SaveOutput')

ans =

on

get_param(mdl,'RL32LogTETModifier')

ans =

on
```

These parameters are set to `'on'` by default.

3 Verify that **Signal logging buffer size in doubles** is set to a value large enough to accommodate the number of signals that you are logging.

```
get_param(mdl,'RL32LogBufSizeModifier')

ans =

100000
```

The default value of 100000 units is large enough for this model.

4 Save the model under a new name.

```
save_system(mdl,'ex_slrt_outport_osc');
```

Log the Data

1 Build the real-time application.

```
rtwbuild(mdl);
```

2 Start execution.

```
tg = slrt;
tg.stoptime = 1;
start(tg);
```

3 Stop execution.

```
stop(tg);
```

Download and Plot the Data

1 Download and plot the logged times and output values from the target computer. In the Command Window, type:

```
tg = slrt;
timelog = tg.TimeLog;
outputlog = tg.OutputLog;
plot(timelog, outputlog)
```

2 Download and plot the task execution times for the target computer. In the Command Window, type:

```
tetlog = tg.TETLog;
plot(timelog, tetlog)
```

The plot shown is the result of a real-time execution.

3   In the Command Window, type:

```
tg.AvgTET
```

```
ans =
```

```
  5.7528e-006
```

The percentage of CPU performance is the average TET divided by the sample time.

For a single-rate model, task execution time (TET) measures how long it takes the kernel to update the model state, propagate the new model state to the outputs, and log the

model state during a sample interval. For a multirate model, TET measures how long it takes the kernel to perform those tasks for the base-rate portion only. You can plot the TET to show under what circumstances subsystems are executed and the additional CPU time required for those executions.

See Also

Real-Time Application Properties | `SimulinkRealTime.target.getlog`

Related Examples

- "Simulate Simulink Model Using MATLAB Language"
- "Log Signal Data with Outport Block and Simulink Real-Time Explorer" on page 5-92

More About

- "Signal Logging Buffer Size" on page 5-105

Signal Logging Buffer Size

Your real-time application sets aside a buffer for data logging. You specify the buffer size in the **Code Generation** > **Simulink Real-Time Options** pane of the Configuration Parameters dialog box. Set **Signal logging buffer size in doubles** to a value large enough to accommodate the logged signals.

The default buffer size is `100000` units (`800000` bytes). Specify only the number of units that you need. Memory dedicated to data logging is not available for scopes and other Simulink Real-Time features.

The Simulink Real-Time software calculates the number of samples `N` for a signal using this formula:

```
N = Buffer size in doubles / Logged signals
```

In this equation, `Logged signals`, the number of logged signals, breaks down as follows:

- `1` for time
- `1` for task execution time
- `1` for each logged output
- `1` for each logged state

The scopes copy the last `N` samples from the log buffer to the target object logs (`tg.TimeLog`, `tg.OutputLog`, `tg.StateLog`, and `tg.TETLog`).

Configure File Scopes Using MATLAB Language

This procedure shows how to trace signals with file scopes using the Simulink model xpcosc as an example. You must have already built and downloaded the real-time application for this model. It also assumes that you are using a serial link.

Note: The signal data file can quickly increase in size. To gauge the growth rate of the file, examine the file size between runs. If the signal data file grows beyond the available space on the disk, the signal data is corrupted.

---

1 Create a target object tg that represents target computer TargetPC1. Type:

```
tg = SimulinkRealTime.target('TargetPC1')
```

2 To get a list of signals, type:

```
tg.ShowSignals = 'on'
```

The Command Window displays a list of the target object properties for the available signals. For example, these signals are part of the model xpcosc:

```
Target: TargetPC1
   Connected           = Yes
   Application         = xpcosc
.
.
.
   Scopes              = 1
   NumSignals          = 7
   ShowSignals         = on
   Signals             =
     INDEX   VALUE      Type    BLOCK NAME        LABEL
     0       0.000000   DOUBLE  Gain
     1       0.000000   DOUBLE  Gain1
     2       0.000000   DOUBLE  Gain2
     3       0.000000   DOUBLE  Integrator
     4       0.000000   DOUBLE  Integrator1
     5       0.000000   DOUBLE  Signal Generator
     6       0.000000   DOUBLE  Sum
.
.
.
```

3   Start running your real-time application. Type:

```
start(tg)
```

4   Create a scope to be displayed on the target computer. For example, to create a scope with an identifier of 2 and a scope object name of sc2, type:

```
sc2 = addscope(tg, 'file', 2)
```

No name is initially assigned to FileName. After you start the scope, Simulink Real-Time assigns a name for the file to acquire the signal data. This name typically consists of the scope object name, ScopeId, and the beginning letters of the first signal added to the scope.

```
sc2 =

Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId             = 2
   Status              = Interrupted
   Type                = File
   NumSamples          = 250
   NumPrePostSamples   = 0
   Decimation          = 1
   TriggerMode         = FreeRun
   TriggerSignal       = -1
   TriggerLevel        = 0.000000
   TriggerSlope        = Either
   TriggerScope        = 2
   TriggerSample       = 0
   FileName            = unset
   WriteMode           = Lazy
   WriteSize           = 512
   AutoRestart         = off
   DynamicFileName     = off
   MaxWriteFileSize    = 536870912
   Signals             = no Signals defined
```

5   Add signals to the scope object. For example, to add Integrator1 and Signal Generator, type:

```
addsignal(sc2, [4,5])

sc2 =
```

```
Simulink Real-Time Scope
   Application          = xpcosc
   ScopeId              = 2
   Status               = Interrupted
   Type                 = File
.
.
.
   FileName             = unset
   WriteMode            = Lazy
   WriteSize            = 512
   AutoRestart          = off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = 4  : Integrator1
                          5  : Signal Generator
```

The target computer displays the following messages:

```
Scope: 2, signal 4 added

Scope: 2, signal 5 added
```

After you add signals to a scope object, the file scope does not acquire signal values until you start the scope.

6   **Caution** Before starting the scope, copy previously acquired data to the development computer. When the file scope starts, the software overwrites previously acquired data in files of the specified name or name pattern. A partially overwritten file or a file opened but left unwritten loses its original contents.

Start the scope. For example, to start scope `sc2`, type:

```
start(sc2)
```

The Command Window displays a list of the scope object properties. `FileName` is assigned a default file name to contain the signal data for the file scope. This name typically consists of the scope object name, `ScopeId`, and the beginning letters of the first signal added to the scope.

```
Application= xpcosc
   ScopeId    = 2
   Status     = Pre-Acquiring
```

```
   Type       = File
.
.
.
   FileName   = c:\sc2Integ.dat
   Mode       = Lazy
   WriteSize  = 512
   AutoRestart= off
   DynamicFileName      = off
   MaxWriteFileSize     = 536870912
   Signals              = 4  : Integrator1
                          5  : Signal Generator
```

7   Stop the scope. Type:

```
stop(sc2)
```

8   Stop the real-time application. In the Command Window, type:

```
stop(tg)
```

See Also
```
plot | SimulinkRealTime.fileSystem |
SimulinkRealTime.utils.getFileScopeData
```

Related Examples
- "Using SimulinkRealTime.fileSystem Objects" on page 12-4
- "Monitor Signals Using MATLAB Language" on page 5-6

Tune Parameters Using Simulink Real-Time Explorer

You can use Simulink Real-Time Explorer to change parameters in your real-time application while it is running or between runs. You do not need to rebuild the Simulink model, set the Simulink interface to external mode, or connect the Simulink interface with the real-time application.

Using model xpcosc, complete the setup tasks in "Create Host Scopes Using Simulink Real-Time Explorer" on page 5-54.

1 Select the real-time application in the **Applications** pane (for example, **xpcosc**).

2 To start execution, click the real-time application and then click the **Start** button on the toolbar.

3 To start **Scope 1**, click **Scope 1** in the **Scopes** pane and then click the **Start Scope** button on the toolbar.
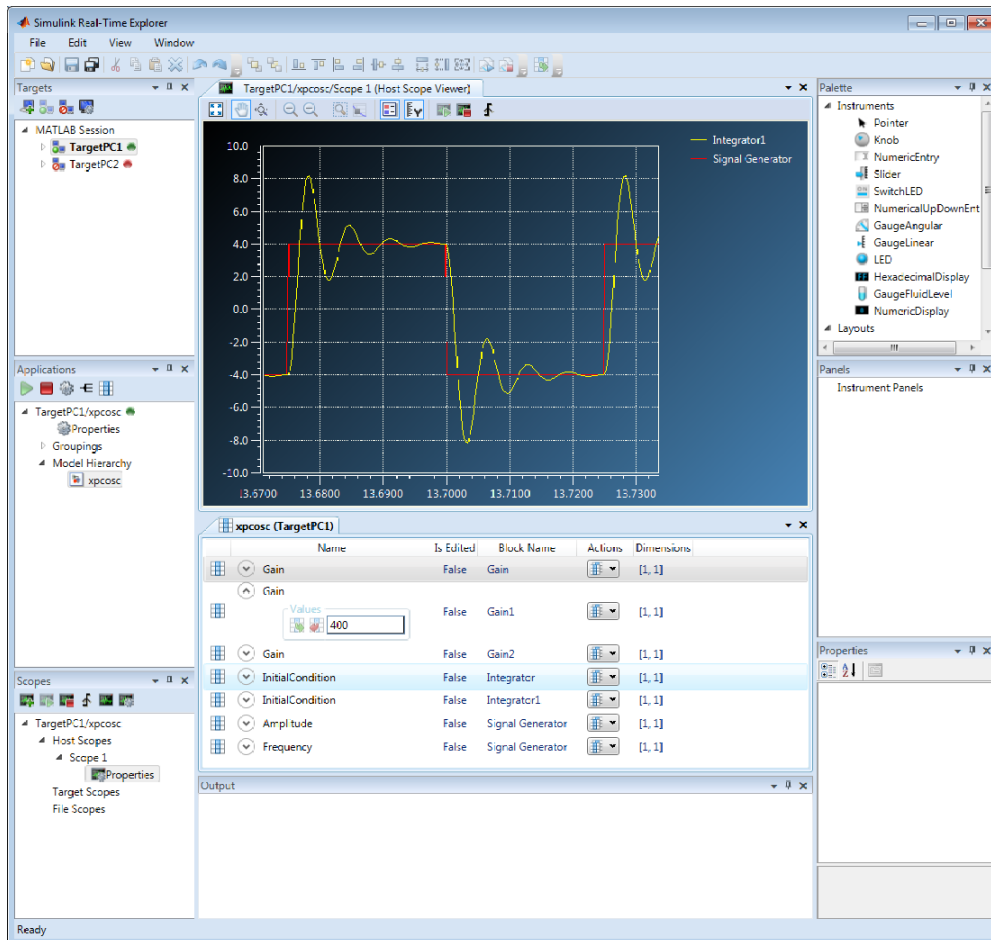
4 In the **Applications** pane, expand both the real-time application node and node **Model Hierarchy**.

5 Select the model node, and then click the **View Parameters** button ⊞ on the toolbar.

The Parameters workspace opens, showing a table of parameters with properties and actions.

6 Click the arrow next to the `Gain` for block `Gain1`. The **Values** text box opens, containing the initial value `400`.

7 Type `100` into the text box, and then press **Enter**.

To revert the `Gain` for block `Gain1` to its previous value, click the **Revert** button 💔.

8 Click the **Apply parameter value(s) changes** button 🖱️.

Simulink Real-Time Explorer looks like this figure.
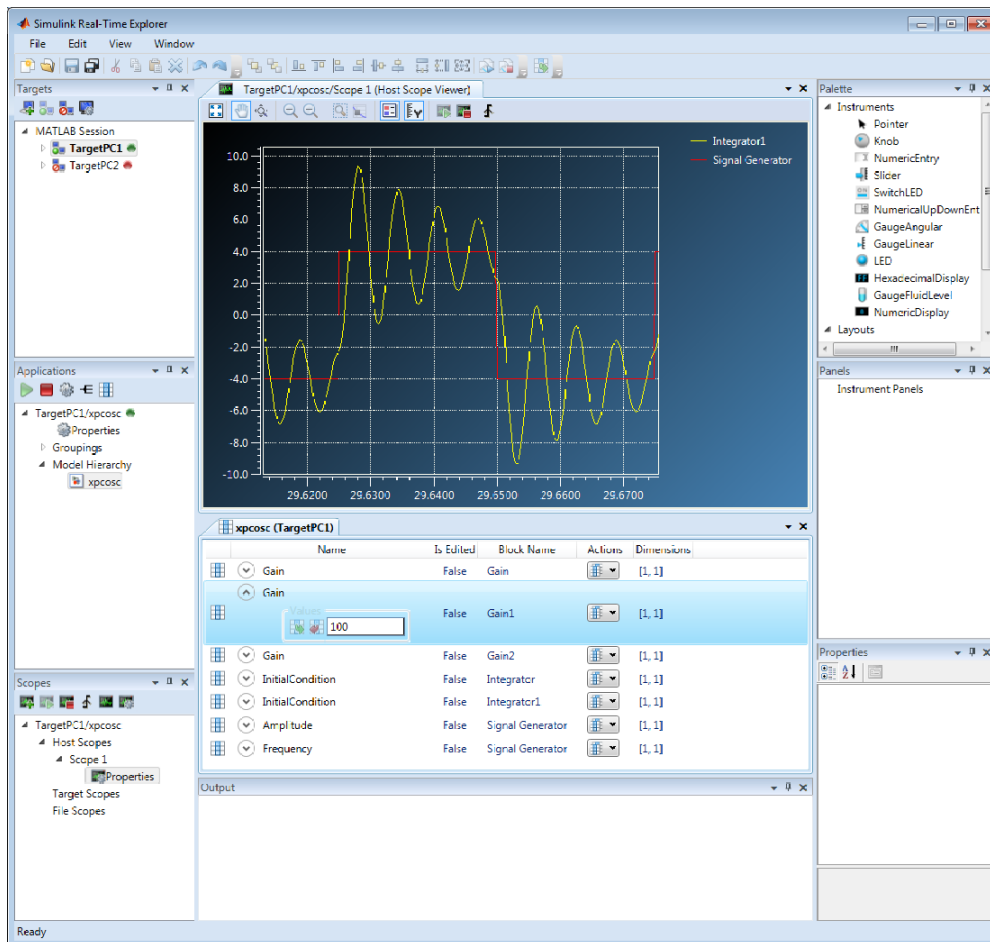
9   To stop **Scope 1**, click **Scope 1** in the **Scopes** pane, and then click the **Stop Scope** button ![icon] on the toolbar.

10 To stop execution, click the real-time application, and then click the **Stop** button ![icon] on the toolbar.

Simulink Real-Time does not support parameters of multiword data types.

To make both workspaces visible at the same time, drag one workspace tab down until the ![icon] icon appears in the middle of the dialog box. Continue to drag until the cursor reaches the required quadrant, and then release the mouse button.

To save your Simulink Real-Time Explorer layout, click **File** > **Save Layout**. In a later session, you can click **File** > **Restore Layout** to restore your layout.

Related Examples

•   "Create Parameter Groups Using Simulink Real-Time Explorer" on page 5-114

More About

•   "Nonobservable Parameters" on page 5-137

Create Parameter Groups Using Simulink Real-Time Explorer

When testing a complex model composed of many reference models, you tune parameters from multiple parts and levels of the model. To do so, create a parameter group.

This procedure uses the model xpcosc as an example. You must have already completed the following setup:

1 Built and downloaded the real-time application to the target computer using Simulink (⊞ on the toolbar).

2 Run Simulink Real-Time Explorer (**Tools** > **Simulink Real-Time**).

3 Connected to the target computer in the **Targets** pane (⊡ on the toolbar).

To create a parameter group:

1 In the **Applications** pane, expand the real-time application node, and then right-click the **Groupings** node.

2 Click **New Parameter Group**.

3 In the Add New Parameter Group Item dialog box, enter a name in the **Name** text box (for example, **ParamGroup1.par**). In the **Location** text box, enter a folder for the group file.

4 Click **OK**. A new parameter group appears, along with its Parameter Group workspace.

5 In the **Applications** pane, expand both the real-time application node and the node **Model Hierarchy**.

6 Select the model node, and then click the **View Parameters** button ⊞ on the toolbar.

The Parameters workspace opens, showing a table of parameters with properties and actions.

7 In the Parameters workspace, to add parameter Amplitude to **ParamGroup1.par**, drag parameter Amplitude to the **ParamGroup1.par** properties workspace.

8 Add parameter Frequency to **ParamGroup1.par** in the same way.

9
   Press **Enter**, and then click the **Save** button 🖫 on the toolbar.
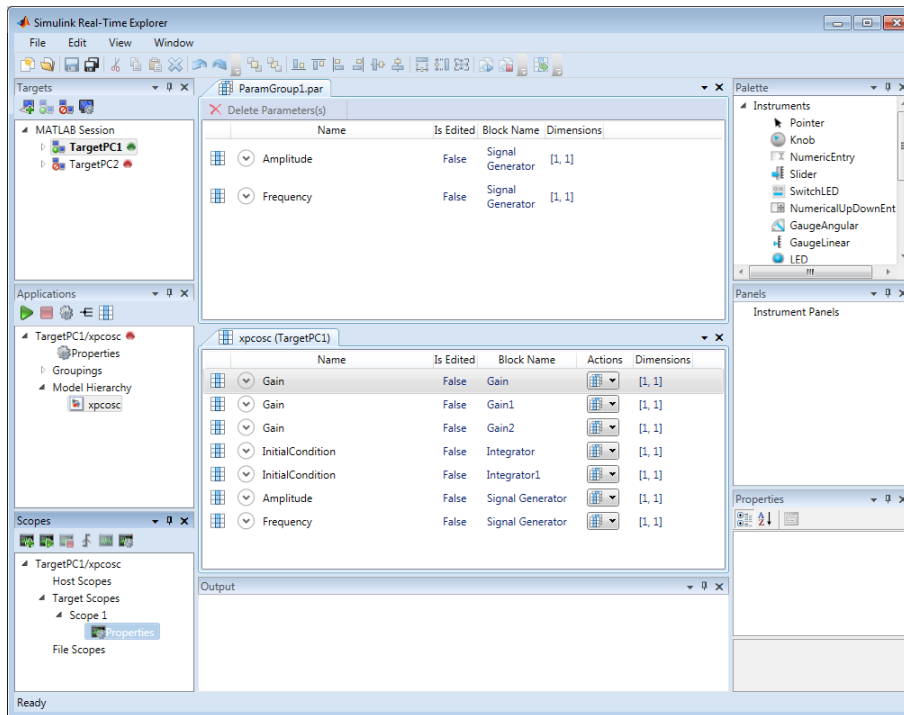
Parameters are defined within a particular real-time application. To open a parameter group from the **File** > **Open** > **Group** menu, you must first select an application.

To remove parameters from the parameter group, select the parameter items in the group list and click **Delete Parameters**.

To remove the parameter group, navigate to the parameter group under **Groupings** > **Parameters**, right-click the parameter group, and click **Remove**.

To make both workspaces visible at the same time, drag one workspace tab down until the icon appears in the middle of the dialog box. Continue to drag until the cursor reaches the required quadrant, and then release the mouse button.

To save your Simulink Real-Time Explorer layout, click **File** > **Save Layout**. In a later session, you can click **File** > **Restore Layout** to restore your layout.

Related Examples

- "Tune Parameters Using Simulink Real-Time Explorer" on page 5-110

Tune Parameters Using MATLAB Language

You use the MATLAB functions to change block parameters. With these functions, you do not need to set the Simulink interface to external mode. You also do not need to connect the Simulink interface with the real-time application.

You can download parameters to the real-time application while it is running or between runs. You can change parameters in your real-time application without rebuilding the Simulink model and change them back to their original values. using Simulink Real-Time functions.

Note:

· Simulink Real-Time does not support parameters of multiword data types.

· Parameter access by parameter index will be removed in a future release. Access parameters by parameter name instead.

· Method names are case-sensitive and must be complete. Property names are not case-sensitive and do not need to be complete, as long as they are unique.

This procedure uses the Simulink model xpcosc as an example. You must have already created and downloaded the real-time application to the default target computer.

1 In the Command Window, type:

```
tg = slrt;
```

```
start(tg)
```

The target computer displays the following message:

```
System: execution started (sample time: 0.001000)
```

2 Display a list of parameters. Type:

```
tg.ShowParameters = 'on'
```

The ShowParameters command displays a list of properties for the target object.

```
Target: TargetPC1
   Connected  = Yes
```

```
   Application= xpcosc
.
.
.
   NumParameters        = 7
   ShowParameters       = on
   Parameters =

      VALUE    TYPE    SIZE     PARAMETER NAME     BLOCK NAME
      1000000  DOUBLE  Scalar   Gain               Gain
      400      DOUBLE  Scalar   Gain               Gain1
      1000000  DOUBLE  Scalar   Gain               Gain2
      0        DOUBLE  Scalar   InitialCondition   Integrator
      0        DOUBLE  Scalar   InitialCondition   Integrator1
      4        DOUBLE  Scalar   Amplitude          Signal Generator
      20       DOUBLE  Scalar   Frequency          Signal Generator
```

3   Change the gain. For example, to change the Gain1 block, type:

```
pt = setparam(tg, 'Gain1', 'Gain', 800)
```

The setparam method returns a structure that stores the source information, the previous value, and the new value.

When you change parameters, the changed parameters in the target object are downloaded to the real-time application. The development computer displays the following message:

```
pt =

     Source: {'Gain1'  'Gain'}
   OldValues: 400
   NewValues: 800
```

The real-time application runs. The plot frame updates the signals for the active scopes.

4   Stop the real-time application. In the Command Window, type:

```
stop(tg)
```

5   To reset to the previous values, type:

```
pt = setparam(tg, pt.Source{1}, pt.Source{2}, pt.OldValues)

pt =
```

```
      Source: {'Gain1'  'Gain'}
   OldValues: 800
   NewValues: 400
```

More About

· "Nonobservable Parameters" on page 5-137

Tune Parameters Using Simulink External Mode

You use Simulink external mode to connect your Simulink model to your real-time application. The model becomes a user interface to your real-time application. You set up the Simulink interface in external mode to establish a communication channel between your Simulink model and your real-time application.

In Simulink external mode, when you change parameters in the Simulink model, Simulink downloads those parameters to the real-time application while it is running. You can change parameters in your program without rebuilding the Simulink model to create a new real-time application.

Note: Simulink Real-Time does not support parameters of multiword data types.

After you download your real-time application to the target computer, you can connect your Simulink model to the real-time application. This procedure uses the Simulink model xpcosc as an example. You must have already built and downloaded the real-time application for that model.

1   In the Simulink editor, click **Simulation** > **Mode** > **External**. A check mark appears next to the menu item **External**, and Simulink external mode is activated.

2   Click the **Run** button ▶ on the toolbar.

    The real-time application begins running on the target computer, and the target computer displays the following message:

    `System: execution started (sample time: 0.000250)`

3   From the Simulation block diagram, double-click the block labeled **Gain1**

4   In the Block Parameters: Gain1 parameter dialog box, the **Gain** text box, enter 800. Click **OK**.

    When you change a model parameter and click **OK**, the changed parameters in the model are downloaded to the real-time application.

5   From the **Simulation** menu, click **Disconnect from Target**.

    The Simulink model is disconnected from the real-time application. If you then change a block parameter in the Simulink model, the real-time application does not change.

6   In the Command Window, type:

```
tg = slrt('TargetPC1')

stop(tg)
```

More About

- "Nonobservable Parameters" on page 5-137

Save and Reload Parameters Using MATLAB Language

After you have a set of real-time application parameter values, save those values to a file on the target computer. You can then later reload these parameter values to the same real-time application.

You can save parameters from your real-time application while the real-time application is running or between runs. You can save and restore parameters in your real-time application without rebuilding the Simulink model. Load parameters to the same model from which you save the parameter file. If you load a parameter file to a different model, the behavior is undefined.

You save and restore parameters with the target object methods `saveparamset` and `loadparamset`.

Requirements:

- You have a real-time application object named `tg`.
- You have assigned `tg` to the target computer.
- You have downloaded a real-time application to the target computer.
- You have parameters to save.

In this section...
"Save the Current Set of Real-Time Application Parameters" on page 5-122
"Load Saved Parameters to a Real-Time Application" on page 5-123
"List the Values of Parameters Stored in a File" on page 5-123

Save the Current Set of Real-Time Application Parameters

To save a set of parameters to a real-time application, use the `saveparamset` method. This example uses the model `ex_slrt_outport_osc`. The real-time application can be stopped or running.

1 Identify the set of parameter values that you want to save.
2 Select a descriptive file name for the parameters. For example, use the model name in the file name.
3 In the Command Window, type:

```
tg = slrt;
saveparamset(tg, 'ex_slrt_outport_osc_param1')
```

The Simulink Real-Time software creates a file named
ex_slrt_outport_osc_param1 in the current folder of the target computer, for
example, C:\ex_slrt_outport_osc_param1.

Load Saved Parameters to a Real-Time Application

To load a set of saved parameters to a real-time application, use the loadparamset
method.

Load parameters to the same model from which you save the parameter file. If you load
a parameter file to a different model, the behavior is undefined. This example uses the
model ex_slrt_outport_osc.

This section assumes that you have a parameters file saved from an earlier run of
saveparamset (see "Save the Current Set of Real-Time Application Parameters" on
page 5-122).

1 From the collection of parameter value files on the target computer, select the one
   that contains the parameter values you want to load.
2 In the Command Window, type:

```
tg = slrt;
```

```
loadparamset(tg, 'ex_slrt_outport_osc_param1')
```

The Simulink Real-Time software loads the parameter values into the real-time
application.

List the Values of Parameters Stored in a File

To list parameters and their values, load the file for a real-time application, and then
turn on the ShowParameters target object property.

You must have a parameters file saved from an earlier run of saveparamset (see "Save
the Current Set of Real-Time Application Parameters" on page 5-122).

1 Stop the real-time application. In the Command Window, type:

```
stop(tg)
```

2 Load the parameter file. Type:

```
tg = slrt;

loadparamset(tg, 'ex_slrt_outport_osc_param1');
```

3 Display a list of parameters. Type:

```
tg.ShowParameters = 'on'
```

The Command Window displays a list of parameters and their values for the target object.

Related Examples

- "Load a parameter set from a file on the designated target file system"
- "Tune Parameters Using Simulink Real-Time Explorer" on page 5-110
- "Tune Parameters Using MATLAB Language" on page 5-117
- "Tune Parameters Using Simulink External Mode" on page 5-120

Parameter Tuning Basics

With Simulink Real-Time, query and modify tunable parameters in your real-time application as it is running, provided the parameters are observable.

Note: Simulink Real-Time does not support parameters of multiword data types.

Parameter tuning uses the C API files generated by Simulink Coder. The Simulink Coder API provides a consistent interface to query and modify tunable parameters from the development computer.

In the code that you generate from a model, there are these kinds of tunable parameters:

- Block parameter — Specify the value as a numeric expression. For example, suppose that you specify the **Gain** parameter of a `Gain` block by the value `5/2`. **Gain** is then a tunable block parameter. Block parameters are observable only when you set the **Default parameter behavior** option to `Tunable`. When using the MATLAB, C, or .NET APIs, you identify a block parameter by the parameter name and the block path in the model hierarchy.

- Model parameter — Workspace variable that you use to specify the values of one or more block parameters in the model. For example, suppose that you specify the **Gain** parameter by the variable `A`. `A` is a `Simulink.Parameter` object that contains the value `3.57`. `A` is then a tunable model parameter. When using the MATLAB, C, or .NET APIs, you identify a model parameter by the variable name only.

Simulink Real-Time provides the same interface to query and modify tunable parameters for both block parameters and model parameters. In this interface, a nonempty block path and a parameter name represent a block parameter. An empty block path (`" "`) and a variable name represent a model parameter.

Model parameters that are structures are observable with Simulink Real-Time. For example, if the **Gain** field of a `Gain` block contains the value `A.Gain`, where `A` is a structure, you can tune that block parameter.

You can improve overall efficiency by setting the **Default parameter behavior** option to `Inlined` in the **Signals and Parameters** pane of the **Optimization** node. By default, you cannot tune inlined parameters. To declare some of the inlined parameters as tunable, create `Simulink.Parameter` objects or use the settings in the Model

5-125

Parameter Configuration dialog box. For more information about inlined parameters, see the Simulink Coder documentation.

See Also

`SimulinkRealTime.target.getparam` | `SimulinkRealTime.target.getparamid` | `SimulinkRealTime.target.setparam`

Related Examples

•   "Configure Model to Tune Inlined Parameters" on page 5-127

More About

•   "Nonobservable Parameters" on page 5-137

Configure Model to Tune Inlined Parameters

This procedure describes how to mark parameters as inline parameters, and then how to specify which of these parameters are tunable model parameters.

Note: Simulink Real-Time does not support parameters of multiword data types.

The following procedure uses the Simulink model `xpcosc` as an example.

1   In the Command Window, type `xpcosc`. The model is displayed in the Simulink Editor.

2   Select the blocks containing the parameters that you want to tune. For example, this procedure makes the `Amplitude` parameter of the `Signal Generator` block tunable. To represent the amplitude, use the variable `A`.

3   Double-click the `Signal Generator` block, and then enter `A` for the `Amplitude` parameter. Click **OK**.

4   In the Command Window, assign a constant to that variable. For example, type:

    A = 4

    The value is displayed in the MATLAB workspace.

5   From the Simulink Editor, click **Simulation** > **Model Configuration Parameters**.

6   In the Configuration Parameters dialog box, select the **Signals and Parameters** node under **Optimization**.

7   In the right pane, set **Default parameter behavior** to `Inlined`.
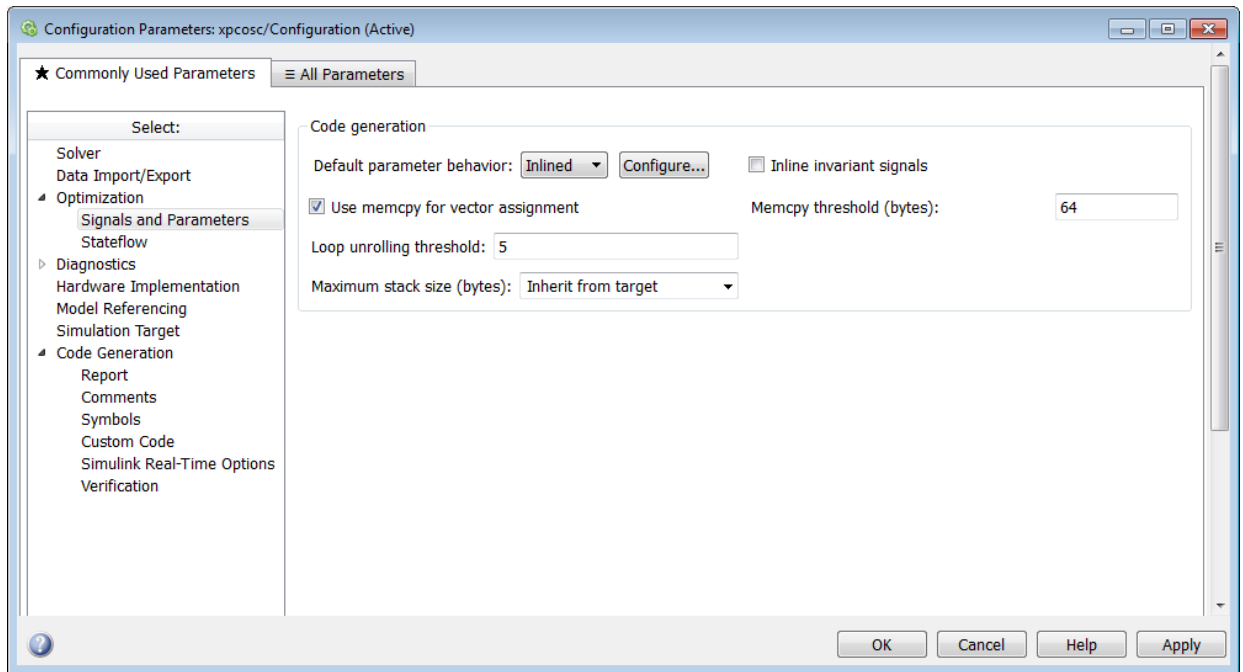
8   Click **Configure**.

The Model Parameter Configuration dialog box opens. The MATLAB workspace contains the constant you assigned to A.

9   Select the line that contains your constant. Click **Add to table**.
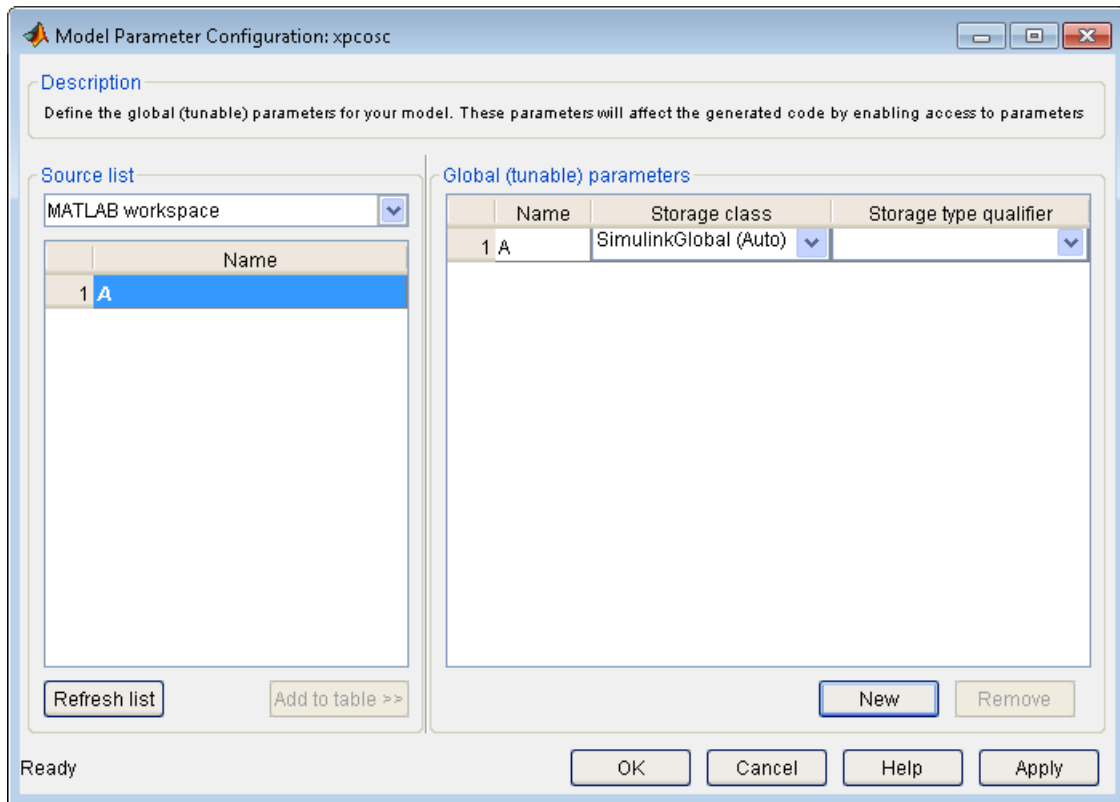
Add the remaining global parameters that you want to tune.

10 Click **Apply**, and then click **OK**.

11 In the Configuration Parameters dialog box, click **Apply**, and then **OK**.

12 Save the model. For example, save it as `ex_slrt_inlined_osc`.

13 Build and download the model to your target computer.

Related Examples

- "Tune Inlined Parameters Using Simulink Real-Time Explorer" on page 5-131
- "Tune Inlined Parameters Using MATLAB Language" on page 5-134

More About

- "Nonobservable Parameters" on page 5-137

Tune Inlined Parameters Using Simulink Real-Time Explorer

This procedure describes how you can tune inlined parameters through the Simulink Real-Time Explorer.

Note: Simulink Real-Time does not support parameters of multiword data types.

The procedure uses the model `ex_slrt_inlined_osc` as an example. You must have already completed the setup tasks in "Create Host Scopes Using Simulink Real-Time Explorer" on page 5-54.

1 Select the real-time application in the **Applications** pane (for example, **ex_slrt_inlined_osc**).

2 To start execution, click the real-time application, and then click the **Start** button ▶ on the toolbar.

3 To start **Scope 1**, click **Scope 1** in the **Scopes** pane, and then click the **Start Scope** button 🔳 on the toolbar.

4 In the **Applications** pane, expand both the real-time application node and the **Model Hierarchy** node.

5 Select the model node, and then click the **View Parameters** button ▦ on the toolbar. The Parameters workspace opens, showing a table of parameters with properties and actions.

6 Click the arrow next to parameter A for block `Model Parameters`. The **Values** text box opens, containing the initial value 4.

The dialog box looks like this figure.
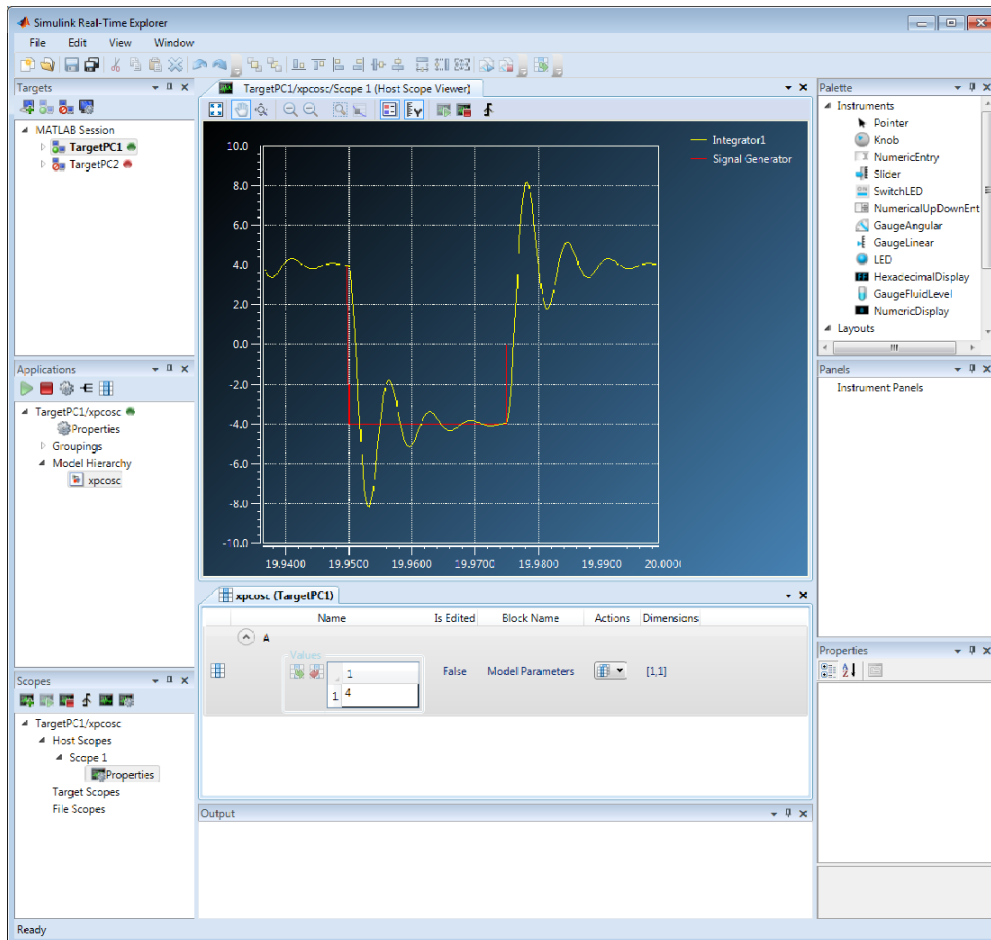
7   Type 2 into the text box, and then press **Enter**.

To revert parameter A for block `Model Parameters` to its previous value, click the **Revert** button 🧰.

8   Click the **Apply parameter value(s) changes** button 🧰.

The dialog box looks like this figure.
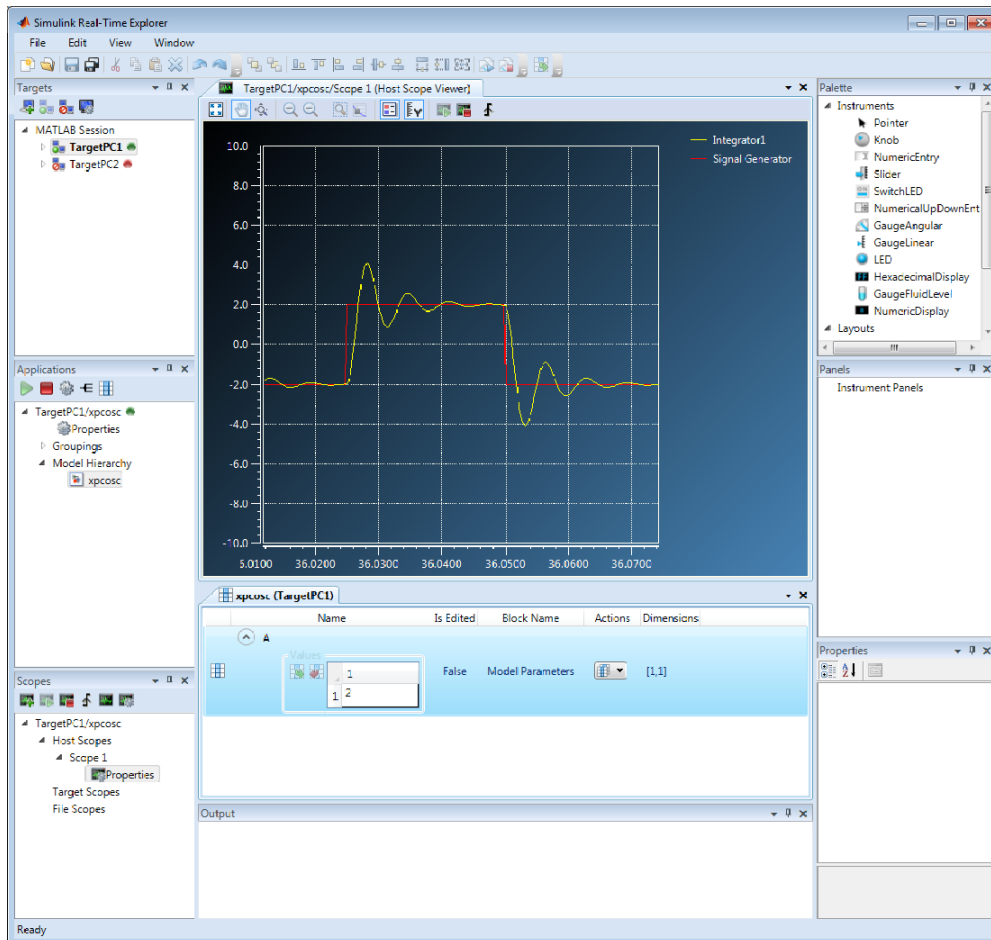
9 To stop **Scope 1**, click **Scope 1** in the **Scopes** pane, and then click the **Stop Scope** button ![icon] on the toolbar.

10 To stop execution, click the real-time application, and then click the **Stop** button ![icon] on the toolbar.

More About

·   "Nonobservable Parameters" on page 5-137

Tune Inlined Parameters Using MATLAB Language

This procedure describes how you can tune inlined parameters through the MATLAB interface. You must have already built and downloaded the model ex_slrt_inlined_osc to the target computer. The model must already be running.

Note: Simulink Real-Time does not support parameters of multiword data types.

You can tune inlined parameters using a parameter ID.

- To get the ID of the inlined parameter that you want to tune, use the getparamid function. For the block_name parameter, leave a blank ('').
- To set the new value for the inlined parameter, use the setparam function .

1 Save the following code in a MATLAB file. For example, change_inlineA.

```
tg = slrt; %Create Simulink Real-Time object

pid = getparamid(tg, '', 'A'); %Get parameter ID of A

if isempty(pid) %Check value of pid.

    error('Could not find A');

end

setparam(tg, pid, 100); %If pid is valid, set parameter value.
```

2 Execute that MATLAB file. Type:

```
change_inlineA
```

3 To see the new parameter value, type:

```
tg.ShowParameters = 'on'
```

The tg object information is displayed, including the parameter lines:

```
NumParameters = 1

ShowParameters = on

Parameters = INDEX   VALUE   TYPE    SIZE   PARAMETER NAME   BLOCK NAME
```

```
         0      100  DOUBLE Scalar A
```

More About

- "Nonobservable Parameters" on page 5-137

Nonobservable Signals

You cannot monitor, trace, or log the following types of signals from the development computer:

- Virtual or bus signals (including signals from bus and virtual blocks). You can access these signals from nonvirtual source blocks.

  - To access a virtual signal, add a `Gain` block with gain 1.0 (unit gain) and observe its output.

  - To access a virtual bus, add a `Gain` block with unit gain to each individual signal.

- Signals that you have optimized by setting the signal storage reuse configuration parameter.

- Signals that you have optimized with block reduction optimization. You can access these signals by making them test points.

- Blocks that buffer their input signals to make them contiguous. Examples include the `To Workspace` block and some `S-function` blocks.

  If you connect a signal to the input port of such a block and branch the same signal to a Simulink Real-Time `Scope` block, the `Scope` block cannot access the signal. To observe such a signal, add a unity `Gain` block before the input to the `Scope` block.

- Signals of complex or multiword data types.

- If a block name consists only of spaces, Simulink Real-Time Explorer does not display a node for signals from that block. To reference such a block:

  - Provide an alphanumeric name for the block

  - Rebuild and download the model to the target computer.

  - Reconnect the MATLAB session to the target computer.

See Also
`Gain`

More About
- "Nonobservable Parameters" on page 5-137
- "Internationalization Issues" on page 5-138

Nonobservable Parameters

Simulink Real-Time does not support parameters of multiword data types.

More About

- "Nonobservable Signals" on page 5-136
- "Internationalization Issues" on page 5-138

Internationalization Issues

Simulink Real-Time inherits the internationalization support of the products it depends upon: Simulink, Simulink Coder, and Embedded Coder®. Signal and parameter names that include Unicode® characters are displayed as expected in Simulink Real-Time Explorer and at the MATLAB command line. In particular, when you use host scopes to observe signals, the non-ASCII signal names are displayed as expected.

Third-party code, such as parsers for vendor configuration files, sometimes does not support cross-locale, cross-platform internationalization. For such code, files and folders must be given locale-specific names. For example, when parsing a configuration file on an English-locale machine, name the file and enclosing folder with English-locale-specific names.

The Simulink Real-Time kernel does not support international (non-ASCII) characters. It generates messages in English using ASCII characters. When interacting with the kernel through the target computer keyboard, you identify signals and parameters by numerical ID, not by name strings.

When you use target scopes to observe signals, the kernel replaces a signal label that includes non-ASCII characters with the numerical ID. It replaces each non-ASCII character in the block path (hierarchical signal name) with the character `?`.

For example, assume that the signal with ID `1` appears in an English-language and a Japanese-language version of the same model. In the English-language version, the signal label is `input1` and the block path is `block1/block2`. In the Japanese-language version, the signal label is 入力1 and the block path is ブロック1/ブロック2.

- In single scope mode, the numerical portion of the screen contains this string for the English-language version:

  `input1: block1/block2`

  It contains this string for the Japanese-language version:

  `1: ????1/????2`

- In multiscope mode, the signal label above the scope contains this string for the English-language version:

  `input1`

  It contains this string for the Japanese-language version:

1

Execution Modes

Execution Modes

The Simulink Real-Time kernel has three mutually-exclusive execution modes. You can execute the real-time application in one non-real-time mode and in two real-time modes.

- Freerun mode — To use this non-real-time mode, on the **Simulink Real-Time Options** pane in the Configuration Parameters dialog box, set **Execution mode** to `Freerun`.

  In this mode, the real-time application thread does not wait for the timer and the kernel runs the application as fast as possible. If the real-time application has conditional code, the time between each execution can vary. Multirate models cannot be executed in `Freerun` execution mode. On the **Solver** pane in the Configuration Parameters dialog box, set **Tasking mode for periodic sample times** to `SingleTasking`. For more information, see "Set Configuration Parameters" and "Execution mode".

- Interrupt mode — To use this real-time mode, on the **Simulink Real-Time Options** pane in the Configuration Parameters dialog box, set **Execution mode** to `Real-Time`.

  In this mode, the scheduler implements real-time single-tasking and multitasking execution of single-rate or multirate systems, including asynchronous events (interrupts). This implementation allows you to interact with the target computer while the real-time application is executing at high sample rates.

- Polling mode — To use this real-time mode, on the **Simulink Real-Time Options** pane in the Configuration Parameters dialog box, set **Execution mode** to `Real-Time`. Then, set a polling rate using a `TLCOptions` setting.

  In this mode, the kernel executes real-time applications at sample times close to the limit of the CPU. Using polling mode with high-speed and low-latency I/O boards and drivers allows you to achieve real-time application sample times that you cannot achieve using the interrupt mode of the Simulink Real-Time software. Because polling mode disables interrupts on the processor core where the model runs, it imposes restrictions on the model architecture and on target communication.

In this section...
"Interrupt Mode" on page 6-3
"Polling Mode" on page 6-4

Interrupt Mode

When you set **Execution mode** to `Real-Time` on the **Simulink Real-Time Options** pane in the Configuration Parameters dialog box, interrupt mode is the real-time execution mode set by default. This mode provides the greatest flexibility. Choose this mode for real-time applications that execute at the given base sample time without overloading the CPU.

In interrupt mode, the scheduler implements real-time single-tasking and multitasking execution of single-rate or multirate systems, including asynchronous events (interrupts). Additionally, background tasks like target communication or updating the target display run in parallel with sample-time-based model tasks. This implementation allows you to interact with the target system while the real-time application is executing at high sample rates. Interaction is made possible by an interrupt-driven real-time scheduler responsible for executing the various tasks according to their priority. The base sample time task can interrupt other tasks (larger sample time tasks or background tasks). Execution of the interrupted tasks resumes as soon as the base sample time task completes operation. This capability gives a quasi parallel priority execution scheme.

In interrupt mode, the kernel is close to optimal for executing code on a PC-compatible system. However, using interrupt mode introduces a constant overhead, or latency, that reduces the minimal possible base sample time to a constant number. The overhead is the sum of various factors related to the interrupt-driven execution scheme. The overhead is referred to as overall interrupt latency. Assuming that the currently executing task is not executing a critical section and has therefore not disabled interrupt sources, the overall latency consists of the following parts:

- Interrupt controller latency — In a PC-compatible system the interrupt controller is not part of the x86-compatible CPU. It is part of the CPU chip set. The controller is accessed over the I/O-port address space, which introduces a read or write latency for each register access.

- CPU latency — Modern CPUs try to predict the next couple of instructions, including branches, using instruction pipelines. If an interrupt occurs, the prediction fails and the pipeline has to be fully reloaded. This process introduces additional latency.

- Interrupt handler entry and exit latency — An interrupt can stop the currently executing task at an arbitrary instruction. When the interrupting task completes execution, the interrupted task has to resume. Its state has to be saved and restored accordingly. The CPU data and address registers, including the stack pointer, must be saved. If the interrupted task executed floating-point unit (FPU) operations, the FPU stack must also be saved. Therefore, additional latency is introduced.

- Interrupt handler content latency — If a background task has been executing for some time, say in a loop, its data is available in cache memory. When an interrupt occurs and the interrupt service handler is executed, the interrupt handler data potentially is purged from the cache. Purging the data from the cache causes the CPU to reload the data from slower RAM. Additional latency is introduced.

The overall latency of interrupt mode is equivalent to a Simulink model containing a hundred nontrivial blocks. Because lower priority tasks have to be serviced as well, at least 5% of headroom is required. This requirement can cause additional cache misses and therefore nonoptimal execution speed.

Polling Mode

Polling mode for the kernel is designed to execute real-time applications at sample times close to the limit of the CPU. Polling mode with high-speed and low-latency I/O boards and drivers allows you to achieve smaller sample times for real-time applications. You cannot achieve these smaller sample times using the interrupt mode of the Simulink Real-Time software.

Polling mode has two main uses:

- Control systems — Control system models of average model size and I/O complexity that are executed at very small sample times (Ts = 5 to 50 µs).
- DSP systems — Sample-based DSP system models (mainly audio and speech) of average model size and I/O complexity that are executed at very high sample rates (Fs = 20 to 200 kHz).

Polling mode for the kernel does not have the constant latency that interrupt mode does. The kernel does not allow interrupts, so the CPU can use this extra time for executing model code.

Polling mode is sometimes seen as a "primitive" or "brute force" real-time execution scheme. When a real-time application executes at a given base sample time in interrupt mode and overloads the CPU, switching to polling mode is often the only alternative for the application to execute at the required sample time.

Polling means that the kernel waits in an empty while loop until the time of the next model step is reached. Then the next model step is executed. A counter must be accessible to the kernel to get a base reference for when the next model step execution must begin. The kernel polls this counter. If the counter must be outside the CPU, for

example in the chip set or on an ISA or PCI board, the counter value can be retrieved only by an I/O or memory access cycle that again introduces latency. This latency consumes the freed-up time of polling mode. Since the introduction of the Pentium CPU family from Intel®, the CPU is equipped with a 64-bit counter on the CPU substrate itself. This counter commences counting at CPU start time and counts up driven by the actual clock rate of the CPU. A highly clocked CPU is unlikely to cause a 64-bit counter to overflow. For a 1 GHz CPU:

overflow_time = 2^{64} * 1e-9 = 584 years

The Pentium counter comes with the following features:

- More precise measurements — Because the counter counts up with the CPU clock rate, time measurements, even in the microsecond range, are very precise, so there are small real-time errors.

- Overflow handler not required— Because the counter is 64 bits wide, overflow does not occur, avoiding the CPU-time overhead of handling overflows.

- Minimal latency — The counter resides on the CPU. Reading the counter value can be done within one CPU cycle, introducing minimal latency.

The polling execution scheme does not depend on interrupt sources to notify the code to continue calculating the next model step. The CPU is freed up. Real-time application code timed by the polling loop is executed as real-time code– even components, which in interrupt mode were executed in background tasks. Do not execute background tasks in polling mode, because these tasks are usually non-real-time tasks and can use a lot of CPU time. To be efficient, execute only the real-time application's relevant parts. For the Simulink Real-Time software, the relevant parts are the code segments that represent the Simulink model itself.

Ethernet links and target display updating are disabled. Polling mode reduces the features of the Simulink Real-Time product to a minimum. Choose this mode only as the last alternative to reach the required base sample time for a given model. Before considering polling mode, do the following:

- Optimize the model execution speed — First, run the model through the Simulink profiler to find possible speed optimizations using alternative blocks. If the model contains continuous states, discretizing these states reduces model complexity significantly. You can avoid a costly fixed-step integration algorithm. If continuous states cannot be discretized, use the integration algorithm with the lowest order that still produces the required numerical results.

- Use the fastest available CPU — Use the CPU with the highest clock rate available for a given target computer form factor. For the desktop form factor, use a CPU with a

clock rate above 3 GHz. For a model of a mobile system (e.g., PC/104 form factor), use a CPU with a clock rate above 1 GHz.

- Use the lowest latency I/O modules and drivers available — Many real-time applications communicate with I/O modules over either an ISA or PCI bus. Each register access introduces a comparably high latency time. Using the lowest latency I/O modules and drivers available is crucial.

Set Polling Mode

Polling mode is an alternative to the default interrupt mode of the kernel. The kernel on the bootable media created by the UI allows running the real-time application in both modes without using another boot disk.

By default, the real-time application executes in interrupt mode. To switch to polling mode, you must set a polling rate using a `TLCOptions` setting.

For a Pentium-class processor, set the polling rate to the target computer CPU clock rate. You can find the CPU clock rate of the target computer by restarting the target computer and checking the screen output during BIOS execution time. The BIOS usually displays the CPU clock rate in MHz right after you start the target computer.

The following example uses `xpcosc`.

1   Open model `xpcosc`.
2   In the Configuration Parameters dialog box, on the **Simulink Real-Time Options** pane, set **Execution mode** to `Real-Time`.
3   In the Command Window, type:

```
set_param('xpcosc','TLCOptions',
          '-axpcCPUClockPoll=CPUClockRateMHz')
```

   For example, if your target computer is a 1.2 GHz AMD Athlon™, type:

```
set_param('xpcosc','TLCOptions','-axpcCPUClockPoll=1200')
```

4   Build and download the real-time application.

After you have downloaded the real-time application, the target display shows the execution mode. If polling mode is activated, it displays the CPU clock rate in MHz. You can check the setting.

If you want to execute the real-time application in interrupt mode again, either remove the setting or assign a CPU clock rate of `0` to the setting:

```
set_param('xpcosc','TLCOptions','')
set_param('xpcosc','TLCOptions','-axpcCPUClockPoll=0')
```

Restrictions on Single- and Multicore Processors

Polling mode runs best on a multicore processor target computer with multicore processing enabled. For more information, see "Multicore Processor Configuration" on page 10-11.

Polling mode disables interrupts on the core where the model is running. Background tasks are inactive on this core, including those for the Ethernet link, target display updating, and UDP transfers. Because the model uses only one core of a multicore target computer, interrupts are still enabled on the other cores. Therefore, the only restriction on a multicore computer is for multirate models. See "Multirate Models Cannot Be Executed in Multitasking Mode" on page 6-7.

The following restrictions apply to single-core target computers:

- "Multirate Models Cannot Be Executed in Multitasking Mode" on page 6-7
- "I/O Drivers Cannot Use Kernel Timing Information" on page 6-8
- "Ethernet Link Unavailable" on page 6-8
- "Target Display Does Not Update" on page 6-9
- "Session Time Does Not Advance" on page 6-9
- "Only Data Logging Is Available" on page 6-9

Multirate Models Cannot Be Executed in Multitasking Mode

You cannot execute real-time applications in multitasking mode because of the polling mode execution algorithm. You also cannot model function-call subsystems to handle asynchronous events (interrupts). You can execute multirate systems in single-tasking mode. However, because of the CPU's scheme for sequential execution of subsystems with different rates, the CPU can overload for the given base sample time. Consequently, polling mode is a feasible alternative to interrupt mode only if the model has a single rate or if it can be converted to a single-rate model.

A single-rate model implies one of the following combinations of states:

- Continuous states only
- Discrete states only

- Mixed continuous and discrete states, provided the continuous and discrete subsystems have the same rate

Select Simulink **Display** > **Sample Time** > **Colors** to check for the single rate requirement. To avoid a possible switch to multitasking mode, on the **Solver** pane in the Configuration Parameters dialog box, set the **Tasking Mode** property to `SingleTasking`.

I/O Drivers Cannot Use Kernel Timing Information

Some Simulink Real-Time drivers use timing information exported from the kernel to, for example, detect timeouts. Because the standard timing engine of the kernel does not run in polling mode, the required timing information is not passed back to the drivers. Therefore, in polling mode you cannot use drivers that import the header file `time_xpcimport.h`. In a future version of polling mode, drivers will use the Pentium counter for timing.

Ethernet Link Unavailable

If you start your real-time application execution in polling mode, for example with

```
start(tg)
```

the Ethernet link is disabled throughout the entire run, until the stop time is reached. Each attempt to issue a command like

```
tg
```

results in a communication-related error message. Even the `start(tg)` command returns such an error message, because the development computer does not receive an acknowledgment from the target before timing out. During the entire run, it is best not to issue target-related commands on the development computer.

You cannot issue a `stop(tg)` command to stop the real-time application execution from the development computer. To stop executing in polling mode, the real-time application has to reach its set stop time. Before starting the execution, you can use

```
tg.StopTime = x
```

but once started, execution continues until the stop time is reached.

To stop the execution interactively before it reaches the real-time application stop time, see "Control Real-Time Application on Single-Core Processor" on page 6-10.

If real-time application execution reaches the stop time and polling mode execution is stopped, the Ethernet link begins functioning again. After polling mode execution stops, if you still get communication error messages, type the command:

```
slrtpingtarget
```

to reset the Ethernet link.

After the communication link is working again, type:

```
tg
```

to resynchronize the target object on the development computer with the most current status of the real-time application.

Target Display Does Not Update

Target display updating is disabled during the entire execution of the real-time application. The **Graphics mode** check box in the **Target Properties** pane of Simulink Real-Time Explorer is not operational. Clear the **Graphics mode** check box, thereby producing text output only.

Session Time Does Not Advance

Because interrupts are disabled during a run, the session time does not advance. Therefore, the session time right before and after the run is the same. This same session time does not cause an issue.

Only Data Logging Is Available

Because the Ethernet link and target display updating are disabled during the entire run, these rapid-prototyping features are not available in polling mode:

- Parameter tuning
- Signal monitoring
- Scope objects
- Programs written using the Simulink Real-Time APIs
- The Internet browser interface
- `SimulinkRealTime.target.viewTargetScreen` and similar utilities

The only rapid-prototyping feature available is signal data logging. The signal data is acquired independently of the development computer, and logged data is retrieved

only after the execution is stopped. Logging data gathers useful information about the behavior of the real-time application. Signal logging is an important part of polling mode.

Control Real-Time Application on Single-Core Processor

When you use polling mode on a single-core processor, you cannot interact with the running real-time application until execution reaches its predefined stop time. However, you can use I/O drivers to implement a minimal level of interactivity.

To stop a real-time application, use a low-latency digital input driver for the digital PCI board in your model. The driver reads in a single digital TTL signal, by default TTL low. You change it to TTL high when the model execution must be stopped.

Connect the output port of the digital input driver block to the input port of a `Stop` simulation block, found in the standard Simulink block library. The `Stop` block stops the execution of the real-time application, depending on the state of the digital input signal.

To generate the TTL signal that stops the real-time application, use one of these methods:

- Connect a physical switch to the target computer TTL input pin.
- Install another digital I/O board in the development computer and connect the TTL output of the development computer board to the TTL input of the target computer board. On the development computer, use the Data Acquisition Toolbox™ MATLAB language API to drive the TTL output pin.

You can use the same setup to pass other control information back and forth at real-time application run time. However, you must implement features other than signal logging at the model level. You must minimize the additional latency introduced by the feature. For example, interactively stopping the real-time application execution requires an additional 1 µs latency to read the digital signal over the digital I/O board.

Real-Time Application Execution

7

Execution Using Graphical User Interface Models

You can use the Simulink interface to create a custom graphical user interface (GUI) for your real-time application. To do this, create an user interface model with the Simulink interface and add-on products like Simulink 3D Animation™ or Altia® Design (a third-party product).

Simulink Real-Time Interface Blocks to Simulink Models

In this section...

Simulink User Interface Model

A user interface model is a Simulink model containing Simulink blocks from add-on products and interface blocks from the Simulink Real-Time block library. This user interface model can connect to a custom graphical interface using Simulink 3D Animation or Altia products. The user interface model runs on the development computer and communicates with your real-time application running on the target computer using `To Target` and `From Target` blocks.

The user interface allows you to change parameters by downloading them to the target computer, and to visualize signals by uploading data to the development computer.

Simulink 3D Animation — The Simulink 3D Animation product enables you to display a Simulink user interface model in 3-D. It provides Simulink blocks that communicate with Simulink Real-Time interface blocks. These blocks then communicate to a graphical interface. This graphical interface is a Virtual Reality Modeling Language (VRML) world displayed with a web browser using a VRML plug-in.

Altia Design — Altia also provides Simulink blocks that communicate with Simulink Real-Time interface blocks. These blocks then communicate with Altia's graphical interface or with a web browser using the Altia ProtoPlay plug-in.
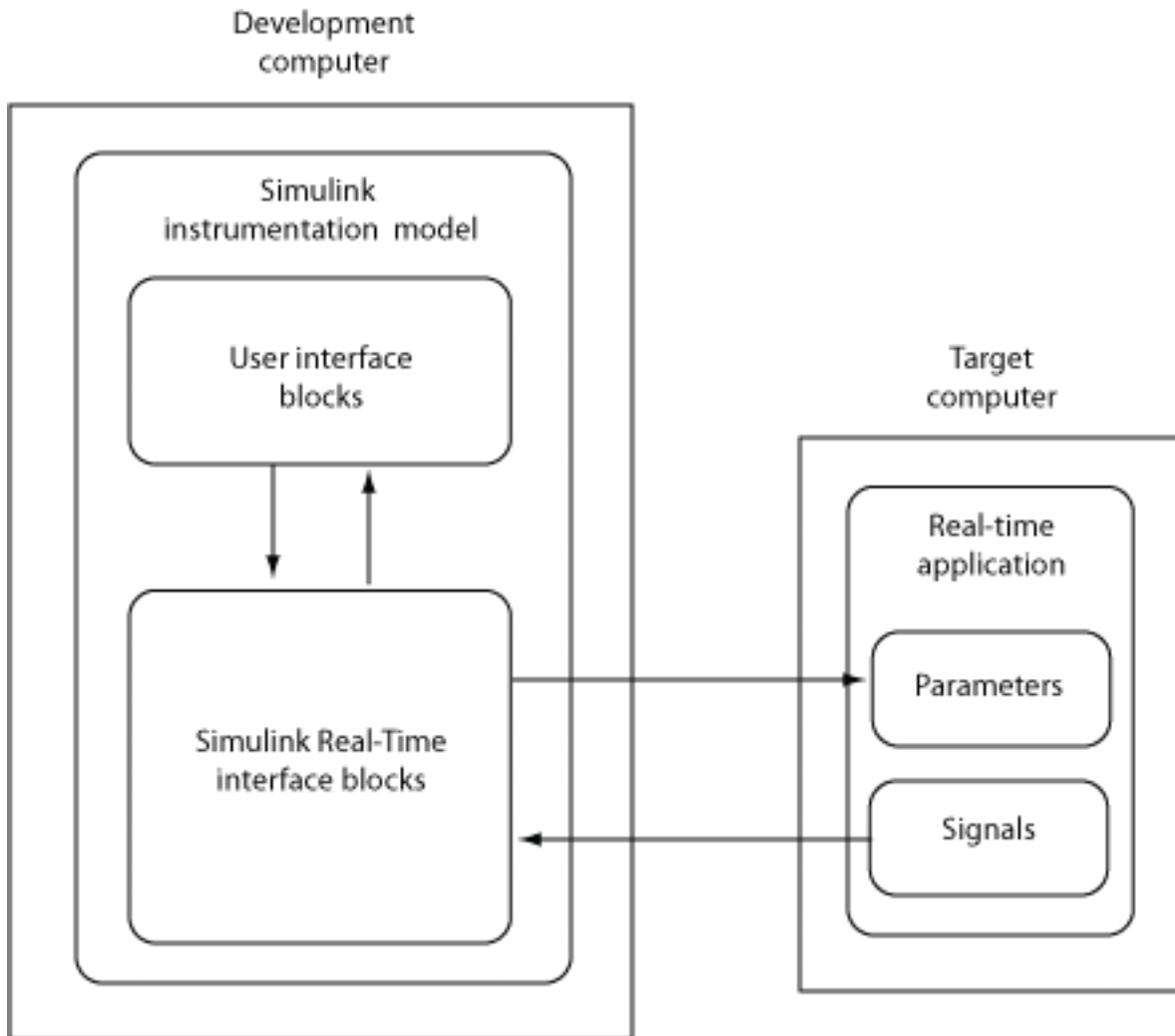
Creating a Custom Graphical Interface

The Simulink Real-Time block library provides Simulink interface blocks to connect graphical interface elements to your real-time application. The steps for creating your own custom user interface are listed below:

1   In the Simulink real-time application model, decide which block parameters and block signals you want to have access to through graphical interface control devices and graphical interface display devices.

2   Tag the block parameters in the Simulink model that you want to be connected to a control device. See "Marking Block Parameters" on page 7-8.

3   Tag the signals in Simulink model that you want to be connected to a display device. See "Marking Block Signals" on page 7-10.

4   In the MATLAB interface, run the function `SimulinkRealTime.utils.createInstrumentationModel` to create the user interface template model. This function generates a new Simulink model containing only the Simulink Real-Time interface blocks (`To Target` and `From Target`) defined by the tagged block parameters and block signals in the real-time application model.

5   To the user interface template model, add Simulink interface blocks from add-on products (Simulink 3D Animation, Altia Design).

   • You can connect Altia blocks to the Simulink Real-Time `To PC Target` interface blocks. `To Target` blocks on the left should be connected to control devices.

   • You can connect Altia and Simulink 3D Animation blocks to the Simulink Real-Time `From Target` interface blocks. `From Target` blocks on the right should be connected to the display devices.

   You can position these blocks to your liking.

6   Start both the real-time application and the Simulink user interface model that represents the application.

To Target Block

This block behaves as a sink and usually receives its input data from a control device. The purpose of this block is to write a new value to a specific parameter in the real-time application.
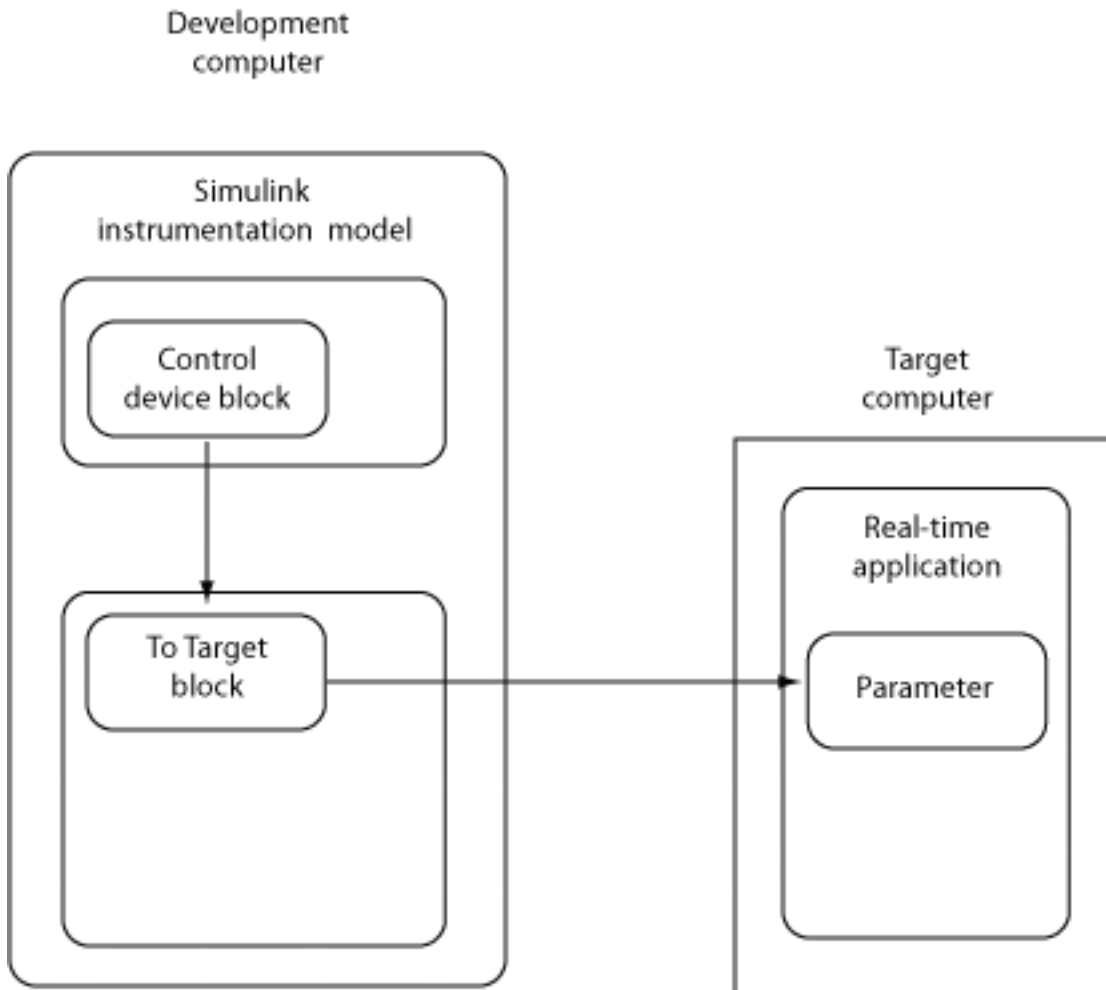
This block is implemented as a MATLAB S-function. The block is optimized so that it only changes a parameter on the real-time application when the input value differs from the value that existed at the last time step. This block uses the parameter downloading feature of the Simulink Real-Time command-line interface. This block is available from the `slrtlib/Displays and Logging` block sublibrary. See `To Target` for further configuration details.

> **Note:** The use of `To Target` blocks requires a connection between the development and target computers. Operations such as opening a model that contains these blocks or copying these blocks within or between models will take significantly longer than normal without a connection between the development and target computers.

`From Target` Block

This block behaves like a source and its output is usually connected to the input of a display device.
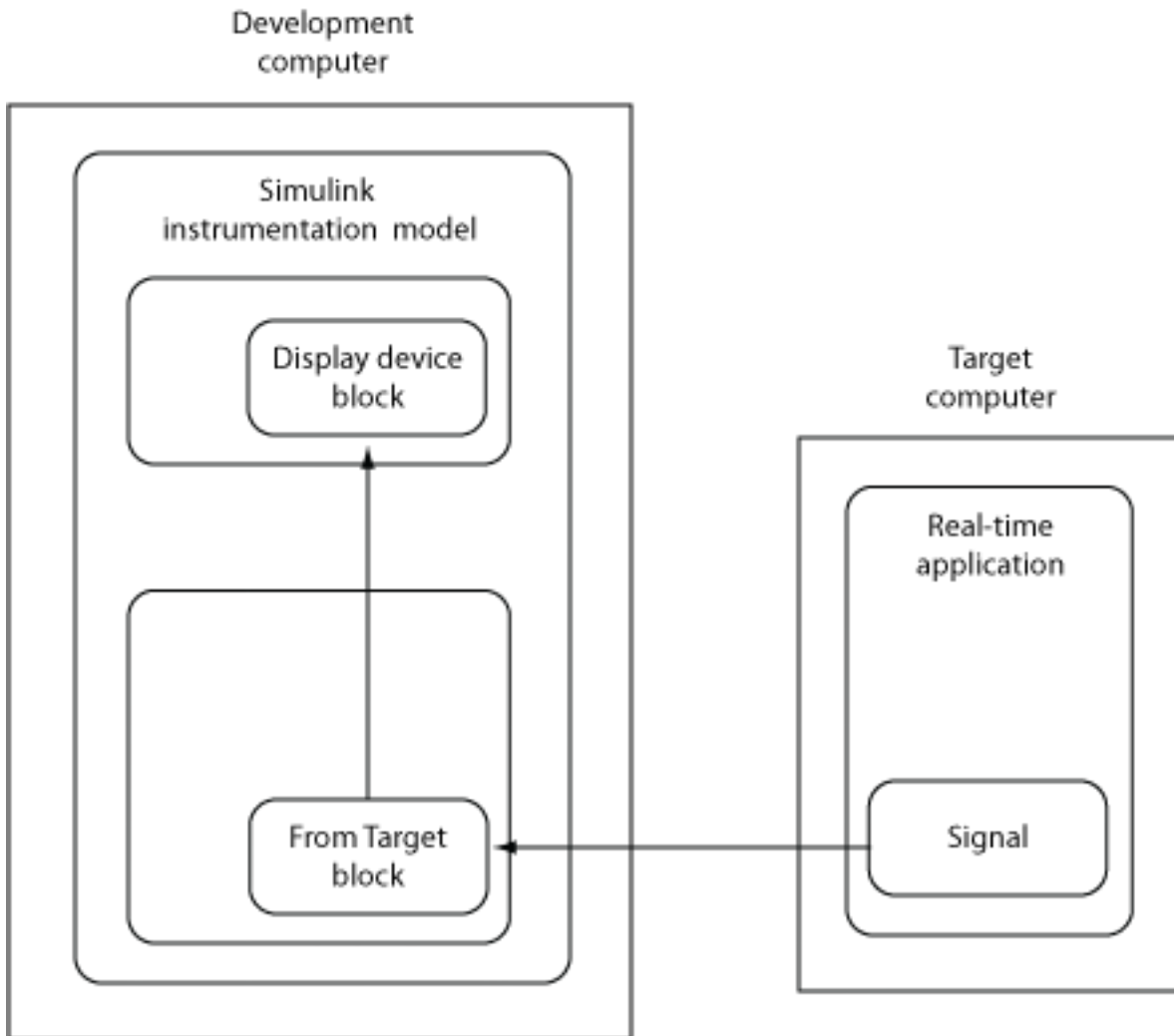
Because only one numerical value per signal is uploaded during a time step, the number of samples of a scope object is set to 1. The block uses the capability of the Simulink Real-Time command-line interface and is implemented as a MATLAB S-function. This block is available from the `slrtlib`/`Displays and Logging` sublibrary. See `From Target` for further configuration details.

Note: The use of `From Target` blocks requires a connection between the development and target computers. Operations such as opening a model that contains these blocks or copying these blocks within or between models will take significantly longer than normal without a connection between the development and target computers.

Creating a Real-Time Application Model

A real-time application model is a Simulink model that describes your physical system, a controller, and its behavior. You use this model to create a real-time application, and you use this model to select the parameters and signals you want to connect to a custom graphical interface.

Creating a real-time application model is the first step you need to do before you can tag block parameters and block signals for creating a custom graphical interface.

See "Marking Block Parameters" on page 7-8 and "Marking Block Signals" on page 7-10 for descriptions of how to mark block properties and block signals.

Marking Block Parameters

Tagging parameters in your Simulink model allows the function `SimulinkRealTime.utils.createInstrumentationModel` to create `To Target` interface blocks. These interface blocks contain the parameters you connect to control devices in your user interface model.

After you create a Simulink model, you can mark the block parameters. This procedure uses the model `xpctank` as an example.

Tip The `xpctank` model blocks and signals may contain placeholder tags illustrating the syntax. As you create your own copy of the model using these procedures, replace these tags with your new tags or add the new tags using the multiple label syntax.

---

1 Open a Simulink model. For example, in the MATLAB Command Window, type

   `xpctank`

2 Point to a Simulink block, and then right-click.

3 From the menu, click **Properties**.

A Block Properties dialog box opens.

4 In the **Description** box, delete the existing tag and enter a tag to the parameters for this block.

For example, the SetPoint block is a constant with a single parameter that selects the level of water in the tank. Enter the tag:

```
xPCTag(1)=water_level;
```

The tag has the following syntax

```
xPCTag(1, . . . index_n)= label_1 . . . label_n;
```

For single dimension ports, the following syntax is also valid:

```
xPCTag=label;
```

index_n -- Index of a block parameter. Begin numbering parameters with an index of 1.

label_n -- Name for a block parameter that will be connected to a `To Target` block in the user interface model. Separate the labels with a space, not a comma.

`label_1...label_n` must consist of the same identifiers as those used by C/C++ to name functions, variables, and so forth. Do not use names like `-foo`.

5 Repeat steps 1 through 3 for the remaining parameters you want to tag.

For example, for the Controller block, enter the tag:

```
xPCTag(1,2,3)=upper_water_level lower_water_level
    pump_flowrate;
```

For the PumpSwitch and ValveSwitch blocks, enter the following tags respectively:

```
xPCTag(2)=pump_switch;
xPCTag(1)=drain_valve;
```

To create the `To Target` blocks in an user interface model for a block with four properties, use the following syntax:

```
xPCTag(1,2,3,4)=label_1label_2label_3label_4;
```

To create the To Target blocks for the second and fourth properties in a block with at least four properties, use the following syntax:

```
xPCTag(2,4)=label_1 label_2;
```

6 From the **File** menu, click **Save as**. Enter a filename for your model. For example, enter

```
xpctank1
```

You next task is to mark block signals if you have not already done so, and then create the user interface template model. See "Marking Block Signals" on page 7-10 and "Creating a Custom Graphical Interface" on page 7-3.

Marking Block Signals

Tagging signals in your Simulink model allows the function `SimulinkRealTime.utils.createInstrumentationModel` to create From Target interface blocks. These interface blocks contain the signals you connect to display devices in your user interface model.

After you create a Simulink model, you can mark the block signals. This procedure uses the model xpctank1 (or xpctank) as an example. See "Creating a Real-Time Application Model" on page 7-8.

Tip The xpctank model blocks and signals may contain placeholder tags illustrating the syntax. As you create your own copy of the model using these procedures, replace these tags with your new tags or add the new tags using the multiple label syntax.

Note that you cannot select signals on the output ports of virtual blocks, such as Subsystem and Mux blocks. Also, you cannot select signals on software-triggered signal output ports.

1 Open a Simulink model. For example, in the MATLAB Command Window, type:

```
xpctank
```

or

```
xpctank1
```

2 Point to a Simulink signal line, and then right-click.

3 From the menu, click **Properties**.

A Signal Properties dialog box opens.

4 Select the **Documentation** tab.

5 In the **Description** box, enter a tag to the signals for this line.

For example, the block labeled TankLevel is an integrator with a single signal that indicates the level of water in the tank. Replace the existing tag with the tag:

```
xPCTag(1)=water_level;
```

The tag has the following format syntax:

```
xPCTag(1, . . . index_n)=label_1 . . . label_n;
```

For single dimension ports, the following syntax is also valid:

```
XPCTag=label:
```

- index_n — Index of a signal within a vector signal line. Begin numbering signals with an index of 1.
- label_n — Name for a signal that will be connected to a From Target block in the user interface model. Separate the labels with a space, not a comma.

label_1...label_n must consist of the same identifiers as those used by C/C++ to name functions, variables, and so forth. Do not use names like -foo.

To create the From Target blocks in an user interface model for a signal line with four signals (port dimension of 4), use the following syntax:

```
xPCTag(1,2,3,4)=label_1 label_2 label_3 label_4;
```

To create the From Target blocks for the second and fourth signals in a signal line with at least four signals, use the following syntax:

```
xPCTag(2,4)=label_1 label_2;
```

Note: Only tag signals from nonvirtual blocks. Virtual blocks are only graphical aids (see "Nonvirtual and Virtual Blocks"). For example, if your model combines two signals into the inputs of a Mux block, do not tag the signal from the output

7-11

of the Mux block. Instead, tag the source signal from the output of the originating nonvirtual block.

6 From the **File** menu, click **Save as**. Enter a filename for your model. For example, enter

```
xpc_tank1
```

You next task is to mark block parameters if you have not already done so. See "Marking Block Parameters" on page 7-8. If you have already marked block signals, return to "Creating a Custom Graphical Interface" on page 7-3 for additional guidance on creating a user interface template model.

8

Execution Using the Target Computer Command Line

Control Real-Time Application at Target Computer Command Line

The Simulink Real-Time software provides a set of commands that you can use to interact with the real-time application after it has been loaded to the target computer, to configure and control the scopes for that application, and to tune parameters.

These commands are particularly useful with standalone real-time applications that are not connected to the development computer. You type commands directly from a keyboard attached to the target computer. As you start to type, a command window appears on the target computer screen.

The target computer commands are case sensitive, but the arguments are not. For more information, see "Target Computer Commands"

Trace Signals at Target Computer Command Line

After you have built and downloaded a real-time application to the target computer, you can use target computer commands to run the application and to create and configure scopes.

To add signals to a scope, you must specify the signals by signal number. For more information, see "Find Signal and Parameter Indexes" on page 8-7.

1   To start the real-time application, in the command line, type:

```
start
```

2   To add a target scope (scope 2), type

```
addscope 2
```

The Simulink Real-Time software adds another scope graphic to the target computer monitor. The command window displays a message to indicate that the new scope has registered.

```
Scope 2, created, type is target
```

3   To add a signal (0) to the new scope, type:

```
addsignal 2=0
```

The command window displays a message to indicate that the new signal has registered.

```
Scope 2, signal 0 added
```

You can add more signals to the scope.

4   To start scope 2, type:

```
startscope 2
```

The target scope 2 starts and displays the signals you added in the default format (graphical).

If you add a target scope from the target computer, you must start that scope manually. If a target scope is in the model, starting the real-time application starts that scope automatically.

5   To collapse scope 2 into an icon, type:

```
hide Scope 2
```

6 To expand scope **2** from an icon, type:

```
show Scope 2
```

7 To check the value of signal **0**, type:

```
s0
```

The command window displays a message to show the value of signal **0**.

```
S0 has value 5.1851
```

8 To stop scope **2**, type:

```
stopscope 2
```

9 To change the number of samples (to **1000**) to acquire in scope **2**, type:

```
numsamples 2=1000
```

You must stop the scope before changing a scope parameter.

10 To start scope **2**, type:

```
startscope 2
```

The target scope **2** starts and displays the signal values with the updated sample count.

11 To stop scope **2**, type:

```
stopscope 2
```

12 To stop the real-time application, type:

```
stop
```

Tune Parameters at Target Computer Command Line

After you have built and downloaded a real-time application to the target computer, you can use target computer commands to tune parameters.

To tune parameters, you must specify them by parameter number. For more information, see "Find Signal and Parameter Indexes" on page 8-7.

1 To check the frequency of the signal generator (parameter 6) of the model `xpcosc`, type:

```
p6
```

The command window displays a message to indicate that the new parameter has registered.

```
p[6] is set to 20.00000
```

2 To change the frequency of the signal generator, type:

```
setpar 6=30
```

The command window displays a message to indicate that the new parameter has registered.

```
p[6] is set to 30.00000
```

The target computer command `setpar` does not work for vector parameters.

3 To change the stop time to 1000, type:

```
stoptime = 1000
```

The parameter changes are made to the real-time application but not to the target object. When you type a Simulink Real-Time command in the MATLAB Command Window, the target computer returns the current properties of the target object.

Alias Commands at Target Computer Command Line

You can use target computer command line variables to tag (or alias) unfamiliar commands, parameter indexes, and signal indexes with more descriptive names.

1 To create the aliases on and off for a parameter (7) that controls a motor, type

```
setvar on = p7 = 1
setvar off = p7 = 0
```

The target computer command window is activated when you start to type, and a command line opens.

2 Type the variable name to run that command sequence. For example, to turn the motor on, type

```
on
```

The parameter P7 is changed to 1, and the motor turns on.

Find Signal and Parameter Indexes

To find signal and parameter indexes using MATLAB language:

1 Build and download the model to the target computer.

2 At the Command Line, type:

```
tg = slrt

Target: TargetPC1
   Connected       = Yes
   Application     = xpcosc
.
.
.
   Scopes          = No Scopes defined
   NumSignals      = 7
   ShowSignals     = off

   NumParameters   = 7
   ShowParameters  = off
```

3 To display signal numbers, type:

```
tg.ShowSignals='on'

Target: TargetPC1
   Connected       = Yes
   Application     = xpcosc
.
.
.
   Scopes          = No Scopes defined
   NumSignals      = 7
   ShowSignals     = on
   Signals         = INDEX   VALUE          BLOCK NAME          . . .
                     0       0.000000       Gain                . . .
                     1       0.000000       Gain1               . . .
                     2       0.000000       Gain2               . . .
                     3       0.000000       Integrator          . . .
                     4       0.000000       Integrator1         . . .
                     5       0.000000       Signal Generator  . . .
                     6       0.000000       Sum                 . . .
```

```
NumParameters     = 7
ShowParameters    = off
```

Use the `Signals INDEX` number in target computer commands such as `addsignal`.

4 To display parameter numbers, type:

```
tg.ShowParameters='on'

Target: TargetPC1
   Connected        = Yes
   Application      = xpcosc
   .
   .
   .
   NumParameters     = 7
   ShowParameters    = on
   Parameters        = INDEX    VALUE    . . .  PARAMETER NAME    . . .
                       0        1000000 . . .  Gain              . . .
                       1        400     . . .  Gain              . . .
                       2        1000000 . . .  Gain              . . .
                       3        0       . . .  InitialCondition . . .
                       4        0       . . .  InitialCondition . . .
                       5        4       . . .  Amplitude         . . .
                       6        20      . . .  Frequency         . . .
```

Use the `Parameters INDEX` number in target computer commands such as `setpar`.

For more information about using signal and parameter numbers, see "Target Computer Commands".

Execution Using the Web Browser Interface

Web Browser Interface

The Simulink Real-Time software includes a web server that allows you to interact with your real-time application through a web browser running on a development computer.

Note: The web interface to the target application will be removed in a future release. Use Simulink Real-Time Explorer instead.

You access the web browser through an Ethernet connection using the Simulink Real-Time web server.

In this section...
"Connecting the Web Interface Through TCP/IP" on page 9-2
"Using the Main Pane" on page 9-3
"Changing WWW Properties" on page 9-6
"Viewing Signals with a Web Browser" on page 9-6
"Trace Signals Using a Web Browser" on page 9-7
"Log Signals Using a Web Browser" on page 9-8
"Tune Parameters Using a Web Browser" on page 9-9
"Changing Access Levels to the Web Browser" on page 9-9

Connecting the Web Interface Through TCP/IP

If your development computer and target computer are connected with a network cable, you can connect the real-time application on the target computer to a web browser on the development computer.

Note: RS-232 communication type has been removed. Configure TCP/IP communication instead.

The TCP/IP stack on the Simulink Real-Time kernel supports only one simultaneous connection, because its main objective is real-time applications. This connection is shared

between the MATLAB interface and the web browser. This shared connection means that only one browser or the MATLAB interface can connect at one time. Close the other connections to the target computer. For example, if you are currently connected to the target computer through Simulink Real-Time Explorer, right-click that target computer icon and select **Disconnect** or click the Disconnect button on the toolbar.

Before you connect your web browser on the development computer, load a real-time application onto the target computer. The real-time application does not have to be running, but it must be loaded. Also, your browser must have JavaScript® and style sheets turned on.

1   In the Command Window, type

     `close(slrt)`

     The MATLAB interface is disconnected from the target computer, and the connection is reset for connecting to another client. If you do not use this command immediately before opening the web interface, your browser might not be able to connect to the target computer.

2   Open a web browser. In the address box, enter the IP address and port number you entered in Simulink Real-Time Explorer. For example, if the target computer IP address is 192.168.0.10 and the port is 22222, type:

     `http://192.168.0.10:22222/`

     The browser loads the Simulink Real-Time web interface frame and panes.

Using the Main Pane

The **Main** pane is divided into four parts, one below the other. The four parts are **System Status**, **Simulink Real-Time Properties**, **Navigation**, and **WWW Properties**.

After you connect a web browser to the target computer, you can use the **Main** pane to control the real-time application:

1   In the left frame, click the **Refresh** button.

     System status information in the top cell is uploaded from the target computer. If the right frame is either the **Signals List** pane or the **Screen Shot** pane, updating the left frame also updates the right frame.

2 Click the **Start Execution** button.

The real-time application begins running on the target computer, the **Status** line is changed from **Stopped** to **Running**, and the **Start Execution** button text changes to **Stop Execution**.

3 Update the execution time and average task execution time (TET). Click the **Refresh** button. To stop the real-time application, click the **Stop Execution** button.

4 Enter new values in the **StopTime** and **SampleTime** boxes, then click the **Apply** button. You can enter `-1` or `Inf` in the **StopTime** box for an infinite stop time.

The new property values are downloaded to the real-time application. Note that the **SampleTime** box is visible only when the real-time application is stopped. You cannot change the sample time while a real-time application is running. (See "Alternative Configuration and Control Methods" for limitations on changing sample times.)

After entering values, the screen looks like this:

System Status

Application	xpcosc
Mode	Real-Time Single-Tasking
Status	Stopped
CPUOverload	none
ExecTime	0.0
SessionTime	88641.1
StopTime	0.2
SampleTime	0.00025
AvgTET	1.04013e-005

Start Execution

Get State Log

Get Output Log

Get TET Log

xPC Target Properties

ViewMode [All ▼]

SampleTime [0.00025]

StopTime [0.2]

Apply   Reset

Navigation

Scopes

Signals   Refresh

Parameters

Screen Shot

WWW Properties

Maximum Signal Width [Inf]

Refresh Interval

Click any button on the left to navigate

Changing WWW Properties

The **WWW Properties** cell in the left frame contains fields that control the display on the web interface itself, and not the real-time application. There are two fields: maximum signal width to display and refresh interval.

1 In the **Maximum Signal Width** box enter `-1`, `Inf` (show all signals), `1` (show only scalar signals), `2` (show scalar and vector signals less than or equal to 2 wide), or `n` (show signals with a width less than or equal to `n`).

Signals with a width greater than the value you enter are not displayed on the **Signals** pane.

2 In the **Refresh Interval** box, enter a value greater than 10. For example, enter 20.

The signal pane updates automatically every 20 seconds. Entering `-1` or `Inf` does not automatically refresh the pane.

Sometimes, both the frames try to update simultaneously, or the auto refresh starts before the previous load has finished. This problem can happen with slow network connections. In this case, increase the refresh interval or manually refresh the browser (set the **Refresh Interval** = `Inf`).

This can also happen when you are trying to update a parameter or property at the same time that the pane is automatically refreshing.

Sometimes, when a race condition occurs, the browser becomes confused about the format, and you might have to refresh it. This should not happen often.

Viewing Signals with a Web Browser

The **Signals** pane is a list of the signals in your model.

After you connect a web browser to the target computer you can use the **Signals** pane to view signal data:

1 In the left frame, click the **Signals** button.

The **Signals** pane is loaded in the right frame with a list of signals and the current values.

2 On the **Signals** pane in the right frame, click the **Refresh** button.

The **Signals** pane is updated with the current values. Vector/matrix signals are expanded and indexed in the same column-major format that the MATLAB interface uses. This can be changed by the **Maximum Signal Width** value you enter in the left frame.

3 In the left frame, click the **Screen Shot** button.

The **Screen Shot** pane is loaded and a copy of the current target computer screen is displayed. The screen shot uses the portable network graphics (PNG) file format.

Trace Signals Using a Web Browser

The web browser interface allows you to visualize data using a user interface.

After you connect a web browser to the target computer, you can use the scopes page to add, remove, and control scopes on the target computer:

1 In the left frame, click the **Scopes** button. The browser loads the **Scopes List** pane into the right frame.

2 Click the **Add Scope** button.

A target scope is created and displayed on the target computer. The **Scopes** pane displays a list of the scopes present. You can add a scope, remove existing scopes, and configure the scopes from this page.

To create a host scope, use the drop-down list next to the **Add Scope** button to select Host. This item is set to Target by default.

3 Click the **Edit** button.

From the scope editing pane, you can configure and control the scope.

4 Click the **Add Signals** button. The browser displays an **Add New Signals** list.

5 Select the check boxes next to the signal names, and then click **Apply**. A **Remove Existing Signals** list is added above the **Add New Signals** list.

You do not have to stop a scope to change scope parameters. If the scope is running, the web interface stops the scope automatically and then restarts it when the changes are made. It does not restart the scope if the state was originally stopped.

The **Get Data** button becomes available under these circumstances:

- The host scope is stopped (**Scope State** is set to `Interrupted`)
- The host scope finishes one cycle of acquisition (**Scope State** is set to `Finished`).

If you click the **Get Data** button, the scope data is retrieved in comma-separated value (CSV) format. The signals in the scope are spread across columns. Each row corresponds to one sample of acquisition. The first column corresponds to the time each sample was acquired.

If **Scope State** is set to `Interrupted`, the scope was stopped before it completed a full cycle of acquisition. The number of rows in the CSV data still correspond to a full cycle. The last few rows (for which data was not acquired) are set to `0`.

Log Signals Using a Web Browser

When you stop the model execution, you see another section of the web browser interface where you can download logging data. This data is in comma-separated value (CSV) format. The MATLAB interface reads CSV format using the `dlmread` function. Most spreadsheet programs can read CSV format files.

You see this section of the web browser interface only if you have enabled data logging. Buttons become available only for those logs (states, output, and TET) that are enabled. If time logging is enabled, the first column of the CSV file is the time at which data (states, output, and TET values) was acquired. If time logging is not enabled, only the data is in the CSV file, without time information.

Note:

- The data logging variables `tout`, `xout`, `yout`, and `logsout` are available only when you use Simulink to simulate the model in non-real-time.

- You cannot use Simulink Data Inspector to create a data log on the target computer. You can log only signals that are connected to an `Outport` block.

You plot the outputs and states of your real-time application to observe the behavior of your model with different input signal values and parameter settings.

To do data logging, first carry out the following setup.

1 Click **Simulation** > **Model Configuration Parameters**.

2 To allow Simulink to log signals, in the **Data Import/Export** pane, verify that the **Signal logging** check box is selected. This check box is selected by default.

3 To plot the task execution time, in the **Simulink Real-Time Options** pane, verify that the **Log Task Execution Time** check box is selected. This check box is selected by default.

4 In the **Simulink Real-Time Options** pane of the Configuration Parameters dialog box, set **Signal logging buffer size in doubles** to a value large enough to accommodate the number of signals you are logging.

For a single-rate model, task execution time (TET) measures how long it takes the kernel to update the model state, propagate the new model state to the outputs, and log the model state during a sample interval. For a multirate model, TET measures how long it takes the kernel to perform those tasks for the base-rate portion only. You can plot the TET to show under what circumstances subsystems are executed and the additional CPU time required for those executions.

Tune Parameters Using a Web Browser

The **Parameters** pane displays the tunable parameters in your model. Row and column indices for vector/matrix parameters are also shown.

After you connect a web browser to the target computer, use the **Parameters** page to change parameters in your real-time application while it is running.

1 In the left frame, click **Parameters**. The browser loads the **Parameter List** pane into the right frame.

If the parameter is a scalar parameter, the current parameter value is shown in a box that you can edit.

If the parameter is a vector or matrix, click the **Edit** button to view the vector or matrix. You can edit the parameter in this pane.

2 In the **Value** box, enter a new parameter value, and then click the **Apply** button.

The new parameter values are uploaded to the real-time application.

Changing Access Levels to the Web Browser

The web browser interface allows you to set access levels to the real-time application. The different levels limit access to the real-time application. The highest level, 0, is the

default level and allows full access. The lowest level, 4, only allows signal monitoring and tracing with your real-time application.

1   In Simulink Editor, click **Simulation** > **Model Configuration Parameters**.

The Configuration Parameters dialog box for the model is displayed.

2   Click the **Code Generation** node.

The code generation pane opens.

3   In the **Target selection** section, access levels are set in the **System target file** box. For example, to set the access level to 1, enter

```
slrt.tlc -axpcWWWAccessLevel=1
```

If you do not specify `-axpcWWWAccessLevel`, the highest access level (`0`) is set.

4   Click **OK**.

The various fields disappear, depending on the access level. For example, if your access level does not allow you access to the parameters, you do not see the button for parameters.

There are various access levels for monitoring, which allow different levels of hiding. The proposed setup is described below. Each level builds on the previous one, so only the incremental hiding of each successive level is described.

Level 0 — Full access to the panes and functions.

Level 1 — Cannot change the sample and stop times. Cannot change parameters, but can view parameters.

Level 2 — Cannot start and stop execution of the real-time application or log data.

Level 3 — Cannot view parameters. Cannot add new scopes, but can edit existing scopes.

Level 4 — Cannot edit existing scopes on the **Scopes** pane. Cannot add or remove signals on the **Scopes** pane. Cannot view the **Signals** pane and the **Parameters** pane, and cannot get scope data.

10

Tuning Performance

Improve Performance of Multirate Model

Required Products: Simulink, Simulink Real-Time

Other Requirements:

- One Windows development computer with an Ethernet card
- One target computer
- One crossover cable for communication between the development and target computers

Performance Advisor detects blocks and parameter settings that can reduce performance. It determines the lower limit on sample time that does not produce a CPU overload.

This example uses model `ex_slrt_perfadv`.

In `ex_slrt_perfadv`, the configuration parameter **Fixed-step size (fundamental sample time)** is set to `auto`. The sample time is set in the referenced subsystems using a MATLAB variable, `Ts`. You can change the base sample time by changing the value of `Ts`.

Generate Baseline

Before you optimize model `ex_slrt_perfadv` using Performance Advisor, generate a baseline.

1 Open `ex_slrt_perfadv`.
2 From the **Analysis** menu, click **Performance Tools** > **Performance Advisor**.
3 Set **Activity** to `Execute real-time application`.

4 Under node **Performance Advisor**, select all of the **Baseline** checks except **Determine minimum sample time**.

Determining the minimum sample time can be a lengthy process for a large model with a long execution time.

5 Select node **Baseline**, and then click **Run selected checks**.
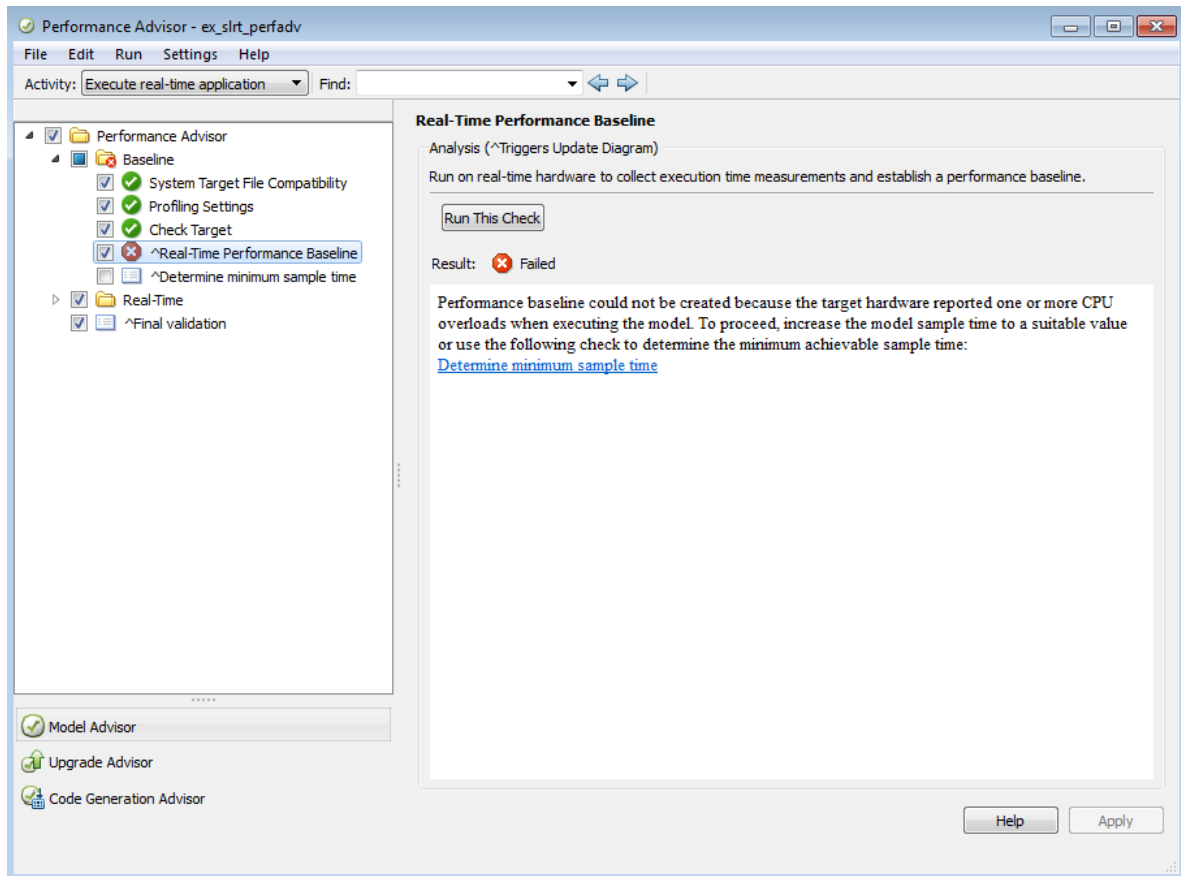
For this model, the **Real-Time Performance Baseline** action fails because running the real-time application produced a CPU overload on the target computer.

6 To remove this condition, increase the sample time to a value greater than the minimum value that does not cause a CPU overload. To find the minimum sample

time, select the **Determine minimum sample time** check box, and then click **Run this check**.
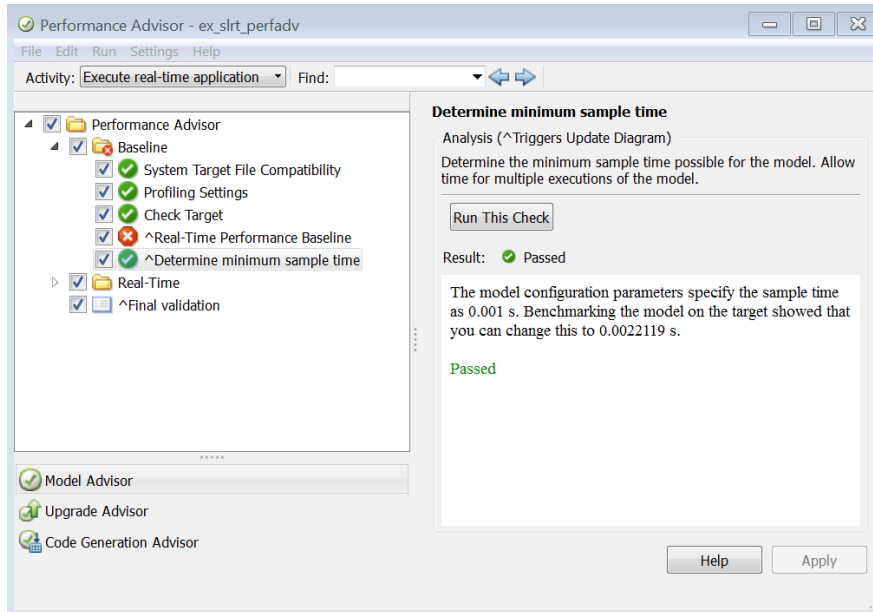
To avoid the overloads that random variations can cause, set Ts to a value above the lower limit. For example, set it to 0.003 s.

7 In the Command Window, type:

```
Ts = 0.003
```

8 Save `ex_slrt_perfadv` and its reference subsystems.

9 Clear the **Determine minimum sample time** check box, select the **Real-Time Performance Baseline** check box, and then click **Run this check**.
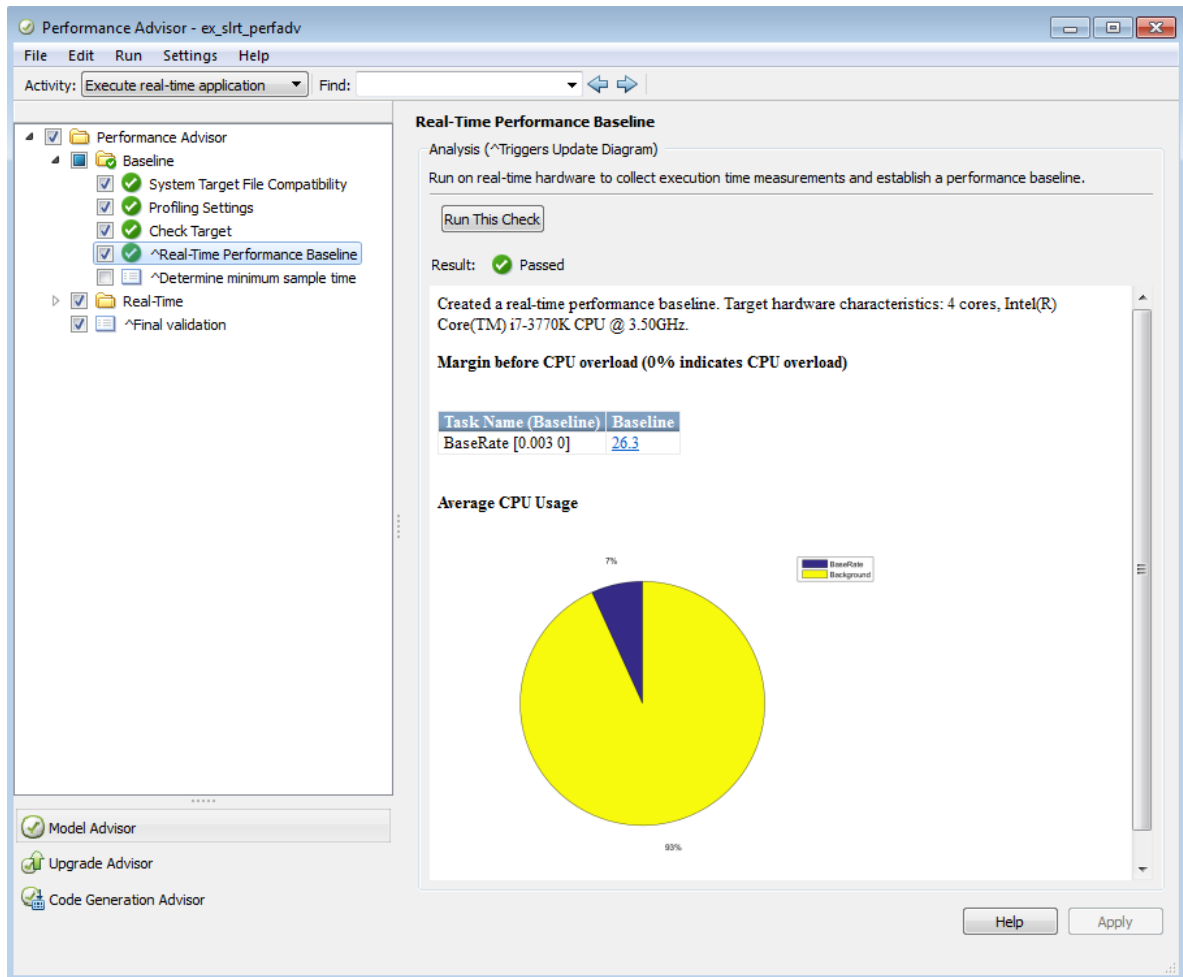
Perform Real-Time Checks

To perform the real-time performance checks on model `ex_slrt_perfadv`, first create a baseline (see "Generate Baseline" on page 10-2). Then carry out the following steps using Performance Advisor.

1   Under node **Performance Advisor**, select all of the top-level **Real-Time** checks.

If you have a license for Simscape™ or its related products, such as Simscape Driveline™ and Simscape Electronics™, clear those checks. There are no Simscape or related blocks in the model.

2 Select the **Real-Time** node, and then click **Run selected checks**.
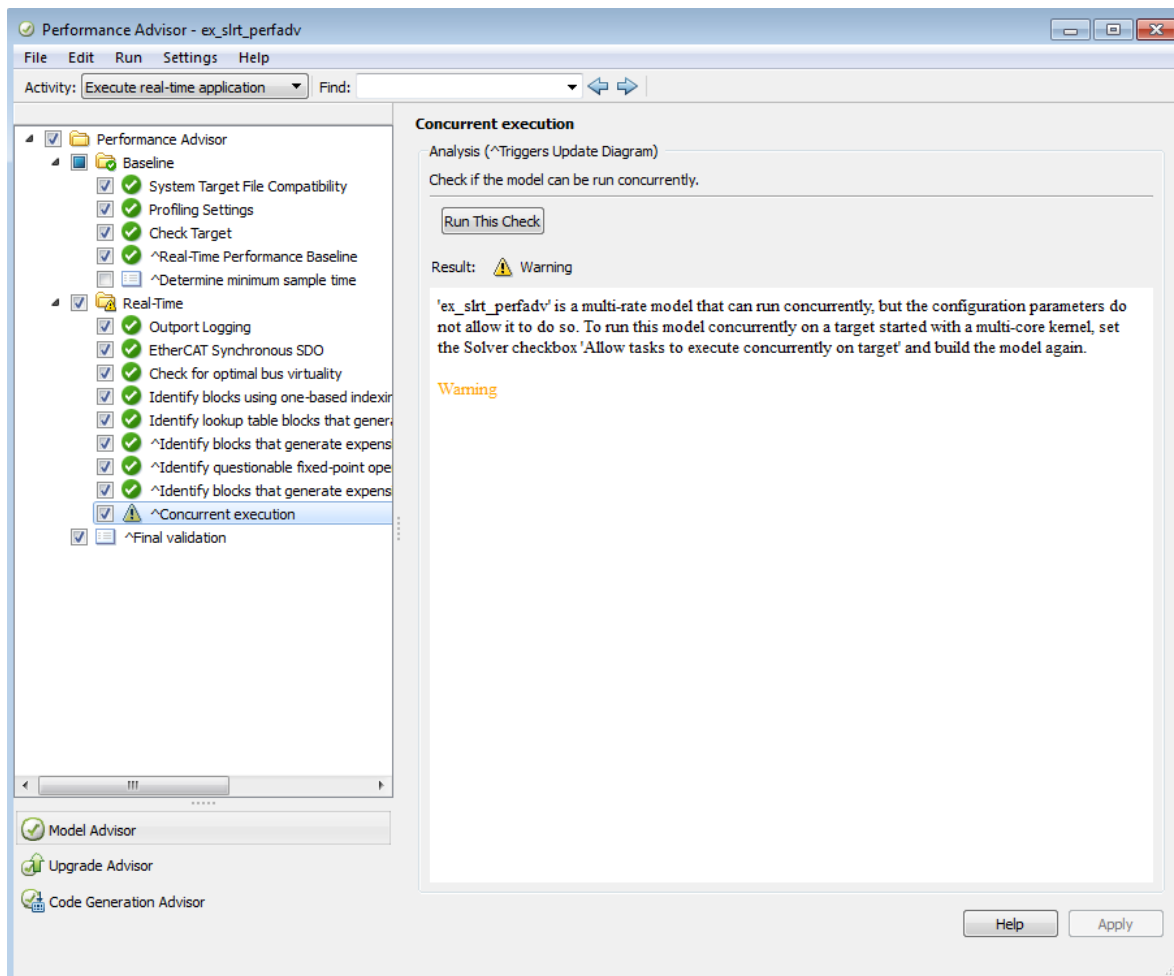
The model is a multirate model running on a multicore target computer, but it is not configured to use more than one core.

3 In the **Solver** pane under **Additional options**, select the check box **Allow tasks to execute concurrently on target**. Select the same setting for the reference subsystems `ex_slrt_perfadv_ref1` and `ex_slrt_perfadv_ref2`.

4 Save `ex_slrt_perfadv` and its reference subsystems.

5 Select the **Concurrent execution** check box, and then click **Run this check**.
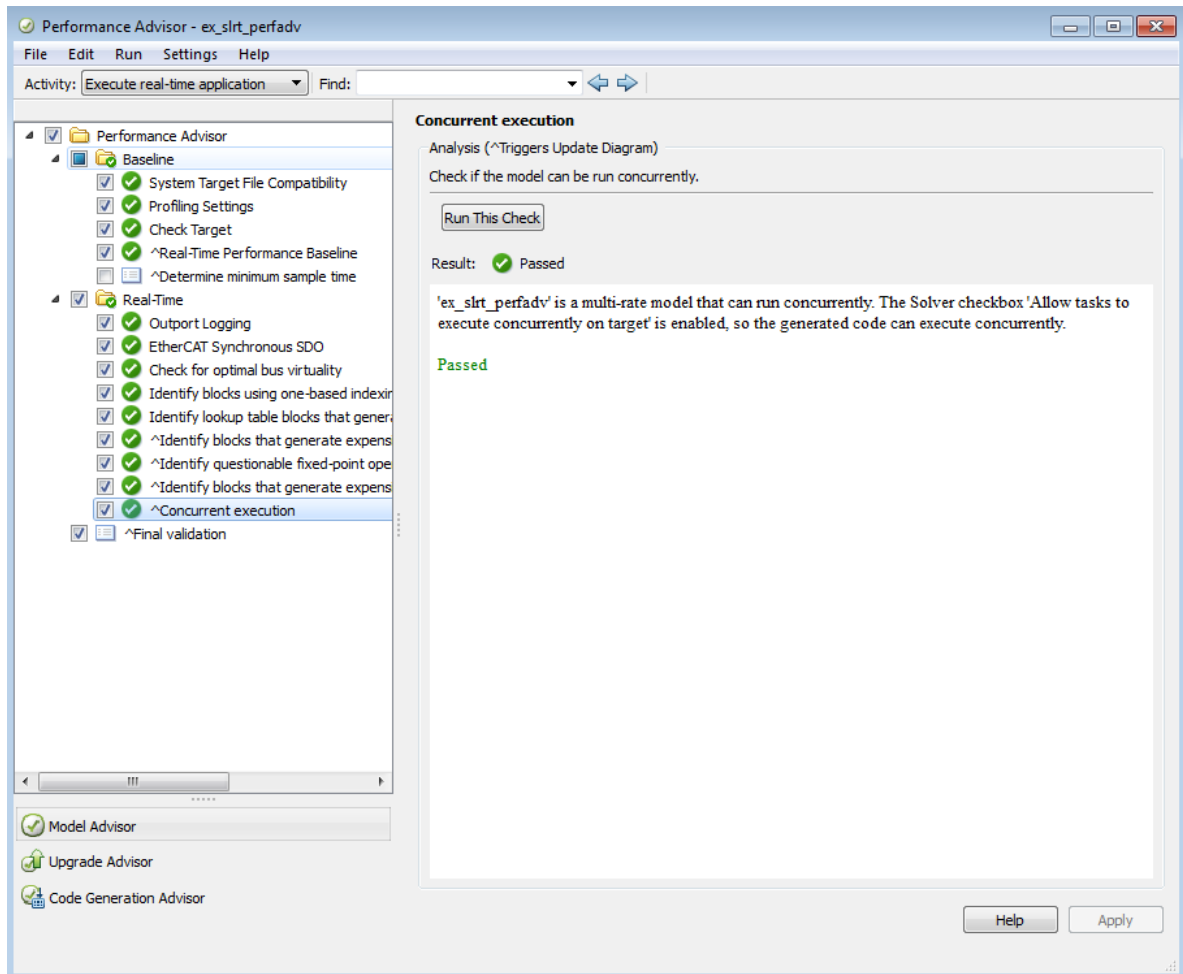
6 To improve the minimum sample time, select the **Determine minimum sample time** check box, and then click **Run this check**.

The result shows a sample time less than `0.0003` s. To avoid the overloads that random variations can cause, set `Ts` to a value above the lower limit. For example, set it to `0.001` s.

7 In the Command Window, type:

    Ts = 0.001

8 Save `ex_slrt_perfadv` and its reference subsystems.

Final Validation

The final validation check tests whether model `ex_slrt_perfadv` works after the actions in "Generate Baseline" on page 10-2 and "Perform Real-Time Checks" on page 10-5.

1 Select the **Final validation** check box, and then click **Run this check**.
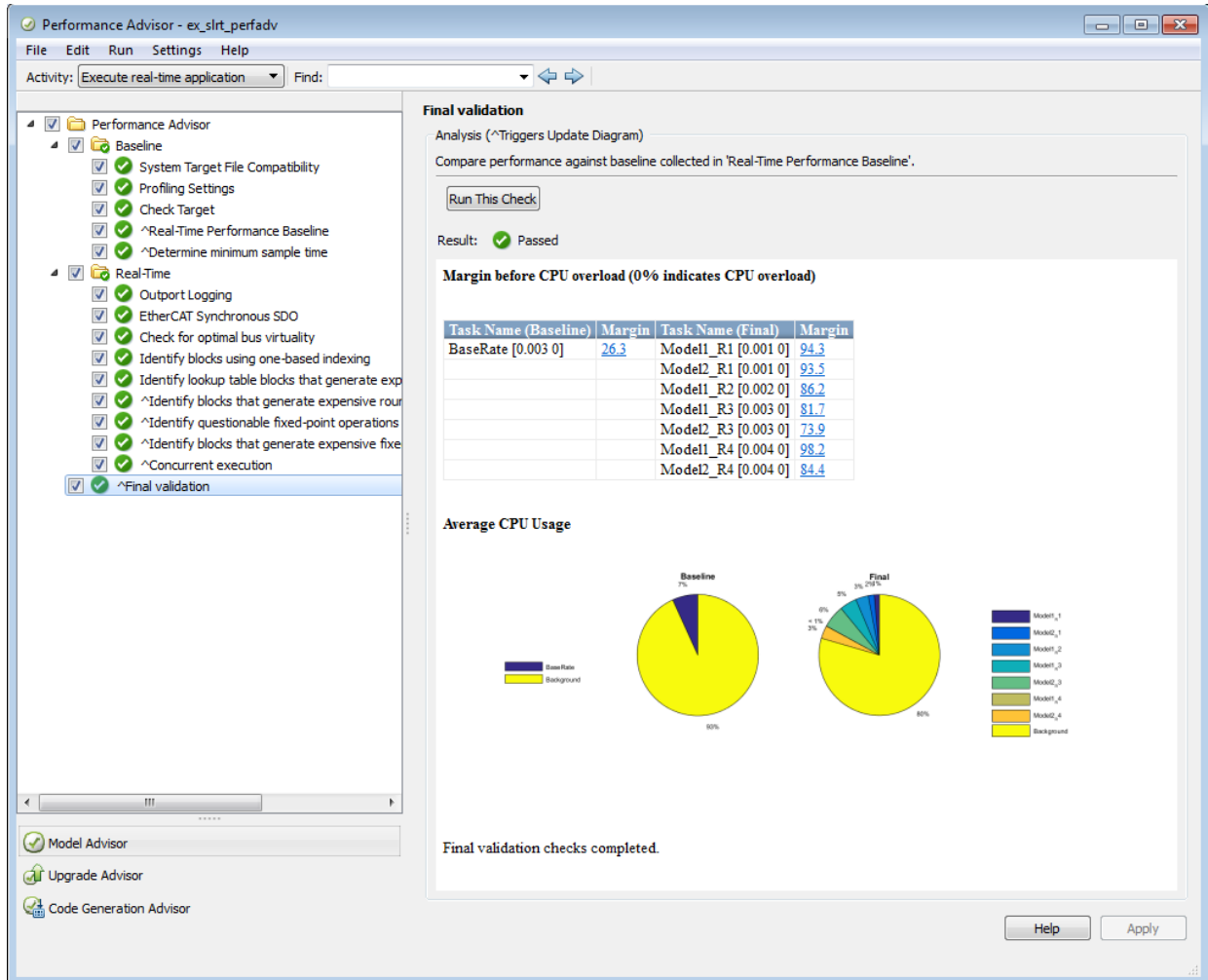
2 To improve the sample time further, run `profile_slrt` and examine the task execution and function execution times in detail.

See Also

ex_slrt_perfadv | `profile_slrt`

Related Examples

- "Execution Profiling for Real-Time Applications" on page 10-15
- "Multicore Processor Configuration" on page 10-11
- "Maximizing Target Computer CPU Usage" on page 10-13

More About

- "Multicore Programming with Simulink"
- "BIOS Settings"
- "Multicore Processors"

Multicore Processor Configuration

For better performance on your target computer, you can run multirate real-time applications on multiple cores. Use this capability if your target computer has a multicore processor and you want to take advantage of it for multirate models. Before you consider enabling this capability, see "BIOS Settings" for the effects of BIOS settings.

To build and download multirate models on your multicore target computer:

1   Type `slrtexplr` in the MATLAB Command Window.

2   In the **Targets** pane, expand the target computer node.

3   Click the Target Properties button ![icon] in the toolbar or double-click **Properties**.

4   Verify that the **Multicore CPU** check box is selected in the **Target settings** pane.

5   Open your model in Simulink Editor.

6   Add a `Rate Transition` block to transition between rates.

Note: Multirate models must use `Rate Transition` blocks. If your model uses other blocks for rate transitions, building the model generates an error.

---

7   Select the **Ensure data integrity during data transfer** check box of the `Rate Transition` block parameters.

8   Clear the **Ensure deterministic data transfer (maximum delay)** check box of the `Rate Transition` block parameters. This setting forces the `Rate Transition` block to use the most recent data available.

Note: Because this box is cleared, the transferred data can differ from run to run.

---

9   In Simulink Editor, select **View** > **Model Explorer**.

10 In Simulink Model Explorer, right click in the **Model Hierarchy** pane and select **Configuration** > **Add configuration for concurrent execution**

11 In the new configuration, select **Solver**.

12 Check **Enable concurrent tasking**.

13 Click **Configure Tasks**.

See Also

`Rate Transition`

More About

· "Multicore Programming with Simulink"

Maximizing Target Computer CPU Usage

Sometimes a real-time application running on the target computer does not have enough time to complete processing before the next time step. This condition is called a CPU overload. Typically, the Simulink Real-Time kernel halts model execution when it encounters a CPU overload.

Some real-time applications can tolerate several CPU overloads without significant loss of data, for example during startup. For such applications, you can maximize target computer CPU usage by allowing a specified number and configuration of CPU overloads. You do this using the TLCOptions settings xPCMaxOverloads, xPCMaxOverloadLen, and xPCStartupFlag.

If your real-time application experiences a CPU overload after execution starts, the kernel ignores timer interrupts if the task is already running. The application continues running, subject to the values of xPCMaxOverloads and xPCMaxOverloadLen. If the kernel allows, the application then executes at the next step.

Consider the following cases:

- xPCMaxOverloads is 2. The software tolerates the first two overloads and stops execution at the third.
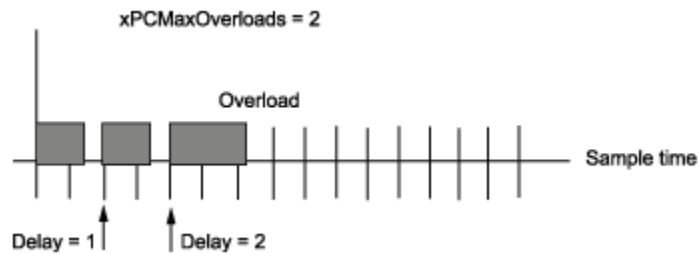
- xPCMaxOverloads is 3 and xPCMaxOverloadLen is 2. The software tolerates the first three overloads and halts the model at the fourth.
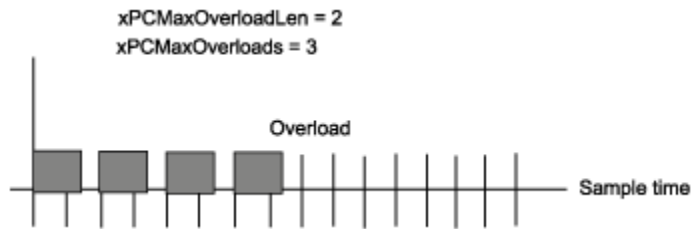
- `xPCStartupFlag` is 4. The kernel ignores overloads for the first four executions.
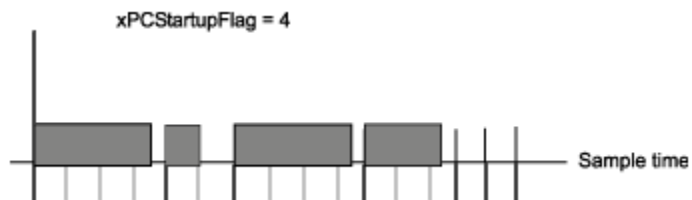
The three properties interact. When the Simulink Real-Time kernel runs the model, it checks the number of CPU overloads against the values of `xPCMaxOverloads` and `xPCMaxOverloadLen`. When the number of CPU overloads reaches the lower of these two values, the kernel stops executing the model.

Suppose that you enter the following `TLCOptions` settings for model `xpcosc`.

```
set_param('xpcosc','TLCOptions','-axPCMaxOverloads=30
          -axPCOverLoadLen=2 -axPCStartupFlag=5')
```

The software ignores CPU overloads for the first five iterations through the model. After the first five iterations, the software allows up to 30 CPU overloads, allowing at most two consecutive CPU overloads.

You can use the blocks `Set Overload Counter` and `Get Overload Counter` to set and track CPU overload numbers. You can use the `Time Stamp Counter` block to profile your model

See Also

`Get Overload Counter` | `Set Overload Counter` | `Time Stamp Counter` | TLCOptions Properties

Execution Profiling for Real-Time Applications

You can profile the task execution time and function execution time of your real-time application running on the target computer. Using that information, you can then tune its performance.

Profiling is especially useful if the real-time application is configured to take advantage of multicore processors on the target computer. To profile the real-time application:

1   In the Configuration Parameters for the model, enable the collection of profile data during execution.
2   Build, download, and execute the model.
3   Display and evaluate the profile data.

Profiling slightly increases the execution time of the real-time application.

Related Examples

· "Configure Real-Time Application for Profiling" on page 10-16
· "Generate Real-Time Application Execution Profile" on page 10-19
· "Multicore Processor Configuration" on page 10-11

Configure Real-Time Application for Profiling

This example shows how to configure model `dxpcmds6t` for task execution profiling.

1   Open model `dxpcmds6t`.

2   In the top model, open the Configuration Parameters dialog box, and select **Code Generation** > **Verification**.

3   Select the **Measure task execution time** check box.

For a single-rate model, task execution time (TET) measures how long it takes the kernel to update the model state, propagate the new model state to the outputs, and log the model state during a sample interval. For a multirate model, TET measures how long it takes the kernel to perform those tasks for the base-rate portion only. You can plot the TET to show under what circumstances subsystems are executed and the additional CPU time required for those executions.

4   If you also want to profile function execution times, select the **Measure function execution times** check box.
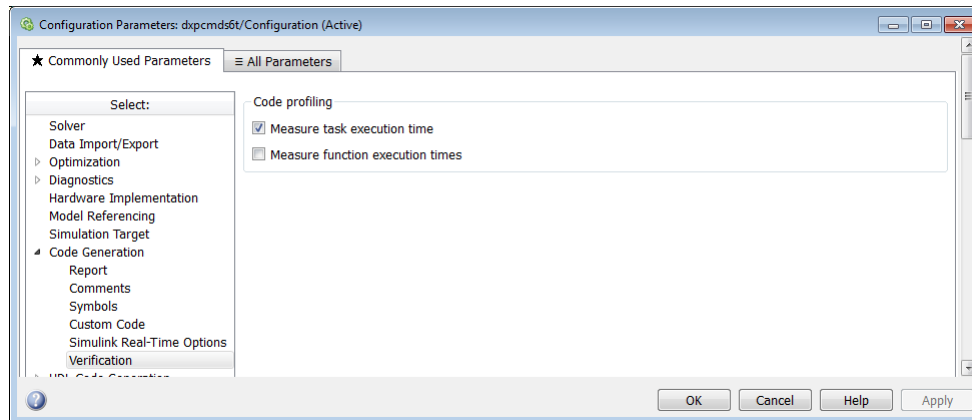
5   Under **Code Generation**, select **Simulink Real-Time Options**.

6   Type a value for **Number of profiling events (each uses 20 bytes)**.

By default, the software logs 5000 events for profiling. You can increase or decrease this number to manage memory usage. When the software logs the specified number of events or the model stops, the software stops collecting the data. The software writes the data to `current_working_folder\xPCTrace.csv` on the target computer.
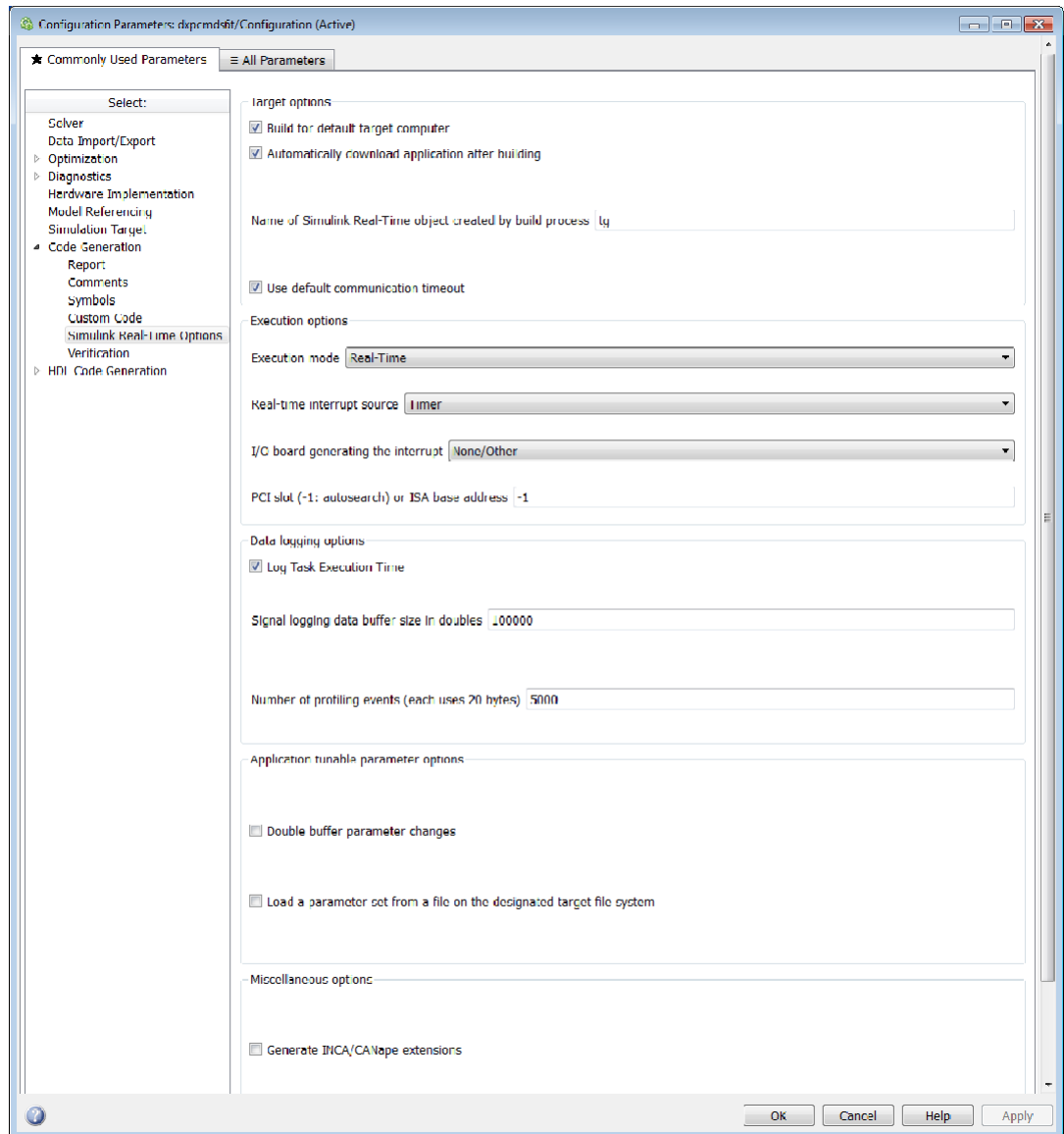
7 Click **OK**.

8 Save model `dxpcmds6t`.

See Also

`profile_slrt`

Generate Real-Time Application Execution Profile

This example shows how to generate profile data for model `dxpcmds6t` using default settings on a multicore target computer.

This procedure assumes that you have configured the target computer to take advantage of multiple cores. It also assumes that you previously configured the model for task execution profiling.

Build and download the model.

```
mdl = 'dxpcmds6t';
open_system(mdl)
rtwbuild(mdl)
```

When you include profiling, the Code Generation Report is generated by default. It contains links to the generated C code and include files. By clicking these links, you can examine the generated code and understand the Code Execution Profile Report better.
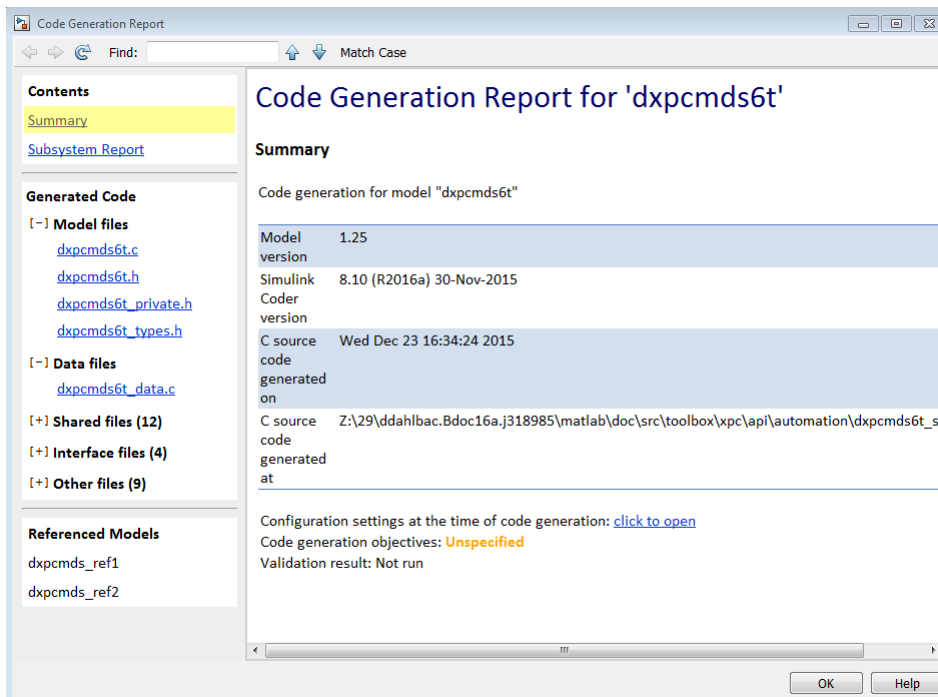
Execute the real-time application.

```
tg = slrt;
start(tg)
pause(2)
stop(tg)
```

Profile the real-time application execution.

```
profileInfo.modelname = 'dxpcmds6t.mdl';
profData = profile_slrt(profileInfo);
```

The Execution Profile plot shows the allocation of execution cycles across the four processors, indicated by the colored horizontal bars.
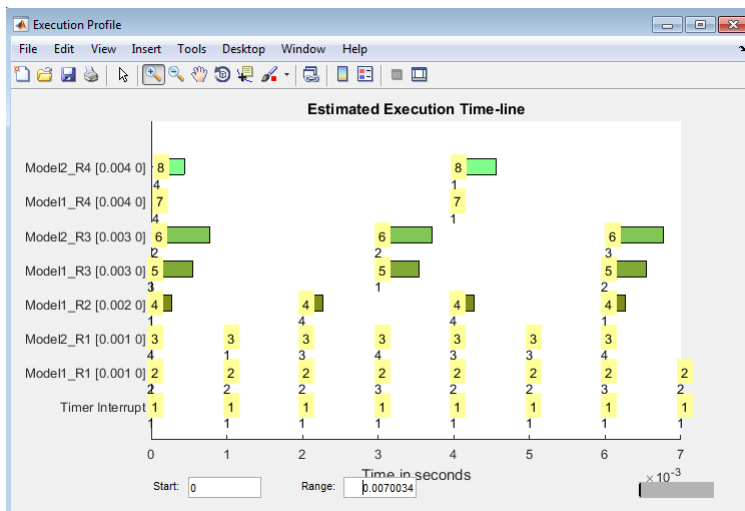
The Code Execution Profiling Report displays model execution profile results for each task.

Code Execution Profiling Report for dxpcmds6t

The code execution profiling report provides metrics based on data collected from real-time simulation. Execution times are calculated from data recorded by instrumentation probes added to the generated code. See Code Execution Profiling for more information.

1. Summary

Total time (seconds × 1e-09)	433849547
Measured time display options	('Units', 'Seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f')
Timer frequency (ticks per second)	3.5e+09
Profiling data created	23-Dec-2015 16:46:46

2. Profiled Sections of Code

Section	Maximum Turnaround Time	Average Turnaround Time	Maximum Execution Time	Average Execution Time	Calls	
Timer Interrupt	2507	1304	2507	1304	536	
Model1_R1 [0.001 0]	56233	54472	56233	54472	535	
Model2_R1 [0.001 0]	63489	62936	63489	62936	535	
Model1_R2 [0.002 0]	269858	267742	269858	267742	268	
Model1_R3 [0.003 0]	543646	540907	543646	540907	179	
Model2_R3 [0.003 0]	717853	713750	717853	713750	179	
Model1_R4 [0.004 0]	8224	7611	8224	7611	134	
Model2_R4 [0.004 0]	546613	544622	546613	544622	134	

Profile Data	Description
Maximum turnaround time	Longest time between when the task starts and finishes. This time includes task preemptions (interrupts).
Average turnaround time	Average time between when the task starts and finishes. This time includes task preemptions (interrupts).
Maximum execution time	Longest time between when the task starts and finishes. This time does not include task preemptions (interrupts).
Average execution time	Average time between when the task starts and finishes. This time does not include task preemptions (interrupts).
Calls	Number of times the generated code section is called.

To display the profile data for the generated code section, click the **Membrane** button ◭ in the Coder Execution Profiling Report.

Close the model.

```
close_system(mdl,0)
```

See Also
```
profile_slrt
```

Related Examples
- "Multicore Processor Configuration" on page 10-11
- "Configure Real-Time Application for Profiling" on page 10-16

Building Referenced Models in Parallel

The Simulink Real-Time software allows you to build referenced models in parallel on a compute cluster. In this way, you can more quickly build and download real-time applications to the target computer.

The following procedure assumes that you have a functioning Simulink Real-Time installation on your development computer.

1   Identify a set of worker computers, which can be separate cores on your development computer or computers in a remote cluster running under Windows.

2   If you intend to use separate cores on the development computer, install Parallel Computing Toolbox™ on the development computer.

3   If you intend to use computers in a remote cluster:

    a   Install the following on each cluster computer:

        ·   MATLAB

        ·   Parallel Computing Toolbox

        ·   MATLAB Distributed Computing Server™

        ·   Simulink Real-Time

        ·   Build compiler

        Install the same compiler and compiler version at the same location as on the development computer.

    b   Start and configure the remote cluster according to the instructions at www.mathworks.se/support/product/DM/installation/ver_current.

4   Run MATLAB on the development computer.

5   In MATLAB, call the `parpool` function to open a parallel pool on the cluster.

6   To configure the compiler for the remote workers as a group, call the `pctRunOnAll` function. For example:

```
pctRunOnAll('slrtsetCC(''VisualC'',
    ''C:\Program Files\Microsoft Visual Studio 9.0'')')
```

In this configuration, the development computer and the remote workers have installed Microsoft Visual Studio® 9.0 at `C:\Program Files\Microsoft Visual Studio 9.0`.

7   Build and download your model.

See Also

parpool | pctRunOnAll

More About

- "Reduce Build Time for Referenced Models"

Execution Using MATLAB Scripts

11

Real-Time Applications and Scopes in the MATLAB Interface

MATLAB Language Interface

An important part of the "Rapid Control Prototyping Process" and "Hardware-in-the-Loop Simulation Process" workflows is preparing stress test and regression test scripts. The Simulink Real-Time product includes specialized MATLAB classes and functions for setting up the target environment, booting the target computer, loading and running the real-time application, and displaying and recording the results. You do these tasks using the MATLAB language interface.

Real-Time Application Objects

The Simulink Real-Time software uses a `SimulinkRealTime.target` object to represent the target kernel and your real-time application. Use real-time application object functions to run and control real-time applications on the target computer with scope objects to collect signal data.

An understanding of the real-time application object properties and functions helps you to control and test your real-time application on the target computer.

A real-time application object on the development computer represents the interface to a real-time application and the kernel on the target computer. You use real-time application objects to run and control the real-time application.

When you change a real-time application object property on the development computer, information is exchanged with the target computer and the real-time application.

To create a real-time application object:

1   Build a real-time application. The Simulink Real-Time software creates a real-time application object during the build process.

2   Use the real-time application object function `SimulinkRealTime.target`. In the MATLAB Command window, type `tg = SimulinkRealTime.target`.

A `SimulinkRealTime.target` object has properties and functions specific to that object. The real-time application object functions allow you to control a real-time application on the target computer from the development computer. You enter real-time application object functions in the MATLAB window on the development computer, or you can use MATLAB code scripts. To access the help for these functions from the command line, use the syntax:

```
doc SimulinkRealTime.target/function_name
```

If you want to control the real-time application from the target computer, use target computer commands (see "Control Real-Time Application at Target Computer Command Line" on page 8-2).

Create Real-Time Application Objects

To create a real-time application object:

1 Build a real-time application. The Simulink Real-Time software creates a real-time application object during the build process.

2 To create a specific real-time application object, or to create multiple real-time application objects in your system, use the real-time application object function `SimulinkRealTime.target` with arguments. For example, to create a real-time application object for target `TargetPC1`, in the MATLAB Command Window, type:

```
tg = SimulinkRealTime.target('TargetPC1')
```

The resulting real-time application object is `tg`.

Using this function clarifies which application object is associated with a particular target computer.

3 To check a connection between development and target computers, use the target function `SimulinkRealTime.target.ping`. For example, type:

```
ping(tg)
```

4 To create a real-time application object for the default target computer, use the creation function `SimulinkRealTime.target` without arguments. For example, in the MATLAB Command Window, type:

```
tg = SimulinkRealTime.target
```

The resulting real-time application object is `tg`.

Note: If you use `SimulinkRealTime.target` without arguments to create a real-time application object, use Simulink Real-Time Explorer to configure your target computer. Doing so clarifies which real-time application object is associated with a particular target computer.

Display Application Object Properties

You might want to list the real-time application object properties to monitor a real-time application. The properties include the execution time and the average task execution time.

After you build a real-time application and real-time application object from a Simulink model, you can list the real-time application object properties. This procedure uses the default real-time application object name `tg` as an example.

1   In the MATLAB window, type:

```
tg = slrt;
```

The current real-time application properties are uploaded to the development computer. MATLAB displays a list of the real-time application object properties with the updated values.

The real-time application object properties for `TimeLog`, `StateLog`, `OutputLog`, and `TETLog` are not updated at this time.

2   Type:

```
start(tg)
```

The `Status` property changes from `stopped` to `running`. The log properties change to `Acquiring`.

For a list of real-time application object properties with a description, see `SimulinkRealTime.target`.

Set Real-Time Application Object Property Values

You can change a real-time application object property by using the Simulink Real-Time dot notation on the development computer. (For limitations on target property changes to sample times, see "Alternative Configuration and Control Methods".)

With the Simulink Real-Time software, you can use object property syntax to change the real-time application object properties.

```
target_object.property_name = new_property_value
```

For example, to change the stop time for the real-time application running on target `tg`, in the MATLAB window, type:

```
tg = slrt;
tg.StopTime = 1000
```

When you change a real-time application object property, the new property value is downloaded to the target computer. The Simulink Real-Time kernel then receives the information and changes the behavior of the real-time application.

To get a list of the writable properties, type `target_object`. The build process assigns the default name of the real-time application object to `tg`.

Get Real-Time Application Object Property Values

You can list a property value in the MATLAB window or assign that value to a MATLAB variable. With the Simulink Real-Time software, you can use object property syntax.

```
target_object.property_name
```

For example, to access the stop time for the real-time application running on target tg, in the MATLAB window, type:

```
tg = slrt;
endrun = tg.StopTime
```

To get a list of readable properties, type `target_object`. Without assignment to a variable, the property values are listed in the MATLAB window.

Signals are not real-time application object properties. To get the value of the Integrator1 signal from the model xpcosc, in the MATLAB window, type:

```
outputvalue = getsignal(tg, 0)
```

0 is the signal index.

Note: Function names are case sensitive. You must type the entire name. Property names are not case sensitive. You do not need to type the entire name, as long as the characters you do type are unique for the property.

Use Real-Time Application Object Functions

Use the `function_name(target_object, argument_list)` syntax to run a real-time application object function.

Unlike properties, for which partial but unambiguous names are permitted, you must enter function names in full, in lowercase. For example, to add a target scope with a scope index of 1, in the MATLAB window, type:

```
tg = slrt;
addscope(tg,'target',1)
```

Real-Time Scope Objects

The Simulink Real-Time software uses scope objects to represent scopes on the target computer. Use scope object functions to view and collect signal data.

The Simulink Real-Time software uses scopes and scope objects as an alternative to using Simulink scopes and external mode. A scope can exist as part of a Simulink model system or outside a model system.

- A scope that is part of a Simulink model system is a scope block. You add a Simulink Real-Time scope block to the model, build a real-time application from that model, and download that application to the target computer.
- A scope that is outside a model is not a scope block. For example, if you create a scope with the `SimulinkRealTime.target.addscope` function, that scope is not defined within the model. After the model has been downloaded and initialized, you add this scope to the model.

This difference affects when and how the scope executes to acquire data.

Scope blocks inherit sample times. A scope block in the root model or a normal subsystem executes at the sample time of its input signals. A scope block in a conditionally executed (triggered/enabled) subsystem executes whenever the containing subsystem executes. In the latter case, the scope might acquire samples at irregular intervals.

A scope that is not part of a model executes at the base sample time of the model. Therefore, it might acquire repeated samples. For example, if the model base sample time is 0.001, and you add to the scope a signal whose sample time is 0.005, the scope acquires five identical samples for this signal, and then the next five identical samples, and so on.

Understanding the structure of scope objects helps you to use the MATLAB command-line interface to view and collect signal data. A scope object on the development computer represents a scope on the target computer. You use scope objects to observe the signals from your real-time application during a real-time run or analyze the data after the run is finished.

To create a scope object:

- Add a Simulink Real-Time scope block to your Simulink model, build the model to create a scope, and then use the real-time application object function `SimulinkRealTime.target.getscope` to create a scope object.

- Use the real-time application object function
  `SimulinkRealTime.target.addscope` to create a scope, create a scope object, and
  assign the scope properties to the scope object.

Upon creation, the Simulink Real-Time software assigns the required scope object class
for the scope type:

- Target scopes — `SimulinkRealTime.targetScope`, created by calling
  `SimulinkRealTime.target.getscope` with scope type `target`.
- Host scopes — `SimulinkRealTime.hostScope`, created by calling
  `SimulinkRealTime.target.getscope` with scope type `host`.
- File scopes – `SimulinkRealTime.fileScope`, created by calling
  `SimulinkRealTime.target.getscope` with scope type `file`.

A scope object has associated properties and functions specific to that scope type, as
well as properties and functions held in common with the other scopes. The scope object
functions allow you to control scopes on your target computer.

If you want to control the real-time application from the target computer, use target
computer commands (see "Control Real-Time Application at Target Computer Command
Line" on page 8-2).

Display Scope Object Properties for One Scope

To list the properties of a single scope object, `sc1`, in the MATLAB window, type:

```
tg = slrt;
sc1 = getscope(tg,1)
```

MATLAB creates the scope object `sc1` from a previously created scope.

The current scope properties are uploaded to the development computer. MATLAB displays a list of the scope object properties with the updated values. Because `sc1` is a vector with a single element, you could also type `sc1(1)` or `sc1([1])`.

Note: Only scopes of type `host` store data in the properties `scope_object.Time` and `scope_object.Data`.

For a list of real-time application object properties with a description, see the target function `SimulinkRealTime.target`.

Display Scope Object Properties for Multiple Scopes

To list the properties of the current scope objects associated with the real-time application object `tg`, in the MATLAB window, type:

```
tg = slrt;
getscope(tg)
```

`SimulinkRealTime.target.getscope` supports vector arguments. For example, to list the first and third scopes, type:

```
getscope(tg,[1,3])
```

To assign a list of current scopes to a variable, type:

```
allscopes = getscope(tg)
```

For a list of real-time application object properties, see the target function `SimulinkRealTime.target`.

Set Scope Property Values

With the Simulink Real-Time software, you can use object property syntax to set a single scope object property.

```
scope_object.property_name = new_property_value
```

For example, to change the trigger mode for scope 1, in the MATLAB window, type:

```
tg = slrt;
sc1 = getscope(tg, 1);
sc1.triggermode = 'signal'
```

You cannot use dot notation to set vector object properties. To assign a property value to a vector of scopes, use the set function. For example, assume you have two scopes, 1 and 2. First assign a vector containing these scopes to the variable sc12:

```
sc12 = getscope(tg, [1,2]);
```

To set the NumSamples property of these scopes to 300, type:

```
set(sc12, 'NumSamples', 300);
```

To get a list of the writable properties, type scope_object.

Note:

- You cannot set a property of a vector of scopes to a vector of property values. For example, you cannot set property NumSamples of vector sc12 to [100,200].

- Function names are case sensitive. You must type the entire name. Property names are not case sensitive. You do not need to type the entire name as long as the characters you do type are unique for the property.

Get Scope Property Values

You can list a property value in the MATLAB window or assign that value to a MATLAB variable. With the Simulink Real-Time software, you can use object property syntax to get scope property values.

```
scope_object_vector(index_vector).property_name
```

For example, to get the number of samples from scope 1, in the MATLAB window, type:

```
tg = slrt;
sc1 = getscope(tg, 1);
sc1.NumSamples
```

To get the values of vector object properties set using the `set` function, you can use the corresponding `get` function. For example, assume you have two scopes, 1 and 2, with a `NumSamples` property of 300.

First assign a vector containing these scopes to the variable `sc12`.

```
sc12 = getscope(tg, [1,2]);
```

To get the value of `NumSamples` for these scopes, type:

```
get(sc12, 'NumSamples')
```

You get a result like:

```
ans =
    [300]
    [300]
```

Although you cannot use dot notation to set the values of vector object properties, you can use it to get those values:

```
sc12.NumSamples
```

You get a result like:

```
ans =
    300

ans =
    300
```

To get a list of readable properties, type `scope_object`. The property values are listed in the MATLAB window.

Note: Function names are case sensitive. You must type the entire name. Property names are not case sensitive. You do not need to type the entire name as long as the characters you do type are unique for the property.

Use Scope Object Functions

Use the function syntax to run a scope object functions:

```
function_name(scope_object, argument_list)
```

Unlike properties, for which partial but unambiguous names are permitted, enter function names in full, in lowercase. For example, to add signals to the first scope in a vector of all scopes, in the MATLAB window, type:

```
allscopes = getscope(tg)
addsignal(allscopes(1), [0,1])
```

Acquire Signal Data with File Scopes

You can acquire signal data into a file on the target computer. To do so, you can include a real-time file scope in your Simulink Real-Time model. Alternatively, after you build the real-time application and download it to the target computer, you can add a file scope to that application.

For example, to add a file scope named `sc` to the real-time application, and to add signal 4 to that scope:

1   In the MATLAB window, type:

```
tg = slrt;
sc = addscope(tg, 'file')
```

The Simulink Real-Time software creates a file scope for the real-time application.

2   To add signal 4, type:

```
addsignal(sc, 4)
```

3   **Caution** Before starting the scope, copy previously acquired data to the development computer. When the file scope starts, the software overwrites previously acquired data in files of the specified name or name pattern. A partially overwritten file or a file opened but left unwritten loses its original contents.

To start the scope, type:

```
start(sc)
```

4   To start the real-time application, type:

```
start(tg)
```

The Simulink Real-Time software adds signal 4 to the file scope. When you start the scope and the real-time application, the scope saves the signal data for signal 4 to a file, by default named `C:\data.dat`.

- For more information on file scopes, see "Configure Real-Time File Scope Blocks" on page 5-72.
- To retrieve the file programmatically from the target computer for analysis, see "Using SimulinkRealTime.fileSystem Objects" on page 12-4.

- To acquire signal data into multiple files, see "Acquire Signal Data into Dynamically Named Files" on page 11-19.

Acquire Signal Data into Dynamically Named Files

You can acquire signal data into multiple, dynamically named files on the target computer. For example, you can acquire data into multiple files to examine one file while the scope continues to acquire data into other files.

To acquire data into multiple files, you can include a real-time file scope in your Simulink Real-Time model. Alternatively, after you build a real-time application and download it to the target computer, you can add a file scope to that application. You can then configure that scope to log signal data to multiple files.

For example, configure a file scope named sc to the real-time application. The file scope has these characteristics:

- Logs signal data into up to nine files whose sizes do not exceed 4096 bytes.
- Creates files whose names contain the string file_.dat.
- Contains signal 4.

1   In the MATLAB window, type:

```
tg = slrt;
tg.StopTime = Inf;
```

This parameter value directs the real-time application to run indefinitely.

2   To add a file scope, type:

```
sc = addscope(tg, 'file');
```

3   To enable the file scope to create multiple log files, type:

```
sc.DynamicFileName = 'on';
```

Enable this setting to enable logging to multiple files.

4   To enable file scopes to collect data up to the number of samples, and then start over again, type:

```
sc.AutoRestart = 'on';
```

Use this setting for the creation of multiple log files.

5   To limit each log file size to 4096, type:

```
sc.MaxWriteFileSize = 4096;
```

You must use this property. Set `MaxWriteFileSize` to a multiple of the `WriteSize` property.

6 To enable the file scope to create multiple log files with the same name pattern, type:

```
sc.Filename = 'file_<%>.dat';
```

This sequence directs the software to create up to nine log files, `file_1.dat` to `file_9.dat` on the target computer file system.

7 To add signal 4 to the file scope, type:

```
addsignal(sc, 4);
```

8 **Caution** Before starting the scope, copy previously acquired data to the development computer. When the file scope starts, the software overwrites previously acquired data in files of the specified name or name pattern. A partially overwritten file or a file opened but left unwritten loses its original contents.

To start the scope, type

```
start(sc)
```

9 To start the real-time application, type

```
start(tg)
```

The software creates a log file named `file_1.dat` and writes data to that file. When the size of `file_1.dat` reaches 4096 bytes (value of `MaxWriteFileSize`), the software closes the file and creates `file_2.dat`. When its size reaches 4096 bytes, the software closes it and creates `file_3.dat`, and so on.

The software repeats this sequence until it fills the last log file, `file_9.dat`. If the real-time application continues to run and collect data after `file_9.dat`, the software reopens `file_1.dat` and overwrites the existing contents. It cycles through the other log files sequentially.

- For more information on file scopes, see "Configure Real-Time File Scope Blocks" on page 5-72.
- To retrieve the file programmatically from the target computer for analysis, see "Using SimulinkRealTime.fileSystem Objects" on page 12-4.
- To acquire signal data into a single file, see "Acquire Signal Data with File Scopes" on page 11-17.

Scope Trigger Configuration

You can configure Simulink Real-Time scopes to acquire data right away, or define triggers for scopes so that the Simulink Real-Time scopes wait until they are triggered to acquire data. You can configure Simulink Real-Time scopes to start acquiring data when a predefined trigger condition is met. The exact condition depends on the trigger mode that you select.

- Freerun — Acquires data as soon as the scope is started (default).
- Software — Acquires data in response to a user request, such as a call to one of the Scope functions (SimulinkRealTime.fileScope.trigger, SimulinkRealTime.hostScope.trigger, and SimulinkRealTime.targetScope.trigger) or a call to a C or .NET API function (xPCScSoftwareTrigger, xPCScope.Trigger).
- Signal — Acquires data when a particular signal has crossed a preset level.
- Scope — Acquires data when another (triggering) scope starts.

You can use several properties to further refine when a scope starts to acquire data. For example, if you want the scope to be triggered when another signal crosses a certain value, use Signal trigger mode. Specify:

- The signal to trigger the scope.
- The trigger level that the signal must cross to trigger the scope to start acquiring data.
- Whether the scope is triggered on a rising signal, falling signal, or either one.

The *trigger point* is the sample at which the scope trigger condition is satisfied. For signal triggering, the trigger point is the sample at which the trigger signal passes through the trigger level. At the trigger point, the scope acquires the first sample. By default, scopes start acquiring data from the trigger point onwards. You can modify this behavior using pretriggering and posttriggering with the NumPrePostSamples scope property. See "Pretriggering and Posttriggering of Scopes" on page 11-22.

Pretriggering and Posttriggering of Scopes

By default, the scope starts acquiring data at the same time as the trigger event (the trigger point). In some cases, you want to observe the sequence of values that led to the trigger, so you start acquiring data a given number of samples before the trigger event (pretriggering). In other cases, you want to observe the system settling down after the trigger event, so you delay acquiring data a given number of samples after the trigger event (posttriggering).

Use the `NumPrePostSamples` scope property to specify pretriggering and posttriggering. A negative value indicates pretriggering and a positive value indicates posttriggering. For example, suppose `P` is the value of `NumPrePostSamples` for Scope 1 and `TP` is the trigger point, the sample where the trigger event occurs.

- `P = 0` — Scope 1 starts acquiring data immediately at trigger point `TP`.

Simple triggering (P = 0)
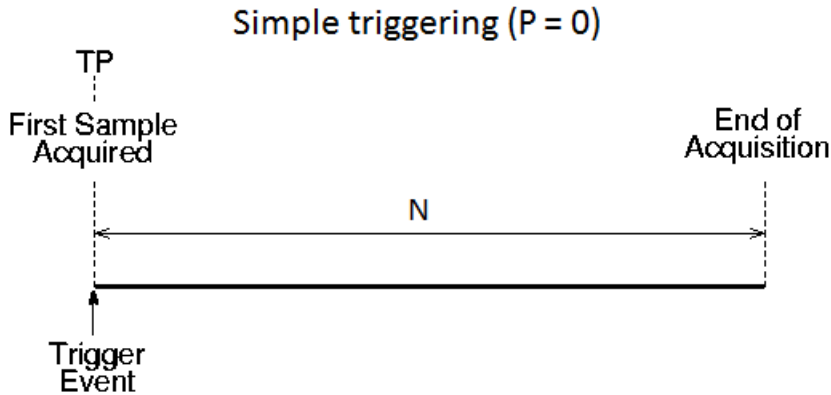
- `P < 0` — Scope 1 starts acquiring data |P| samples before trigger point TP.

Pre-triggering (P < 0)

First Sample
Acquired

End of
Acquisition

N

TP

|P|

(N + P)
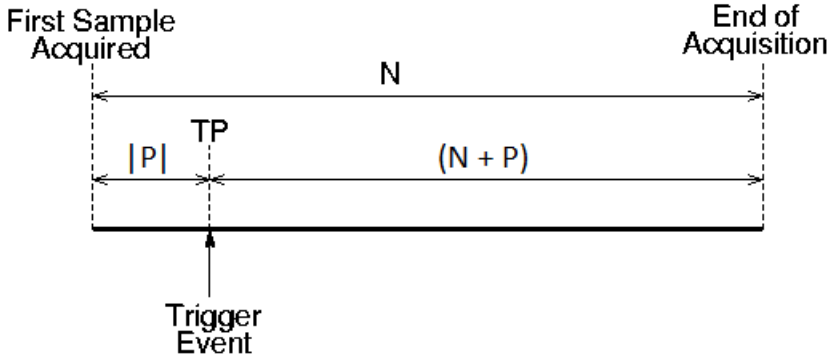
Trigger
Event

- P > 0 — Scope 1 starts acquiring data P samples after trigger point TP.

Post-triggering (P > 0)

TP

End of
Acquisition

(N + P)

P

N

First Sample
Acquired
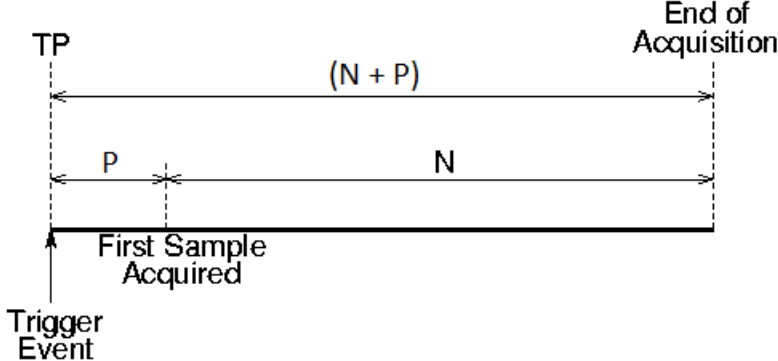
Trigger
Event

Trigger One Scope with Another Scope

When you have started two scopes that you want to keep synchronized, you can trigger one scope with another to acquire data. Set up the first scope with the trigger of your choice, and then trigger the second scope from the first.

In this setup, Scope 1 triggers Scope 2.

1   Two scope objects are configured as a vector with the command:

```
tg = slrt;
sc = addscope(tg, 'host', [1 2]);
```

2   For Scope 1, set these values:

```
sc(1).ScopeId = 1
sc(1).NumSamples = N1
sc(1).NumPrePostSamples = P1
```

3   For Scope 2, set these values:

```
sc(2).ScopeId = 2
sc(2).NumSamples = N2
sc(2).TriggerMode = 'Scope'
sc(2).TriggerScope = 1
sc(2).NumPrePostSamples = P2
```

Because Scope 2 is triggered by Scope 1, the trigger point TP is the same for both scopes. However, different samples can be acquired by Scopes 1 and 2.

In this section...
"Scope-Triggered Data Acquisition" on page 11-24
"Trigger Sample Setting" on page 11-27

Scope-Triggered Data Acquisition

Some representative relationships between data acquisitions by Scope 1 and Scope 2 are shown in the figures. P1 and P2 are the values of NumPrePostSamples for Scopes 1 and 2. TP is the trigger point, the sample where a trigger event occurs, for both Scopes 1 and 2. Scope 2 begins acquiring data as described.

- P1 = 0 and P2 = 0 — Scopes 1 and 2 start acquiring data immediately at trigger point TP.

TP   Scope 1 Simple triggering (P1 = 0)

First Sample Acquired

End of Acquisition

N1

TP   Scope 2 Simple triggering (P2 = 0)

First Sample Acquired

End of Acquisition

N2

Trigger Event
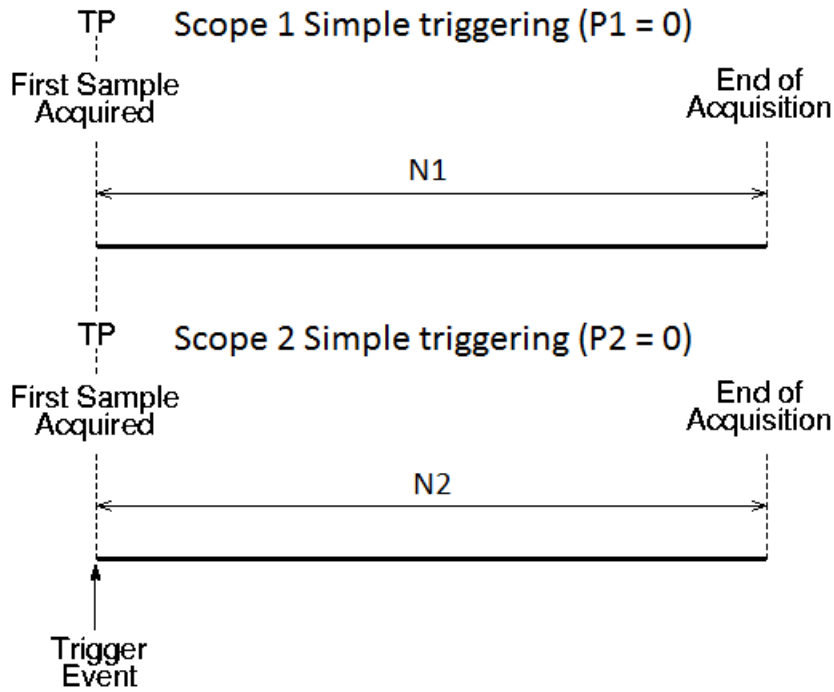
- P1 < 0 and P2 > 0 — Scope 1 starts acquiring data |P1| samples before trigger point TP. Scope 2 starts acquiring data P2 samples after trigger point TP.

Scope 1 Pre-triggering (P1 < 0)

First Sample Acquired

End of Acquisition

N1

TP

|P1|

(N1 + P1)

Scope 2 Post-triggering (P2 > 0)

End of Acquisition

TP

(N2 + P2)

P2

N2

First Sample Acquired

Trigger Event
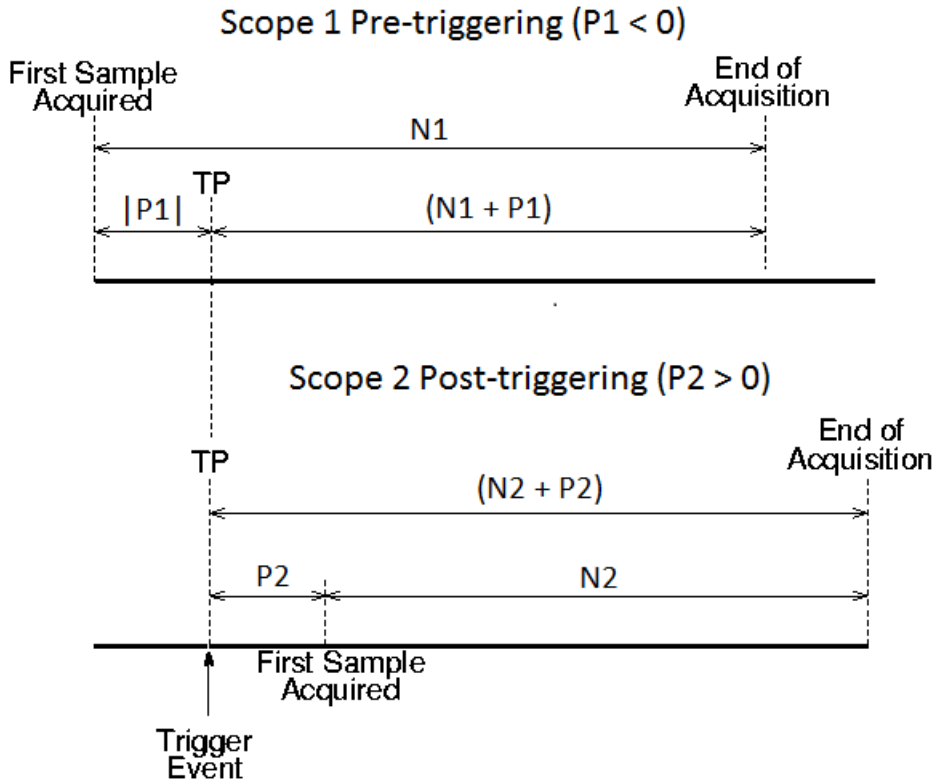
- $P1 > 0$ and $P2 < 0$— Scope 1 starts acquiring data $P1$ samples after trigger point TP. Scope 2 starts acquiring data $|P2|$ samples before trigger point TP.

Scope 1 Post-triggering (P1 > 0)

TP

End of Acquisition

(N1 + P1)

P1

N1

First Sample Acquired

Scope 2 Pre-triggering (P2 < 0)

First Sample Acquired

End of Acquisition

N2

|P2|

TP

(N2 + P2)

Trigger Event
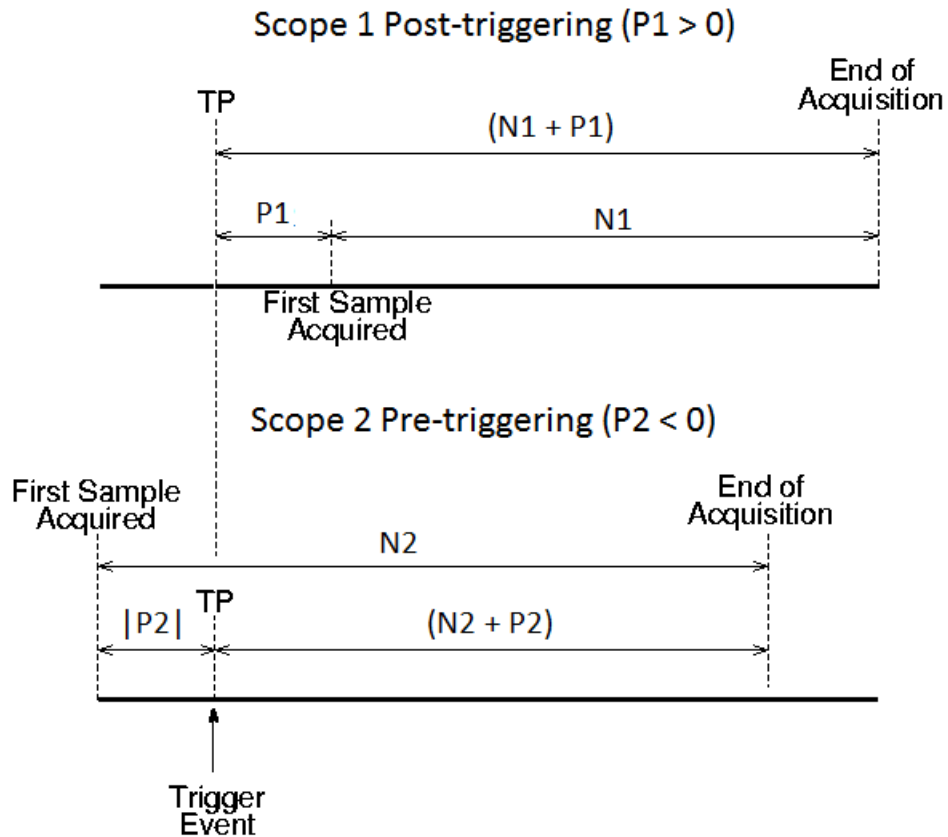
Trigger Sample Setting

For additional flexibility in scope triggering, you can use the Scope 2 trigger sample setting.

```
sc(2).TriggerSample = range 0 to (N + P1 - 1)
```

- `sc(2).TriggerSample = 0` (default) — Scope 2 triggers when Scope 1 triggers. Trigger point TP is the same sample for both scopes.

TP   Scope 1 Simple triggering (P1 = 0)

First Sample
Acquired

End of
Acquisition

N1

TP   Scope 2 Simple triggering (P2 = 0)

First Sample
Acquired

End of
Acquisition

N2
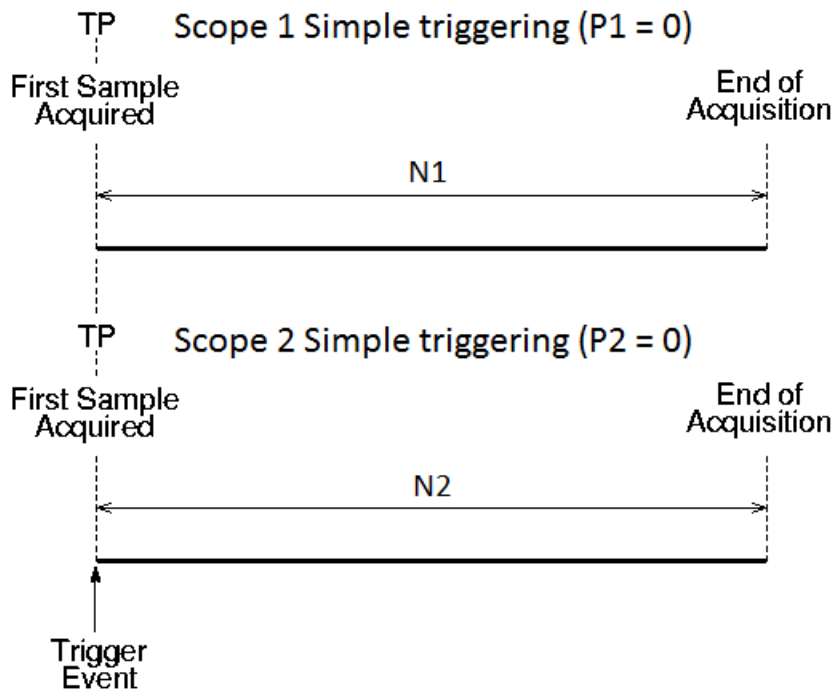
Trigger
Event

- `sc(2).TriggerSample = ts > 0` — Scope 2 triggers `ts` samples after Scope 1 is triggered. Trigger point `TP2` for Scope 2 is `ts` samples after `TP1` for Scope 1.

Scope 2 Delayed triggering (ts > 0)
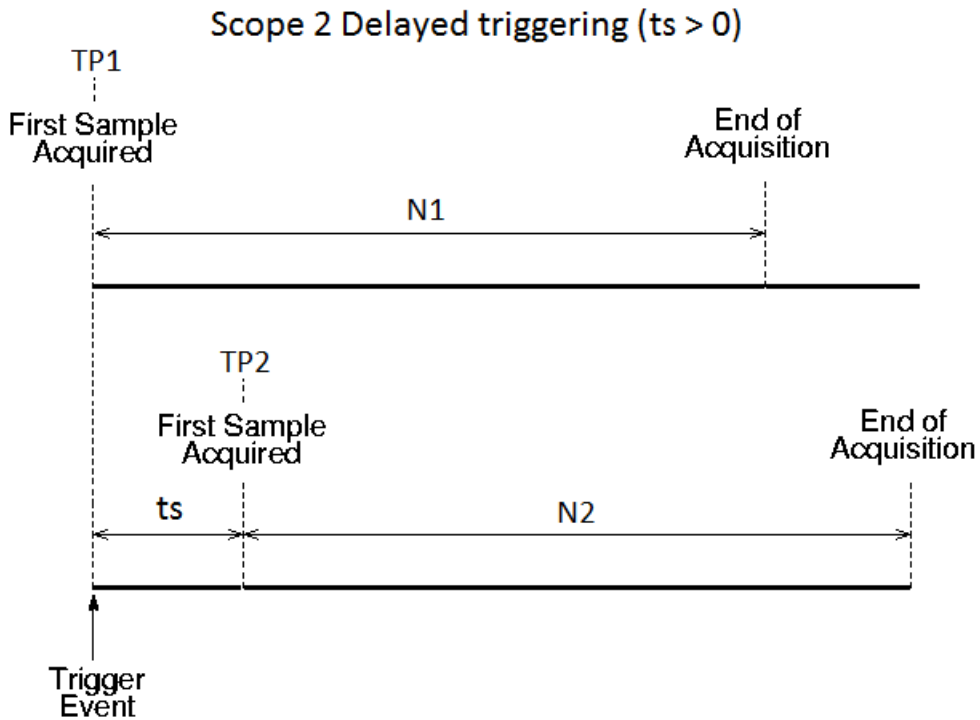
Setting `sc(2).TriggerSample` to a value `ts` larger than `(N + P - 1)` does not cause an error. It implies that Scope 2 cannot be triggered, because Scope 1 cannot acquire more than `(N + P - 1)` samples after TP.

- `sc(2).TriggerSample = -1` (special case) — Causes Scope 2 to start acquiring data from the sample after Scope 1 stops acquiring.

Scope 2 Special delayed triggering (ts = -1)
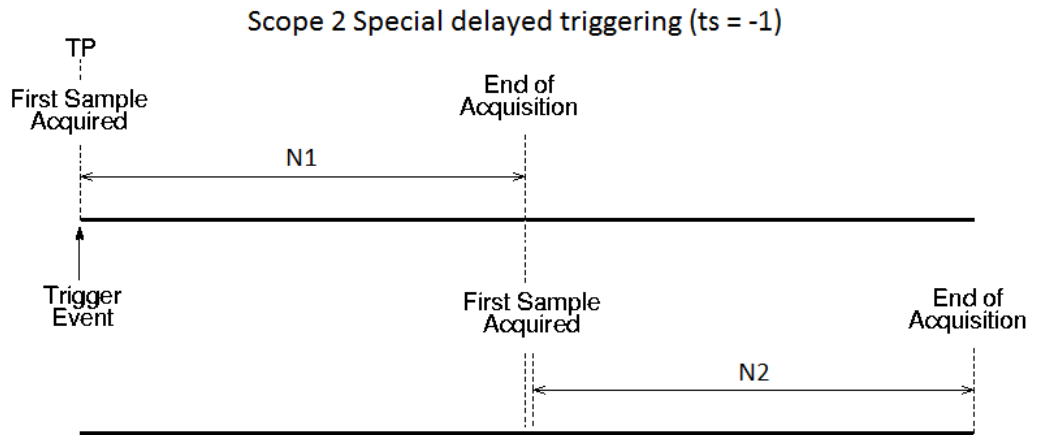
Acquire Gap-Free Data Using Two Scopes

With two scopes, you can acquire gap-free data. Gap-free data is data that two scopes acquire consecutively, without overlap. The first scope acquires data up to sample N, then stops. The second scope begins to acquire data at sample N+1.

In this example, the TriggerMode property of Scope 1 is set to 'Software'. This setting allows Scope 1 to be triggered when it receives the MATLAB command trigger(sc1).
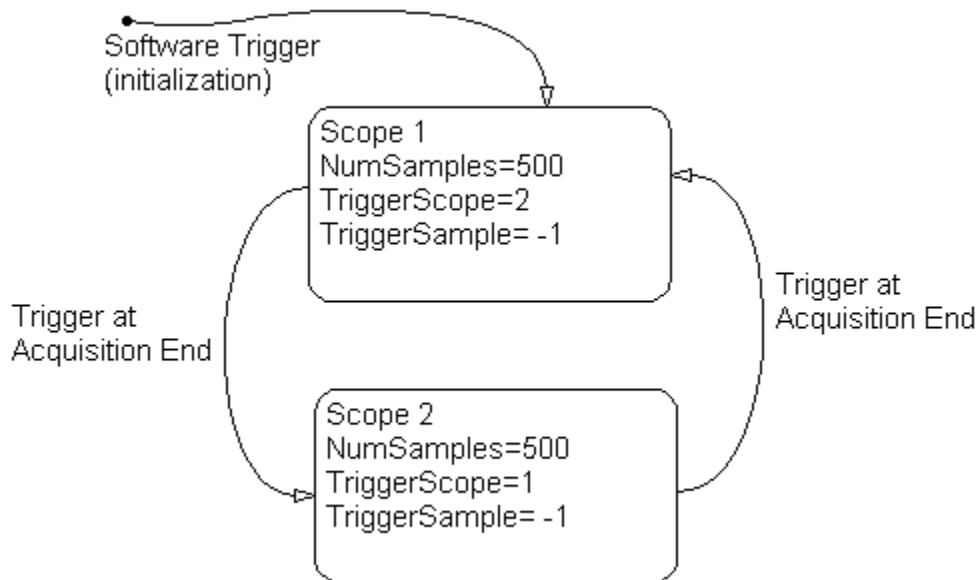
To programmatically acquire gap-free data with two scopes:

1 Build and download the Simulink model xpcosc to the target computer.

2 In the MATLAB Command Window, assign tg to the target computer and set the StopTime property to 1. For example:

```
tg = slrt;
tg.StopTime = 1;
```

3 Add two host scopes to the real-time application. You can assign the two scopes to a vector, sc, so that you can work with both scopes with one command.

```
sc = addscope(tg, 'host', [1 2]);
```

4 Add the signals of interest (0 and 1) to both scopes.

```
addsignal(sc,[0 1]);
```

5 Set the `NumSamples` property for both scopes to `500` and the `TriggerSample` property for both scopes to `-1`. With this property setting, each scope triggers the next scope *at the end* of its 500 sample acquisition.

```
sc.NumSamples = 500
sc.TriggerSample = -1
```

6 Set the `TriggerMode` property for both scopes to `'Scope'`. Set the `TriggerScope` property such that each scope is triggered by the other.

```
sc.TriggerMode = 'Scope';
sc(1).TriggerScope = 2;
sc(2).TriggerScope = 1;
```

7 Set up storage for time, `t`, and signal, `data` acquisition.

```
t    = [];
data = zeros(0, 2);
```

8 Start both scopes and the model.

```
start(sc);
start(tg);
```

Both scopes receive exactly the same signals, 0 and 1.

9 Trigger scope 1 to start acquiring data.

```
scNum = 1;
trigger(sc(scNum));
```

Setting `scNum` to 1 indicates that scope 1 will acquire data first.

10 Start acquiring data using the two scopes to double buffer the data.

```
while (1)
  % Wait until this scope has finished acquiring 500 samples
  % or the model stops (scope is interrupted).
  while ~(strcmp(sc(scNum).Status, 'Finished') || ...
          strcmp(sc(scNum).Status, 'Interrupted')), end
  % Stop buffering data when the model stops.
  if strcmp(tg.Status, 'stopped')
```

```
        break
      end
      % Save the data.
      t(   end + 1 : end + 500)     = sc(scNum).Time;
      data(end + 1 : end + 500, :) = sc(scNum).Data;
      % Restart this scope.
      start(sc(scNum));
      % Switch to the next scope.
    % Shortcut for if(scNum == 1) scNum = 2;else scNum = 1,end
    scNum = 3 - scNum;
    end
```

11 When done, remove the scopes.

```
% Remove the scopes we added.
remscope(tg,[1 2]);
```

Following is a complete code listing for the preceding double-buffering data acquisition procedure. After you download the model (xpcosc) to the target computer, you can copy and paste this code into a MATLAB file and run it. The communication speed between the development and target computers must be fast enough to handle the number of samples and can acquire the full data set before the next acquisition cycles starts. In a similar way, you can use more than two scopes to implement a triple- or quadruple-buffering scheme.

```
% Assumes model xpcosc.mdl has been built and loaded on the target computer.
% Attach to the target computer and set StopTime to 1 sec.
tg = slrt;
tg.StopTime = 1;
% Add two host scopes.
sc = addscope(tg,'host', [1 2]);
% [0 1] are the signals of interest.  Add to both scopes.
addsignal(sc,[0 1]);
% Each scope triggers next scope at end of a 500 sample acquisition.
sc.NumSamples = 500;
sc.TriggerSample', -1;
sc.TriggerMode = 'Scope';
sc(1).TriggerScope = 2;
sc(2).TriggerScope = 1;
% Initialize time and data log.
t = [];
data = zeros(0, 2);
% Start the scopes and the model.
start(sc);
start(tg);
% Start things off by triggering scope 1.
scNum = 1;
trigger(sc(scNum));
% Use the two scopes as a double buffer to log the data.
while (1)
```

```
   % Wait until this scope has finished acquiring 500 samples
   % or the model stops (scope is interrupted).
   while ~(strcmp(sc(scNum).Status, 'Finished') || ...
           strcmp(sc(scNum).Status, 'Interrupted')), end
   % Stop buffering data when the model stops.
   if strcmp(tg.Status, 'stopped')
     break
   end
   % Save the data.
   t(   end + 1 : end + 500)    = sc(scNum).Time;
   data(end + 1 : end + 500, :) = sc(scNum).Data;
   % Restart this scope.
   start(sc(scNum));
   % Switch to the next scope.
   scNum = 3 - scNum;
end
% Remove the scopes we added.
remscope(tg,[1 2]);
% Plot the data.
plot(t,data); grid on; legend('Signal 0','Signal 1');
```

Logging Signal Data with File System Objects

File System Basics

Simulink Real-Time file scopes create files on the target computer. To work with these files from the development computer, see Using File System Objects. The `SimulinkRealTime.fileSystem` object allows you to perform file system-like operations on the target computer file system.

You cannot direct the scope to write the data to a file on the Simulink Real-Time development computer. Once the software has written the signal data file to the target computer, you can access the contents of the file for plotting or other inspection from the development computer.

The software can write data files to:

- Target computer hard drive. This can be a serial ATA (SATA) or parallel ATA (PATA)/ Integrated Device Electronics (IDE) drive. The Simulink Real-Time software supports file systems of type FAT-12, FAT-16, or FAT-32 only.

  Verify that the hard drive is not cable-selected and that the target computer BIOS can detect it. The maximum file size is limited by the file system type (FAT-12, FAT-16, or FAT-32). The target computer file system uses the 8.3 file name convention. A target computer file name cannot exceed 8 characters and a file extension cannot exceed 3 characters (ABCD5678.ABC).

  A Simulink Real-Time file scope can access the target computer hard drive, provided it is formatted with a FAT-12, FAT-16, or FAT-32 file system. Simulink Real-Time ignores other file systems.

- Disks connected to a secondary IDE controller. The software supports up to four drives through the second IDE controller. By default, it works with drives configured as the primary master. If you want to use a secondary IDE controller, you must configure the Simulink Real-Time software for it (see "Converting Simulink Real-Time File Format Content to Double Precision Data" on page 12-8). The software searches for another drive in the first four ports of the target computer.

- 3.5-inch disk drive. Note that writing data files to 3.5-inch disk drives is considerably slower than writing to hard drives.

There are the following limitations:

- A maximum of eight files can be open on the target computer at the same time.
- The largest single file that you can create is 4 GB.

- A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.
- A fully qualified file name can have a maximum of 260 characters. The file part can have 12 characters — a maximum of eight characters for the file name, one character for the period, and a maximum of three characters for the file extension. If the file name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

You can access signal data files, or other target computer system files, in one of the following ways:

- If running the target computer standalone, you can access a file by restarting the target computer under an operating system such as DOS and accessing the file through the operating system utilities.
- If running the target computer linked to a development computer, you can access the target computer file system from the development computer using a `SimulinkRealTime.fileSystem` function.

You can perform file transfer operations using the functions `SimulinkRealTime.copyFileToHost` and `SimulinkRealTime.copyFileToTarget`.

You can perform file system-like tasks using functions such as `SimulinkRealTime.fileSystem.fopen` and `SimulinkRealTime.fileSystem.fread` on the signal data file. File system functions work like the corresponding MATLAB file I/O functions.

The `SimulinkRealTime.fileSystem` class also includes file system utilities that allow you to collect target computer file system information for the disk and disk buffers.

This topic focuses primarily on using the `SimulinkRealTime.fileSystem` functions to work with target computer data files that you generate from a real-time `Scope` of type `file`.

For an example of how to perform data logging with file scopes, see "Data Logging With a File Scope".

Using SimulinkRealTime.fileSystem Objects

In this section...
"Copying a File from the Target Computer to the Development Computer" on page 12-6
"Copying a File from the Development Computer to the Target Computer" on page 12-6
"Accessing File Systems on a Specific Target Computer" on page 12-7
"Reading the Contents of a File on the Target Computer" on page 12-7
"Removing a File from the Target Computer" on page 12-9
"Getting a List of Open Files on the Target Computer" on page 12-10
"Getting Information about a File on the Target Computer" on page 12-11
"Getting Information about a Disk on the Target Computer" on page 12-12

The `fileSystem` object enables you to work with the target computer file system from the development computer. You enter target object functions in the MATLAB window on the development computer or use scripts. The `fileSystem` object has functions that allow you to use

- `SimulinkRealTime.fileSystem.cd` to change folders
- `SimulinkRealTime.fileSystem.dir` to list the contents of the current folder
- `SimulinkRealTime.fileSystem.mkdir` to make a folder
- `SimulinkRealTime.fileSystem.pwd` to get the current working folder path
- `SimulinkRealTime.fileSystem.rmdir` to remove a folder
- `SimulinkRealTime.fileSystem.diskinfo` to get information about the specified disk
- `SimulinkRealTime.fileSystem.fclose` to close a file (similar to MATLAB `fclose`)
- `SimulinkRealTime.fileSystem.fileinfo` to get information about a particular file
- `SimulinkRealTime.fileSystem.filetable` to get information about files in the file system
- `SimulinkRealTime.fileSystem.fopen` to open a file (similar to MATLAB `fopen`)
- `SimulinkRealTime.fileSystem.fread` to read a file (similar to MATLAB `fread`)

- `SimulinkRealTime.fileSystem.fwrite` to write a file (similar to MATLAB `fwrite`)
- `SimulinkRealTime.fileSystem.getfilesize` to get the size of a file in bytes
- `SimulinkRealTime.fileSystem.removefile` to remove a file from the target computer

Useful global functions:

- `SimulinkRealTime.copyFileToHost` to retrieve a file from the target computer to the development computer
- `SimulinkRealTime.copyFileToTarget` to place a file from the development computer on the target computer
- `SimulinkRealTime.utils.getFileScopeData`, to interpret the raw data from the `fread` function

These procedures assume that the target computer has a signal data file created by a Simulink Real-Time file scope. This file has the pathname `C:\data.dat`.

There are the following limitations:

- A maximum of eight files can be open on the target computer at the same time.
- The largest single file that you can create is 4 GB.
- A fully qualified folder name can have a maximum of 248 characters, including the drive letter, colon, and backslash.
- A fully qualified file name can have a maximum of 260 characters. The file part can have 12 characters — a maximum of eight characters for the file name, one character for the period, and a maximum of three characters for the file extension. If the file name is longer than eight characters, the software truncates it to six characters and adds '~1' to the end of the file name.

In this section...
"Copying a File from the Target Computer to the Development Computer" on page 12-6
"Copying a File from the Development Computer to the Target Computer" on page 12-6
"Accessing File Systems on a Specific Target Computer" on page 12-7
"Reading the Contents of a File on the Target Computer" on page 12-7

Copying a File from the Target Computer to the Development Computer

You can copy a data file from the target computer to the development computer using a `SimulinkRealTime` package function on the development computer.

For example, to retrieve a file named **data.dat** from the target computer C:\ drive (default):

1　If you have not already done so, in the MATLAB window, type the following to assign the default `SimulinkRealTime.target` object to a variable.

```
tg = slrt;
```

2　Type

```
SimulinkRealTime.copyFileToHost(tg,'data.dat')
```

This retrieves the file and saves that file to the variable **data**. This content is in the Simulink Real-Time file format.

Copying a File from the Development Computer to the Target Computer

You can copy a file from the development computer to the target computer using a `SimulinkRealTime` package function on the development computer.

For example, to copy a file named **data2.dat** from the development computer to the target computer C:\ drive (default),

1　If you have not already done so, in the MATLAB window, type the following to assign the default `SimulinkRealTime.target` object to a variable.

```
tg = slrt;
```

2　Type the following to save that file to the variable **data**.

```
SimulinkRealTime.copyFileToTarget(tg,'data2.dat')
```

Accessing File Systems on a Specific Target Computer

You can access specific target computer files from the development computer for the `SimulinkRealTime.fileSystem` object.

Use the `SimulinkRealTime.fileSystem` creator function. If your system has multiple targets, you can access specific target computer files from the development computer for the `SimulinkRealTime.fileSystem` object.

For example, to list the name of the current folder of target computer `'TargetPC1'`:

1   In the MATLAB window, type a command like the following to assign the `SimulinkRealTime.fileSystem` object for the default computer to a variable.

```
fsys = SimulinkRealTime.fileSystem;
```

2   Type

```
dir(fsys)
```

Alternatively, you can use the `SimulinkRealTime.target` constructor to construct a target object for a specific computer, then use that target object as an argument to `SimulinkRealTime.fileSystem`.

1   In the MATLAB window, type a command like the following to assign the `SimulinkRealTime.target` object for target computer `'TargetPC1'` to a variable.

```
tg1 = SimulinkRealTime.target('TargetPC1');
```

2   Type the following command to assign the `SimulinkRealTime.fileSystem` object to a variable.

```
fsys = SimulinkRealTime.fileSystem(tg1);
```

3   Type

```
dir(fsys)
```

Reading the Contents of a File on the Target Computer

You can read the contents of a data file from the target computer by using `SimulinkRealTime.fileSystem` functions on the development computer. This is an alternate method to "Configure File Scopes Using MATLAB Language" on page 5-106.

Use the `function_name(fs_object, argument_list)` syntax to run a `SimulinkRealTime.fileSystem` object function. For example, to retrieve the contents of a file named `data.dat` from the target computer C:\ drive (default),

1 If you have not already done so, in the MATLAB window, type the following to assign the `SimulinkRealTime.fileSystem` object to a variable.

   ```
   fsys = SimulinkRealTime.fileSystem;
   ```

2 Type

   ```
   h = fopen(fsys, 'data.dat');
   ```

   This opens the file `data.dat` for reading and assigns the file identifier to `h`.

3 Type

   ```
   data2 = fread(fsys,h);
   ```

   This reads the file `data.dat` and stores the contents of the file to `data2`. This content is in the Simulink Real-Time file format.

4 Type

   ```
   fclose(fsys, h)
   ```

   This closes the file `data.dat`.

Before you can view or plot the contents of this file, you must convert the contents. See "Converting Simulink Real-Time File Format Content to Double Precision Data" on page 12-8.

Converting Simulink Real-Time File Format Content to Double Precision Data

The Simulink Real-Time software provides the function `SimulinkRealTime.utils.getFileScopeData` to convert Simulink Real-Time file format content (in bytes) to double precision data representing the signals and timestamps. The `SimulinkRealTime.utils.getFileScopeData` function takes in data from a file in Simulink Real-Time format. The data must be a vector of bytes (`uint8`). To convert the data to `uint8`, use a command like the following:

```
data2 = uint8(data2');
```

This section assumes that you have a variable, `data2`, that contains data in the Simulink Real-Time file format (see "Reading the Contents of a File on the Target Computer" on page 12-7).

1   In the MATLAB window, change folder to the folder that contains the Simulink Real-Time format file.

2   Type

```
new_data2 = SimulinkRealTime.utils.getFileScopeData(data2);
```

`SimulinkRealTime.utils.getFileScopeData` converts the format of `data2` from the Simulink Real-Time file format to an array of bytes. It also creates a structure for that file in `new_data2`, of which one of the elements is an array of doubles, `data`. The `data` member is also appended with a time stamp vector. The data is returned as doubles, which represent the real-world values of the original Simulink signals at the specified times during target execution.

You can view or examine the signal data. You can also plot the data with `plot(new_data2.data)`.

If you are using the Simulink Real-Time software in StandAlone mode, you can extract the data from the data file if you know the number of signals in the scope and file header size. If you know these numbers, you can extract the data. Note the following:

- First determine the file header size. To obtain the file header size, ignore the first eight bytes of the file. The next four bytes store the header size as an unsigned integer.

- After the header size number of bytes, the file stores the signals sequentially as doubles. For example, assume the scope has three signals, `x`, `y`, and `z`. Assume that `x[0]` is the value of `x` at sample 0, `x[1]` is the value at sample 1, and so forth, and `t[0]`, t[1] are the simulation time values at samples 0, 1, and so forth, respectively. The file saves the data using the following pattern:

```
x[0] y[0] z[0] t[0] x[1] y[1] z[1] t[1] x[2] y[2] z[2] t[2]...
x[N] y[N] z[N] t[N]
```

`N` is the number of samples acquired. The file saves `x`, `y`, `z`, and `t` as doubles at 8 bytes each.

Removing a File from the Target Computer

You can remove a file from the target computer by using Simulink Real-Time functions on the development computer for the `SimulinkRealTime.fileSystem` object. If you have not already done so, close this file first with `fclose`.

Use the `function_name(fs_object, argument_list)` syntax to run a `SimulinkRealTime.fileSystem` object function. For example, to remove a file named `data2.dat` from the target computer C:\ drive (default),

1   If you have not already done so, in the MATLAB window, type the following to assign the `SimulinkRealTime.fileSystem` object to a variable.

```
fsys = SimulinkRealTime.fileSystem;
```

2   Type the following to remove the specified file from the target computer.

```
removefile(fsys,'data2.dat')
```

Getting a List of Open Files on the Target Computer

You can get a list of open files on the target computer file system from the development computer by using Simulink Real-Time functions on the development computer for the `SimulinkRealTime.fileSystem` object. Do this to identify files you can close. The target computer file system limits the number of open files you can have to eight.

Use the `function_name(fs_object, argument_list)` syntax to run a `SimulinkRealTime.fileSystem` object function. For example, to get a list of open files for the file system object `fsys`,

1   If you have not already done so, in the MATLAB window, type the following to assign the `SimulinkRealTime.fileSystem` object to a variable.

```
fsys = SimulinkRealTime.fileSystem;
```

2   Type

```
filetable(fsys)
```

If the file system has open files, a list like the following is displayed:

```
ans =
Index    Handle  Flags      FilePos  Name
-------------------------------------------
    0  00060000  R__           8512  C:\DATA.DAT
    1  00080001  R__              0  C:\DATA1.DAT
    2  000A0002  R__           8512  C:\DATA2.DAT
    3  000C0003  R__           8512  C:\DATA3.DAT
    4  001E0001  R__              0  C:\DATA4.DA
```

3 The table returns the open file handles in hexadecimal. To convert a handle to one that other `SimulinkRealTime.fileSystem` functions, such as `fclose`, can use, use the `hex2dec` function. For example,

```
h1 = hex2dec('O01E0001')
h1 =
1966081
```

4 To close that file, use the `SimulinkRealTime.fileSystem` `fclose` function. For example,

```
fclose(fsys, h1)
```

Getting Information about a File on the Target Computer

You can display information for a file on the target computer file system from the development computer by using Simulink Real-Time functions on the development computer for the `SimulinkRealTime.fileSystem` object.

Use the `function_name(fs_object, argument_list)` syntax to run a `SimulinkRealTime.fileSystem` object function. For example, to display the information for the file identifier `fid1`,

1 If you have not already done so, in the MATLAB window, type the following to assign the `SimulinkRealTime.fileSystem` object to a variable.

```
fsys = SimulinkRealTime.fileSystem;
```

2 Type

```
fid1 = fopen(fsys, 'data.dat');
```

This opens the file `data.dat` for reading and assigns the file identifier to `fid1`.

3 Type

```
fileinfo(fsys, fid1)
```

This returns disk information like the following for the C:\ drive file system.

```
ans =
                FilePos: O
          AllocatedSize: 12288
          ClusterChains: 1
    VolumeSerialNumber: 1.0450e+009
```

```
          FullName: 'C:\DATA.DAT'
```

Getting Information about a Disk on the Target Computer

You can display information for a disk on the target computer file system from the development computer by using Simulink Real-Time functions on the development computer for the `SimulinkRealTime.fileSystem` object.

Use the `function_name(fs_object, argument_list)` syntax to run a `SimulinkRealTime.fileSystem` object function. For example, to display the disk information for the C:\ drive,

1   If you have not already done so, in the MATLAB window, type the following to assign the `SimulinkRealTime.fileSystem` object to a variable.

```
fsys = SimulinkRealTime.fileSystem;
```

2   Type

```
diskinfo(fsys, 'C:\');
```

This returns disk information like the following for the C:\ drive file system.

```
ans =
                 Label: 'SYSTEM '
            DriveLetter: 'C'
               Reserved: ''
           SerialNumber: 1.0294e+009
     FirstPhysicalSector: 63
                 FATType: 32
                FATCount: 2
            MaxDirEntries: 0
           BytesPerSector: 512
         SectorsPerCluster: 4
             TotalClusters: 2040293
               BadClusters: 0
              FreeClusters: 1007937
                     Files: 19968
                FileChains: 22480
                FreeChains: 1300
           LargestFreeChain: 64349
```

13

Deploy the MATLAB Application as a Standalone Executable

MATLAB Runtime Setup

As part of developing and testing your model, you can write MATLAB applications for:

- Parameter extreme value testing
- Regression testing
- Putting the model into a consistent state for testing another part of the system

To run an application on a Windows computer that does not have MATLAB installed, use MATLAB Compiler to deploy the application as a standalone executable.

To deploy a MATLAB application, first set up MATLAB Runtime:

1 Open MATLAB.

2 To find the `MCRInstaller` program, type:

```
mcrinstaller
```

The Command Window displays output similar to this output:

```
The WIN64 MATLAB Runtime Installer, version 9.0.1, is:
    C:\Program Files\MATLAB\R2016a\toolbox\compiler\deploy...
        \win64\MCRInstaller.exe
```

Note the version number of the installer program, here `9.0.1`.

3 Copy and paste the full path to the `MCRInstaller` program into the Windows Run text box.

4 Press **Enter**, and then follow the prompts.

Place the MATLAB Runtime in the following location:

```
C:\Program Files\MATLAB\MATLAB Runtime\v901\bin\win64
```

5 Open a Windows command prompt window.

6 Type the following command:

```
set PATH=C:\Program Files\MATLAB\MATLAB Runtime\v901\bin\win64;%PATH%
```

This command sets the MATLAB Runtime path only for the command prompt window within which you typed it. To make this environment variable setting visible throughout the operating system, see the Windows documentation.

Related Examples

*   "Deploy MATLAB Application to Control Real-Time Application" on page 13-4

Deploy MATLAB Application to Control Real-Time Application

Required Products: Simulink, Simulink Real-Time, MATLAB Compiler, and MATLAB Compiler SDK™

This example shows how to deploy a test script as a standalone executable by using MATLAB Compiler. The test script performs a frequency-response test of the xpcosc model. Using this information, in the design phase, you can modify the internal parameters of the model to meet your frequency requirements. In the production phase, you can bin manufactured parts based on frequency response.

The test script is slrt_freq_sweep_test.

In this section...
"Prerequisites" on page 13-4
"Package the MATLAB Application" on page 13-4
"Run the MATLAB Application" on page 13-6

Prerequisites

This procedure assumes that you have:

- Completed the steps in "MATLAB Runtime Setup" on page 13-2.
- Built the xpcosc real-time application.
- Configured TCP/IP communication between the development and target computers and recorded the required settings in the test script.

Package the MATLAB Application

1   Open **Apps** > **Application Compiler**.
2   Enter the name of the application as slrt_freq_sweep_test. Add summary information as required.
3   To save the project, click **Save**. Save the project under a name such as slrt_freq_sweep_test.prj.
4   Click the **Add main file** button ➕, and then navigate to the file slrt_freq_sweep_test.m.

5 Select the **Runtime included in package** check box.
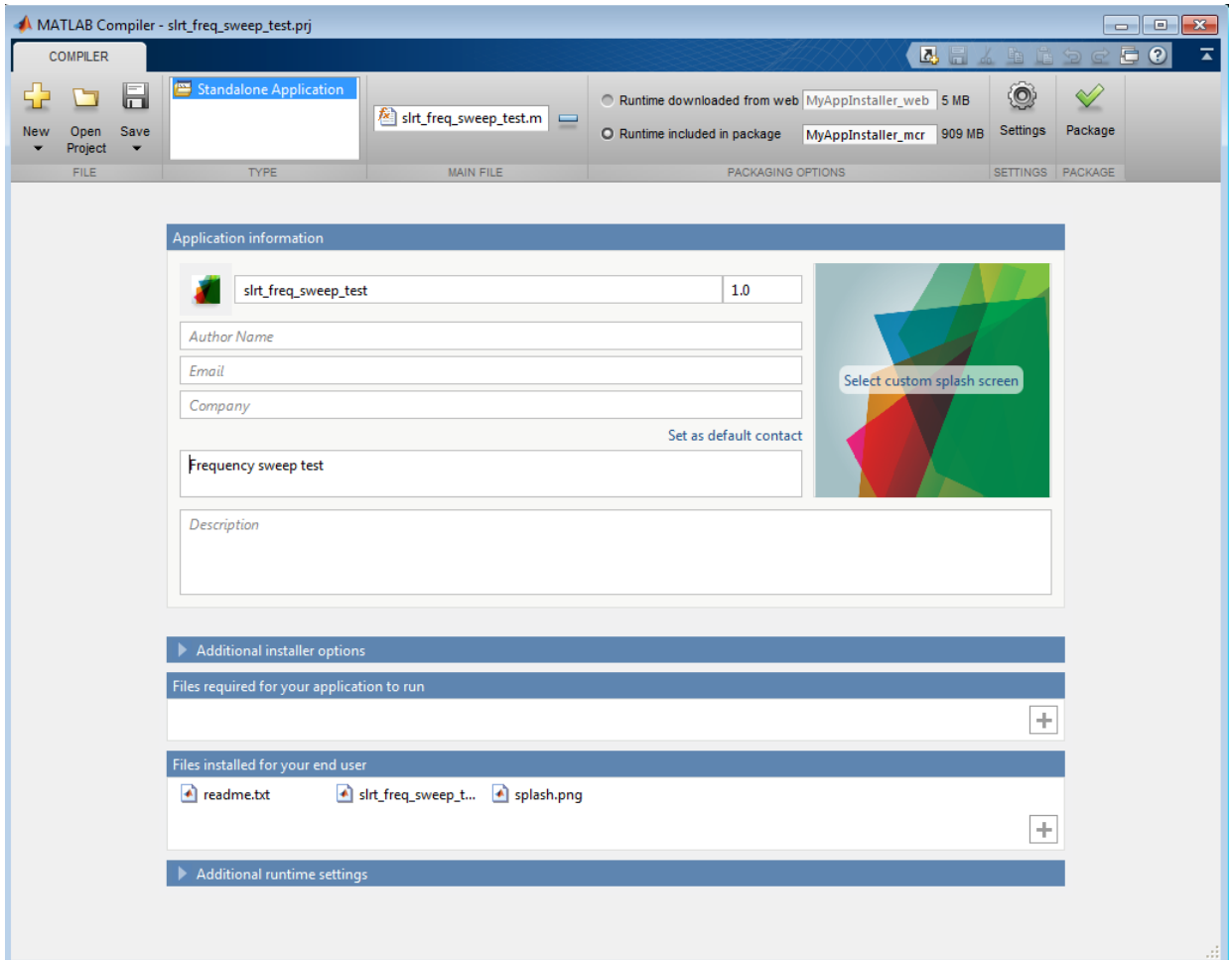
6 Click the **Package** button ✅.

The compiler generates the application and opens the `slrt_freq_sweep_test` folder.

Run the MATLAB Application

1   In Windows Explorer, navigate to `slrt_freq_sweep_test`
   `\for_redistribution_files_only`.

2   Copy the real-time application file (`xpcosc.dlm`) into `slrt_freq_sweep_test`
   `\for_redistribution_files_only`.

   The application assumes that the model file is in the folder where you run the
   application.

3   If you are connected to the target computer within MATLAB, close the connection.
   Use the `close(tg)` command.

4   To run the application, click `slrt_freq_sweep_test.exe`.

   The application runs and displays a plot for each frequency.
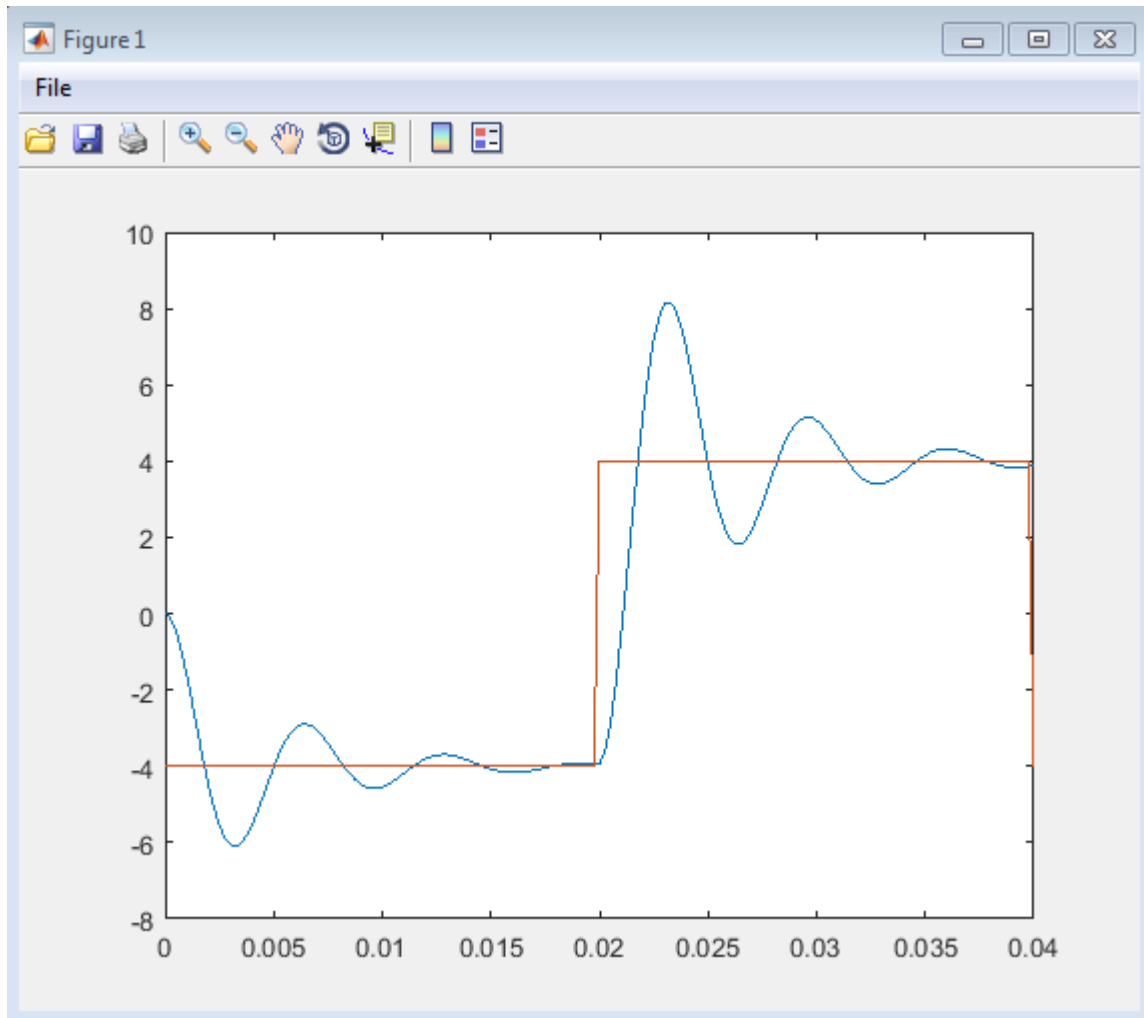
After the run is complete, the application displays a text box containing the test results.
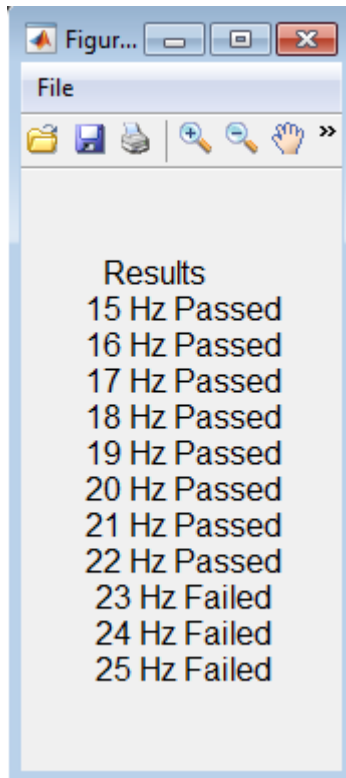
More About

- "Write Deployable MATLAB Code"

Automated Test by Using Simulink Test

Test Real-Time Application

Required Products: Simulink, Simulink Test, Simulink Real-Time

This example shows how to perform a frequency-response test of the `ex_slrt_slt_osc` model. Using this information, in the design phase, you can modify the internal parameters of the model to meet your frequency requirements. In the production phase, you can bin manufactured parts based on frequency response.

Model xpcosc
Simulink Real-Time example model

Copyright 1999-2013 The MathWorks, Inc.
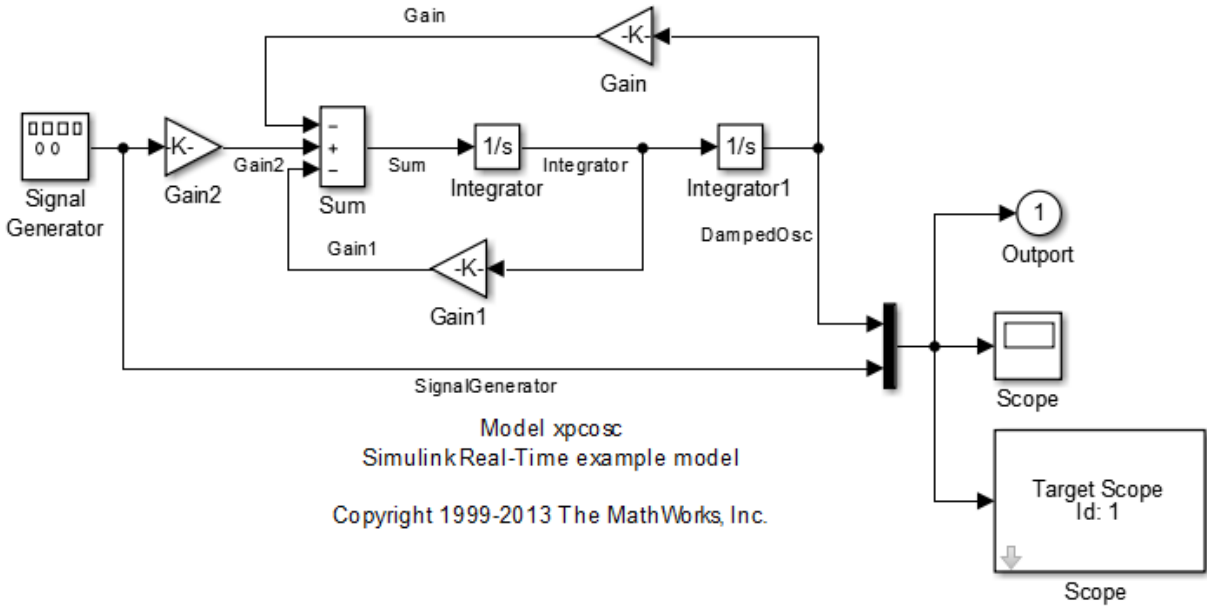
The figure shows representative output from a real-time application running on a target computer. At low frequencies, the output of the `Integrator1` block settles to the same value as the output of the `Signal Generator` block. At high frequencies, the output of the `Integrator1` block is still ringing at the end of each pulse.

The test determines the highest frequency at which the output values of the `Integrator` and `Signal Generator` blocks are within a specified criterion of each other. The test uses the model itself as a signal source and uses a test harness to compare the outputs of the `Integrator` and `Signal Generator` blocks. The test harness uses the information in these tables.

Constants

Identifier	Value	Notes
`StartFreq`	15.0	Start frequency of parameter sweep
`EndFreq`	25.0	End frequency of parameter sweep
`FreqIncr`	1.0	Frequency increment
`Criterion`	0.025	Maximum normalized amplitude difference between `Signal Generator` and `Integrator1` within the time window
`WOpen`	0.90	Start of time window for evaluating `criterion`
`WClose`	0.99	End of time window for evaluating `criterion`

Variables

Identifier	Notes
`freq`	Frequency at each iteration.
`w_open`	Window opens once in each half-period
`w_close`	Window closes once in each half-period

In this section...

Set Model Configuration Parameters

1   Open `ex_slrt_slt_osc` in a writable folder.

2   Open the Configuration Parameters dialog box.

3   Open the **Model Referencing** node, and then set **Total number of instances allowed per top model** to One.

4   Open the **Data Import/Export** node and make the following settings:

- Set **Format** to Structure with time.
- Set the **Time** and **Output** check boxes.
- Clear the **States**, **Final states**, **Signal logging**, **Data stores**, and **Log Dataset data to file** check boxes.

Create Test Harness

1   In **Analysis**, select **Test Harness** > **Create for Model**.

The software creates a test harness with the default name ex_slrt_slt_osc_Harness1.

2   Select the **Save Test Harnesses Externally** check box.

3   For Component under Test:

- Input — Select None.
- Output — Select Outport.

4   Select the **Add separate assessment block** check box.

5   In the **Initial harness configuration** box, select Verification.

6   In the **Verification Mode** box, select Normal.

7   Select the **Open harness after creation** check box.

14-5
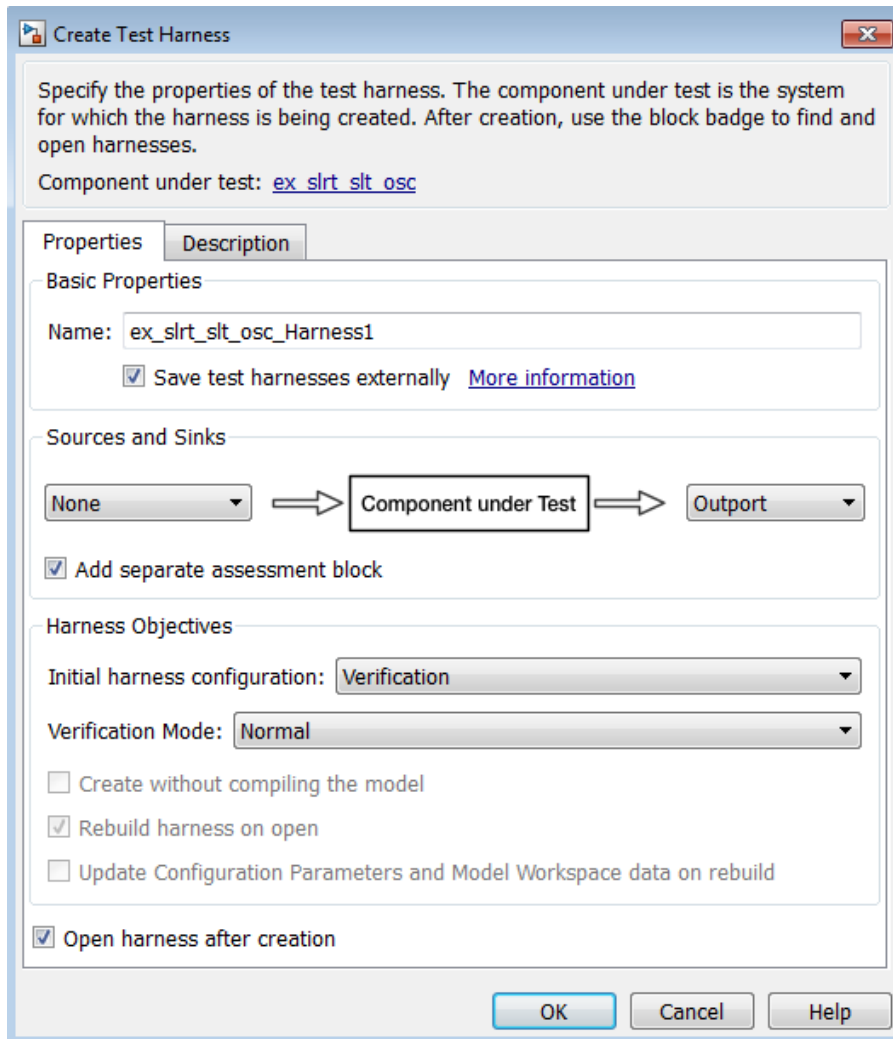
8 Click **OK**.

The test harness looks like this figure.

Test Assessment
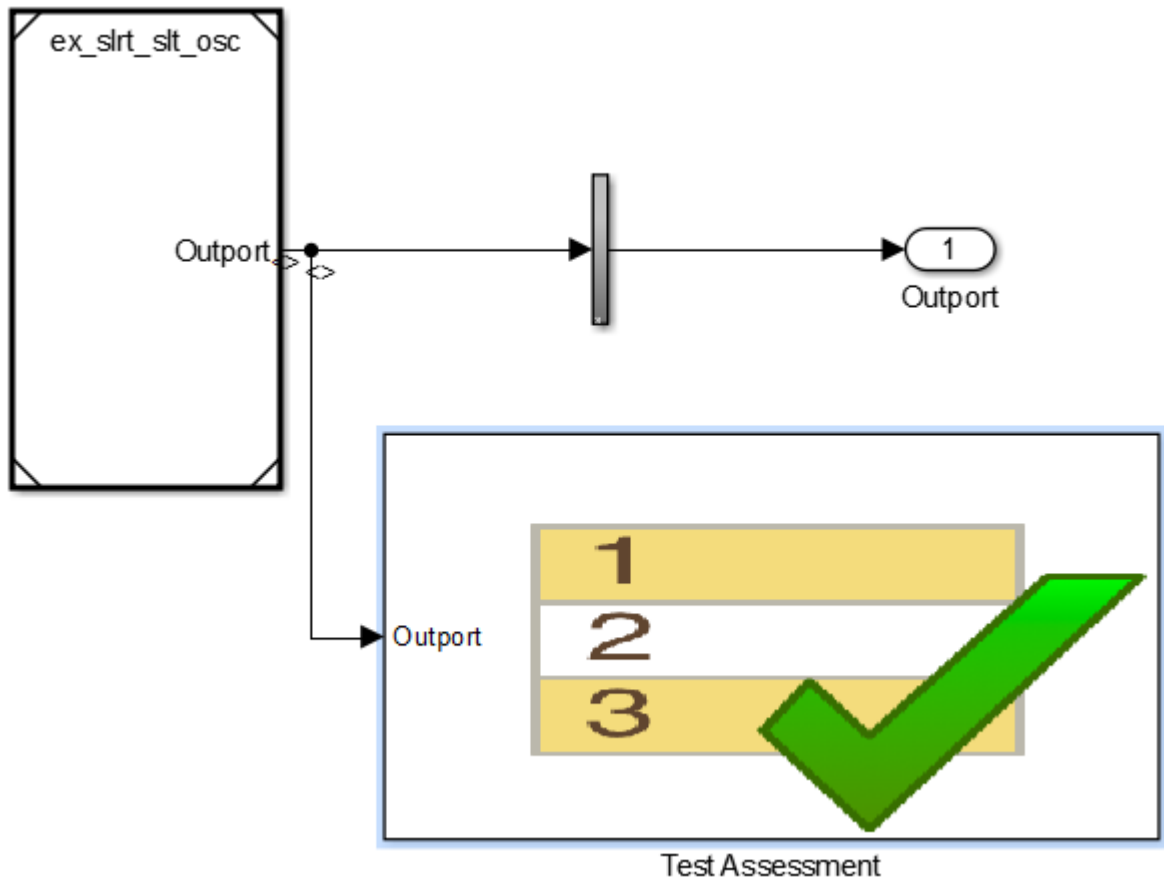
Configure Test Harness

1 Open ex_slrt_slt_osc_Harness1.
2 Open the Configuration Parameters dialog box.
3 Open the **Model Referencing** node, and then set **Total number of instances allowed per top model** to One.
4 Open the **Data Import/Export** node and make the following settings:

   - Set **Format** to Structure with time.

- Set the **Time** and **Output** check boxes.

- Clear the **States**, **Final states**, **Signal logging**, **Data stores**, and **Log Dataset data to file** check boxes.

5 Open the `Test Assessment` block.

6 To simplify the test assessment configuration, in the **Input** symbol list, replace input Outport with inputs Int1 and SigGen.

7 In `ex_slrt_slt_osc_Harness1`, connect a `Demux` block to the xpxosc_slt/Outport.

8 In the `Demux` block dialog box, set **Number of outputs** to 2.

9 To make the `Demux` outputs visible to the `Test Assessment` block, connect unitary `Gain` blocks to each of the `Demux` block outputs.

10 Connect the top `Demux` block output to Test Assessment/Int1 and the bottom output to Test Assessment/SigGen.
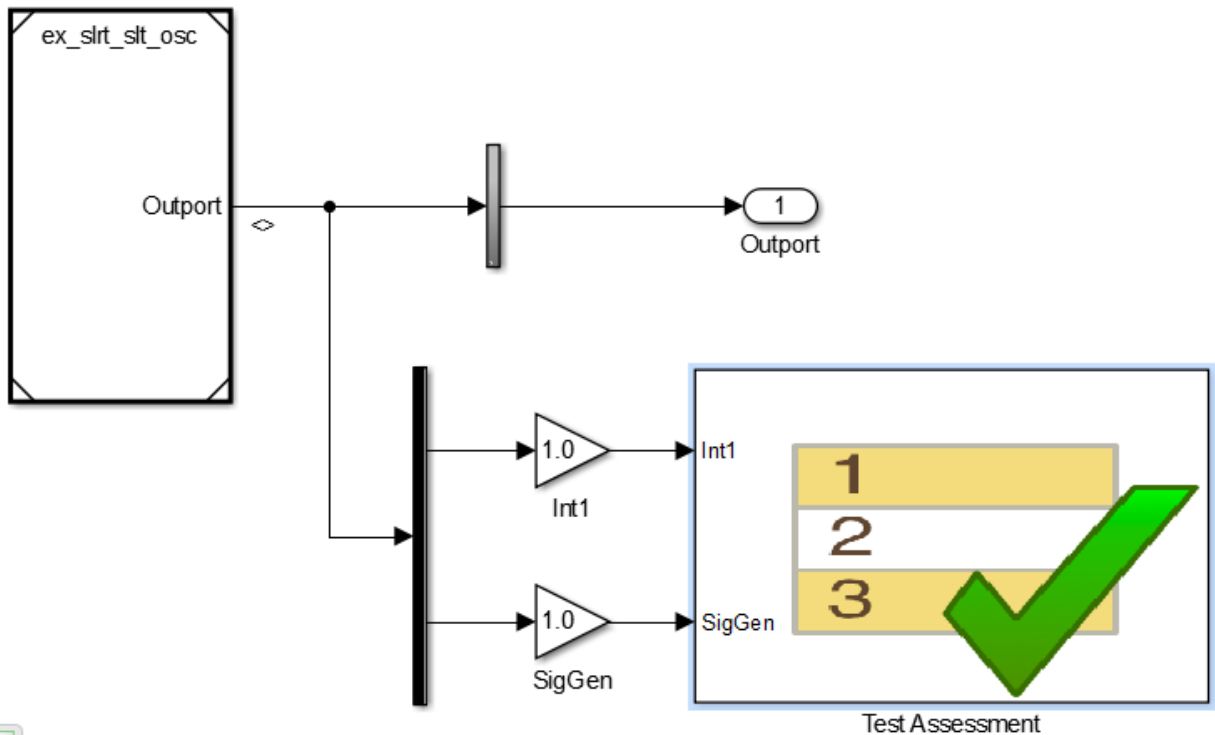
Configure Test Assessment Block

1 In the `Test Assessment` block, add these parameters to the **Parameter** symbol list:

  - `Criterion`
  - `w_open`
  - `w_close`

2 On the toolbar, click the **Model Explorer** button 　.

3 Click node **ex_slrt_slt_osc_Harness1** > **Model Workspace**.

4 In the toolbar, click the **Add Simulink Parameter** button 　.

5 Add the following data object:

  - **Name** — `Criterion`
  - **Value** — `0`
  - **DataType** — `double`

6 In a similar manner, add data objects `w_open` and `w_close`.
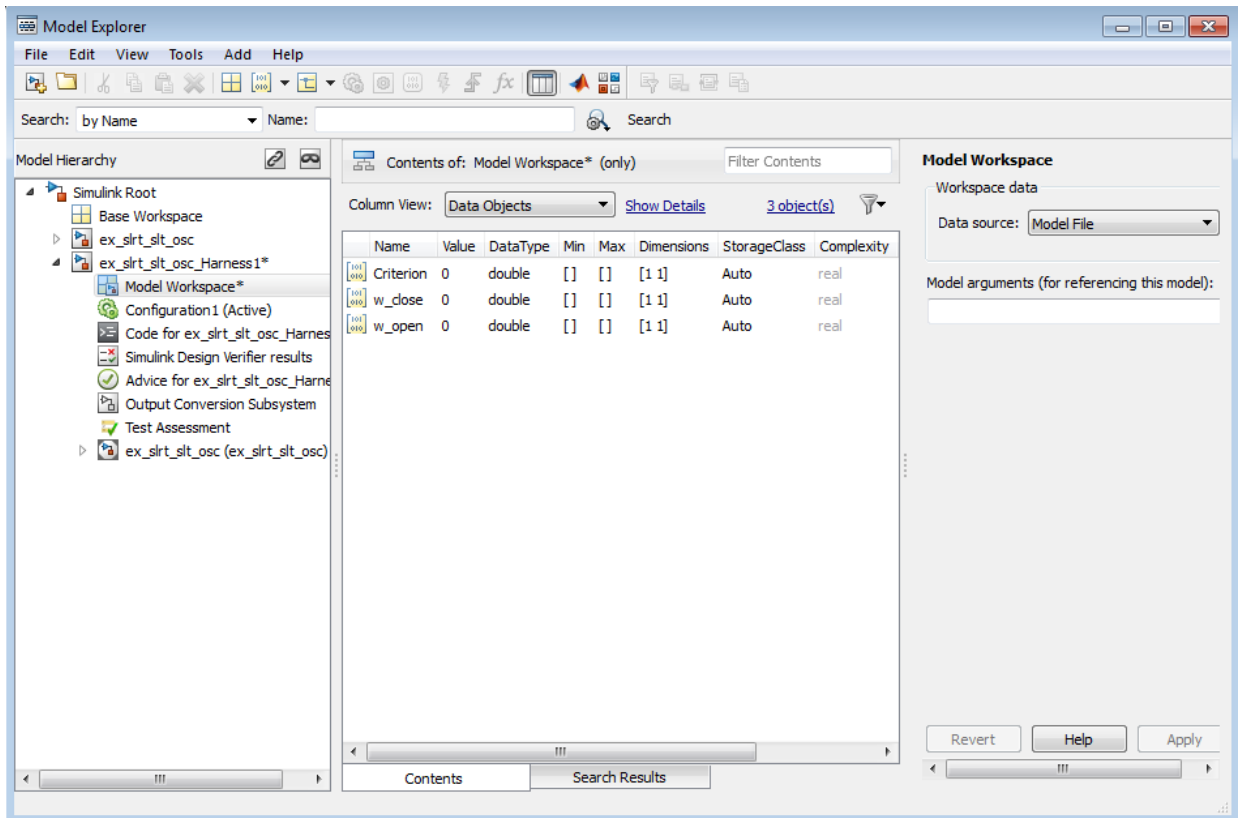
Program Test Assessment Steps

1 Open the `Test Assessment` block

2 In the **Step** column, to add a step, move the cursor to the top row, click **Add step after**, and type:

```
CheckSetting
```

3 Right-click step `CheckSetting` and set the **When decomposition** check box.

4 To add a substep to `CheckSetting`, click **Add sub-step**, and type:

```
Hi when (SigGen > 0)
```

The `when` expression selects one half of the waveform.

5 Right-click substep `Hi` and set the **When decomposition** check box.

6 To substep `Hi`, add substep:

```
HiCheck when ((et >= w_open) && (et <= w_close))
verify((abs(Int1) >= abs(SigGen) * (1.O - Criterion)) && ...
    (abs(Int1) <= abs(SigGen) * (1.O + Criterion)));
```

The `when` expression selects the time window for testing the acceptance criterion. The `verify` command tests the acceptance criterion.

7 In a similar manner, to step `CheckSetting`, add substeps:

```
Lo when (SigGen < O)
    LoCheck when ((et >= w_open) && (et <= w_close))
    verify((abs(Int1) >= abs(SigGen) * (1.O - Criterion)) && ...
        (abs(Int1) <= abs(SigGen) * (1.O + Criterion)));
```

8 Insert `DefaultStep` substeps at the end of each level of nesting to satisfy the requirements of when decomposition. When decomposition requires at least two steps at each level of nesting, and one nondecomposed step at the end of each list of steps.
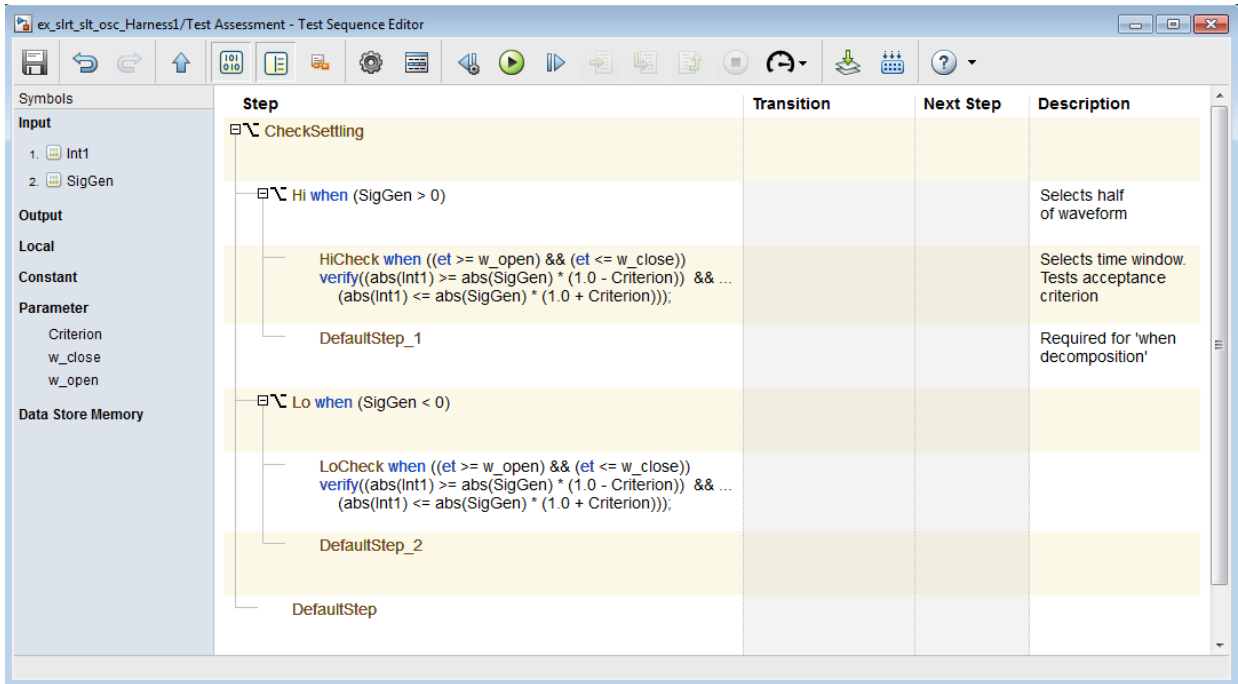
Initialize Test Suite

1   Open `ex_slrt_slt_osc`.

2   Open **Analysis** > **Test Manager**.

3   Select **New** > **Test File**.

- Right-click the default test file node, and then name the file node with a name such as **ex_slrt_slt_osc_slx_tests**.

- Right-click the default test suite node, and then name the suite node with a name such as **ex_slrt_slt_osc_stress_tests**.

- Right-click the default test case node, and then delete it.

4   Select **New** > **Real-Time Test**.

- Accept the default test type, `Simulation`, and then click **OK**.

- Right-click the test case node, and then name the case node with a name such as **FrequencySweep**.

Initialize System Under Test

1 In Test Manager, select node **FrequencySweep**.

2 Select tab **System Under Test**.

3 In the **Load Application From** box, select Model.

4 In the **Model** text box, enter ex_slrt_slt_osc.

5 In the **Target** text box, enter TargetPC1.

6 In the **Harness** box, select ex_slrt_slt_osc_Harness1.

7 Select the **Stop Time** check box.

Initialize Parameter Overrides

1 In Test Manager, select tab **Parameter Overrides**.

2 Click the **Add** button ➕.

A dialog box opens containing a list of parameters. If parameters are not visible, click the **Refresh** button ↻. The refresh builds and downloads the model and uploads the parameters from the model.

3 Open **Parameter Set 1** and select the **Frequency**, **w_close**, **w_open**, and **Criterion** check boxes.
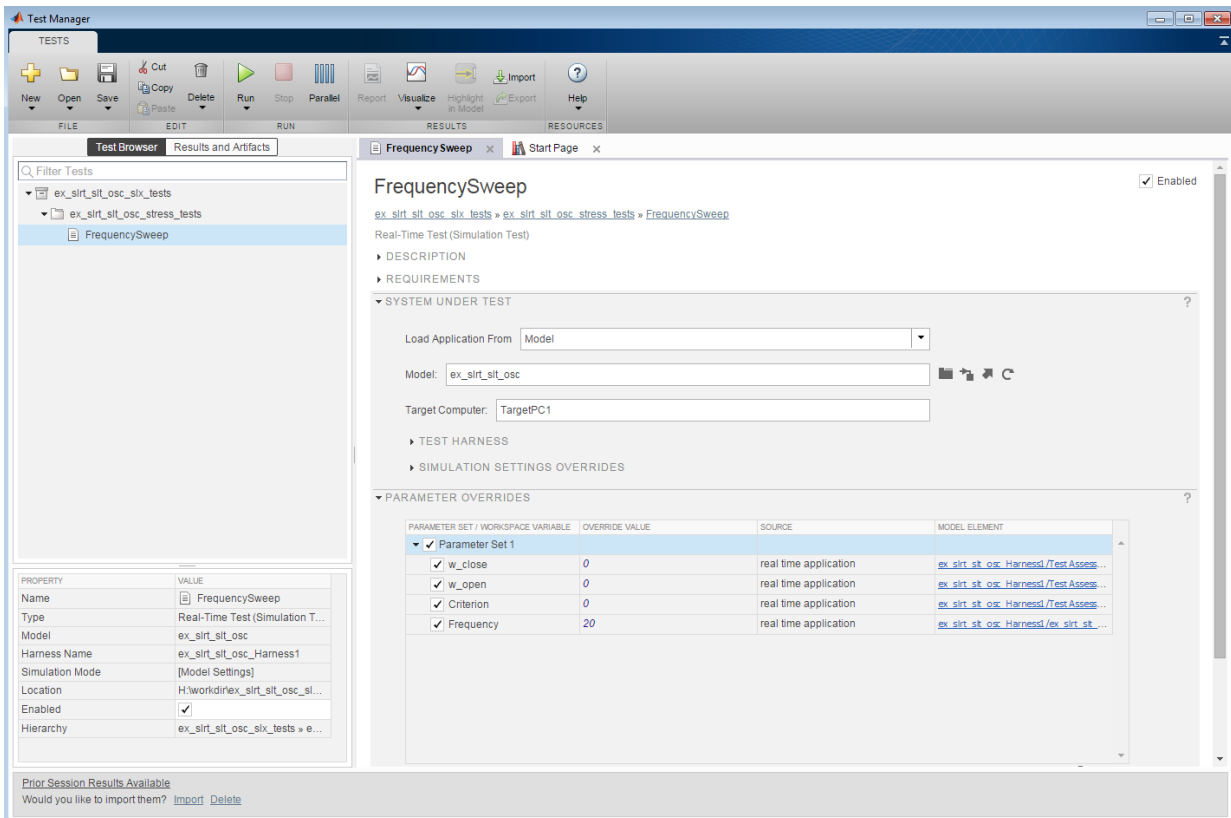
Create Scripted Iterations

1. In Test Manager, select tab **Iterations** > **Scripted Iterations**.

2. In the text box, enter the following code:

```
% Initialize constants and variables
StartFreq = 15.0;
StopFreq = 25.0;
FreqIncr = 1.0;
WOpen = 0.90;
WClose = 0.99;
Criterion = 0.025;

% Loop through test frequencies
```

```
for freq = StartFreq:FreqIncr:StopFreq

    % Create a new iteration
    testItr = sltest.testmanager.TestIteration();

    % Calculate the time window
    half_period = 0.5 * (1.0/freq);
    w_open = half_period * WOpen;
    w_close = half_period * WClose;

    % Set the parameters for the iteration
    testItr.setVariable('Name','Frequency','Source', ...
        'ex_slrt_slt_osc/Signal Generator','Value',freq);
    testItr.setVariable('Name','w_open','Source', ...
        'Test Assessment','Value', w_open);
    testItr.setVariable('Name','w_close','Source', ...
        'Test Assessment','Value', w_close);
    testItr.setVariable('Name','Criterion','Source',
        'Test Assessment','Value', Criterion);

    % Name and add the iteration to the testcase
    str = sprintf('%.0f Hz', freq);
    addIteration(sltest_testCase, testItr, str);
end
```

To resize the **Scripted Iterations** text box, click and drag the lower-right corner of the box.
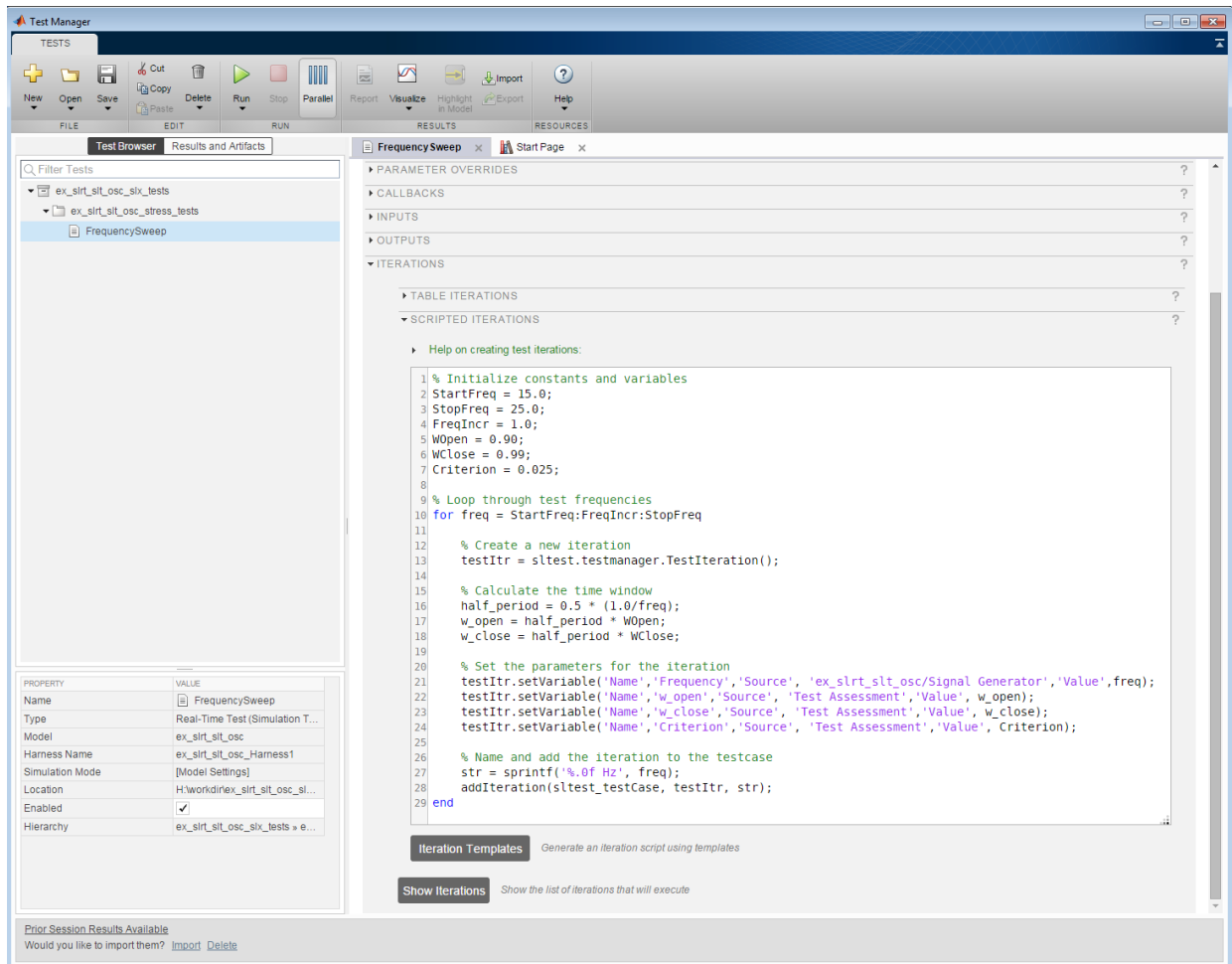
Run Test and Display Results

1 In Test Manager, on the toolbar, click the **Run** button ▷.

2 To view test results, in the left column, click **Results and Artifacts**.

   In this case, the test failed at iteration **23 Hz**.

3 To view the failing results, open nodes **23 Hz** > **Verify Statements** and **23 Hz** > **Sim Output (ex_slrt_slt_osc)**.

4 In the Simulation Data Inspector pane, select the two **Test Assessment** check boxes and the two **Out** check boxes.

5 Click the **Zoom in Time** button ⊝ and click near the first failure.

The first half-waveform succeeded, as shown by the rectangle on the left with its top near 0. The rest of the test failed, as shown by the rectangles with a spike at the top.
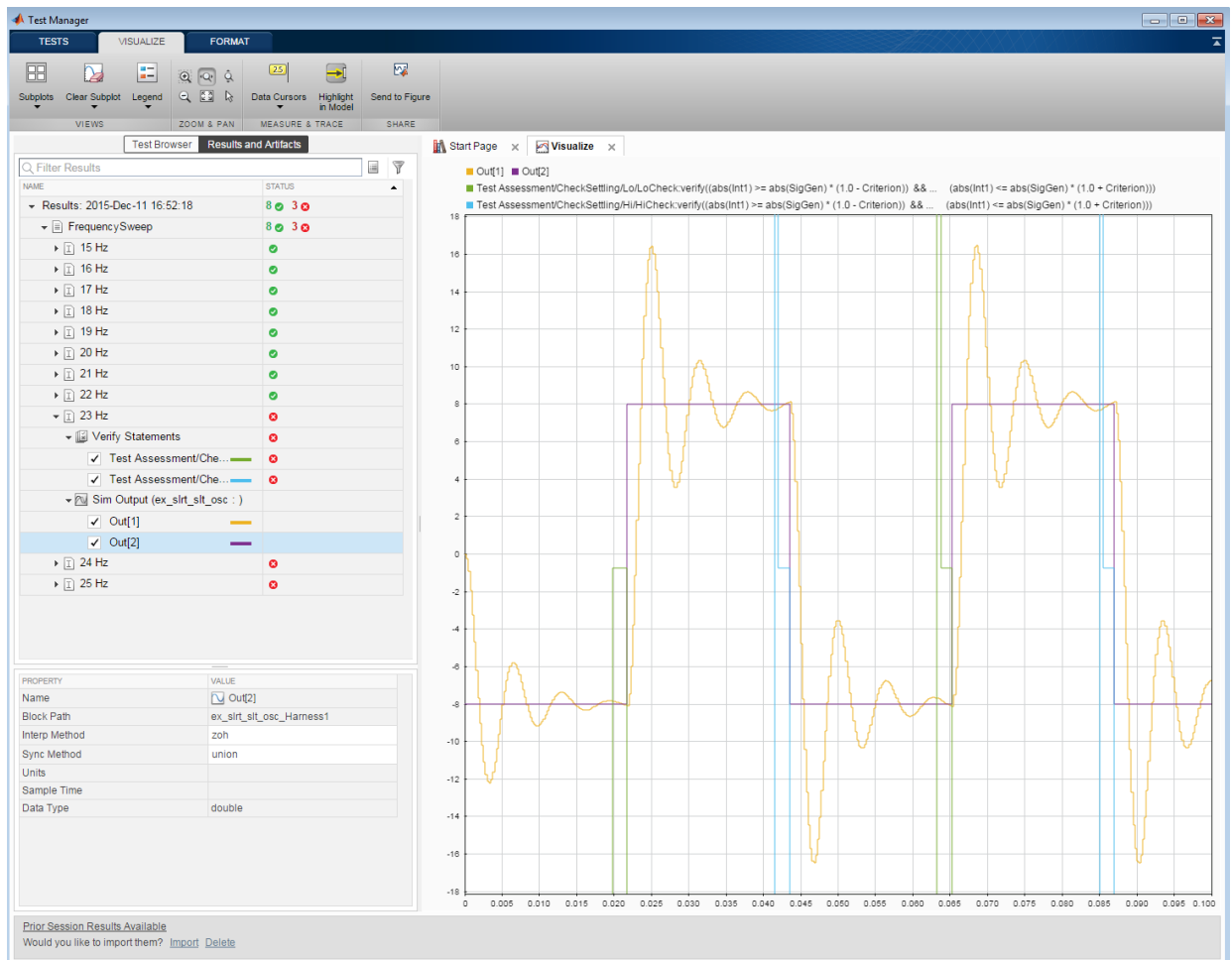
See Also
`Test Assessment` | `Test Sequence`

Related Examples
- "Test Models in Real Time"

More About
- "Real-Time Testing"

Troubleshooting

Getting Started with Troubleshooting

Refer to these guidelines, hints, and tips for questions or issues you might have about your installation of the Simulink Real-Time product. For more specific troubleshooting solutions, go to the MathWorks® Support Simulink Real-Time web site:

`www.mathworks.com/support/search_results.html?q=product:"Simulink+Real-Time"`

Troubleshooting Procedure

A Simulink Real-Time installation can sometimes fail. Causes include development and target computer failures, changes in underlying system software, I/O module failures, and procedural errors. Follow this procedure to address these problems:

1 Run the confidence test (see "Run Confidence Test on Configuration").

Tip Run the confidence test as the first step in troubleshooting, as well as in validating your initial product installation and configuration.

---

2 If one or more tests fail, see the following information about the specific failure:

- "Test 1: Ping Using System Ping" on page 16-2
- "Test 2: Ping Using slrtpingtarget" on page 16-5
- "Test 3: Software Restart Target Computer" on page 16-7
- "Test 4: Build and Download `slrttestmdl`" on page 16-9
- "Test 5: Check Ethernet Link" on page 16-12
- "Test 6: Download Prebuilt Real-Time Application" on page 16-13
- "Test 7: Execute Real-Time Application" on page 16-14
- "Test 8: Upload Logged Data and Compare Results" on page 16-15

3 Check the categorized questions and answers for clues to the root cause of the problem.

4 If the tests run, but task execution time is slow or the CPU overloads, see the questions and answers for Real-Time Application Performance.

5 Check the MathWorks Support web site, MATLAB Answers, and MATLAB Central for tips. If these do not solve your problem, call MathWorks Technical Support. See "Getting Simulink Real-Time Support" on page 28-2.

Confidence Test Failures

This topic describes guidelines, hints, and tips for questions or issues you might have while using the Simulink Real-Time product. For specific troubleshooting solutions, refer to the MathWorks Support Simulink Real-Time web site:

www.mathworks.com/support/search_results.html?q=product:"Simulink +Real-Time".

- "Test 1: Ping Using System Ping" on page 16-2
- "Test 2: Ping Using slrtpingtarget" on page 16-5
- "Test 3: Software Restart Target Computer" on page 16-7
- "Test 4: Build and Download `slrttestmdl`" on page 16-9
- "Test 5: Check Ethernet Link" on page 16-12
- "Test 6: Download Prebuilt Real-Time Application" on page 16-13
- "Test 7: Execute Real-Time Application" on page 16-14
- "Test 8: Upload Logged Data and Compare Results" on page 16-15

Test 1: Ping Using System Ping

If you are using a network connection, this test is a standard system `ping` to your target computer.

Note: The confidence test skips test 1 for serial connections.

Troubleshoot failures with the following procedure:

1 Open a DOS shell and type the IP address of the target computer:

ping xxx.xxx.xxx.xxx

Check the messages on your screen.

If DOS displays a message similar to the following, system `ping` succeeds even though test 1 fails.

Pinging xxx.xxx.xxx.xxx with 32 bytes of data:
Reply from xxx.xxx.xxx.xxx: bytes-32 time<1O ms TTL=59

If the DOS shell displays the following message, the system `ping` command failed.

Pinging xxx.xxx.xxx.xxx with 32 byte of data:
Request timed out.

2 Ping succeeds — Ethernet addresses OK?

If `ping` succeeds, check whether you entered the required IP and gateway addresses in Simulink Real-Time Explorer:

a Type `slrtexplr` in the MATLAB Command Window.

b In the **Targets** pane, expand the target computer node.

c Click the Target Properties button 🖳 in the toolbar or double-click **Properties**.

d Select **Host-to-Target communication**.

e Verify that **IP address**, **Subnet mask**, and **Gateway** boxes contain the required values.

f Select **Boot configuration**.

g Click **Create boot disk**.

h Reboot the target computer with the new kernel.

3 Ping fails — Cables OK?

If `ping` fails, first check your network cables. You might have a faulty network cable or, if you are using a coaxial cable, the terminators might be missing.

4 Ping fails — Simulink Real-Time properties OK?

Check that you have entered the required properties in Simulink Real-Time Explorer:

a Type `slrtexplr` in the MATLAB Command Window.

b In the **Targets** pane, expand the target computer node.

c Click the Target Properties button 🖥️ in the toolbar or double-click **Properties**.

d Select **Host-to-Target communication**.

e Verify that **IP address**, **Subnet mask**, and **Gateway** boxes contain the required values.

f Verify that the bus settings match those of the target computer:

- For a PCI computer: check that **Bus type** is set to **PCI** instead of **ISA**.

- For an ISA computer:

  - Check that **Bus type** is set to **ISA** instead of **PCI**.

  - Check that **Address** is set to the required I/O port base address and that the address does not conflict with that of another resource.

  - Check that **IRQ** is set to the required IRQ line and that the IRQ line does not conflict with that of another resource.

  - If the target computer motherboard contains a PCI chip set, check whether the target computer BIOS reserves the IRQ line used by the ISA bus Ethernet card.

g Select **Boot configuration**.

h Click **Create boot disk**.

i Reboot the target computer with the new kernel.

5 Ping fails — Ethernet interface operating?

Verify that your Ethernet protocol interface is operating. For example, check that the green "ready" light goes on when the cable is connected to the Ethernet card.

6 **Ping fails — Ethernet card supported?**

Verify that you are using a supported Ethernet card on the target computer. See "PCI Bus Ethernet Setup" and "USB-to-Ethernet Setup" for further details.

7 **Ping fails — Not a locally mounted folder?**

Run `slrttest` from a locally mounted folder, such as `Z:\work`, rather than from a UNC network folder, such as `\\Server\user\work`.

8 If these steps do not solve your problem, check the questions and answers for Serial and Ethernet Links and section "Faulty BIOS Settings on Target Computer" on page 18-2.

9 If you still cannot solve your problem, see "Getting Simulink Real-Time Support" on page 28-2.

Test 2: Ping Using `slrtpingtarget`

This test is a Simulink Real-Time ping to your target computer. Troubleshoot failures with the following procedure:

1   In the MATLAB Command Window, type

```
tg = SimulinkRealTime.target('argument-list')
```

where `argument-list` is the connection information that indicates which target computer you are working with. If you do not specify arguments, the software assumes that you are communicating with the default target computer.

Check the messages in the MATLAB Command Window.

MATLAB should respond with the following messages:

```
Target: TargetPC1
   Connected          = Yes
   Application        = loader
```

2   **Not connected — Bad target boot kernel?**

If you do not get the preceding messages, you could have a bad target boot kernel. To solve this problem, create a new target boot kernel and restart the target computer with the new kernel. See "Target Boot Methods".

3   **Not connected — Environment variables set?**

Use Simulink Real-Time Explorer to check the environment variables, in particular the target computer IP address. If test 1 passes but test 2 fails, you might not have entered the required IP address.

4   **Not connected — Ethernet card supported?**

Make sure you are using a supported Ethernet card (see "Test 1: Ping Using System Ping" on page 16-2).

5   **Not connected — RS-232 configuration?**

Note: RS-232 communication type has been removed. Configure TCP/IP communication instead.

---

6   If these steps do not solve your problem, check the questions and answers for Ethernet and Serial Links and section "Faulty BIOS Settings on Target Computer" on page 18-2.

7   If you still cannot solve your problem, see "Getting Simulink Real-Time Support" on page 28-2.

Test 3: Software Restart Target Computer

This test tries to boot your target computer using a Simulink Real-Time command.

Note: This procedure assumes that you have set environment settings with Simulink Real-Time Explorer. See "PCI Bus Ethernet Setup" or "USB-to-Ethernet Setup".

Troubleshoot failures with the following procedure:

1 In the MATLAB Command Window, type

```
slrttest('-noreboot')
```

This command reruns the test without using the `SimulinkRealTime.target.reboot` command and displays the message

```
### Test 3, Software reboot the target computer: ... SKIPPED
```

2 **Build Succeeded — Software restart supported?**

Check the results of `Test 4, Build and download a Simulink Real-Time application using model slrttestmdl`. If `slrttest` skips the `SimulinkRealTime.target.reboot` command but builds and loads the real-time application without producing an error message, the problem could be that the target computer does not support the Simulink Real-Time `reboot` command. In this case, you need to restart using a physical reset button.

3 **Build Failed — Kernel not loaded?**

If you saw the following error, the kernel might not be loaded when the development computer initiates communication with the target computer.

```
ReadFile Error: 6
```

Older Simulink Real-Time releases might receive this error. As a workaround, run `slrttest` with the `noreboot` option. For example,

```
slrttest('-noreboot')
```

This command runs the test without trying to restart the target computer. It displays the following message:

```
### Test 3, Software reboot the target computer: ... SKIPPED
```

4 **Build Failed — Example model modified?**

If you directly or indirectly modify the `slrttestmdl` example model supplied with the product, test 3 is likely to fail.

Note: Do not modify the files installed with the Simulink Real-Time software. If you want to modify one of these files, copy the file and modify the copy.

Restore the `slrttestmdl` example model to its original state by one of the following methods:

· Recreate the original model by editing it in the following location:

   matlabroot`\toolbox\rtw\targets\xpc\xpcdemos`

· Reinstall the software.

5 If these steps do not solve your problem, check the questions and answers for-Target Computer Boot Process and section "Faulty BIOS Settings on Target Computer" on page 18-2.

6 If you still cannot solve your problem, see "Getting Simulink Real-Time Support" on page 28-2.

Test 4: Build and Download `slrttestmdl`

This test tries to build and download the model `slrttestmdl`. Troubleshoot failures with the following procedure:

1 In the MATLAB Command Window, check the error messages.

These messages help you locate where there is a problem.

2 Build Failed — Loader not ready?

If you get the following error message, restart your target computer:

```
Simulink real-Time loader not ready
```

This error message is sometimes displayed even if the target screen shows that the loader is ready.

3 Build Failed — Using full duplex?

If the link between the development computer and target computers is Ethernet, set the development computer network interface card (NIC) card and hub to half-duplex mode. Do not set the mode to full-duplex mode.

4 Build Failed — Compiler not supported?

Verify that a supported compiler is being used and that the blocks in the model can be compiled with the given compiler and compiler version.

5 Build Failed — Compiler path?

All Microsoft Visual C++ compiler components must be in the Microsoft Visual Studio folder after installation. If the compiler is not installed at the required location, you might get one of the following errors:

```
Error executing build command: Error using ==> make_rtw
Error using ==> rtw_c (SetupForVisual)
Invalid DEVSTUDIO path specified
```

or

```
Error executing build command: Error using ==> make_rtw
Error using ==> rtw_c
Errors encountered while building model "slrttestmdl"
```

along with the following MATLAB Command Window error:

```
NMAKE: fatal error U1064: MAKEFILE not found and no target
specified
Stop.
```

Verify your compiler setup:

a   In the MATLAB command window, type:

```
slrtsetCC('setup')
```

This function queries the development computer for C compilers that the Simulink Real-Time environment supports. It returns output like the following:

```
Select your compiler for Simulink Real-Time.

[1] Microsoft Visual C++ Compilers 2008 Professional Edition (SP1) in
    c:\Program Files (x86)\Microsoft Visual Studio 9.0
[2] Microsoft Visual C++ Compilers 2010 Professional in
    C:\Program Files (x86)\Microsoft Visual Studio 10.0

[0] None

Compiler:
```

b   At the `Compiler` prompt, enter the number for the compiler that you want to use. For example, `2`.

The function verifies your selection:

```
Verify your selection:

Compiler: Microsoft Visual C++ Compilers 2010 Professional
Location: C:\Program Files (x86)\Microsoft Visual Studio 10.0

Are these correct [y]/n?
```

c   Type `y` or press **Enter** to verify the selection.

The function finishes the dialog.

```
Done...
```

6   **Build Failed — COM port read failed?**

If you see the following MATLAB Command Window error:

```
ReadFile failed while reading from COM-port
```

- Check the state of your target computer. If it is unresponsive, you might need to restart the target computer.

- In Simulink Real-Time Explorer, try to connect to the target computer again. Be sure to also check the link between the development and target computers.

7 If these steps do not solve your problem, check the questions and answers for Model Compilation, Real-Time Application Download, and Ethernet and Serial Links, and section "Faulty BIOS Settings on Target Computer" on page 18-2.

8 If you still cannot solve your problem, see "Getting Simulink Real-Time Support" on page 28-2.

Test 5: Check Ethernet Link

This error occurs only when the environment variable settings are out of date. Troubleshoot failures with the following procedure:

1 Type `slrtexplr` in the MATLAB Command Window.

2 In the **Targets** pane, expand the target computer node.

3 Click the Target Properties button 🖳 in the toolbar or double-click **Properties**.

4 Select **Host-to-Target communication** and make the required changes to the communication properties.

Note: RS-232 communication type has been removed. Configure TCP/IP communication instead.

---

5 Select **Boot configuration**.

6 Set the required **Boot mode**.

   For information on boot options, see "Target Boot Methods".

7 Click **Create boot disk**

8 Reboot the target computer.

9 Rerun `slrttest`.

10 If these steps do not resolve the issue, recreate the target boot kernel using `SimulinkRealTime.createBootImage`, restart the target computer, and rerun `slrttest`.

11 If these steps do not solve your problem, check the questions and answers for Ethernet and Serial Links and section "Faulty BIOS Settings on Target Computer" on page 18-2.

12 If you still cannot solve your problem, see "Getting Simulink Real-Time Support" on page 28-2.

Test 6: Download Prebuilt Real-Time Application

This test runs the basic target object constructor, `slrt`. This error rarely occurs unless an earlier test has failed.

1. Verify that the preceding steps completed without producing an error message.

2. Configure, build and download the tutorial model and record whatever error messages appear (see "Build and Download Real-Time Application").

3. If these steps do not solve your problem, check the questions and answers for Real-Time Application Download and Ethernet and Serial Links, and section "Faulty BIOS Settings on Target Computer" on page 18-2.

4. If you still cannot solve your problem, see "Getting Simulink Real-Time Support" on page 28-2.

Test 7: Execute Real-Time Application

This test executes a real-time application (`slrttestmdl`) on the target computer. This test fails if you change the `slrttestmdl` model start time to something other than `0`, such as `0.001`. This change causes the test, and the MATLAB interface, to halt. To address this failure:

1 Set the `slrttestmdl` model start time back to `0`.

2 Rerun the test.

3 If these steps do not solve your problem, check the questions and answers for Real-Time Application Execution, Real-Time Application Performance, Real-Time Application Signals, and Real-Time Application Parameters and section "Faulty BIOS Settings on Target Computer" on page 18-2.

4 If you still cannot solve your problem, see "Getting Simulink Real-Time Support" on page 28-2.

Test 8: Upload Logged Data and Compare Results

This test executes a real-time application (`slrttestmdl`) on the target computer. This test might fail if you change the `slrttestmdl` model (for example, if you remove the Outport block).

Note: Do not modify the files installed with the Simulink Real-Time software. If you want to modify one of these files, copy the file and modify the copy.

---

1 To eliminate this problem, restore the `slrttestmdl` example model to its original state by one of the following methods:

   - Recreate the original model by editing it in the following location:

     matlabroot\toolbox\rtw\targets\xpc\xpcdemos

   - Reinstall the software.

2 If you are running a new Simulink Real-Time release, be sure that you have a new target boot kernel for this release. See "Installing Simulink Real-Time Software Updates" on page 28-3.

3 If you are installing another version of the Simulink Real-Time software on top of an existing version, check the version number of the current installation. At the MATLAB command line, type `slrtlib`. The version number appears at the bottom of the Simulink Real-Time block library window. If the version number is not the one to which you want to upgrade, reinstall the software.

4 If these steps do not solve your problem, check the questions and answers for Real-Time Application Execution, Real-Time Application Performance, Real-Time Application Signals, and Real-Time Application Parameters and section "Faulty BIOS Settings on Target Computer" on page 18-2.

5 If you still cannot solve your problem, see "Getting Simulink Real-Time Support" on page 28-2.

Development Computer Configuration

Why Does Boot Drive Creation Halt?

If your development computer MATLAB interface halts while creating a Simulink Real-Time boot disk or network boot image:

- Use another drive to create a new Simulink Real-Time boot drive or network boot image.
- If your development computer has antivirus software, it might conflict with the MATLAB software. Disable the software while using the MATLAB interface.
- Verify that the development computer drive is accessible. If it is not accessible, replace the drive.

Target Computer Configuration

Faulty BIOS Settings on Target Computer

The BIOS settings of a target computer influence how the target computer works. If you experience problems using the Simulink Real-Time software, check the system BIOS settings of the target computer. These settings are beyond the control of the Simulink Real-Time product. See "BIOS Settings".

Faulty BIOS settings can cause issues like the following:

- Why is my target not booting?
- Why can `SimulinkRealTime.targetgetPCIInfo` detect PCI boards, but `autosearch -l` cannot?
- Why can my standalone real-time application run on some target computers, but not others?
- Why is my target computer crashing while downloading real-time applications?
- Why is my target PC104 hanging on boot?
- Why is my boot time slow?
- Why is my real-time application not running in real time?
- Why are my USB ports not working?

Why is My Hard Drive Not Recognized?

Simulink Real-Time only recognizes a target computer hard drive formatted as FAT-12, FAT-16, or FAT-32.

Simulink Real-Time does not recognize a SATA hard drive if the target computer BIOS is set to AHCI or RAID mode. Change the BIOS to IDE mode if Simulink Real-Time does not recognize your target hard drive.

Allowable File Systems on the Target Hard Drive

Simulink Real-Time only supports target computer hard drives with a file system of type FAT-12, FAT-16, or FAT-32.

File System Disabled on the Target Computer

If your target computer does not have a FAT hard disk, the monitor on the target computer displays the following error:

```
ERROR -4: drive not found
No accessible disk found: file system disabled
```

If you do not want to access the target computer file system, you can ignore this message. If you want to access the target computer file system, add a FAT hard disk to the target computer and restart the target computer.

Tip Verify that the hard drive is not cable-selected and that the BIOS can detect it.

Adjust the Target Computer Stack Size

To discover and adjust the stack size used by the real-time threads on the target computer:

1 Add the following blocks to your model:

- `Current Available Stack Size` — Outputs the number of bytes of stack memory currently available to the real-time application thread.
- `Minimum Available Stack Size` — Outputs the number of bytes that have not been used in the stack since the thread was created.

  The block traverses the entire stack at every time step to find and report unused bytes. `Minimum Available Stack Size` should be used only for diagnostic purposes.

2 Execute the real-time application, monitoring the stack size and minimal stack size.

3 Calculate a stack size that allows execution to proceed.

Note:

- To meet the memory requirements, you might have to reconfigure your target computer.

- The Simulink Real-Time kernel supports a maximum of 4 GB of target computer memory.

---

4 Adjust the stack size of the real-time threads using a `TLCOptions` setting.

For example, to set the stack size for real-time application `xpcosc` to `512` kBytes, type the following in the MATLAB Command Window:

```
set_param('xpcosc','TLCOptions','-axPCModelStackSizeKB=512')
```

See Also

`Current Available Stack Size` | `Minimum Available Stack Size` | TLCOptions Properties

Where to Find PCI Board Information

Information about the PCI devices in your target computer is useful if you want to determine what PCI boards are installed in your Simulink Real-Time system, or if you have multiple boards of a particular type in your system. Before you start, determine what boards are installed in your target computer by typing the following in the MATLAB Command Window:

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

If you have or want to use multiple boards of a particular type in your system, verify that the I/O driver supports multiple boards. See the "Multiple board support" entry for this board type in the Simulink Real-Time library or the Simulink Real-Time Interactive Hardware Selection Guide:

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

If you confirm that the board type supports multiple boards, and these boards are installed in the Simulink Real-Time system, do the following to obtain the bus and slot information for these boards:

1 In the PCI devices display, note the contents of the `Bus` and `Slot` columns of the PCI devices in which you are interested.

2 Enter the bus and slot numbers as vectors into the **PCI Slot** parameter of the PCI device. For example:

```
[1 9]
```

where `1` is the bus number and `9` is the slot number.

For additional information about PCI bus I/O devices, refer to "PCI Bus I/O Devices".

How to Diagnose My Board Driver

If you encounter issues using the Simulink Real-Time I/O drivers:

1   Display the input/output behavior of the board using an external instrument, such as an oscilloscope or logic analyzer.
2   Verify that you have configured the I/O board driver according to the manufacturer's data sheet.
3   Verify that you are using the latest version of the I/O board driver and of the Simulink Real-Time software. See "Installing Simulink Real-Time Software Updates" on page 28-3.
4   Verify that the behavior persists when you run the real-time application on a different target computer.
5   Verify that the behavior persists when you install another instance of the I/O board in the target computer.
6   Download the manufacturer I/O driver and diagnostic software from the manufacturer web site, install the driver and software on your computer, and test the I/O board using the manufacturer's software.
7   Report the issue to MathWorks Support at `www.mathworks.com/support/contact_us`.

Ethernet and Serial Links

Is There a Link Between the Development and Target Computers?

Use the following MATLAB commands from the development computer to validate the link between the development and target computers:

- `slrtpingtarget`

  The `slrtpingtarget` command performs a basic communication check between the development and target computers. This command returns `success` only if the Simulink Real-Time kernel is loaded and running and the development and target computers are linked. Use this command for a quick check of the communication between the development and target computers.

- `slrttest`

  The `slrttest` command performs a series of tests on your Simulink Real-Time system. These tests range from performing a basic link check to building and running real-time applications. At the end of each test, the command returns an `OK` or failure message. If the test is inappropriate for your setup, the command returns a `SKIPPED` message. Use this command for a thorough check of your Simulink Real-Time installation.

Link errors might also occur in the following instances:

- The target computer is running an old Simulink Real-Time boot kernel that is not in sync with the Simulink Real-Time release installed on the development computer. Create a new target boot kernel for each new release.

- If the communication between the development and target computers is Ethernet, set the development computer network interface card (NIC) card and hub to half-duplex mode. Do not set the mode to full-duplex mode.

- If you have an active firewall in your system, you might experience communication errors. For example, build errors might occur if you try to build and download a model with a thermocouple board (causing a slower initialization time) in a system that contains a firewall. To work around this issue, you can add the MATLAB interface to the firewall exception list. See also "Boards with Slow Initialization" on page 19-4

- To diagnose BIOS problems, see:

  - "Faulty BIOS Settings on Target Computer" on page 18-2
  - "BIOS Settings"

- If multiple Ethernet cards or chips are installed in the target computer, see "Timeout with Multiple Ethernet Cards" on page 19-6.

Boards with Slow Initialization

Some Simulink Real-Time boards take a long time to initialize. This situation might cause the software to run out of time before a model downloads, causing the development computer to disconnect from the target computer.

By default, if the development computer does not get a response from the target computer after downloading a real-time application and waiting five seconds, the development computer software times out. The target computer responds only after downloading and initializing the real-time application.

Usually five seconds is enough time to initialize a real-time application, but in some cases it might not be long enough. The time to download a real-time application mostly depends on your I/O module. For example, thermocouples (such as the PCI-DAS-TC board) take longer to initialize. With slower modules, you might also get errors when building and downloading an associated model. Even though the target computer is fine, a false timeout is reported and you might get an error like the following:

```
"cannot connect to ping socket"
```

This is not a fatal error. You can reestablish communication with the following procedure:

1 Type `slrtpingtarget` at the MATLAB command prompt.

2 Wait for the system to return from the `slrtpingtarget` command. If `slrtpingtarget` finds a working connection between the development and target computers, the response is something like:

```
ans =

success
```

3 Restart the target object.

Alternatively, you can increase the timeout value, using the following procedure:

1 In your Simulink model, select **Simulation** > **Model Configuration Parameters**, and navigate to the **Simulink Real-Time Options** node.

2 Clear the **Use default communication timeout** parameter.

The **Specify the communication timeout in seconds** parameter appears.

3 Specify a new timeout value, in seconds. For example, enter 20 in parameter **Specify the communication timeout in seconds**.

4 Click **OK**.

5 In Simulink Editor, from the **Code** menu, click **C/C++ Code** > **Build Model**.

In this case, the development computer waits for about 20 seconds before declaring that a timeout has occurred. It does not take 20 seconds for every download. The development computer polls the target computer about once every second, and if a response is returned, returns the success value. Only in the case where a download really fails does it take the full 20 seconds.

Timeout with Multiple Ethernet Cards

The Simulink Real-Time product supports a number of Ethernet cards and chips, as described in "PCI Bus Ethernet Setup" and "USB-to-Ethernet Setup". If your target computer has more than one of these cards or chips installed, you could experience timeout problems. For example, suppose you are using the Network Boot option to boot the target computer. If the development computer boots the target computer using Ethernet A on the target computer, it associates the IP address of the target computer with the Media Access Control (MAC) address of Ethernet adapter A. If, after it does so, the target computer BIOS connects the target computer to Ethernet B, the Simulink Real-Time software cannot connect the development and target computers because they are connected to different Ethernet controllers.

First, try to disable or remove the Ethernet controller that you will not use. For example, if you have both an on-board Ethernet controller and a separate Ethernet card, you could disable the on-board Ethernet controller through the target computer BIOS. If you are required to have multiple Ethernet adapters of the same type in the target computer, you might need to experiment to determine which Ethernet adapter the software has chosen.

If you are not using the Network Boot option to boot the target computer and cannot establish a link between the development and target computers:

1   Switch the network cable to the other Ethernet port and try again.
2   If you can establish communication, use this Ethernet port to connect the development computer to the target computer.

If you are using the Network Boot option and experience this issue, do the following:

1   Connect the network cable to Ethernet adapter B.
2   In the Command Window, type

    ```
    !arp -d
    ```

    This command removes the association between the target computer address and the network address of Ethernet adapter A from the cache of the development computer. This removal allows a new connection (and association) to be made.
3   Change the Ethernet adapter card that the Network Boot option uses. You can do this in one of the following ways:

    · Change the target computer BIOS to change the Ethernet adapter to the one that the Network Boot option is looking for.

- Follow the procedure "Ethernet Card Selection by Index" on page 4-17..

Recovery from Board Driver Errors

If an error in a driver causes the Simulink Real-Time system to crash, a timeout occurs and `slrtpingtarget` fails with an error message. In such an event, you need to restart the target and reestablish communication between the development and target computers.

To get the Simulink Real-Time system back up and running:

1 Remove the reference to the problem driver from the model.

2 Reboot the target computer.

3 At the MATLAB command line, issue `slrtpingtarget` to reestablish communications.

4 If the driver with which you are having problems is one provided by MathWorks, try to pinpoint the problem area (for example, determine whether certain settings in the driver block cause problems).

Alternatively, you can exit and restart the MATLAB interface.

How Can I Diagnose Network Problems?

If you experience network problems when using this product, use an available computer with Internet access to refer to the MathWorks Support Simulink Real-Time web site:

`www.mathworks.com/products/simulink-real-time/expert-contact.html`.

Target Computer Start Process

Why Won't the Target Computer Start?

If your target computer cannot start with the Simulink Real-Time boot disk, removable boot drive, or network boot image:

- Recreate the target boot kernel using new media.
- Verify using `SimulinkRealTime.getTargetSettings` that the current properties in the Simulink Real-Time kernel correspond with the environment variables displayed in the **Host-to-Target communication** and **Target settings** panes of Simulink Real-Time Explorer.

Tip To display the current values of Simulink Real-Time environment properties for the default target computer, type `SimulinkRealTime.getTargetSettings` without arguments. To display their allowed values, type:

```
tgs = SimulinkRealTime.getTargetSettings;
tgs.set
```

- Verify that the Simulink Real-Time boot disk or removable boot drive contains files like the following:

  - `BOOTSECT.RTT`
  - `XPCTGB1.RTA`

Note: The name of the last file varies depending on the communication method.

- If the .RTT and .RTA files are not complete, reinstall the software.
- If the target computer does not start and shows only a blank screen, to display messages that the kernel prints during startup, type:

```
tgs = SimulinkRealTime.getTargetSettings;
tgs.TargetScope = 'Disabled'
```

Restart the target computer.

If you see an error message similar to the following:

```
This target computer does not have an APIC,
which is required for the multicore kernel.
Please turn the MulticoreSupport target
setting off, recreate the boot image, and
```

restart the target computer.

set `tgs.MulticoreSupport = 'off'`, recreate the boot image, and restart the target computer.

If the target computer boots without an error message, set `tgs.TargetScope = 'Enabled'`.

- If the target setting `LegacyMultiCoreConfig` is set to `'off'`, the Simulink Real-Time kernel may not be able to discover system resources that are not compliant with the Advanced Configuration and Power Interface (ACPI) standard. To allow the kernel to discover such devices by using the legacy MPFPS in the target computer BIOS:

```
tgs = SimulinkRealTime.getTargetSettings;
tgs.LegacyMultiCoreConfig =  'on'
```

By default, `LegacyMultiCoreConfig` is set to `'on'`.

- If you are doing a network boot and the procedure displays a message similar to `TFTP Timeout`:

  - Verify that the `xpctftpserver` program is running. If it is not, recreate the network boot image.
  - Temporarily disable the Internet security (firewall) software on the development computer. If you can now start:

    - Follow the Internet security software instructions to allow the start procedure to work in its presence. For example, add the MATLAB interface to the firewall exception list.
    - Reenable the Internet security software.

- If problems persist, see the questions and answers for Target Computer Start Process.
- If you still cannot start the target computer from a boot disk or removable boot drive, you might need to replace the target computer disk drive.

Why Won't the Kernel Load?

When starting the target computer, you might see a message like the following:

```
Simulink Real-Time 4.X loading kernel..@@@@@@@@@@@@@@@@@@@@@@
```

The target computer displays this message when it cannot read and load the kernel from the target boot disk.

The probable cause is a bad boot kernel. To diagnose this problem, recreate the target boot kernel. If you have a removable boot drive, reformat the drive or use a new formatted drive. If you have a boot CD, create a new boot disk. If you are using network boot, recreate the network boot image.

Why Is the Target Medium Not Bootable?

When starting the target computer, you might get a message similar to the following:

```
Not a bootable medium or NTLDR is missing
```

Selecting either **DOS Loader** or **Stand Alone** mode instead of **Removable Disk** mode can cause this message.

To solve this problem:

1 Type `slrtexplr` in the MATLAB Command Window.

2 In the **Targets** pane, expand the target computer node.

3 Click the Target Properties button in the toolbar or double-click **Properties**.

4 Select **Boot configuration** and select the desired entry in the **Boot mode** list.

5 Click **Create boot disk.**.

Why Is the Target Computer Halted?

If your target computer displays a `System Halted` message while starting:

- Verify in the **Host-to-Target communication** pane of Simulink Real-Time Explorer that the **Target driver** parameter is configured as required by your network.

- Recreate the target boot kernel using new media and use the new kernel to restart the target computer.

- Verify that the Simulink Real-Time software supports your model of target computer. Be sure to verify the Ethernet protocol interface hardware.

Modeling

- "How Do I Handle Encoder Register Rollover?" on page 21-2
- "Why Does the From USB Video Device Block Drop Frames?" on page 21-3

How Do I Handle Encoder Register Rollover?

Encoder boards have a fixed size counter register of 16 bits, 24 bits, or 32 bits. Regardless of the size, the register eventually overflows and rolls over. Registers can roll over in either the positive or negative direction.

Some boards provide a mechanism to account for overflows or rollovers. As a best practice, you should design your model to deal with overflows or rollovers. Defining an initial count can handle the issue for some system models.

To handle register rollovers, you can use standard Simulink blocks to design the following counter algorithm types:

- Rollover Counter — Counts the number of times the output of an encoder block has rolled over. This counter should count up for positive direction rollovers and down for negative direction rollovers.

- Extended Counter — Provides a rollover count not limited by register size. For an n-bit register, this counter should be able to count values greater than $2^{(n-1)}$.

The Incremental Encoder/Utilities/Rollover sublibrary of the Simulink Real-Time library contains example blocks for these two types of counters. See Rollover Counter and Extended Counter for further details. You can use these blocks in your model as is, or modify them for your model. Connect the output of the encoder block to these blocks.

Note: To view the algorithms used in these implementations, right-click the subsystem and select **Mask > Look Under Mask**.

Keep the following requirements in mind when using these blocks:

- Some driver blocks allow an initial starting value to be loaded into the register. You must pass this value to the rollover blocks to adjust for that offset.

- The rollover block needs to know how many counts each rollover represents. Typically, this number is 2^n, where n is the size of the register in bits.

Why Does the `From USB Video Device` Block Drop Frames?

When executed on a single-core target computer, the `From USB Video Device` block can drop frames if insufficient time is available to process frames received through the USB port. If the block is dropping frames, select a larger frame interval, lengthen the sample time, or use a multicore target computer.

Model Compilation

Compiler Errors from Models Linked to DLLs

When building real-time applications, the Simulink Real-Time software supports links to static link libraries (.lib) **only**, not links to dynamic link libraries (.dll), such as Windows libraries. When you build your models, verify that you link only to static link libraries. Linking to static libraries is not an issue when you compile with Simulink Real-Time S-functions.

Real-Time Application Download

Why Does My Download Time Out?

If the development and target computers are not connected, or you have not entered the required environment properties, the download process terminates after about 5 seconds with a timeout error. Be sure that you have followed the instructions outlined in "System Configuration" before continuing.

To diagnose the problem, use the following procedure:

1   Type `slrtexplr` in the MATLAB Command Window.
2   In the **Targets** pane, expand the target computer node.
3   Click the Target Properties button ![icon] in the toolbar or double-click **Properties**.
4   Select **Host-to-Target communication** and make the required changes to the communication properties.

> **Note:** RS-232 communication type has been removed. Configure TCP/IP communication instead.

5   Select **Boot configuration** and click **Create boot disk**.
6   Restart the target computer and try downloading the real-time application again.
7   In some cases, the download might have completed even though you get a timeout error. To detect this condition, wait until the target display shows

    `System:initializing application finished.`
8   Type `slrtpingtarget` at the MATLAB command prompt.

    If `slrtpingtarget` finds a working connection between the development and target computers, the response is something like:

    `ans =`

    `success`
9   Right-click the target computer in question and select **Connect**.

    If the connection resumes, the connection is working. If the connection times out consistently for a particular model, the timeout needs to be increased. See "Increase the Time for Downloads" on page 23-3.

For information on setting up the Simulink Real-Time environment, see "PCI Bus Ethernet Setup" and "USB-to-Ethernet Setup", and then see "Target Boot Methods".

Increase the Time for Downloads

By default, if the development computer does not get a response from the target computer after downloading a real-time application and waiting about 5 seconds, the development computer software times out. On the other hand, the target computer responds only after downloading and initializing the real-time application.

Usually 5 seconds is enough time to download a real-time application, but in some cases it may not be long enough. The time to download a real-time application mostly depends on your I/O module. For example, a thermocouple takes longer to initialize. In this case, even though the target computer is fine, a false timeout is reported.

You can increase the timeout value in one of the following ways:

- At the model level, open the **Simulink** > **Model Configuration Parameters** dialog box and navigate to the **Simulink Real-Time Options** node. Clear the **Use default communication timeout** parameter and enter a new desired timeout value in the **Specify the communication timeout in seconds** parameter. For example, enter 20 to increase the value to 20 s.

- At the real-time application level, set the `CommunicationTimeOut` property to the desired timeout value. For example, to increase the value to 20 s:

```
tg = slrt;
tg.CommunicationTimeOut = 20
```

For both methods, the development computer polls the target computer about once every second, and if a response is returned, returns the success value. Only if a download really fails does the development computer wait the full twenty seconds.

Why Does the Download Halt?

If the MATLAB interface freezes and there are target ping errors, this failure is likely the result of an active firewall, a long initialization process, or both combined. To diagnose this problem, see:

24

Real-Time Application Execution

Sample Time Deviates from Expected Value

You might notice that the sample time you measure from your model is not equal to the sample time you requested. This difference depends on your target computer. Your model sample time is as close to your requested time as the target computer CPU allows.

However, digital processing does not allow infinite precision in setting the spacing between the timer interrupts. This limitation can cause the divergent sample times.

For the supported target computers, the only timer that can generate interrupts is based on a 1.193 MHz clock. For the Simulink Real-Time system, the timer is set to a fixed number of ticks of this frequency between interrupts. If you request a sample time of 1/10000 seconds, or 100 microseconds, you do not get exactly 100 ticks. Instead, the Simulink Real-Time software calculates that number as:

```
100 x 10⁻⁶ s X 1.193 x 10⁶ ticks/s = 119.3 ticks
```

The Simulink Real-Time software rounds this number to the nearest whole number, 119 ticks. The actual sample time is then:

```
119 ticks/(1.193 X 10⁶ ticks/s) = 99.75 X 10⁻⁶ s
(99.75 microseconds)
```

Compared to the requested original sample time of 100 microseconds, this value is 0.25% faster.

As an example of how you can use this value to derive the expected deviation for your target computer, assume the following:

- Output board that generates a 50 Hz sine wave (expected signal)
- Sample time of 1/10000
- Measured signal of 50.145 Hz

The difference between the expected and measured signals is 0.145 Hz, which deviates from the expected signal value by 0.29% (`0.145 / 50`). Compared to the previously calculated value of 0.25%, there is a difference of 0.04% from the expected value.

If you want to further refine the measured deviation for your target computer, assume the following:

- Output board that generates a 50 Hz sine wave (expected signal)

- Sample time of 1/10200
- Measured signal of 50.002 Hz

```
1/10200 s X 1.193 x 10^6 ticks/s = 116.96 ticks
```

Round this number to the nearest whole number of 117 ticks. The resulting frequency is then

```
(116.96 ticks/117)(50) = 49.983 Hz
```

The difference between the expected and measured signal is 0.019, which deviates from the expected signal value by 0.038% (`0.019 / 50.002`). The deviation when the sample time is 1/10000 is 0.04%.

Some amount of error is common for most computers, and the margin of error varies from machine to machine.

Note: Most high-level operating systems, like Microsoft Windows or Linux®, occasionally insert extra long intervals to compensate for errors in the timer. Be aware that the Simulink Real-Time software does not attempt to compensate for timer errors. For this product, close repeatability is more important for most models than exact timing. However, some chips might have inherent designs that produce residual jitters that could change your system behavior. For example, some Intel Pentium chips might produce residual jitters on the order of 0.5 microsecond from interrupt to interrupt.

Allowed Range of Sample Times

The complexity of your model and the speed of your target computer primarily determine the minimum sample time that you can specify. If you specify too short a sample time, the target computer can experience a CPU overload. For more about CPU overloads, see "Why Does Model Execution Produce CPU Overloads?" on page 27-4.

The kernel enforces lower and upper bounds on sample time. The lower bound is `8e-6` seconds. The upper bound is `10` seconds. The kernel sets sample times smaller than `8e-6` seconds to `8e-6` seconds. It sets sample times larger than `10` seconds to `10` seconds.

Changing Sample Time at Run Time Not Supported

Some blocks do not handle sample time changes at run time. If you include such blocks in your model, the software displays a warning message during model build. To avoid incorrect results, change the sample time in the original model, and then rebuild and download the model.

Change of Stop Time

If you change the step size of a real-time application after it has been built, it is possible that the real-time application will execute for fewer steps than you expect. The number of execution steps is:

```
floor(stop time/step size)
```

When you compile code for a model, Simulink Coder calculates a number of steps based on the current step size and stop time. If the stop time is not an integral multiple of the step size, Simulink Coder adjusts the stop time for that model based on the original stop time and step size. If you later change a step size for a real-time application but do not recompile the code, Simulink Real-Time uses the new step size and the previously adjusted stop time. The resulting model may execute for fewer steps than you expect.

For example, if a model has a stop time of 2.4 and a step size of 1, Simulink Coder adjusts the stop time of the model to 2 at compilation. If you change the step size to 0.6 but do not recompile the code, the expected number of steps is 4, but the actual number of steps is 3 because Simulink Real-Time uses the previously adjusted stop time of 2.

To avoid this problem, verify that the original stop time (as specified in the model) is an integral multiple of the original step size.

Kernel Error Code Definitions

Sometimes, the Simulink Real-Time kernel unexpectedly stops running on the target computer, displaying an error message or an error code on the target computer screen. Examples of error codes include `-9`, meaning `file not found`, or `-16`, meaning `access denied`.

If you see an error code, contact MathWorks technical support and report the error code.

More About

· "Getting Simulink Real-Time Support" on page 28-2

Cannot Use Web Interface

The web interface to the target computer requires a connection between a web browser and the IP address and port by which you access the target. If this IP address and port is already in use because you connected to the target via Simulink, Simulink Real-Time Explorer, or a MATLAB command such as slrt, the web interface cannot connect and will fail.

Tip Type the MATLAB command close(slrt) immediately before opening the web interface.

Real-Time Application Parameters

Why Does the getparamid Function Return Nothing?

The `SimulinkRealTime.target.getparamid` and
`SimulinkRealTime.target.getsignalid` functions accept `block_name` parameters.
For these functions, enter for `block_name` the mangled name that the Simulink Coder
software uses for code generation. You can determine the `block_name` as follows:

- If you do not have special characters in your model, use the `gcb` function.
- If the blocks of interest have special characters, retrieve the mangled name with
  `tg.showsignals='on'` or `tg.showparam = 'on'`.

For example, if carriage return `'\n'` is part of the block path, the mangled name returns
with carriage returns replaced by spaces.

Real-Time Application Signals

How Do I Fix Invalid File IDs?

You might get `Error -10: Invalid File ID` on the target computer if you are acquiring signal data with a file scope. This error occurs because the size of the signal data file exceeds the available space on the disk. The signal data is most likely corrupt and irretrievable. You should delete the signal data file and restart the Simulink Real-Time system. To prevent this occurrence, monitor the size of the signal data file as the scope acquires data.

For additional information, refer to the MathWorks Support Simulink Real-Time web site:

`www.mathworks.com/support/search_results.html?q=product:"Simulink +Real-Time"`.

Cannot View Mux Output in External Mode

In external mode, when you connect a `Scope` block to the output of a `Mux` block, sometimes you cannot view the signals from the mux. To address this problem, insert a `Signal Conversion` block at the input to the `Scope` block. Set the **Output** parameter to `Signal copy`.

See Also
`Signal Conversion`

27

Real-Time Application Performance

How Can I Improve Run-Time Performance?

To improve runtime performance and reduce the task execution time (TET) of a model `model`:

1 Run Performance Advisor from the Simulink **Analysis** > **Performance Tools** menu and follow the offered advice. See "Improve Performance of Multirate Model" on page 10-2.

2 Run `profile_slrt` on `model` and record where the time is being spent. See "Execution Profiling for Real-Time Applications" on page 10-15.

3 If the model contains many states (for example, more than 20 states), clear the **States** check box in the **Data Import/Export** pane of the Configuration Parameters dialog box. This disables state logging, making more memory available for the real-time application.

4 In the **Data Import/Export** pane of the Configuration Parameters dialog box, clear the **Time**, **States**, **Output**, **Final states**, **Signal logging** parameters. This turns logging off, making more computing time available for calculating the model.

5 Clear the **Log Task Execution Time** check box in the **Simulink Real-Time Options** pane of the Configuration Parameters dialog box. This disables TET logging for the real-time application.

6 Increase **Fixed-step size (fundamental sample time)** in the **Solver** pane of the Configuration Parameters dialog box. Executing with a very short sample time might overload the CPU.

7 Use polling mode, if you do not need background processes (see "Polling Mode" on page 6-4 for more on setting this mode).

8 Disable the target scope display. To do this, clear the **Graphics mode** check box in the **Target settings** pane of Simulink Real-Time Explorer.

9 Use fewer scopes in the model.

10 Eliminate or minimize target computer disk I/O in your model.

11 Reduce the number of I/O channels in the model.

12 Consider partitioning the model and running it on a multicore system (see "Multicore Programming with Simulink").

To use your target computer in multicore mode, verify that the **Multicore CPU** check box is selected in the **Target settings** pane of Simulink Real-Time Explorer.

13 Consider partitioning the model and running it on multiple target computers. This optimization might require multitarget synchronization using CAN, UDP, parallel port, or reflective memory.

14 Check the questions and answers under Real-Time Application Performance for tips on eliminating CPU overloads and improving task execution time.

15 Check the MathWorks Support web site, MATLAB Answers, and MATLAB Central for tips. If these do not solve your problem, call MathWorks Technical Support. See "Getting Simulink Real-Time Support" on page 28-2.

Why Does Model Execution Produce CPU Overloads?

A CPU overload indicates that the CPU was unable to complete processing a model time step before being asked to restart. When an overload occurs, one of the following can happen:

- The Simulink Real-Time kernel halts model execution.
- If the overload is allowed, the model execution continues until a predefined event (see "Can I Allow CPU Overloads?" on page 27-7 for details). If a model continues running after a CPU overload, the model time step is as long as the time required to finish the execution. This behavior delays the following time step.

This error might occur if you have:

- **Real CPU overloads** — Those caused by model design or target computer resources. For example, a model is trying to do more than can be done in the allocated time on the target computer. Possible reasons are:

  - The target computer is too slow or the model sample time is too small (see "How Small Can the Sample Time Be?" on page 27-6).
  - The model is too complex (algorithmic complexity).
  - The model does disk I/O on the target computer hard drive.
  - Channel I/O latency, where each I/O channel used introduces latency into the system. This might cause the execution time to exceed the model time step.

    To find latency numbers for boards supported by the block library, you can use the Simulink Real-Time Interactive Guide:

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

    For example, if your real-time application includes the National Instruments® PCI-6713 board, and you want to use four outputs:

    1 Look up the board in the Simulink Real-Time Interactive Guide:

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

    From the table, the D/A latency is $1 + 2.4 \times N$.

2 To get the latency for four outputs, calculate the latency

1 + (2.4 × 4) = 10.6 microseconds

3 Include this value in your sample time calculations.

- **Spurious CPU overloads** — Commonly caused by factors outside of the model design. These overloads are most likely caused by one of the following:

  - Advanced Power Management
  - Plug-and-Play (PnP) operating system
  - System Management Interrupts (SMIs)

  Enabling these properties can cause non-real-time behavior from the target computer. You must disable these BIOS properties for the target computer to run the application as a real-time application. See "BIOS Settings".

  Some BIOS do not allow you to disable SMIs.. However, for some chip sets, you can programmatically prevent or disable SMIs. For a solution to disabling SMIs in the Intel ICH5 family, see `www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=18832&objectType=file"`.

For further information and test models, see `www.mathworks.com/matlabcentral/fileexchange/23507`.

How Small Can the Sample Time Be?

If the model has too small a sample time, a CPU overload can occur. This error indicates that to run the real-time application, executing one step of the model requires more time than the sample time for the model (`Fixed step size` property) allows.

When this error occurs, the target object property `CPUoverload` changes from `none` to `detected`. To diagnose the issue:

1 Run `SimulinkRealTime.utils.minimumSampleTime` at the MATLAB command line.
2 Change the model `Fixed step size` property to slightly above the indicated value and rebuild the model. Use the **Solver** node in the Simulink model Configuration Parameters dialog.

Actual obtainable sample times depend on a number of factors, including:

- Processor performance
- Model complexity
- I/O block types
- Number of I/O channels

If these steps do not solve your problem, see:

"How Can I Improve Run-Time Performance?" on page 27-2.

Can I Allow CPU Overloads?

Typically, the Simulink Real-Time kernel halts model execution when it encounters a CPU overload. However, you can direct the Simulink Real-Time environment to allow CPU overloads using the `TLCOptions` settings `xPCMaxOverloads`, `xPCMaxOverloadLen`, and `xPCStartupFlag`. For more information, see "Maximizing Target Computer CPU Usage" on page 10-13.

Task Execution Time for Multirate Model

For a single-rate model, task execution time (TET) measures how long it takes the kernel to update the model state, propagate the new model state to the outputs, and log the model state during a sample interval. For a multirate model, TET measures how long it takes the kernel to perform those tasks for the base-rate portion only. You can plot the TET to show under what circumstances subsystems are executed and the additional CPU time required for those executions.

Simulink Real-Time Support

Getting Simulink Real-Time Support

For support on general MATLAB or Simulink issues, see the MathWorks Support web site:

`www.mathworks.com/support`.

For support on Simulink Real-Time issues, see the following:

- Simulink Real-Time Support web site: `www.mathworks.com/support/ search_results.html?q=product:"Simulink+Real-Time"`
- Simulink Real-Time Answers:

  - `www.mathworks.com/matlabcentral/answers/?term=Simulink+Real- Time`
  - `www.mathworks.com/matlabcentral/answers/?term=xPC+Target`

- Simulink Real-Time Central File Exchange:

  - `www.mathworks.com/matlabcentral/fileexchange/?term=Simulink +Real-Time`
  - `www.mathworks.com/matlabcentral/fileexchange/?term=xPC+Target`

If you cannot solve your problem after searching these resources, do the following:

1   Call function `SimulinkRealTime.getSupportInfo` to retrieve diagnostic information for your Simulink Real-Time configuration. This function writes the diagnostic information to the file `slrtinfo.txt` in the current folder.

Note: The `slrtinfo.txt` file might contain information sensitive to your organization. Review the contents of this file before disclosing it to MathWorks.

2   Contact the Simulink Real-Time Technical Team directly for online or phone support:

`www.mathworks.com/products/simulink-real-time/expert- contact.html`.

Installing Simulink Real-Time Software Updates

The general procedure for updating Simulink Real-Time is the same as for other MathWorks products:

1 Navigate to the MathWorks download page:

   `www.mathworks.com/downloads`.

2 Navigate to the page for the Simulink Real-Time software version you want and download it to your development computer.

3 Install and integrate the new release software.

After updating Simulink Real-Time, you must carry out the following steps to recreate your Simulink Real-Time environment.

1 Type `slrtexplr` in the Command Window.

2 In the **Targets** pane, expand the target computer node.

3 Click the Target Properties button ![icon] in the toolbar or double-click **Properties**.

4 Select **Host-to-Target communication** and select the required communication method between your development and target computers ("PCI Bus Ethernet Setup" or "USB-to-Ethernet Setup").

Note: RS-232 communication type has been removed. Configure TCP/IP communication instead.

---

5 Select **Boot configuration** and click **Create boot disk**.

6 Reboot the target computer.

7 In Simulink Editor, from the **Code** menu, click **C/C++ Code** > **Build Model** for each model to be executed.