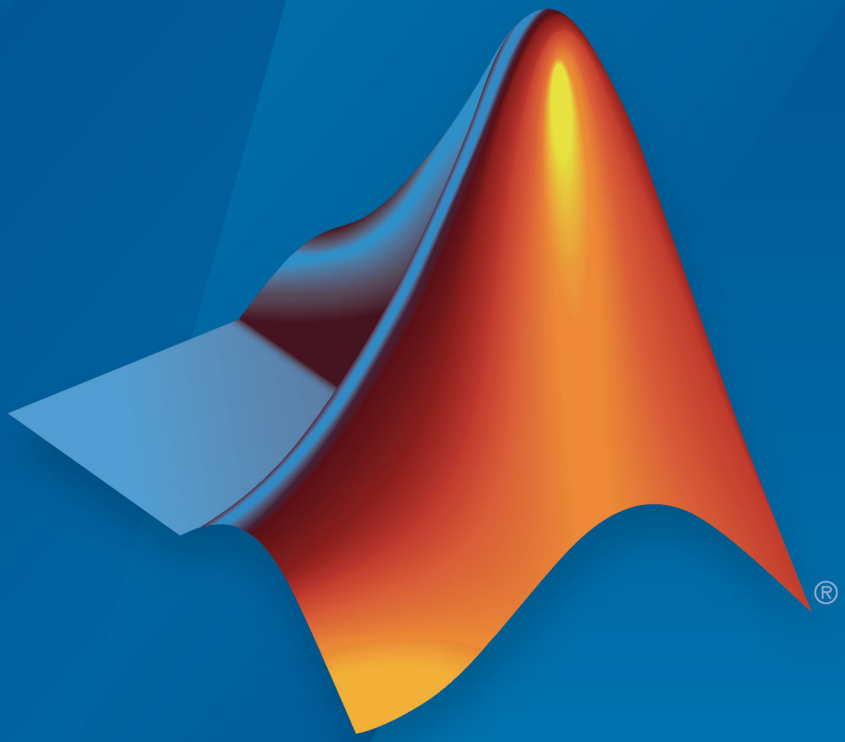
Simulink® Real-Time™

User's Guide



MATLAB® & SIMULINK®

R2018a

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

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

Simulink® Real-Time ™ User's Guide

© COPYRIGHT 1999–2018 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

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)
September 2016	Online only	Revised for Version 6.5 (Release 2016b)
March 2017	Online only	Revised for Version 6.6 (Release 2017a)
September 2017	Online only	Revised for Version 6.7 (Release 2017b)
March 2018	Online only	Revised for Version 6.8 (Release 2018a)

Contents

Model Architectures

1 FPGA Models

Speedgoat FPGA Support with HDL Workflow Advisor . . . **1-2**

FPGA Programming and Configuration **1-5**

Interrupt Configuration **1-17**
FPGA Domain Model **1-17**
Simulink Real-Time Domain Model **1-18**

FPGA Subsystem Plan **1-20**
Target Device **1-20**
FPGA Synchronization Mode **1-20**
FPGA Inports and Outports **1-21**
FPGA Clock Frequency **1-22**
FPGA Deployment **1-22**

FPGA Synchronization Modes **1-25**

2 Functional Mockup Units and Simulink Real-Time

Apply Functional Mockup Units with Simulink Real-Time **2-2**

3 Third-Party Calibration Support

Calibrate Real-Time Application **3-2**

Prepare ASAP2 Data Description File **3-4**
Initial Setup **3-6**
Set Up Parameters **3-6**
Set Up Signals **3-7**
Set Up Lookup Tables **3-8**
Generate Data Description File **3-8**

Calibrate Parameters with Vector CANape **3-10**
Prepare Project **3-10**
Prepare Device **3-10**
Configure Signals and Parameters **3-11**
Perform Signal Measurement and Parameter Calibration **3-11**

Vector CANape Limitations **3-13**

Troubleshoot Vector CANape Operation **3-14**
What This Means **3-14**
Try This **3-14**

Calibrate Parameters with ETAS Inca **3-16**
Prepare Database **3-16**
Prepare Project **3-16**
Prepare Workspace **3-17**
Prepare Experiment **3-17**
Configure Signals and Parameters **3-17**
Perform Signal Measurement and Parameter Calibration **3-17**

ETAS Inca Limitations **3-19**

Troubleshoot ETAS Inca Operation **3-20**
What This Means **3-20**
Try This **3-20**

4 Incorporating Fortran S-Functions

Fortran S-Functions **4-2**
Prerequisites **4-2**
Simulink S-Function Example **4-2**
Steps to Incorporate Fortran **4-2**

Real-Time Application Setup

5 Real-Time Application Environment

Default Target Computers **5-2**

Command-Line C Compiler Configuration **5-3**

Command-Line Setup **5-5**

Command-Line PCI Bus Ethernet Setup **5-6**
PCI Bus Ethernet Protocol Hardware **5-6**
Command-Line PCI Bus Ethernet Settings **5-7**

Command-Line USB-to-Ethernet Setup **5-10**
USB-to-Ethernet Protocol Hardware **5-10**
Command-Line USB-to-Ethernet Settings **5-12**

Ethernet Card Selection by Index **5-15**

Command-Line Ethernet Card Selection by Index **5-17**

Command-Line Target Computer Settings **5-20**

Command-Line Target Computer Boot Methods **5-22**

Command-Line Network Boot Method **5-23**

Command-Line CD/DVD Boot Method **5-25**

Command-Line DOS Loader Boot Method **5-26**

Command-Line Removable Disk Boot Method **5-28**

Command-Line Standalone Boot Method **5-30**
DOS Environment Restrictions **5-30**
Command-Line Standalone Settings **5-31**
Real-Time Application Build **5-31**
Real-Time Application Transfer and Boot Configuration **5-32**

6 Signals and Parameters

Signal Monitoring Basics **6-4**

Monitor Signals with Simulink Real-Time Explorer **6-6**

Monitor Signals with MATLAB Language **6-9**

Instrument a Stateflow Subsystem **6-11**
Configure Stateflow States as Test Points **6-11**
Monitor Stateflow States with Simulink Real-Time Explorer **6-13**

Signal Group Monitoring Formats **6-16**

Monitor Stateflow States with MATLAB Language **6-17**

Animate Stateflow Charts with Simulink External Mode **6-18**

Signal Tracing Basics **6-20**

Simulink Real-Time Scope Usage **6-21**

Target Scope Usage **6-23**

Configure Real-Time Target Scope Blocks **6-25**

Create Target Scopes with Simulink Real-Time Explorer **6-31**

Configure Scope Sampling with Simulink Real-Time Explorer **6-36**

Trigger Scopes with Simulink Real-Time Explorer **6-40**
Freerun Triggering **6-40**
Software Triggering **6-40**
Signal Triggering **6-43**
Scope Triggering **6-47**

Configure Target Scopes with Simulink Real-Time Explorer **6-51**

Configure Target Scopes with MATLAB Language **6-55**

Create Signal Groups with Simulink Real-Time Explorer **6-58**

Host Scope Usage **6-61**

Configure Real-Time Host Scope Blocks **6-62**

Create Host Scopes with Simulink Real-Time Explorer . **6-66**
Set Up Model **6-66**
Configure Host Scope **6-66**
View Host Scope **6-68**

Configure the Host Scope Viewer **6-71**

Trace Signals with Simulink External Mode **6-73**

Inspect Simulink® Real-Time™ Data with Simulation Data Inspector **6-76**

Trace or Log Data with Simulation Data Inspector **6-82**
Set Up Model **6-82**
Set Up Simulation Data Inspector **6-83**
View Simulation Data **6-83**

External Mode Usage **6-87**

Signal Logging Basics **6-88**

File Scope Usage **6-90**

Configure Real-Time File Scope Blocks **6-93**

Create File Scopes with Simulink Real-Time Explorer . . **6-98**

Configure File Scopes with Simulink Real-Time Explorer **6-102**

Log Signal Data into Multiple Files **6-106**

Log Signal Data with Outport Blocks and Simulink Real-Time Explorer **6-110**
Data Logs **6-111**
Configure the Model for Data Logging **6-112**
Log the Data **6-112**
Download and Plot the Data **6-113**

Log Signal Data with Outport Block and MATLAB Language **6-117**
Data Logs **6-118**
Configure the Model for Data Logging **6-119**
Log the Data **6-120**
Download and Plot the Data **6-120**

Signal Logging Buffer Size **6-124**

Configure File Scopes with MATLAB Language **6-125**

Tune Parameters with Simulink Real-Time Explorer . . **6-129**
Set Up Host Scope **6-129**
Initial Values **6-130**
Updated Values **6-131**

Create Parameter Groups with Simulink Real-Time Explorer **6-134**

Tune Parameters with MATLAB Language **6-137**

Tune Parameters with Simulink External Mode **6-140**

Save and Reload Parameters with MATLAB Language **6-142**
Save the Current Set of Real-Time Application Parameters **6-142**
Load Saved Parameters to a Real-Time Application . . . **6-143**
List Parameter Values Stored in a File **6-143**

Tunable Block Parameters and Tunable Global Parameters **6-145**
Tunable Parameters **6-145**
Inlined Parameters **6-146**
Tuning in External Mode **6-146**
Tuning with Simulink Real-Time Explorer **6-146**
Tuning with MATLAB Language **6-147**

Tune Inlined Parameters with Simulink Real-Time Explorer **6-148**
Configure Model to Tune Inlined Parameters **6-148**
Initial Value **6-151**
Updated Value **6-152**

Tune Inlined Parameters with MATLAB Language **6-155**

Tune Parameter Structures with Simulink Real-Time Explorer **6-157**
Create Parameter Structure **6-157**
Replace Block Parameters with Parameter Structure Fields **6-159**
Tune Parameters in a Parameter Structure **6-159**
Save and Load Parameter Structure **6-161**

Tune Parameter Structures with MATLAB Language . . **6-163**
Create Parameter Structure **6-163**
Replace Block Parameters with Parameter Structure Fields **6-164**
Tune Parameters in a Parameter Structure **6-164**
Save and Load Parameter Structure **6-166**

Define and Update Inport Data **6-167**
File Dependencies **6-167**
Map Inport to Use Square Wave **6-167**
Update Inport to Use Sawtooth Wave **6-170**

Define and Update Inport Data with MATLAB Language **6-173**
File Dependencies **6-173**
Map Inport to Use Square Wave **6-173**
Update Inport to Use Sawtooth Wave **6-175**

Inport Data Mapping Limitations **6-178**

Display and Filter Hierarchical Signals and Parameters **6-179**
Hierarchical Display **6-179**
Filtered Display **6-180**
Grouped Display **6-182**

Troubleshoot Signals Not Accessible by Name **6-184**
What This Means **6-184**
Try This **6-184**

Troubleshoot Parameters Not Accessible by Name . . . **6-186**
What This Means **6-186**
Try This **6-186**

Troubleshoot Internationalization Issues **6-187**
What This Means **6-187**
Try This **6-187**

Internationalization Issues **6-188**

7 Execution Modes

Execution Modes **7-2**
Interrupt Mode **7-3**
Polling Mode **7-3**

8 Execution with User Interface Models

Simulink Real-Time Interface Blocks to Simulink Models **8-2**
Simulink User Interface Model **8-2**
Creating a Custom Graphical Interface **8-3**
To Target Block **8-4**
From Target Block **8-6**
Creating a Real-Time Application Model **8-8**
Marking Block Parameters **8-8**
Marking Block Signals **8-10**

9 Execution Using the Target Computer Command Line

Control Real-Time Application at Target Computer Command Line **9-2**
Trace Signals at Target Computer Command Line **9-2**
Tune Parameters at Target Computer Command Line . . . **9-4**
Alias Commands at Target Computer Command Line . . . **9-5**
Find Signal and Parameter Indexes **9-5**

10 Tuning Performance

Improve Performance of Multirate Model **10-2**

Multicore Processor Configuration **10-10**

Limits on Sample Time **10-12**

CPU Overload Options **10-14**
Option Behavior **10-14**
Violation of xPCMaxOverloads **10-16**
Violation of xPCMaxOverloadLen **10-17**
Violation of xPCStartupFlag **10-17**

Execution Profiling for Real-Time Applications **10-19**

Building Referenced Models in Parallel **10-26**

Sample Time and Throughput in Real-Time Applications **10-28**
Real-Time Performance Factors **10-28**
Resources **10-28**
Improving Performance with Concurrency **10-30**
Additional Optimizations **10-48**

Execution with MATLAB Scripts

11 Real-Time Applications and Scopes in the MATLAB Interface

Real-Time Application Objects **11-2**
Create Real-Time Application Objects **11-3**
Display Application Object Properties **11-3**
Set Real-Time Application Object Property Values **11-4**
Get Real-Time Application Object Property Values **11-5**
Use Real-Time Application Object Functions **11-5**

Real-Time Scope Objects **11-7**
Display Scope Object Properties for One Scope **11-8**
Display Scope Object Properties for Multiple Scopes . . **11-9**
Set Scope Property Values **11-9**
Get Scope Property Values **11-10**

Use Scope Object Functions 11-11

Acquire Signal Data with File Scopes 11-12

Acquire Signal Data into Dynamically Named Files . . . 11-14

Scope Trigger Configuration 11-16

Pretriggering and Posttriggering of Scopes 11-17

Trigger One Scope with Another Scope 11-19
Scope-Triggered Data Acquisition 11-19
Trigger Sample Setting 11-22

Minimize Data Gaps with Two Scopes 11-26

12 Logging Signal Data with File System Objects

File System Basics 12-2

Using SimulinkRealTime.fileSystem Objects 12-5
Copying Files from the Target Computer to the Development Computer 12-7
Copying Files from the Development Computer to the Target Computer 12-7
Accessing File Systems on a Specific Target Computer 12-8
Reading the Contents of a File on the Target Computer 12-9
Removing a File from the Target Computer 12-11
Getting a List of Open Files on the Target Computer . . 12-11
Getting Information About a File on the Target Computer 12-12
Getting Information About a Disk on the Target Computer 12-13

13 Deploy the MATLAB Application as a Standalone Executable

MATLAB Runtime Setup **13-2**

Deploy MATLAB Application to Control Real-Time Application **13-4**
Prerequisites **13-4**
Package the MATLAB Application **13-4**
Run the MATLAB Application **13-6**

14 Automated Test with Simulink Test

Test Real-Time Application **14-2**

Troubleshooting

15 Troubleshooting Basics

Troubleshoot with Confidence Test **15-2**

16 Confidence Test Failures

Test 1: Ping Target Computer with System Ping **16-2**

Test 2: Ping Target Computer with slrtpingtarget **16-4**

Test 3: Software Restart Target Computer **16-5**

Test 4: Build and Download slrttestmdl **16-7**

Test 5: Check Communication with Target Computer . . **16-9**

Test 6: Download Prebuilt Real-Time Application **16-10**

Test 7: Execute Real-Time Application **16-11**

Test 8: Upload Logged Data and Compare Results **16-12**

17 Development Computer Configuration

Troubleshoot Halted Boot Drive Creation **17-2**
What This Means **17-2**
Try This **17-2**

18 Target Computer Configuration

Troubleshoot Target Computer Stack Size **18-2**
What This Means **18-2**
Try This **18-2**

Troubleshoot Target Computer Ethernet and MAC Address Information **18-4**
What This Means **18-4**
Try This **18-4**

19 Link Between Development and Target Computers

Troubleshoot Communication Failure with Target Computers **19-2**
What This Means **19-2**
Try This **19-2**

Troubleshoot Communication Timeout with Target Computers **19-4**
What This Means **19-4**
Try This **19-4**

Troubleshoot Communication Timeout with Target Computers and Multiple Ethernet Cards **19-6**
What This Means **19-6**
Try This **19-6**

20 Target Computer Boot Process

21 Model Compilation

Troubleshoot Model Links to DLLs **21-2**
What This Means **21-2**
Try This **21-2**

Troubleshoot Build Error for Accelerator Mode **21-3**
What This Means **21-3**
Try This **21-3**

22 Real-Time Application Execution

Troubleshoot Missing or Unreadable Crash Information 22-2
What This Means 22-2
Try This 22-2

Troubleshoot Unexpected Measured Sample Time Value 22-4
What This Means 22-4
Try This 22-5

Troubleshoot Changed Sample Time at Run Time Affects Results 22-6
What This Means 22-6
Try This 22-6

Troubleshoot Unexpected Measured Stop Time Value 22-7
What This Means 22-7
Try This 22-7

23 Real-Time Application Signals

Troubleshoot Invalid File IDs on Target Computer 23-2
What This Means 23-2
Try This 23-2

Troubleshoot Missing Mux Block Output on Scope 23-3
What This Means 23-3
Try This 23-3

24 Real-Time Application Performance

Troubleshoot Unsatisfactory Real-Time Performance . . **24-2**
What This Means **24-2**
Try This **24-2**

Troubleshoot Overloaded CPU from Executing Real-Time Application **24-5**
What This Means **24-5**
Try This **24-5**

Troubleshoot Task Execution Time **24-7**
What This Means **24-7**
Try This **24-7**

Troubleshoot Failed Read of Profiling Data **24-8**
What This Means **24-8**
Try This **24-8**

Troubleshoot Timeout During File System Access **24-9**
What This Means **24-9**
Try This **24-9**

25 Simulink Real-Time Support

Find Simulink Real-Time Support **25-2**

Install Simulink Real-Time Software Updates **25-3**

Model Architectures

FPGA Models

- “Speedgoat FPGA Support with HDL Workflow Advisor” on page 1-2
- “FPGA Programming and Configuration” on page 1-5
- “Interrupt Configuration” on page 1-17
- “FPGA Subsystem Plan” on page 1-20
- “FPGA Synchronization Modes” on page 1-25

Speedgoat FPGA Support with HDL Workflow Advisor

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

For an example that shows the development workflow for FPGA I/O boards, see “FPGA Programming and Configuration” on page 1-5. You do not use these blocks outside of HDL Coder HDL Workflow Advisor.

To use these blocks, open HDL Coder HDL Workflow Advisor and use it to generate a Simulink Real-Time interface subsystem. See “FPGA Programming and Configuration” on page 1-5.

The subsystem mask controls the block parameters. Do not edit the parameters directly. The FPGA I/O board block descriptions are for informational purposes only.

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.com`) FPGA I/O boards.

`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. It also provides two 16-bit 105-MHz analog input channels. This board is based on a Xilinx® Virtex-4 chip with 41,472 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 147,333 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.
`Speedgoat IO333`	The Speedgoat IO333 is a field-programmable gate array (FPGA) board based on a Xilinx Kintex® 7 chip with 325k logic cells. HDL Coder HDL Workflow Advisor supports the Speedgoat IO333-325K-06 configuration. For more information, see *Speedgoat HDL Coder Integration Package for the IO333-325K*.

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

- Install HDL Coder and Xilinx design tools. For the specific tool and version required, see the board reference topic and 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.

See Also

More About

- “Supported Third-Party Tools and Hardware” (HDL Coder)
- “Tool Setup” (HDL Coder)
- “FPGA Programming and Configuration” on page 1-5
- “Digital I/O with Speedgoat FPGA Board”
- “PLL-Based Interrupt Generation from FPGA Input”

External Websites

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

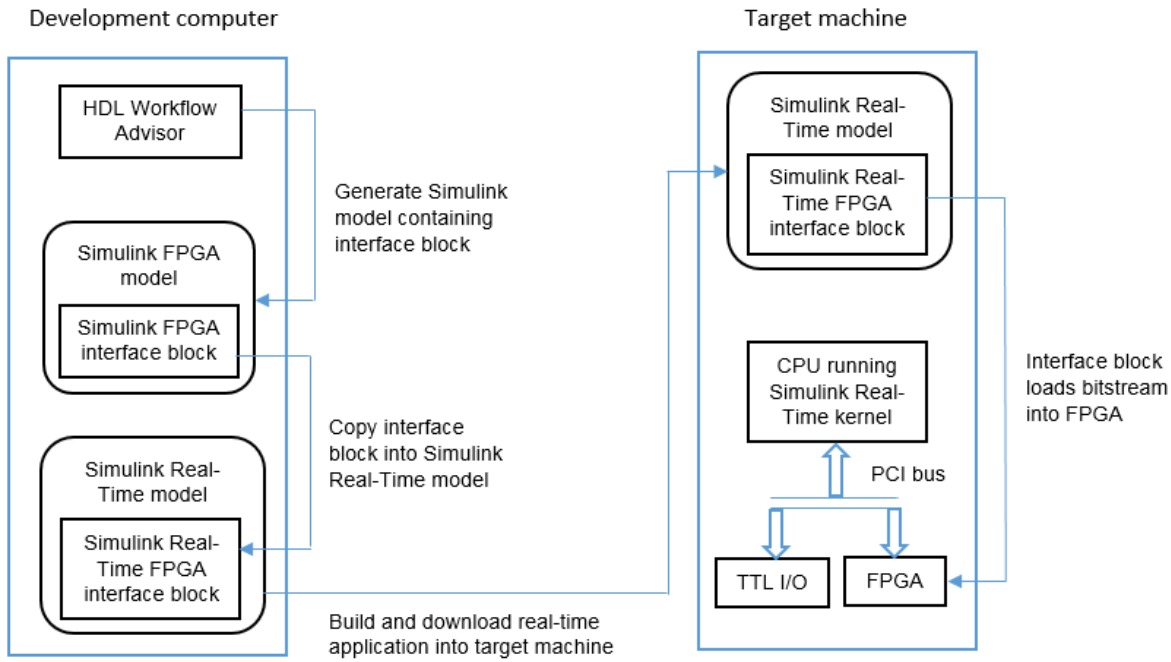
FPGA Programming and Configuration

This example shows how to implement a Simulink® algorithm on a Speedgoat FPGA I/O board by using HDL Workflow Advisor to:

1. Specify an FPGA board and its I/O interface.
2. Synthesize the Simulink algorithm for FPGA programming.
3. 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 entire workflow looks like this figure.



This example uses the Speedgoat IO331. You can use any FPGA I/O module supported by Simulink Real-Time and HDL Coder that meets the speed, size, and pinout requirements of the model.

Requirements and Preconditions

HDL Coder™

Before you start, complete an FPGA subsystem plan.

For the IO331 board, HDL Workflow Advisor requires the Xilinx® ISE toolset. To install this toolset, in the Command Window, type:

```
hdlsetuptoolpath('ToolName', 'Xilinx ISE', 'ToolPath', toolpath)
```

where *`toolpath`* is the full path to the synthesis tool executable.

For the toolset requirements of other boards, see Supported Third-Party Tools and Hardware (HDL Coder).

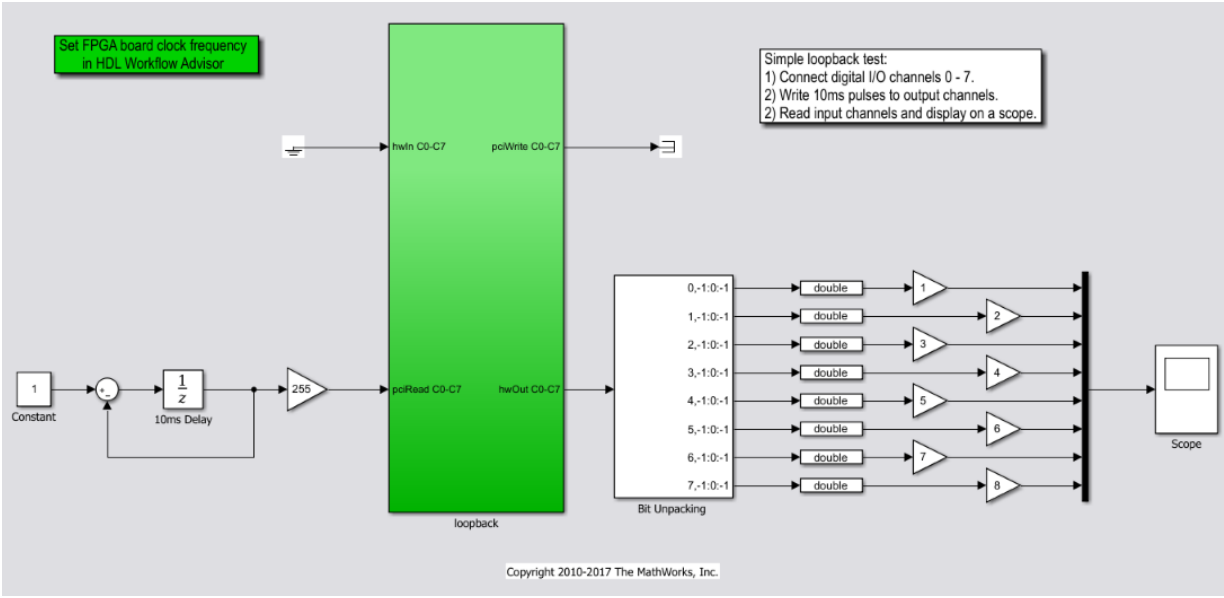
Step 1. Simulink Domain Model

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 that contains the algorithm that you want to load onto the FPGA, in this case a loopback test.
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.
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 IO331 has an onboard 125 MHz clock. One second of simulation equals 125e6 iterations of the model.

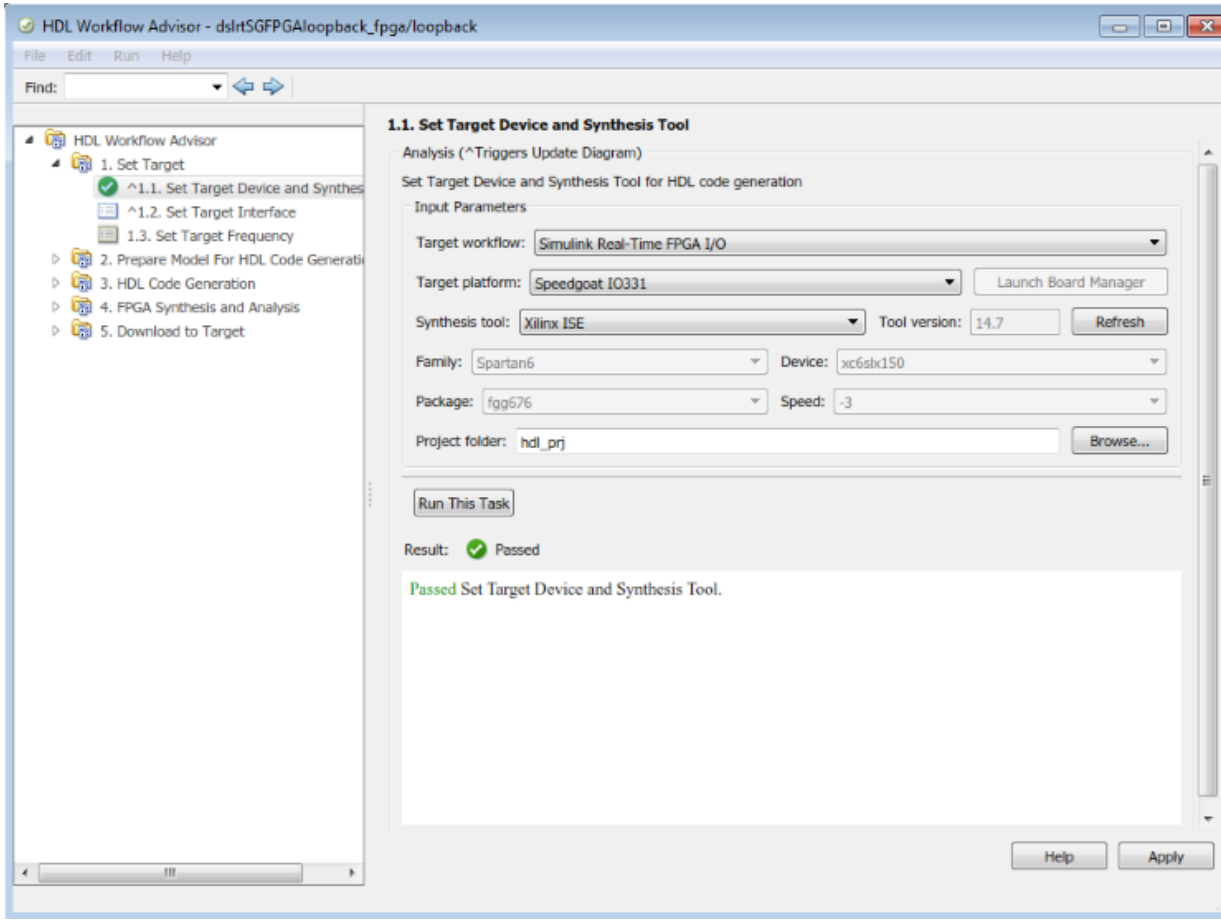
For an example of an FPGA domain model, see `dslrtSGFPGAloopback_fpga`. The `ServoSystem` subsystem contains the FPGA algorithm.



Step 2. FPGA Target Configuration

This procedure uses the `dslrtSGFPGAloopback_fpga` example. You must have already created an FPGA subsystem (algorithm) in an FPGA domain model and developed an FPGA subsystem plan.

1. Open the FPGA domain model `dslrtSGFPGAloopback_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 several tasks for the subsystem. Address only your required 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, in this case the `Speedgoat IO331`. Check that HDL Workflow advisor sets the synthesis tool to the Xilinx® ISE Design Suite.
6. Click **Run This Task**.

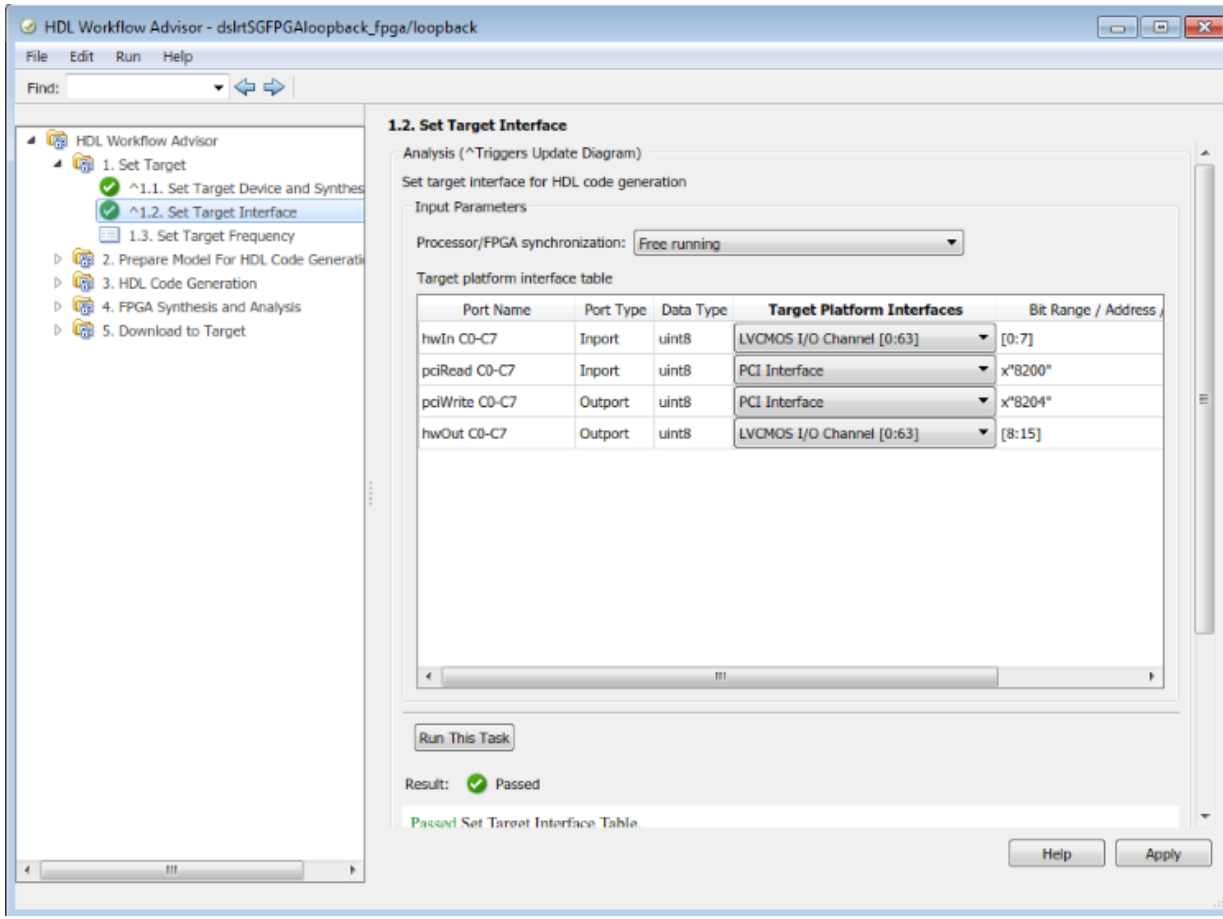


Step 3. FPGA Target Interface Configuration

You must have already configured the FPGA target.

1. In the **Set Target** folder, select task **1.2 Set Target Interface**.
2. In the **Processor/FPGA synchronization** box, select `Free running`.
3. For signals `hwIn` and `hwOut`, in the **Target Platform Interfaces** column, select `LVCMOS I/O Channel [0:63]`. In the **Bit Range/Address/FPGA Pin** column, enter the channel value for each signal, or take the defaults.

4. For signals `pciRead` and `pciWrite`, 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.
5. Click **Run This Task**.



Step 4. FPGA Target Frequency Configuration

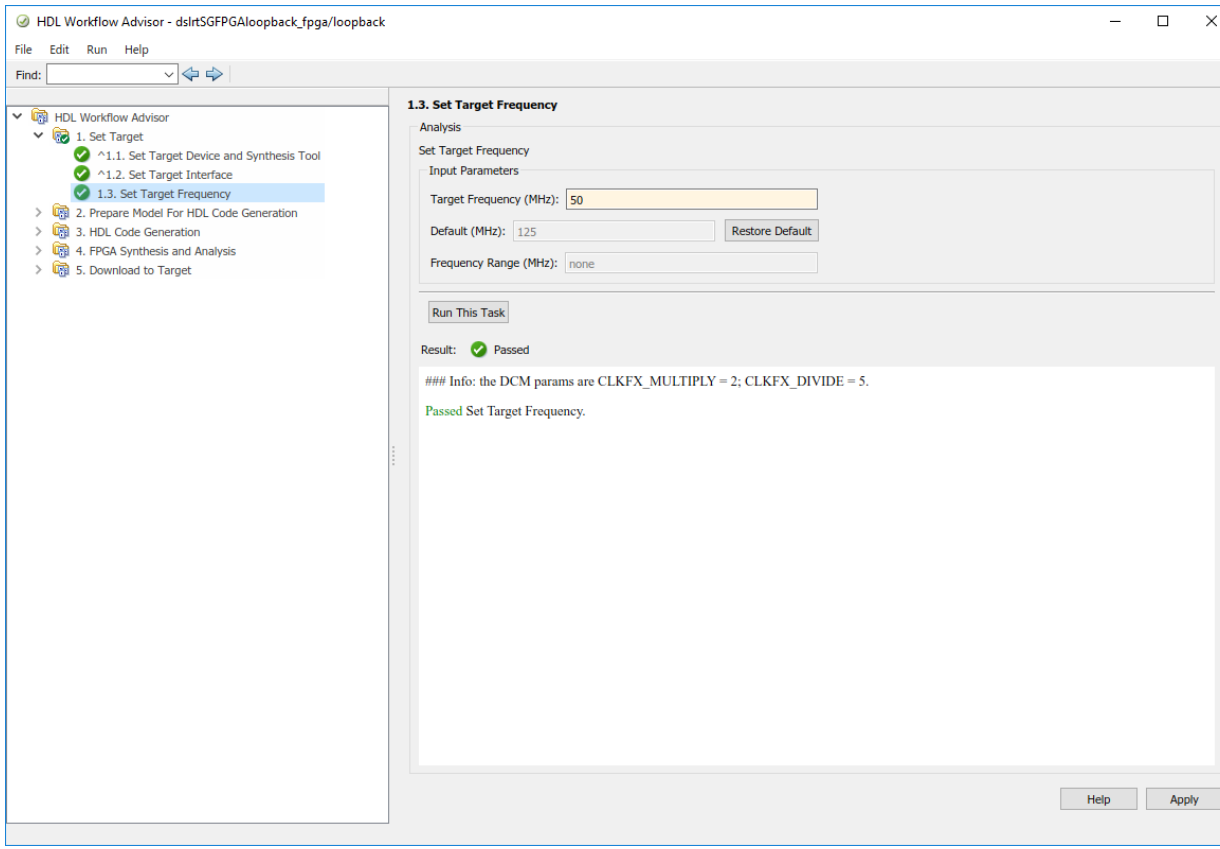
You must have already configured the FPGA target interface.

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 sometimes sets a value different from the one you specified.

3 Click **Run This Task**.



Step 5. Simulink Real-Time Interface Subsystem Generation

This procedure generates an interface subsystem file for the dxpcSGFPGAloopback_fpga example.

Assign distinct names to blocks that contain different HDL code. The name of the interface subsystem file is derived directly from the block name. If two blocks containing different HDL code have the same name, the names collide and one of the blocks gets the wrong code.

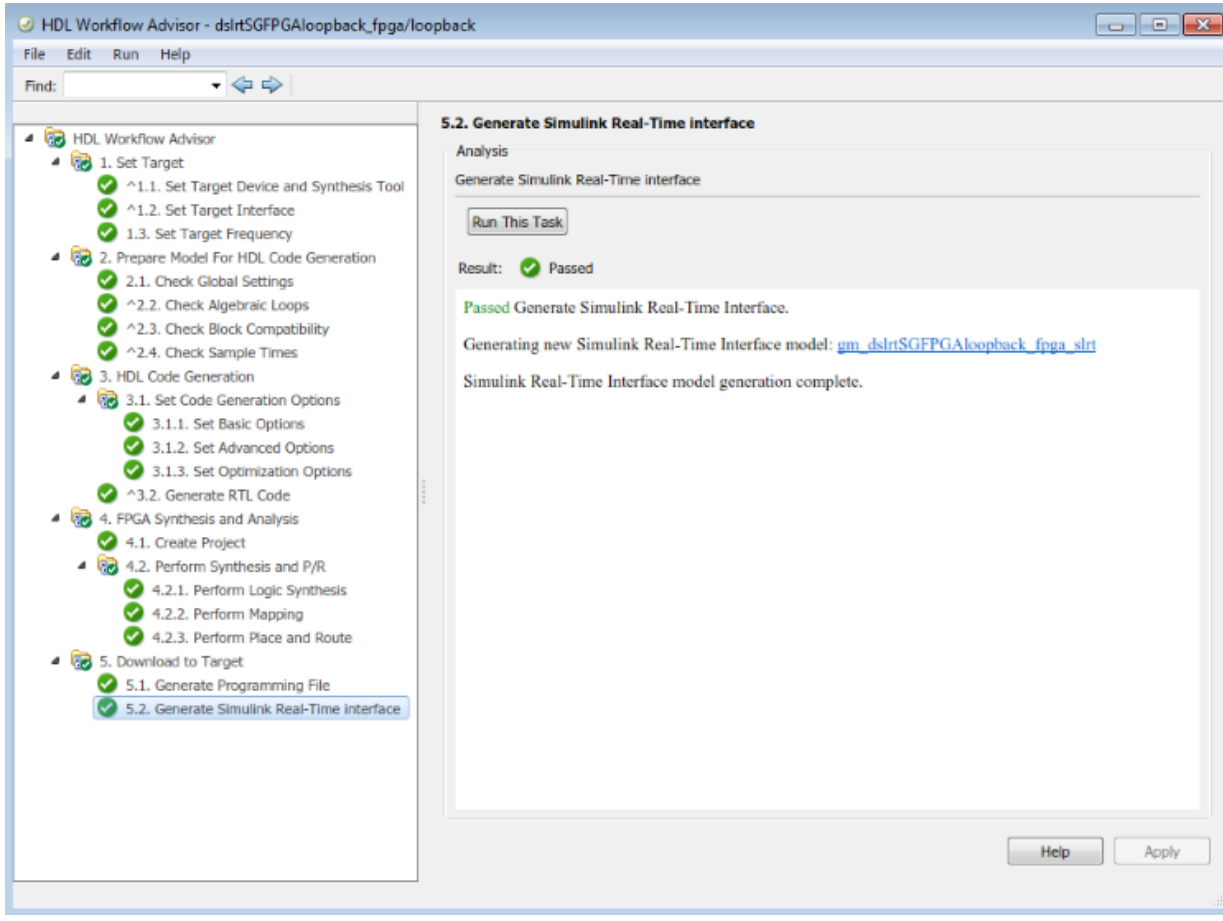
You must have already configured the FPGA target interface and the required target frequency. If you have specified vector inports or outports, you must have already selected the **Scalarize vector ports** check box. This check box is on the **Coding style** tab of node **Global Settings**, under node **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**.

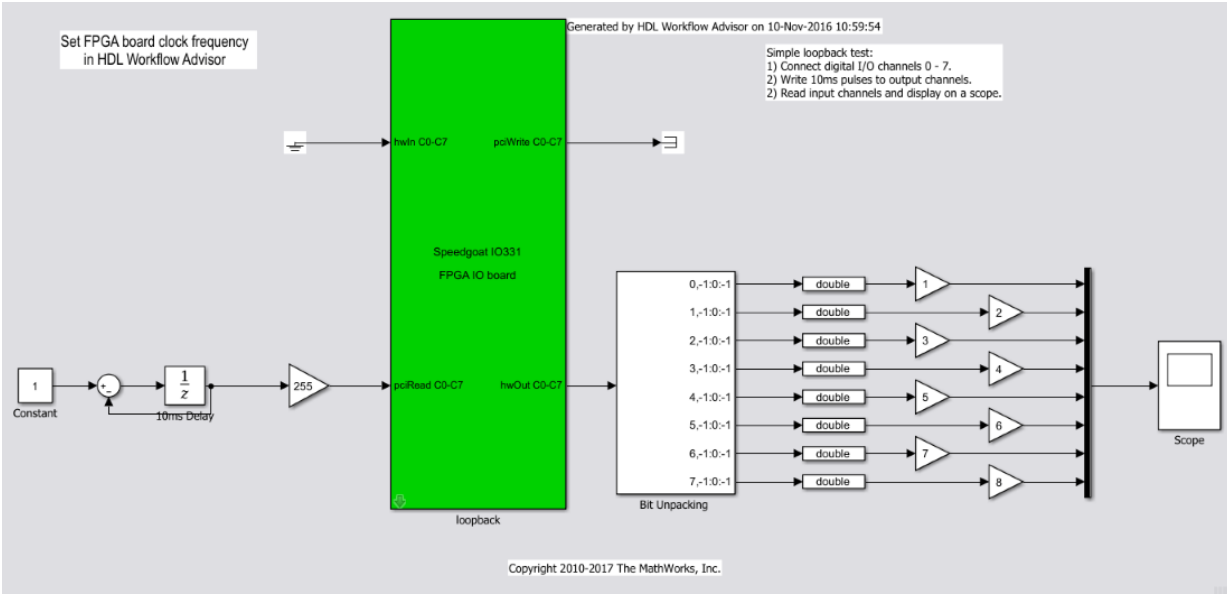
This action:

- 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_dslrtSGFPGAloopback_fpga_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.



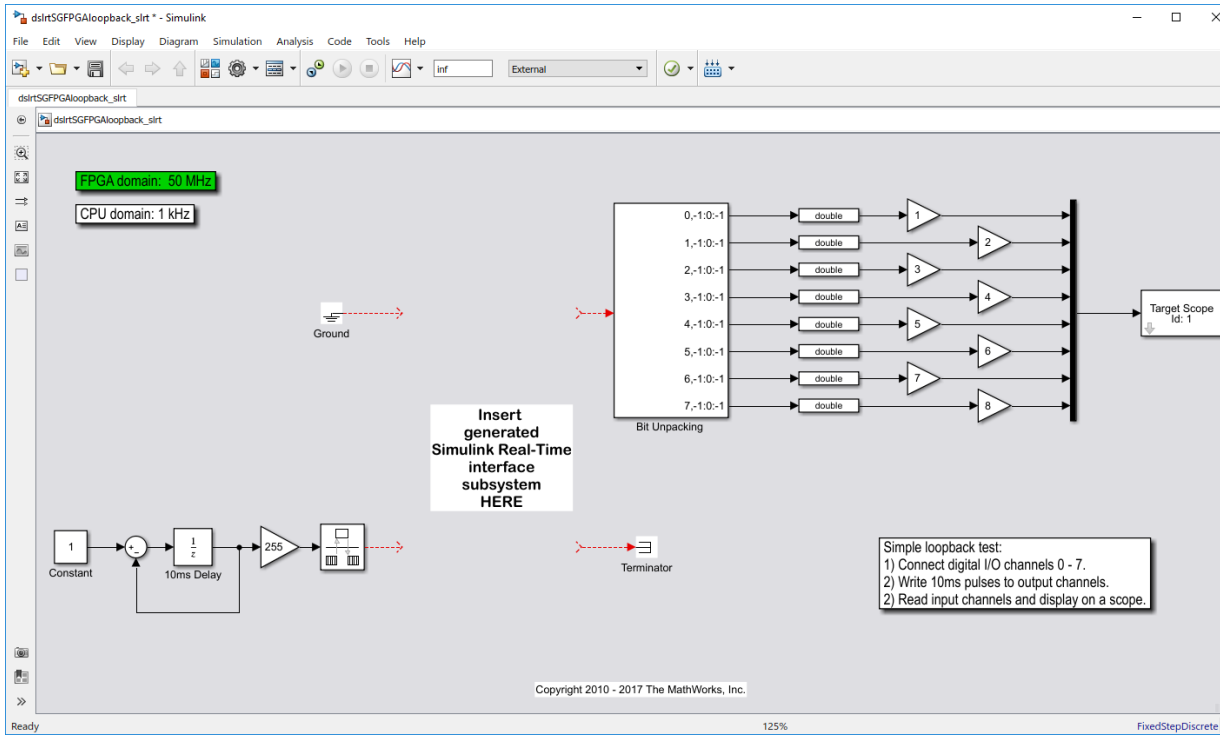
This generated model contains a masked subsystem with the same name as the subsystem in the Simulink FPGA domain model. Although the appearance is similar, this subsystem does not contain the Simulink algorithm. Instead, the algorithm is implemented in an FPGA bitstream. You reference and load this algorithm into the FPGA from this subsystem.

Step 6. Simulink Real-Time Domain Model

Using the Simulink Real-Time software, 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 interface subsystem. You can then 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 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 `dslrtSGFPGAloopback_slrt`.



Step 7. Simulink Real-Time Interface Subsystem Integration

In the Simulink Real-Time interface subsystem mask, set three parameters:

- Device index
- PCI slot
- Sample time

To integrate the interface subsystem:

1. In the Simulink editor, open `gm_dslrtSGFPGAloopback_fpga_slrt`.
2. Copy the Simulink Real-Time interface subsystem and paste it into the Simulink Real-Time domain model.
3. Save or discard `gm_dslrtSGFPGAloopback_fpga_slrt`. You can recreate it as required using the HDL Coder HDL Workflow Advisor.

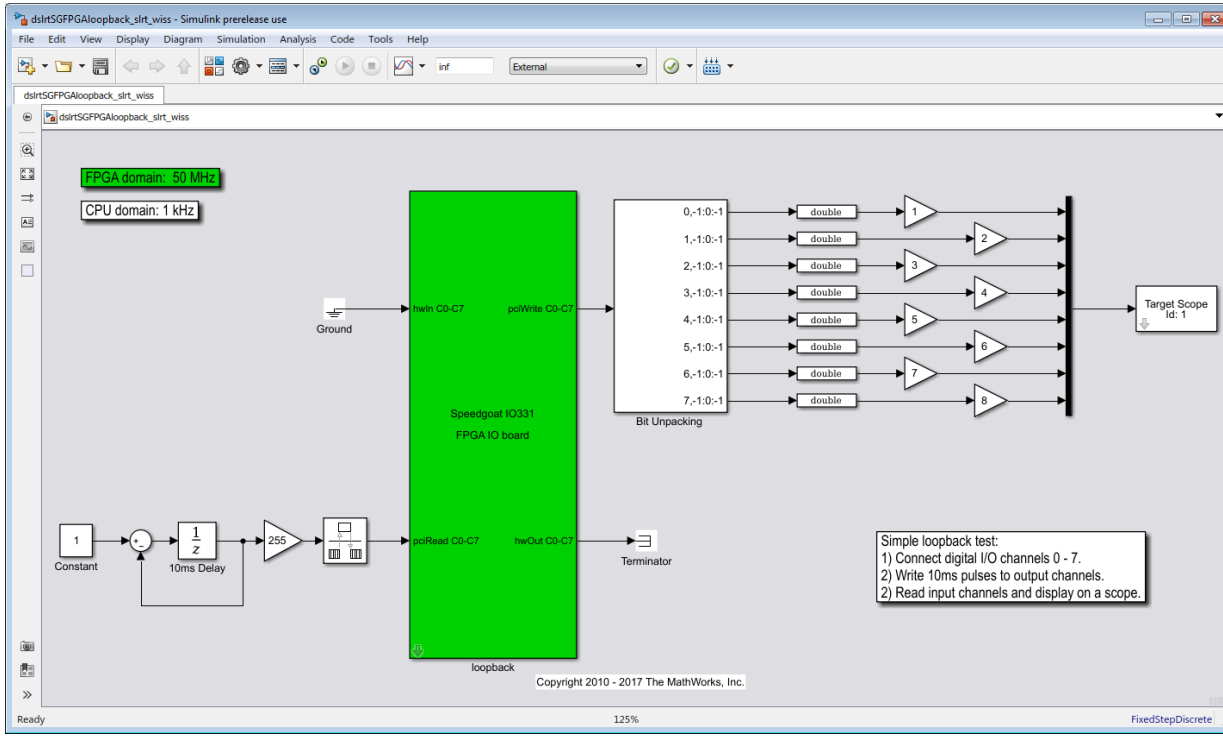
4 In the domain model, connect signals to the inports and outports of the interface subsystem.

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 IO331 boards), specify the PCI slot ([bus, slot]) for each board. Get this information with the SimulinkRealTime.target.getPCIInfo function.

6 Save the model.

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



Step 8. 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 application. The real-time application 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 the 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.

See Also

`SimulinkRealTime.target.getPCIInfo` | Subsystem

More About

- “HDL Workflow Advisor” (HDL Coder)
- “IP Core Generation Workflow for Speedgoat Boards” (HDL Coder)
- “FPGA Subsystem Plan” on page 1-20
- “FPGA Synchronization Modes” on page 1-25
- “FPGA Clock Frequency” on page 1-22
- “Digital I/O with Speedgoat FPGA Board”
- “PLL-Based Interrupt Generation from FPGA Input”

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).

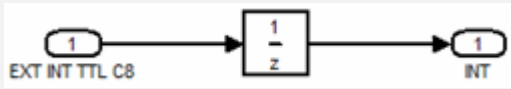
To use FPGA-based interrupts, set up and configure the FPGA domain and Simulink Real-Time domain models.

In this section...
“FPGA Domain Model” on page 1-17 “ Simulink Real-Time Domain Model” on page 1-18

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 from `1` to `0`). 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 labeled 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.

Simulink Real-Time Domain Model

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_IO331`.
6. Add the Simulink Real-Time interface subsystem to the model.
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.

See Also

More About

- “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 1-20
“FPGA Synchronization Mode” on page 1-20
“FPGA Inports and Outports” on page 1-21
“FPGA Clock Frequency” on page 1-22
“FPGA Deployment” on page 1-22

Target Device

First, to decide which FPGA to target for code generation, consult the Speedgoat data sheet for information:

- Availability and cost
- Bus compatibility
- Size
- Pinouts
- Clock speed

The example procedure uses the Simulink Real-Time FPGA workflow and the Speedgoat IO331 FPGA IO board as target platform. This choice requires that you use the Xilinx ISE synthesis tool.

For information about other target devices, see “Supported Third-Party Tools and Hardware” (HDL Coder).

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 1-25.

FPGA Inports and Outports

Inports and outports can transmit signal data between the Speedgoat target machine and the FPGA over the PCI bus. Alternatively, they can 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 and 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 can also map to a predefined specification or role (`PCI Interface`, `Interrupt from FPGA`).

  For information on using FPGA interrupts, see “Interrupt Configuration” on page 1-17.
- **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.

To achieve a simultaneous update of vector port elements, Workflow Advisor automatically inserts a strobe and specifies a strobe offset. For more information, see "IP Core User Guide" (HDL Coder).

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

The FPGA deployment procedure depends upon the FPGA model.

Deploy the IO321 and IO331 FPGAs

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 IO331. Then HDL Coder HDL Workflow Advisor generates the following files:

```
gm_fpga_model.slx
fpga_subsystem_topIO331.c
```

When you build your domain model with the integrated subsystem, the model builder:

1. Reads the C file.
2. Inserts its contents into the real-time application.
3. Packages the real-time application as an MLDATX file.

The model builder 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.

Deploy the IO333 FPGA

When HDL Coder HDL Workflow Advisor generates the programmed FPGA subsystem, it writes an SLX file (`gm_mdlname.slx`) and an MCS file (`blkorrefmdlname_timestamp.mcs`) into the model folder. The SLX file contains the FPGA subsystem. The MCS 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 IO333. Then HDL Coder HDL Workflow Advisor generates the following files:

```
gm_fpga_model.slx
fpga_subsystem_201703301740.mcs
```

When you build your domain model with the integrated subsystem, the model builder:

1. Generates the real-time application.
2. Packages the real-time application and the MCS file as an MLDATX file.

The model builder searches for the MCS file on the MATLAB® path. If you deploy the model to another location on the disk, add the new location to the path.

See Also

More About

- “Supported Third-Party Tools and Hardware” (HDL Coder)

- “IP Core User Guide” (HDL Coder)
- “FPGA Synchronization Modes” on page 1-25
- “Interrupt Configuration” on page 1-17

FPGA Synchronization Modes

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

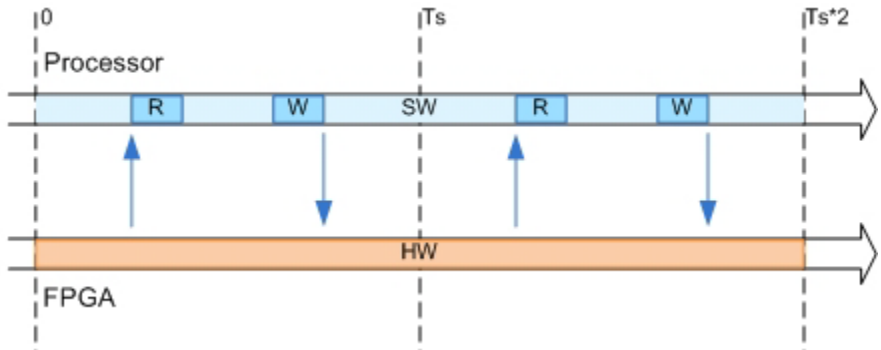
- `Free running`
- `Coprocessing — blocking`
- `Coprocessing — nonblocking with delay`

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

  The CPU:

  1. Strobes data out of the FPGA.
  2. Reads results from the FPGA outputs.
  3. Writes data to the FPGA inputs.
  4. Strobes the data into the FPGA.

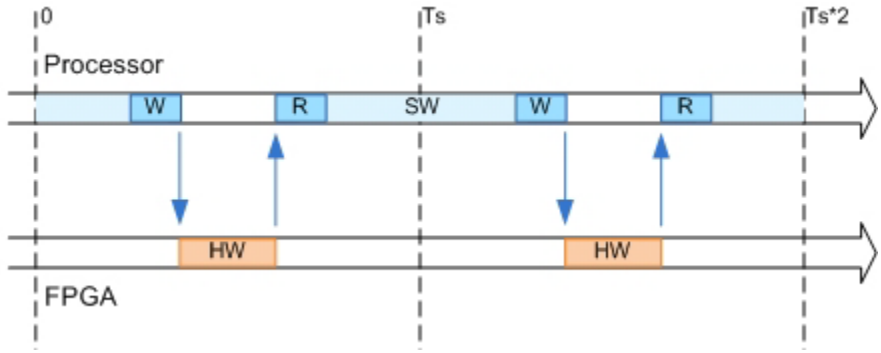
  The shaded areas indicate that the processor and FPGA are running continuously.



- `Coprocessing — blocking` — The CPU of the Speedgoat target machine and the FPGA run synchronized and in tandem. Select this mode when the FPGA execution time is short compared to the target computer sample time. For example, select this mode when the model requires the FPGA results to continue processing.

  The CPU:

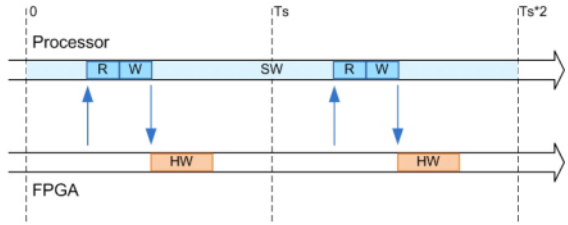
1. Writes data to the FPGA inputs.
2. Strobes the data into the FPGA.
3. Waits for the FPGA to finish executing.
4. Reads results from the FPGA outputs.



- `Coprocessing — nonblocking with delay` — The CPU of the Speedgoat target machine and the FPGA run synchronized and in tandem. Select this mode when the FPGA execution time is long compared to the Speedgoat target machine sample time. For example, select this mode to manage multiple FPGAs effectively in parallel.

  The CPU:

  1. Waits for the FPGA to finish executing.
  2. Reads the data from the previous time step.
  3. Writes new data to the FPGA inputs.
  4. Strobes the data into the FPGA.



2

Functional Mockup Units and Simulink Real-Time

Apply Functional Mockup Units with Simulink Real-Time

After you create a model that contains an FMU block, you can build and download the model to a target computer by using Simulink Real-Time. The following limitations apply:

- Simulink Real-Time supports FMU blocks for Co-Simulation mode. Simulink Real-Time does not support FMU blocks for Model Exchange mode.
- Simulink Real-Time does not support FMU blocks within a referenced model. FMU blocks must be at the top level of the model.
- Simulink Real-Time generates a mask dialog box that contains both numerical-valued and string-valued parameters. However, Simulink Real-Time generates code for only numerical- valued parameters.

To convert a Simulink model that contains FMU blocks to a Simulink Real-Time model, set the model Configuration Parameters to values compatible with real-time execution:

- In the **Code Generation** pane, set **System target file** to `slrt.tlc`.
- In the **Solver** pane:
  - Set **Type** to `Fixed-step`.
  - Set **Fixed-step size** to a step size compatible with the real-time requirements of your model.

You can then build and download the model to a target computer and run the real-time application. The build and download process downloads the required FMU binary files.

To open an example model that contains FMU blocks running in Simulink Real-Time, type `dslrt_bouncing_cs` in the MATLAB Command Window.

See Also

FMU

More About

- “Import FMUs” (Simulink)

External Websites

- http://fmi-standard.org/

Third-Party Calibration Support

- “Calibrate Real-Time Application” on page 3-2
- “Prepare ASAP2 Data Description File” on page 3-4
- “Calibrate Parameters with Vector CANape” on page 3-10
- “Vector CANape Limitations” on page 3-13
- “Troubleshoot Vector CANape Operation” on page 3-14
- “Calibrate Parameters with ETAS Inca” on page 3-16
- “ETAS Inca Limitations” on page 3-19
- “Troubleshoot ETAS Inca Operation” on page 3-20

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.
- 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 a real-time application with signal and parameter attributes that are consistent with A2L (ASAP2) file generation. See “Export ASAP2 File for Data Measurement and Calibration” (Simulink Coder).
- Use the build process to generate *`model`*`_slrt.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 and XCP-related sections and memory addresses.

  If your model includes referenced models, the build creates a *`model`*`_slrt.a2l` file for the real-time application and separate *`refmodel`*`_slrt.a2l` files for each referenced model.

Note You cannot configure third-party software for calibration with only the A2L files that Simulink Coder™ generates. These files do not contain XCP-related sections and memory addresses. Simulink Real-Time adds this information during the build process.

See Also

More About

- “Export ASAP2 File for Data Measurement and Calibration” (Simulink Coder)
- “Prepare ASAP2 Data Description File” on page 3-4
- “Calibrate Parameters with Vector CANape” on page 3-10
- “Calibrate Parameters with ETAS Inca” on page 3-16
- “XCP Master Mode”

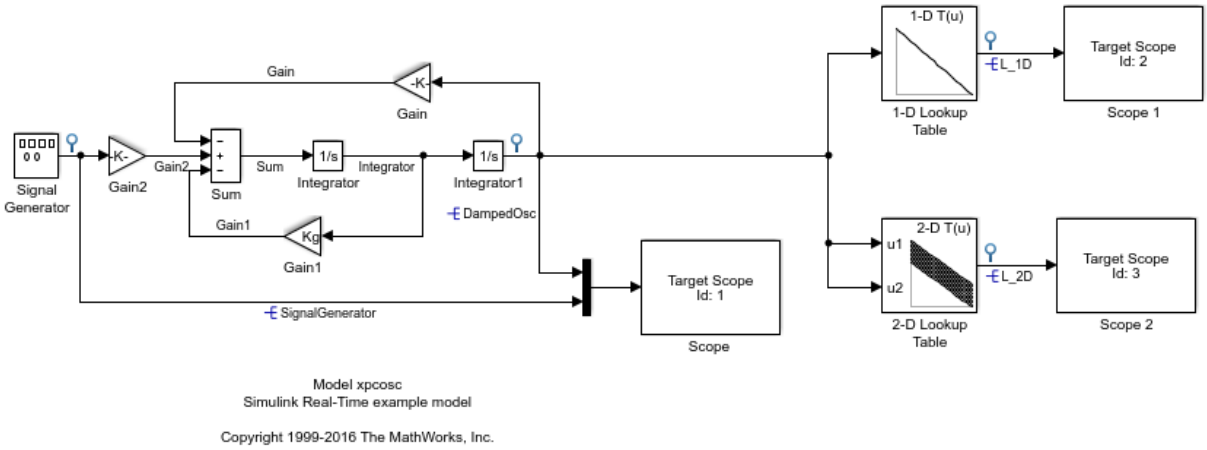
External Websites

- 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.

This example uses `ex_slrt_cal_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_cal_osc')))`), which requires `ex_slrt_cal_osc_data.mat` (`load(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_cal_osc_data.mat')))`).



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

In this section...
“Initial Setup” on page 3-6 “Set Up Parameters” on page 3-6 “Set Up Signals” on page 3-7

In this section...
“Set Up Lookup Tables” on page 3-8 “Generate Data Description File” on page 3-8

Initial Setup

For best results, load the MATLAB workspace variables before you load the model that uses them.

1. Load workspace variables for the example model from `ex_slrt_cal_osc_data.mat` (`load(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', '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. Open `ex_slrt_cal_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_cal_osc')))`).

Set Up Parameters

Set up global 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. Check 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 Check 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 Check that the `DampedOsc` signal object exists and has these properties:

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

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

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

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

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

6 Check 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 Check 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, check 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, check 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 (disables Simulation Data Inspector and Dashboard blocks)** check box.

This option enables real-time applications to generate an 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 (disables Simulation Data Inspector and Dashboard blocks)" | n-D Lookup Table

More About

- "Calibrate Parameters with Vector CANape" on page 3-10
- "Calibrate Parameters with ETAS Inca" on page 3-16

External Websites

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

Calibrate Parameters with 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 3-4.

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 3-10
“Prepare Device” on page 3-10
“Configure Signals and Parameters” on page 3-11
“Perform Signal Measurement and Parameter Calibration” on page 3-11

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 computer 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` in 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 — `0` s
   - Max home time-axis value — `0.1` s
   - Time duration — `0.1` s
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.

See Also

More About

- “Prepare ASAP2 Data Description File” on page 3-4
- “Vector CANape Limitations” on page 3-13
- “Troubleshoot Vector CANape Operation” on page 3-14

External Websites

- www.vector.com

Vector CANape 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 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.

Troubleshoot Vector CANape Operation

My third-party calibration tool (Vector CANape) is not working with the real-time application.

What This Means

You can use the Vector CANape tool to view signals and tune parameters in the real-time application. For more information, see the steps in “Prepare ASAP2 Data Description File” on page 3-4. In addition to the limitations listed in “Vector CANape Limitations” on page 3-13, there are various issues that can prevent operation of this tool.

Try This

Try these workarounds for Vector CANape tool issues.

Simulation Data Inspector in Use

Simulation Data Inspector (SDI) and the third-party calibration tools (Vector CANape and ETAS Inca) are mutually exclusive. If you use SDI to view signal data, you cannot use the calibration tools. If you use the calibration tools, you cannot use SDI to view signal data.

Master Cannot Connect

Check the IP address of the target computer associated with the model and compare it with the address stored in the ASAP2 file.

ASAP2 File Out of Date

When you rebuild a Simulink Real-Time application, update the ASAP2 file loaded in the calibration tool with the new version of the file. The ASAP2 file is valid only until the next time you build the application.

See Also

More About

- “Prepare ASAP2 Data Description File” on page 3-4

- “Vector CANape Limitations” on page 3-13
- “Troubleshoot Vector CANape Operation” on page 3-14

External Websites

- www.vector.com

Calibrate Parameters with 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 3-4.

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 3-16
“Prepare Project” on page 3-16
“Prepare Workspace” on page 3-17
“Prepare Experiment” on page 3-17
“Configure Signals and Parameters” on page 3-17
“Perform Signal Measurement and Parameter Calibration” on page 3-17

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.

See Also

More About

- “Prepare ASAP2 Data Description File” on page 3-4
- “ETAS Inca Limitations” on page 3-19
- “Troubleshoot ETAS Inca Operation” on page 3-20

External Websites

- www.etas.com

ETAS Inca 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 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.

Troubleshoot ETAS Inca Operation

Investigate issues that can occur when ETAS Inca controls a real-time application.

My third-party calibration tool (ETAS Inca) is not working with the real-time application.

What This Means

You can use the ETAS Inca tool to view signals and tune parameters in the real-time application. For more information, see the steps in “Prepare ASAP2 Data Description File” on page 3-4. In addition to the limitations listed in “ETAS Inca Limitations” on page 3-19, there are various issues that can prevent operation of this tool.

Try This

Try these workarounds for ETAS Inca tool issues.

Simulation Data Inspector in Use

Simulation Data Inspector (SDI) and the third-party calibration tools (Vector CANape and ETAS Inca) are mutually exclusive. If you use SDI to view signal data, you cannot use the calibration tools. If you use the calibration tools, you cannot use SDI to view signal data.

Master Cannot Connect

Check the IP address of the target computer associated with the model and compare it with the address stored in the ASAP2 file.

ASAP2 File Out of Date

When you rebuild a Simulink Real-Time application, update the ASAP2 file loaded in the calibration tool with the new version of the file. The ASAP2 file is valid only until the next time you build the application.

Cannot Disable Freeze Mode

Remove the dataset file from the target file system and reset the parameters to the original values specified in your model. The dataset file is named `flashdata_model_name.dat`.

See Also

More About

- “Prepare ASAP2 Data Description File” on page 3-4
- “ETAS Inca Limitations” on page 3-19
- “Troubleshoot ETAS Inca Operation” on page 3-20

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” (Simulink) and “Port Legacy Code” (Simulink).

In this section...
“Prerequisites” on page 4-2
“Simulink S-Function Example” on page 4-2
“Steps to Incorporate Fortran” on page 4-2

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 compilers listed here:

`www.mathworks.com/support/compilers/current_release`

Simulink S-Function Example

The Simulink examples 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. Open “Custom Code and Hand Coded Blocks using the S-function API” (Simulink)
2. Open the associated model.
3. Open the Fortran S-functions example model. 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 subroutines (`*.f`). 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. To validate the compiled Fortran code and wrapper S-function, run a simulation C-MEX file with the Simulink software.
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. 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.

See Also

More About

- “Create Level-2 Fortran S-Functions” (Simulink)
- “Port Legacy Code” (Simulink)
- “Fortran S-Function Examples” (Simulink)

Real-Time Application Setup

Real-Time Application Environment

- “Default Target Computers” on page 5-2
- “Command-Line C Compiler Configuration” on page 5-3
- “Command-Line Setup” on page 5-5
- “Command-Line PCI Bus Ethernet Setup” on page 5-6
- “Command-Line USB-to-Ethernet Setup” on page 5-10
- “Ethernet Card Selection by Index” on page 5-15
- “Command-Line Ethernet Card Selection by Index” on page 5-17
- “Command-Line Target Computer Settings” on page 5-20
- “Command-Line Target Computer Boot Methods” on page 5-22
- “Command-Line Network Boot Method” on page 5-23
- “Command-Line CD/DVD Boot Method” on page 5-25
- “Command-Line DOS Loader Boot Method” on page 5-26
- “Command-Line Removable Disk Boot Method” on page 5-28
- “Command-Line Standalone Boot Method” on page 5-30

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 to 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 5-5.

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

When you call `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 require different compilers for MEX and Simulink Real-Time.

Note By default, the Microsoft Visual Studio® 2015 installer does not install the C++ compiler that Simulink Real-Time requires. To install the C++ compiler, perform a custom install and select the C++ compiler. If you already installed Microsoft Visual Studio with the default configuration, rerun the installer and select the modify option.

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**.

Command-Line Setup

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

You must have installed and configured a C compiler. For more information, see “Command-Line C Compiler Configuration” on page 5-3.

1. “Command-Line PCI Bus Ethernet Setup” on page 5-6 or “Command-Line USB-to-Ethernet Setup” on page 5-10
2. “Command-Line Target Computer Settings” on page 5-20
3. “Command-Line Target Computer Boot Methods” on page 5-22

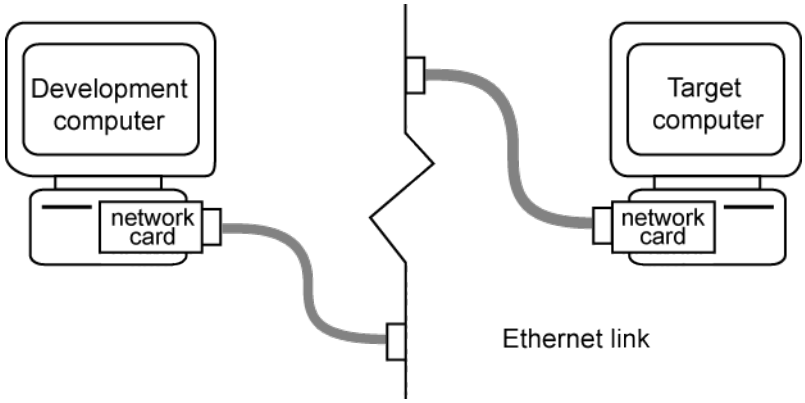
The next task is “Run Confidence Test on Configuration”.

Command-Line PCI Bus Ethernet Setup

If your Speedgoat 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 5-6
“Command-Line PCI Bus Ethernet Settings” on page 5-7

PCI Bus Ethernet Protocol Hardware



To install PCI bus Ethernet protocol interface hardware in your Speedgoat target computer, contact Speedgoat support.

1. If the target computer already contains one or more Ethernet cards, to get a list of these Ethernet cards, in the Command Window, type:

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

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

   Unlike the target computer, the development computer network adapter card can have a dynamic host configuration protocol (DHCP) address and can be accessed from the network. Configure the DHCP server to reserve static IP addresses to prevent these addresses from being assigned to other systems.

3 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

With the installed PCI bus Ethernet card, to build and download a real-time application, first 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);
```

   You make the other settings 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, do not change the default setting. If you communicate from a development computer within a LAN different from your target computer, define a gateway and enter its IP address here. In particular, create a gateway if you access the target computer via the Internet.

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

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

7. Set the target driver to one of `'I210'`, `'I217'`, `'I8254x'`, `'I82559'`, `'X540'`, `'R8139'`, `'R8168'`, 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 a supported Ethernet card is not 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 5-17.

Repeat this procedure as required for each target computer.

The next task is “Command-Line Target Computer Settings” on page 5-20.

See Also

`SimulinkRealTime.target.getPCIInfo`

More About

- “Ethernet Card Selection by Index” on page 5-15

- “Command-Line Ethernet Card Selection by Index” on page 5-17

Command-Line USB-to-Ethernet Setup

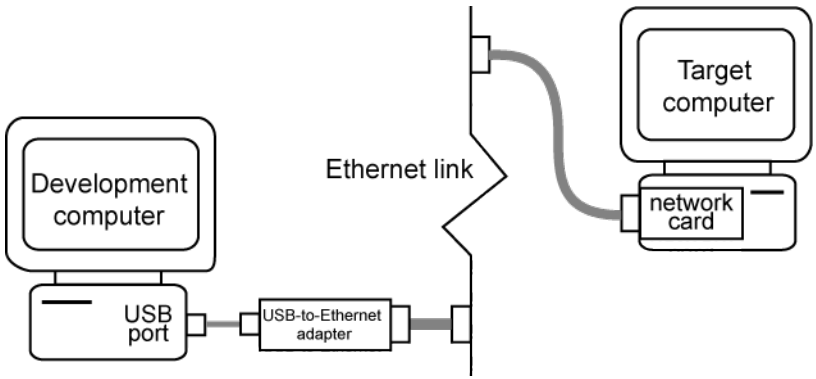
If the target computer has a USB 2.0 port but does not have a supported PCI Ethernet card, use a USB-to-Ethernet adapter.

In this section...
“USB-to-Ethernet Protocol Hardware” on page 5-10
“Command-Line USB-to-Ethernet Settings” on page 5-12

USB-to-Ethernet Protocol Hardware

You can plug the USB-to-Ethernet adapter into the development computer or the target computer. The setup is slightly different for each location.

Development Computer USB Port



To connect the USB-to-Ethernet protocol interface hardware to the USB port on the development computer:

1. Get a list of the Ethernet cards in the target computer. In the Command Window, type:

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

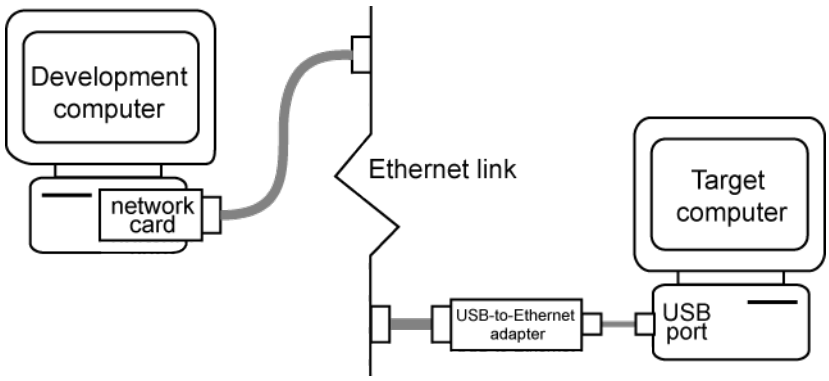
2. Acquire a supported USB-to-Ethernet adapter.
3. Plug the USB-to-Ethernet adapter into the USB port in the development computer.

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

4. Connect the USB-to-Ethernet adapter from your development computer to your LAN using an unshielded twisted-pair (UTP) cable.
5. Connect the target computer Ethernet card to your LAN using another 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.

Target Computer USB Port



To connect the USB-to-Ethernet protocol interface hardware to the USB port on the target computer:

1. Acquire a supported USB-to-Ethernet adapter.
2. Turn off your target computer.
3. Prepare a boot drive according to the instructions in “Target Computer Boot Methods”.

   You cannot use the network boot method with this hardware configuration.
4. Plug the USB-to-Ethernet adapter into the USB port in the target.

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

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

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

Unlike the target computer, the development computer network adapter card can have a dynamic host configuration protocol (DHCP) address and can be accessed from the network. Configure the DHCP server to reserve static IP addresses to prevent these addresses from being assigned to other systems.

7 Connect your development 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, that adapter can have a DHCP address.

Command-Line USB-to-Ethernet Settings

With the installed USB-to-Ethernet adapter, to build and download a real-time application, first 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);
```

You make the other settings 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, do not change the default setting. If you communicate from a development computer within a LAN different from your target computer, define a gateway and enter its IP address here. In particular, create a gateway if you access the target computer via the internet.

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 5-20.

See Also

`SimulinkRealTime.target.getPCIInfo`

More About

- “Ethernet Card Selection by Index” on page 5-15
- “Command-Line Ethernet Card Selection by Index” on page 5-17

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);
   ```

   You make the other settings to this object.

2. At the MATLAB prompt, type:

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

   With `ShowHardware` set, after the kernel starts, the development computer cannot communicate with the target computer. When you have gathered your information, to resume normal functionality, set this property to `'off'`, recreate the boot image, and restart the target computer.

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. To create a boot disk, click **Create boot disk** and follow the prompts.

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

    Check that the boot order allows you to start the target computer from your disk. For more information, see your Speedgoat target computer documentation. 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);
   ```

   You make the other settings to this object.

2. At the MATLAB prompt, type:

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

   With `ShowHardware` set, after the kernel starts, the development computer cannot communicate with the target computer. When you have gathered your information, to resume normal functionality, set this property to `'off'`, recreate the boot image, and restart the target computer.

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

Check that the boot order allows you to start the target computer from your disk. For more information, see your Speedgoat target computer documentation. 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.

Note

- The `NonPentiumSupport` property has ceased to function. Use a target computer with an Intel® Pentium or AMD® K5/K6/Athlon processor.
- In a future release, the `SecondaryIDE` target setting will be read-only and set to `'off'`.
- The `MulticoreSupport` target setting is read-only and set to `'on'`. This property is obsolete. Attempting to set property to `'off'` generates and error. Single-core target computers are not supported.
- The property `MaxModelSize` has no function.
- The **RAM size** check box has been removed from Simulink Real-Time Explorer. The property value `TargetRAMSizeMB` continues to function.

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);
   ```

   You make the other settings to this object.
2. Assign the following target computer settings as required:

   - **Target scope display**

     - `env.TargetScope='Enabled'` (the default) — Use to display information, such as a target scope, in graphic format.
     - `env.TargetScope='Disabled'` — Use 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 to enable USB ports on the target computer; for example, to connect a USB keyboard.
  - `env.USBSupport='off'` — Otherwise.

- **Target RAM size**

  `env.TargetRAMSizeMB='Auto'` (the default) — For more information, see your Speedgoat target computer documentation to determine the amount of memory installed in the target computer.

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

  Target computer memory for the real-time application executable, the kernel, and other uses is limited to a maximum of 4 GB.

Repeat this procedure as required for each target computer.

The next task is “Command-Line Target Computer Boot Methods” on page 5-22.

Command-Line Target Computer 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 Target Computers and Support” for further information.

1. Select one of the following methods:

   - “Command-Line Network Boot Method” on page 5-23
   - “Command-Line CD/DVD Boot Method” on page 5-25
   - “Command-Line DOS Loader Boot Method” on page 5-26
   - “Command-Line Removable Disk Boot Method” on page 5-28
   - “Command-Line Standalone Boot Method” on page 5-30

2. 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 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 can 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 5-6 or “Command-Line USB-to-Ethernet Setup” on page 5-10.

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 can already have the required values.

2. Set network boot method:

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

3. Set a TCP/IP address. Check 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);
   ```

   You make the other settings 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

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

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);
   ```

   You make other settings 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. Check that `autoexec.bat` file is on the DOS boot path (typically the root folder).
8. Check the Speedgoat target computer documentation for how to select the required boot device.
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, use a 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);
   ```

   You make the other settings to this object.
2. In the output of the `setAsDefaultTarget` command, check that property `TargetBoot` is `BootFloppy`.

   You can update property `TargetBoot` 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. 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. 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...
“DOS Environment Restrictions” on page 5-30
“Command-Line Standalone Settings” on page 5-31
“Real-Time Application Build” on page 5-31
“Real-Time Application Transfer and Boot Configuration” on page 5-32

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.

If you want to use real-time Scope blocks to display or record output, there are additional restrictions:

- Use only `Target` or `File` type blocks.
- 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);
   ```

   You make other settings 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 standalone kernel and real-time application with utility files.

1. In the Command Window, open your Simulink model, for example, `ex_slrt_rt_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', '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.
- `slrtkrnl.rtb` — Contains the Simulink Real-Time standalone kernel. This image also contains applicable options, such as the IP address of the target computer.
- `ex_slrt_rt_osc.mldatx` — Contains the real-time application and application-specific data.

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

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

Real-Time Application Transfer and Boot Configuration

After building the kernel and real-time application on a development computer, transfer the files to a target computer by using the `SimulinkRealTime.fileSystem` functions. Configure the target computer to run the real-time application upon startup.

For this procedure, your target computer must support network boot mode. If it does not support network boot mode, see "Application Transfer and Boot Configuration with Flash Drive".

1. Restart the target computer in DOS mode and open the DOS prompt.

   If the target computer was previously started from the network boot image, to disable the network boot capability, kill the boot server from Windows® Task Manager.
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. Start the target computer by using network boot mode.
4. In the Command Window, change to the folder that contains the kernel and real-time application files.
5. Copy these files to the root folder of the target computer `C:\` drive:

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

6. Restart the target computer.

   To boot the kernel and start the real-time application, the target computer executes the following sequence of calls:

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

Repeat this procedure for each target computer that you start in `Stand Alone` mode.

Continue by testing a real-time application in `Stand Alone` mode.

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” on page 6-4
- “Monitor Signals with Simulink Real-Time Explorer” on page 6-6
- “Monitor Signals with MATLAB Language” on page 6-9
- “Instrument a Stateflow Subsystem” on page 6-11
- “Signal Group Monitoring Formats” on page 6-16
- “Monitor Stateflow States with MATLAB Language” on page 6-17
- “Animate Stateflow Charts with Simulink External Mode” on page 6-18
- “Signal Tracing Basics” on page 6-20
- “ Simulink Real-Time Scope Usage” on page 6-21
- “Target Scope Usage” on page 6-23
- “Configure Real-Time Target Scope Blocks” on page 6-25
- “Create Target Scopes with Simulink Real-Time Explorer” on page 6-31
- “Configure Scope Sampling with Simulink Real-Time Explorer” on page 6-36
- “Trigger Scopes with Simulink Real-Time Explorer” on page 6-40
- “Configure Target Scopes with Simulink Real-Time Explorer” on page 6-51
- “Configure Target Scopes with MATLAB Language” on page 6-55
- “Create Signal Groups with Simulink Real-Time Explorer” on page 6-58
- “Host Scope Usage” on page 6-61
- “Configure Real-Time Host Scope Blocks” on page 6-62
- “Create Host Scopes with Simulink Real-Time Explorer” on page 6-66
- “Configure the Host Scope Viewer” on page 6-71
- “Trace Signals with Simulink External Mode” on page 6-73

- “Inspect Simulink® Real-Time™ Data with Simulation Data Inspector” on page 6-76
- “Trace or Log Data with Simulation Data Inspector” on page 6-82
- “External Mode Usage” on page 6-87
- “Signal Logging Basics” on page 6-88
- “File Scope Usage” on page 6-90
- “Configure Real-Time File Scope Blocks” on page 6-93
- “Create File Scopes with Simulink Real-Time Explorer” on page 6-98
- “Configure File Scopes with Simulink Real-Time Explorer” on page 6-102
- “Log Signal Data into Multiple Files” on page 6-106
- “Log Signal Data with Outport Blocks and Simulink Real-Time Explorer” on page 6-110
- “Log Signal Data with Outport Block and MATLAB Language” on page 6-117
- “Signal Logging Buffer Size” on page 6-124
- “Configure File Scopes with MATLAB Language” on page 6-125
- “Tune Parameters with Simulink Real-Time Explorer” on page 6-129
- “Create Parameter Groups with Simulink Real-Time Explorer” on page 6-134
- “Tune Parameters with MATLAB Language” on page 6-137
- “Tune Parameters with Simulink External Mode” on page 6-140
- “Save and Reload Parameters with MATLAB Language” on page 6-142
- “Tunable Block Parameters and Tunable Global Parameters” on page 6-145
- “Tune Inlined Parameters with Simulink Real-Time Explorer” on page 6-148
- “Tune Inlined Parameters with MATLAB Language” on page 6-155
- “Tune Parameter Structures with Simulink Real-Time Explorer” on page 6-157
- “Tune Parameter Structures with MATLAB Language” on page 6-163
- “Define and Update Inport Data” on page 6-167
- “Define and Update Inport Data with MATLAB Language” on page 6-173
- “Inport Data Mapping Limitations” on page 6-178
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179
- “Troubleshoot Signals Not Accessible by Name” on page 6-184
- “Troubleshoot Parameters Not Accessible by Name” on page 6-186
- “Troubleshoot Internationalization Issues” on page 6-187

• “Internationalization Issues” on page 6-188

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.

See Also

More About

- “Troubleshoot Signals Not Accessible by Name” on page 6-184
- “Simulink Real-Time Scope Usage” on page 6-21
- “Target Scope Usage” on page 6-23
- “Host Scope Usage” on page 6-61
- “File Scope Usage” on page 6-90

- “Display and Filter Hierarchical Signals and Parameters” on page 6-179

Monitor Signals with Simulink Real-Time Explorer

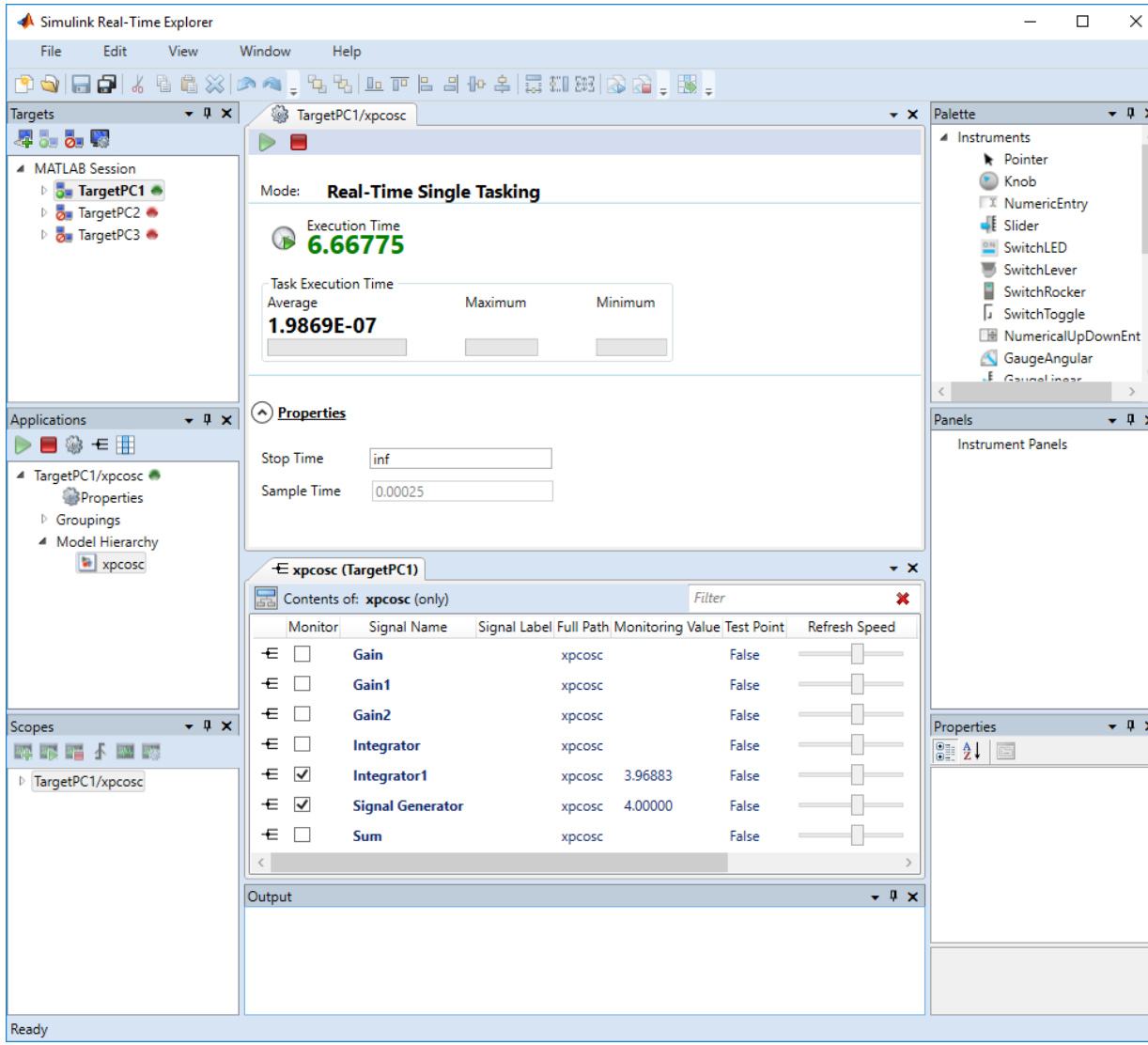
This procedure uses the model `xpcosc`. You must have already completed the following setup:

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 Properties 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 the workspace 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

More About

- “Create Signal Groups with Simulink Real-Time Explorer” on page 6-58
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179
- “Signal Group Monitoring Formats” on page 6-16
- “Troubleshoot Signals Not Accessible by Name” on page 6-184

Monitor Signals with MATLAB Language

This procedure uses the model `ex_slrt_rt_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_rt_osc')))`). You must have already completed the setup in "Prepare Real-Time Application with 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
```

See Also

More About

- “Configure Target Scopes with MATLAB Language” on page 6-55
- “Troubleshoot Signals Not Accessible by Name” on page 6-184

Instrument a Stateflow Subsystem

In this section...
“Configure Stateflow States as Test Points” on page 6-11 “Monitor Stateflow States with Simulink Real-Time Explorer” on page 6-13

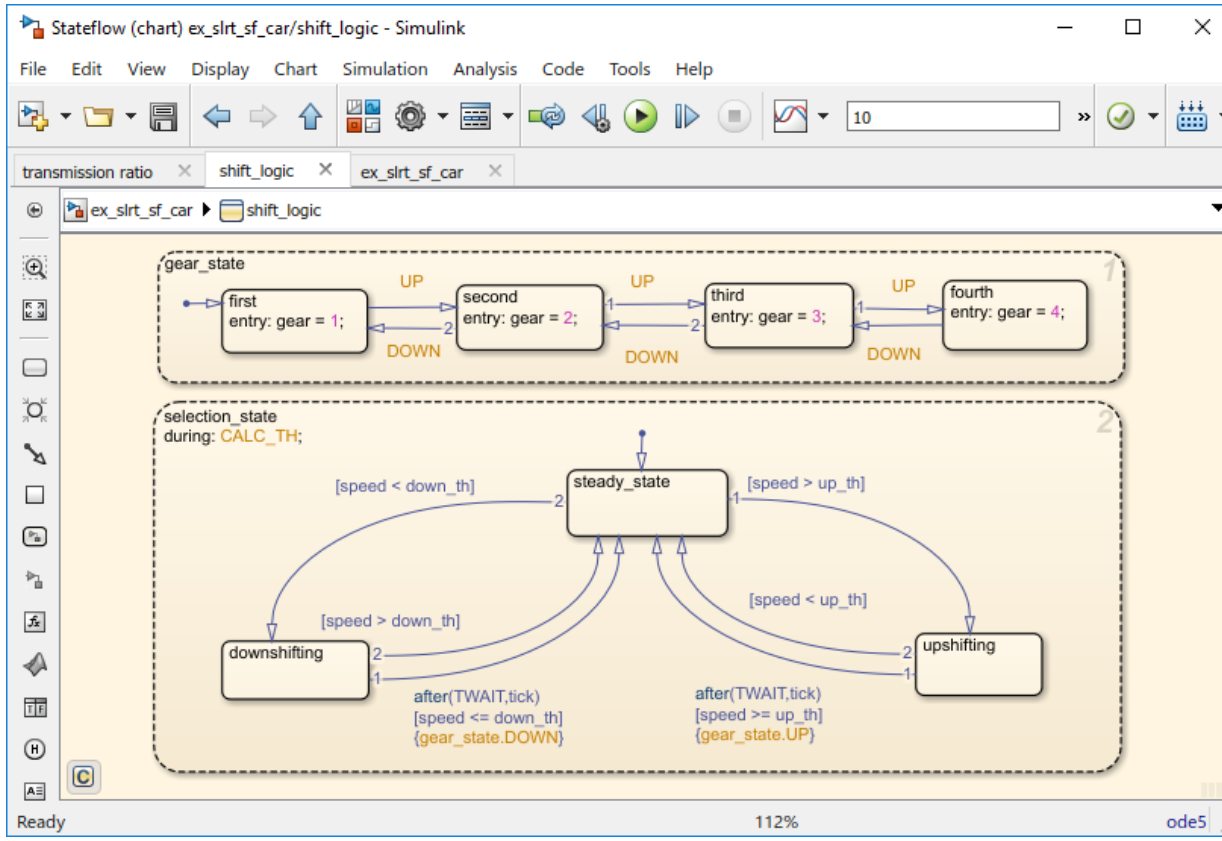
A Simulink Real-Time model that uses Stateflow blocks can present special circumstances. For example, if the model implements a control algorithm as a Stateflow subsystem, the Stateflow signals are not visible to Simulink Real-Time by default.

This procedure uses the model `ex_slrt_sf_car` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_sf_car'))))`.

Configure Stateflow States as Test Points

To make Stateflow signals visible to Simulink Real-Time, mark them as test points:

1. Open `ex_slrt_sf_car`.
2. Double-click the `shift_logic` chart.



3 In the `shift_logic` chart, click **Tools** > **Model Explorer**.

4 In the Model Explorer, expand **ex_slrt_sf_car**, and then **shift_logic**.

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

6 To create a test point for the `first` state, in the **State first** pane **Logging** tab, select the **Test point** check box.

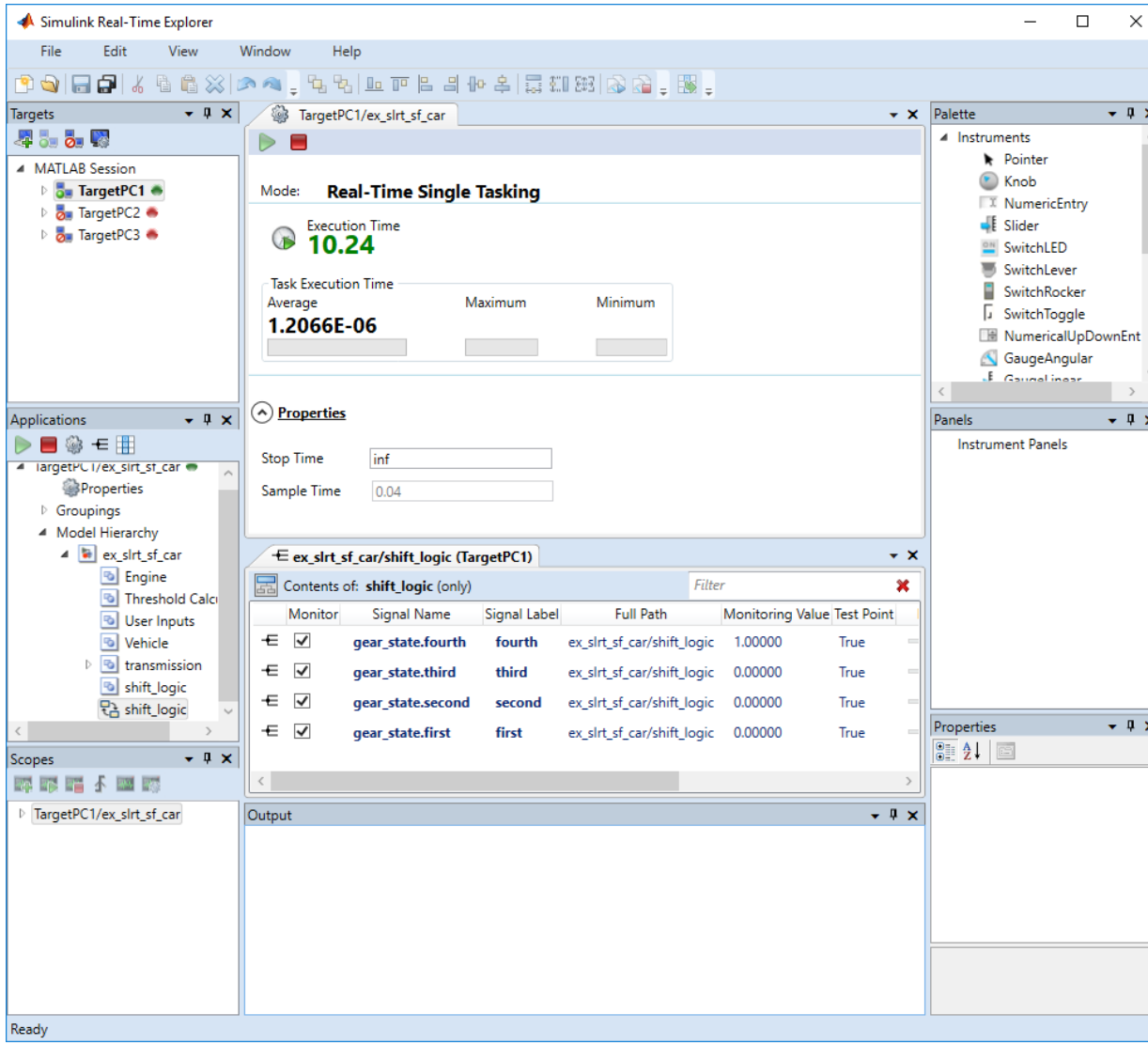
7 Click **Apply**.

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

9 Build and download the `ex_slrt_sf_car` real-time application to the target computer (on the toolbar).

Monitor Stateflow States with Simulink Real-Time Explorer

1. Open Simulink Real-Time Explorer (**Tools** > **Simulink Real-Time**).
2. Connect to the target computer in the **Targets** pane (on the toolbar).
3. In the **Applications** pane, expand the real-time application and the **Model Hierarchy** node.
4. To view the test point, select `shift_logic` and click the **View Signals** button on the toolbar.
5. 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.
6. To start execution, click the real-time application. On the toolbar, click the **Start** button .
7. To stop execution, click the real-time application. On the toolbar, click the **Stop** button .


Simulink Real-Time Explorer
File
Edit
View
Window
Help
Targets
MATLAB Session
TargetPC1
TargetPC2
TargetPC3
TargetPC1/ex_slrt_sf_car
Mode:
Real-Time Single Tasking
Execution Time
10.24
Task Execution Time
Average
Maximum
Minimum
1.2066E-06
Properties
Stop Time
inf
Sample Time
0.04
Palette
Instruments
Pointer
Knob
NumericEntry
Slider
SwitchLED
SwitchLever
SwitchRocker
SwitchToggle
NumericalUpDownEnt
GaugeAngular
Applications
Properties
Groupings
Model Hierarchy
ex_slrt_sf_car
Engine
User Inputs
Vehicle
transmission
shift_logic
Panels
Instrument Panels
ex_slrt_sf_car/shift_logic (TargetPC1)
Contents of: shift_logic (only)
Filter
Monitor
Signal Name
Signal Label
Full Path
Monitoring Value
Test Point
gear_state.fourth
fourth
ex_slrt_sf_car/shift_logic
1.00000
True
gear_state.third
third
0.00000
gear_state.second
second
gear_state.first
first
Properties
Scopes
Output
Ready

See Also

More About

- “Monitor Stateflow States with MATLAB Language” on page 6-17
- “Animate Stateflow Charts with Simulink External Mode” on page 6-18
- “Create Signal Groups with Simulink Real-Time Explorer” on page 6-58
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179
- “Signal Group Monitoring Formats” on page 6-16
- “Troubleshoot Signals Not Accessible by Name” on page 6-184

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 character vectors.

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`.

See Also

More About

- “Create Signal Groups with Simulink Real-Time Explorer” on page 6-58

Monitor Stateflow States with MATLAB Language

You must have already set Stateflow states as test points in model `ex_slrt_sf_car` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_sf_car')))`). If you have not, see “Configure Stateflow States as Test Points” on page 6-11.

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, assume that you set a test point for the `first` state of `gear_state` in the `shift_logic` chart of the `ex_slrt_sf_car` model. The state of interest, `first`, appears as follows in the list of signals in the MATLAB interface:

   ```
   shift_logic:gear_state.first
   ```

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

See Also

More About

- “Troubleshoot Signals Not Accessible by Name” on page 6-184

Animate Stateflow Charts with Simulink External Mode

The Simulink Real-Time software supports the animation of Stateflow charts in your model to provide visual confirmation 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” (Stateflow).

You must have already set Stateflow states as test points in model `ex_slrt_sf_car` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_sf_car')))`). If you have not, see “Configure Stateflow States as Test Points” on page 6-11.

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.

See Also

More About

- “Troubleshoot Signals Not Accessible by Name” on page 6-184

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.

See Also

More About

- “Troubleshoot Signals Not Accessible by Name” on page 6-184
- “Simulink Real-Time Scope Usage” on page 6-21
- “Target Scope Usage” on page 6-23
- “Host Scope Usage” on page 6-61
- “File Scope Usage” on page 6-90
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179

Simulink Real-Time Scope Usage

- 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 unit gain block (a Gain block with a gain of `1`) 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

- “Troubleshoot Signals Not Accessible by Name” on page 6-184
- “Target Scope Usage” on page 6-23
- “Host Scope Usage” on page 6-61

- “File Scope Usage” on page 6-90
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179

Target Scope Usage

- 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

More About

- “Configure Real-Time Target Scope Blocks” on page 6-25
- “Create Target Scopes with Simulink Real-Time Explorer” on page 6-31
- “Simulink Real-Time Scope Usage” on page 6-21
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179

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.

This procedure uses the example model `ex_slrt_rt_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_rt_osc')))`).

1. In the Command Window, open `ex_slrt_rt_osc`.
2. Double-click the block labeled Scope.

   The Scope block 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`, or `Graphical rolling`. (The `Graphical sliding` will be removed in a future release. It behaves like `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 do not 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 numerical 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 format character vector 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 insert a single entry, that entry applies to each signal (scalar expansion). If you enter N label entries for $N+K$signals, 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 character vector 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` 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` 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 one of the following:

    - `FreeRun` or `Software Triggering` — No extra parameters.
    - `Signal Triggering` — enter additional parameters, as required:

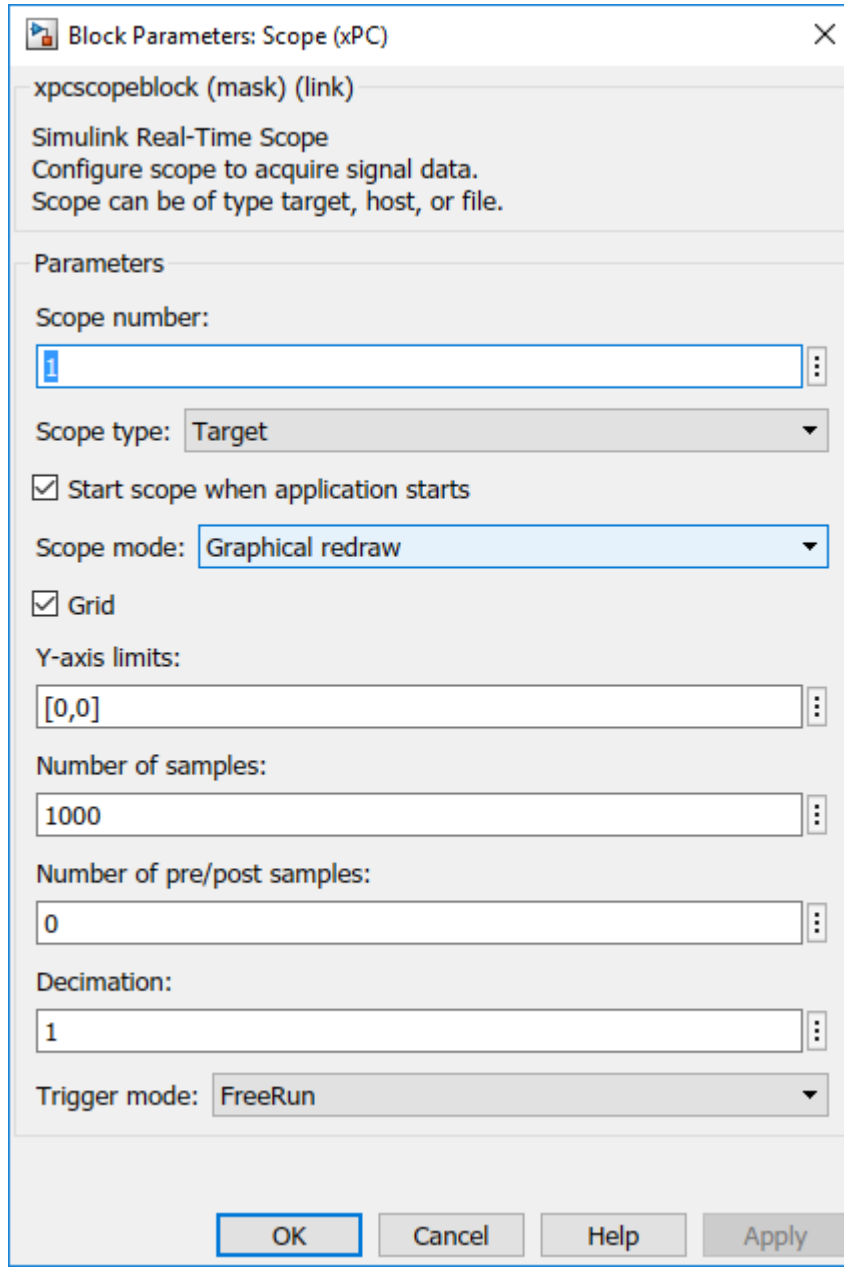
      - In the **Trigger signal** box, enter the index of a signal previously added to the scope.

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`.

- `Scope Triggering` — enter 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, `0`, indicates that the triggered scope starts on the same sample as the triggering scope.

The target scope dialog box looks like this figure.


Block Parameters: Scope (xPC)
xpcscopeblock (mask) (link)
Simulink Real-Time Scope
Configure scope to acquire signal data.
Scope can be of type target, host, or file.
Parameters
Scope number:
1
Scope type:
Target
Start scope when application starts
Scope mode:
Graphical redraw
Grid
Y-axis limits:
[0,0]
Number of samples:
1000
Number of pre/post samples:
0
Decimation:
1
Trigger mode:
FreeRun
OK
Cancel
Help
Apply

14 Click **OK**.

15 From the **File** menu, click **Save As**.

Save the model as `ex_slrt_target_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_target_osc')))`).

See Also

Scope

More About

- “Simulink Real-Time Scope Usage” on page 6-21
- “Target Scope Usage” on page 6-23
- “Trigger One Scope with Another Scope” on page 11-19

Create Target Scopes with Simulink Real-Time Explorer

You can create a 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 6-25 and “Target Scope Usage” on page 6-23.

This procedure uses the model `xpcosc`. 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 target scope:

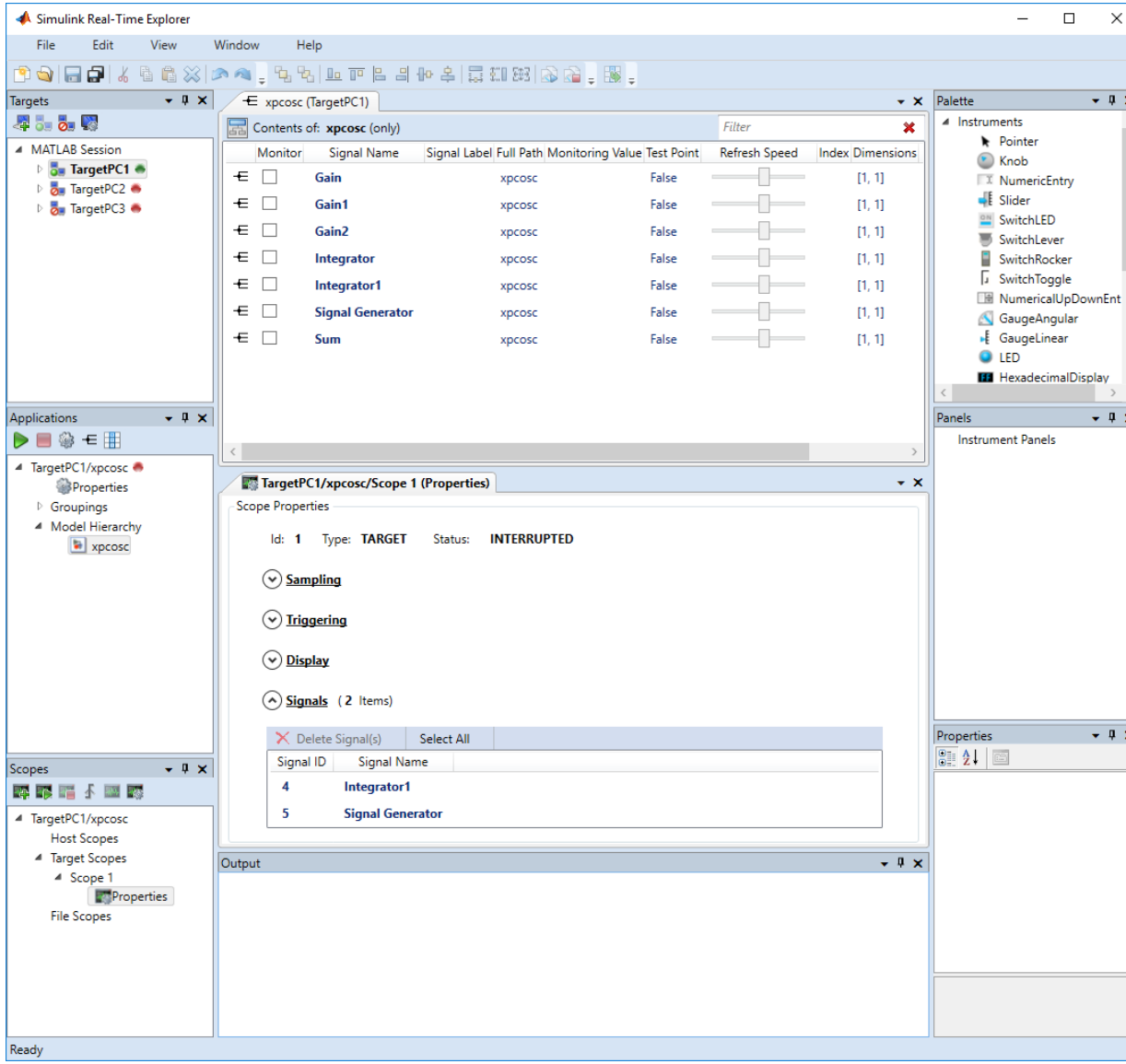
1. In the **Scopes** pane, expand the **xpcosc** node.
2. To add a target scope, select **Target Scopes** and then click the **Add Scope** button on the toolbar.

   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:

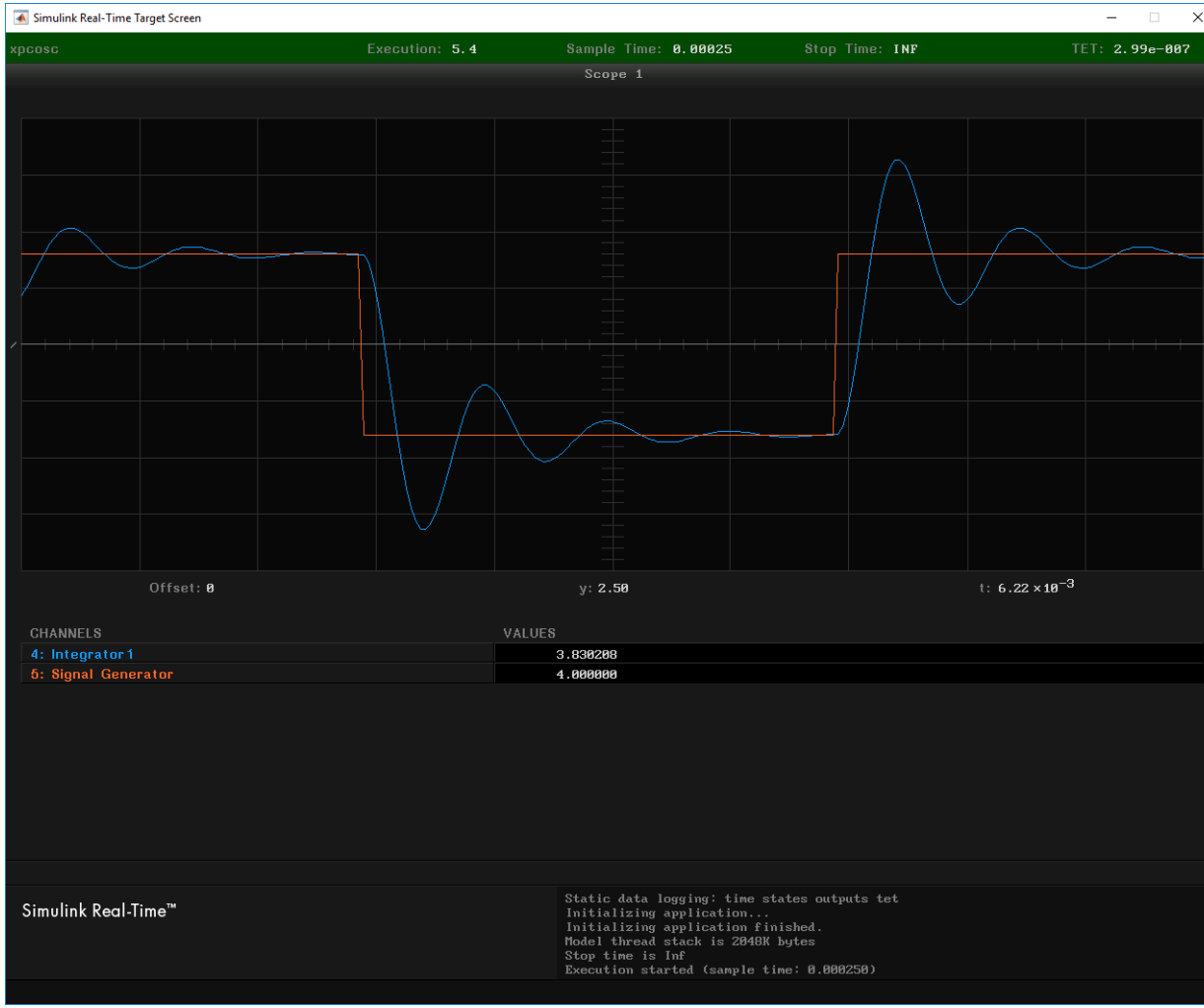
```
Scope: 1, set to state 'interrupted'
```

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 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 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 the workspace 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`

More About

- “Create Signal Groups with Simulink Real-Time Explorer” on page 6-58
- “Configure Target Scopes with Simulink Real-Time Explorer” on page 6-51
- “Configure Scope Sampling with Simulink Real-Time Explorer” on page 6-36
- “Trigger Scopes with Simulink Real-Time Explorer” on page 6-40
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179

Configure Scope Sampling with 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`. You must have already completed the procedure in “Create Target Scopes with Simulink Real-Time Explorer” on page 6-31. Target execution and scopes must be stopped.

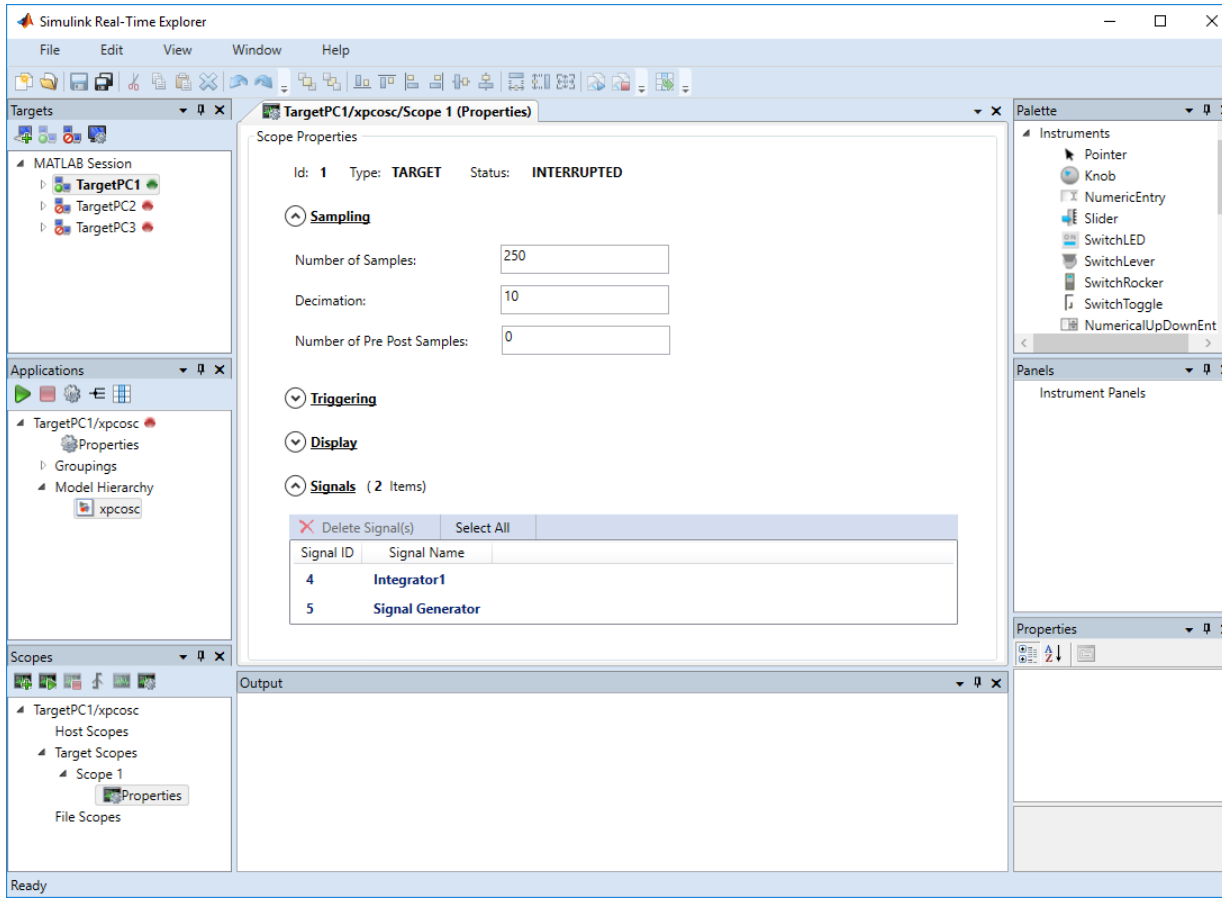
1. Select **Scope 1** and open the Properties pane (on the **Scopes** toolbar).
2. In the **Scope 1** properties pane, click **Sampling**.
3. In the **Number of Samples** box, enter the number of values to be acquired in a data package, here `250`.

   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`.

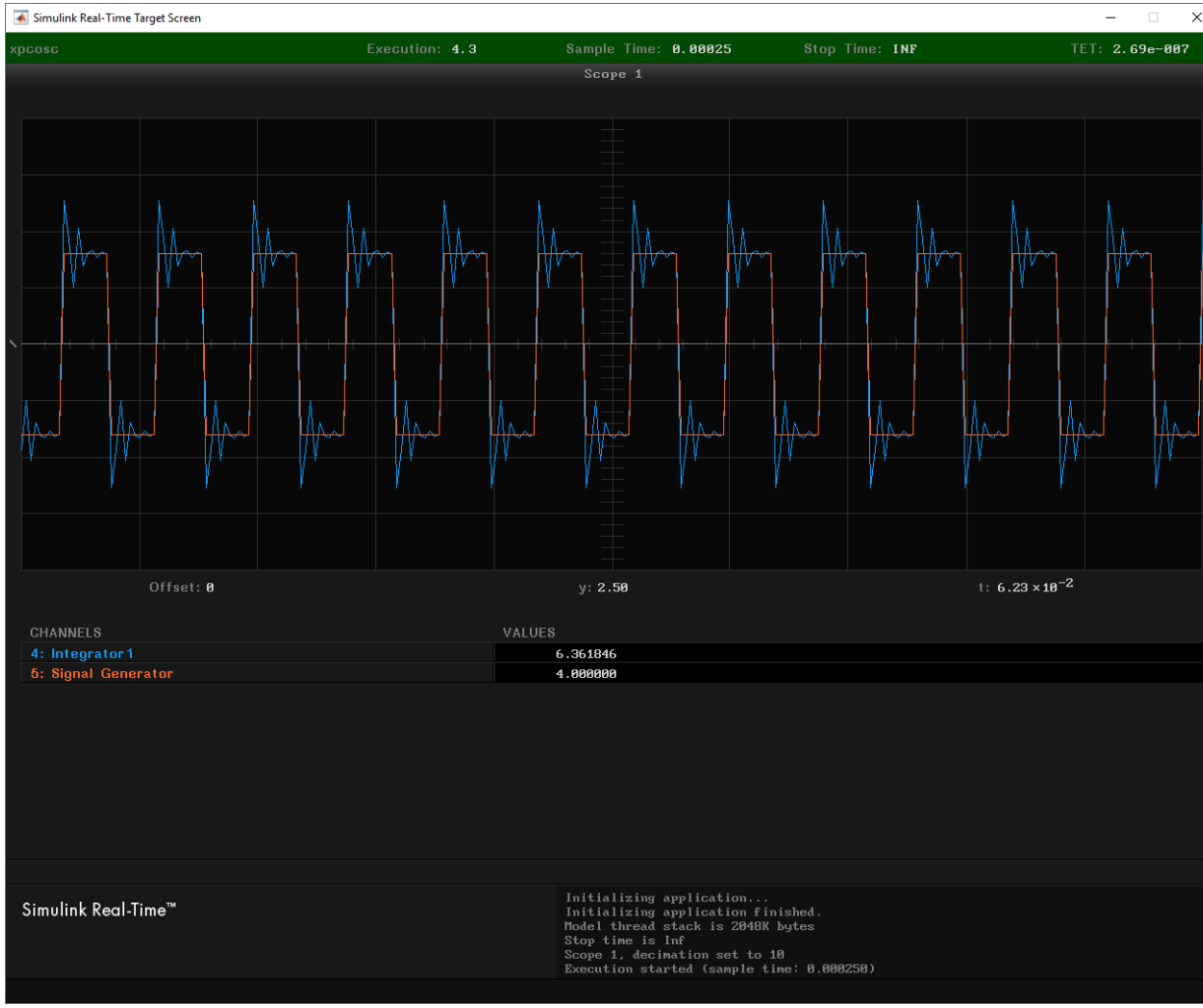
   The dialog box looks like this figure.



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).

See Also

More About

- “Trigger Scopes with Simulink Real-Time Explorer” on page 6-40

Trigger Scopes with Simulink Real-Time Explorer

To facilitate your interaction with the running model, you can configure 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`. You must have already completed the procedure in “Create Target Scopes with Simulink Real-Time Explorer” on page 6-31. Target execution and scopes must be stopped.

In this section...
“Freerun Triggering” on page 6-40
“Software Triggering” on page 6-40
“Signal Triggering” on page 6-43
“Scope Triggering” on page 6-47

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

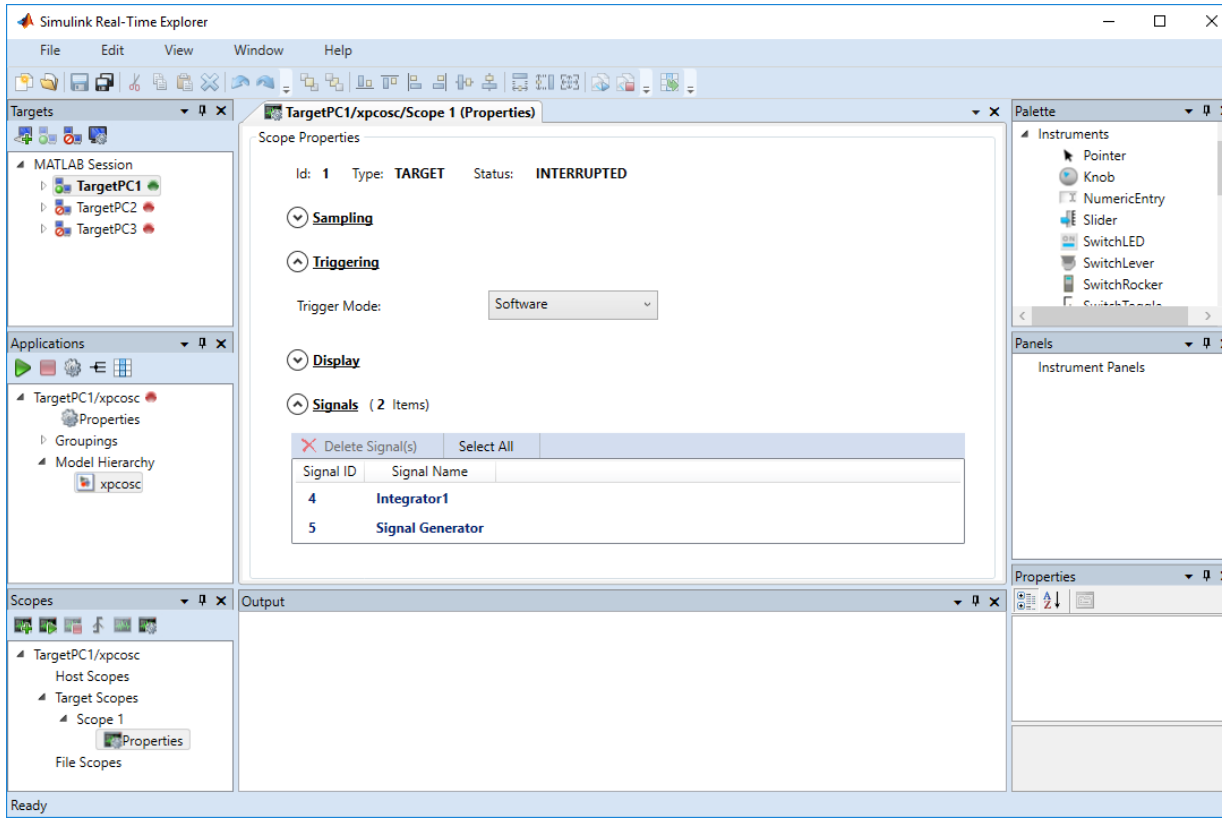
In **Trigger Mode** `Software`, the scope triggers when you select **Scope 1** and then click the **Trigger** button on the toolbar.

1. Start execution (on the **Applications** toolbar).
2. Select **Trigger Mode** `Software`.
3. Start **Scope 1** (on the toolbar).

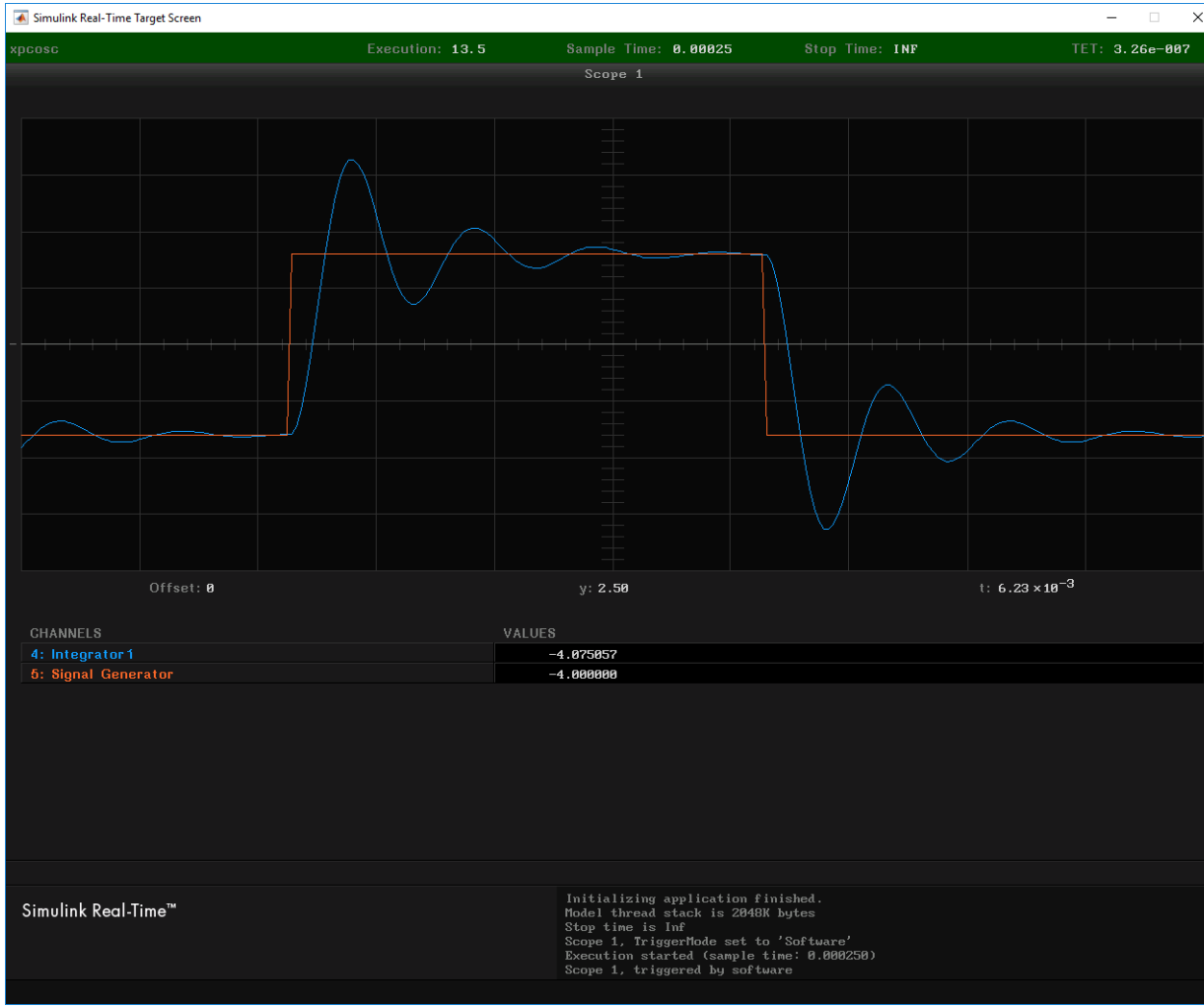
   The **Trigger** button is enabled on the toolbar.
4. Click the **Trigger** button on the **Scopes** toolbar.

   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.



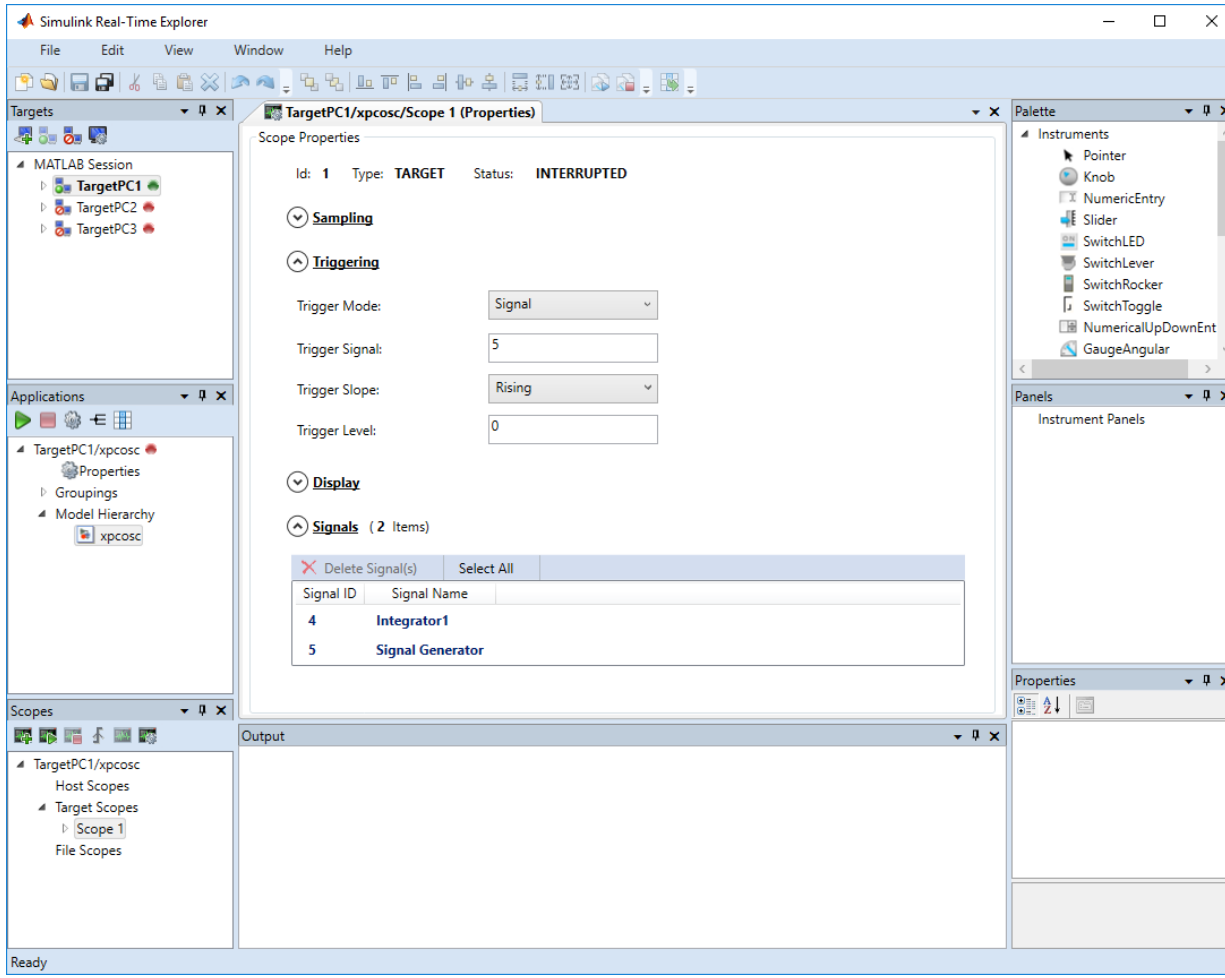
6 Stop execution (on the **Applications** toolbar).

Signal Triggering

In **Trigger Mode** `Signal`, the scope triggers when a signal rises or falls through a specified level.

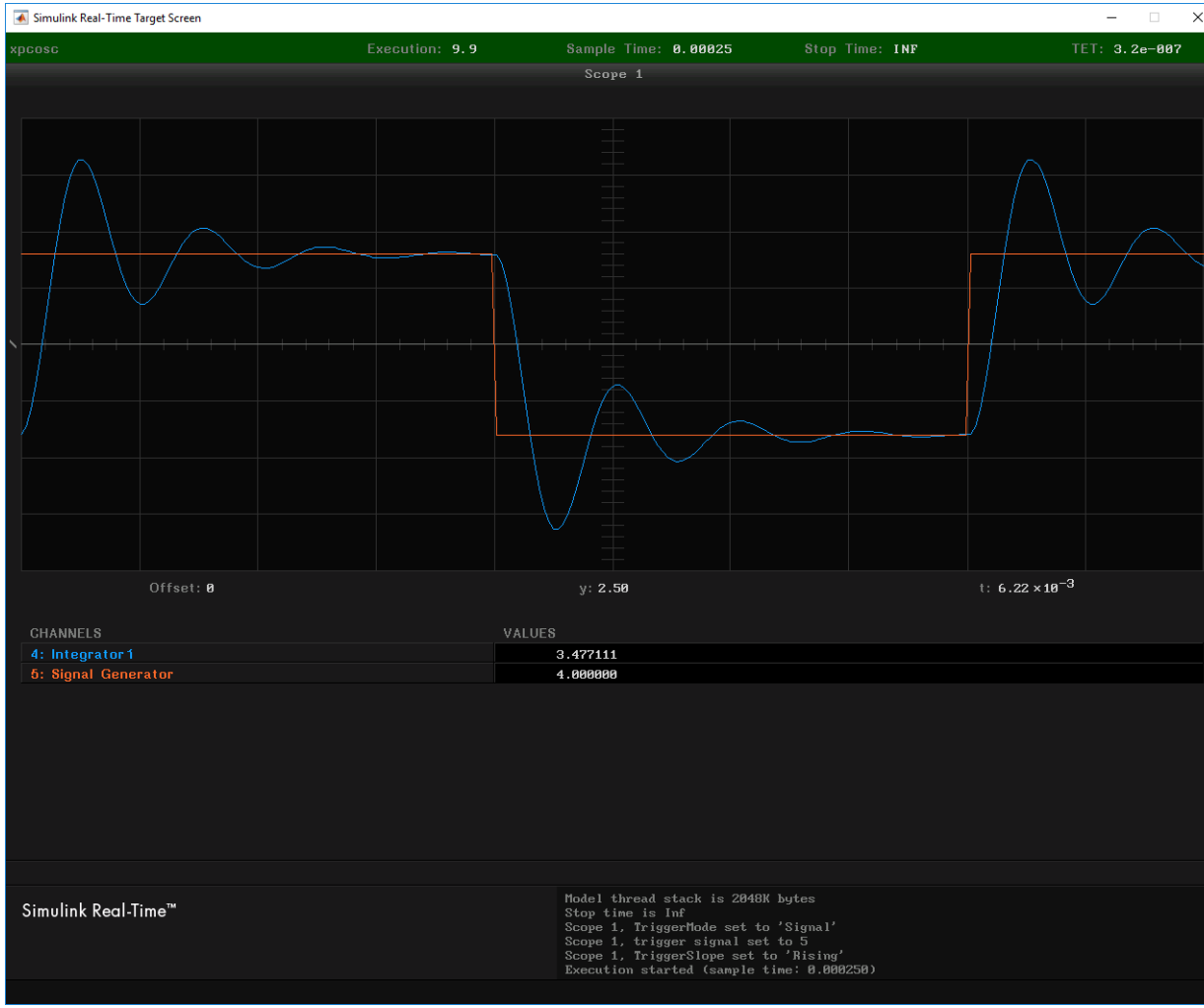
1. Start execution (on the **Applications** toolbar).
2. Select **Scope 1** and open the Properties pane (on the **Scopes** toolbar).
3. In the **Scope 1** pane, click **Triggering**.
4. Select **Trigger Mode** `Signal`.

   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** (on the toolbar).

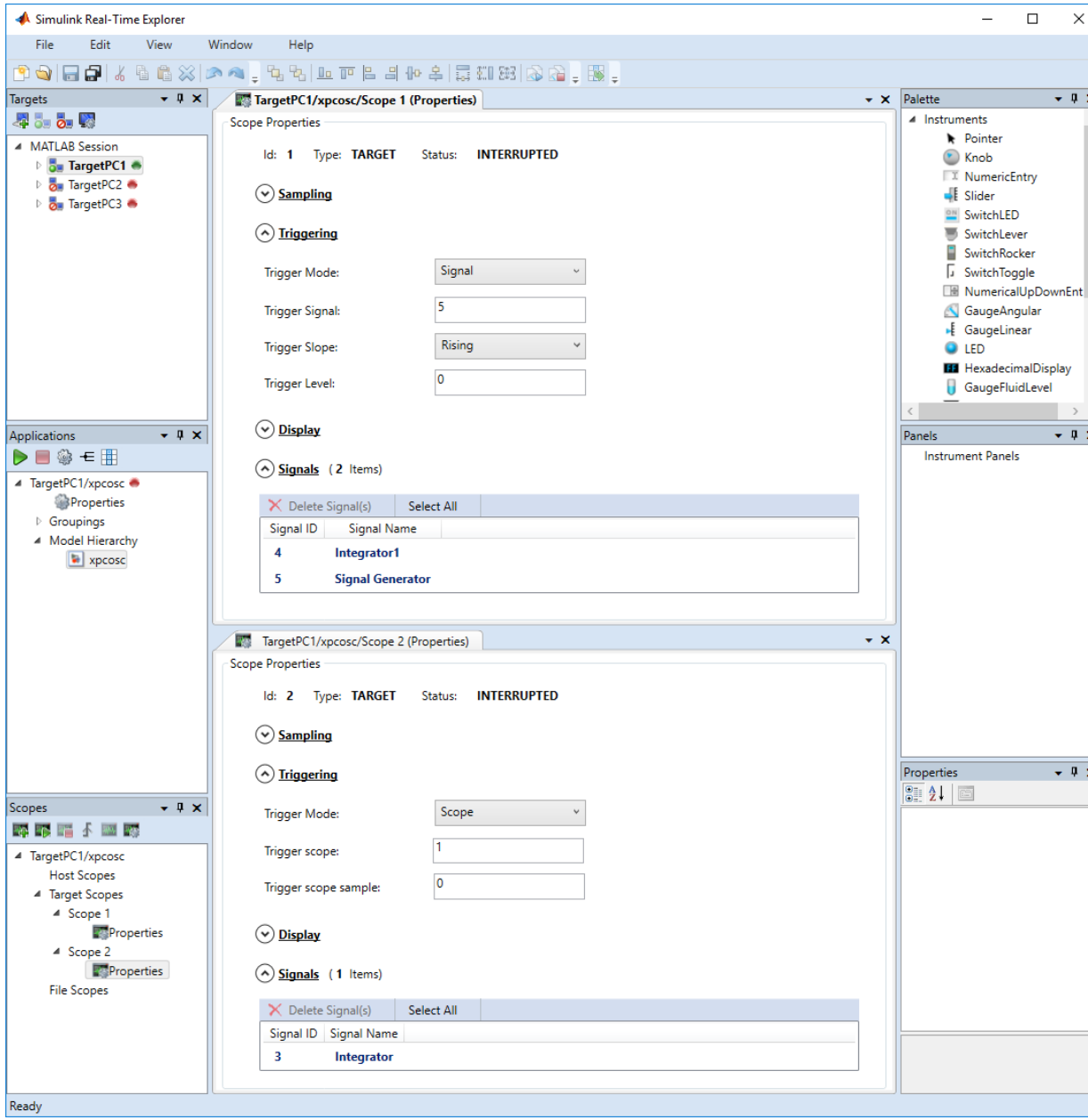
10 Stop execution (on the **Applications** toolbar).

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**.


Simulink Real-Time Explorer
File
Edit
View
Window
Help
Targets
MATLAB Session
TargetPC1
TargetPC2
TargetPC3
Applications
TargetPC1/xpcosc
Properties
Groupings
Model Hierarchy
xpcosc
Scopes
TargetPC1/xpcosc
Host Scopes
Target Scopes
Scope 1
Properties
Scope 2
Properties
File Scopes
TargetPC1/xpcosc/Scope 1 (Properties)
Scope Properties
Id: 1 Type: TARGET Status: INTERRUPTED
Sampling
Triggering
Trigger Mode: Signal
Trigger Signal: 5
Trigger Slope: Rising
Trigger Level: 0
Display
Signals (2 Items)
Delete Signal(s)
Select All
Signal ID
Signal Name
4 Integrator1
5 Signal Generator
TargetPC1/xpcosc/Scope 2 (Properties)
Scope Properties
Id: 2 Type: TARGET Status: INTERRUPTED
Sampling
Triggering
Trigger Mode: Scope
Trigger scope: 1
Trigger scope sample: 0
Display
Signals (1 Items)
Delete Signal(s)
Select All
Signal ID
Signal Name
3 Integrator
Palette
Instruments
Pointer
Knob
NumericEntry
Slider
SwitchLED
SwitchLever
SwitchRocker
SwitchToggle
NumericalUpDownEnt
GaugeAngular
GaugeLinear
LED
HexadecimalDisplay
GaugeFluidLevel
Panels
Instrument Panels
Properties
Ready

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).

See Also

More About

- “Display and Filter Hierarchical Signals and Parameters” on page 6-179

Configure Target Scopes with 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.

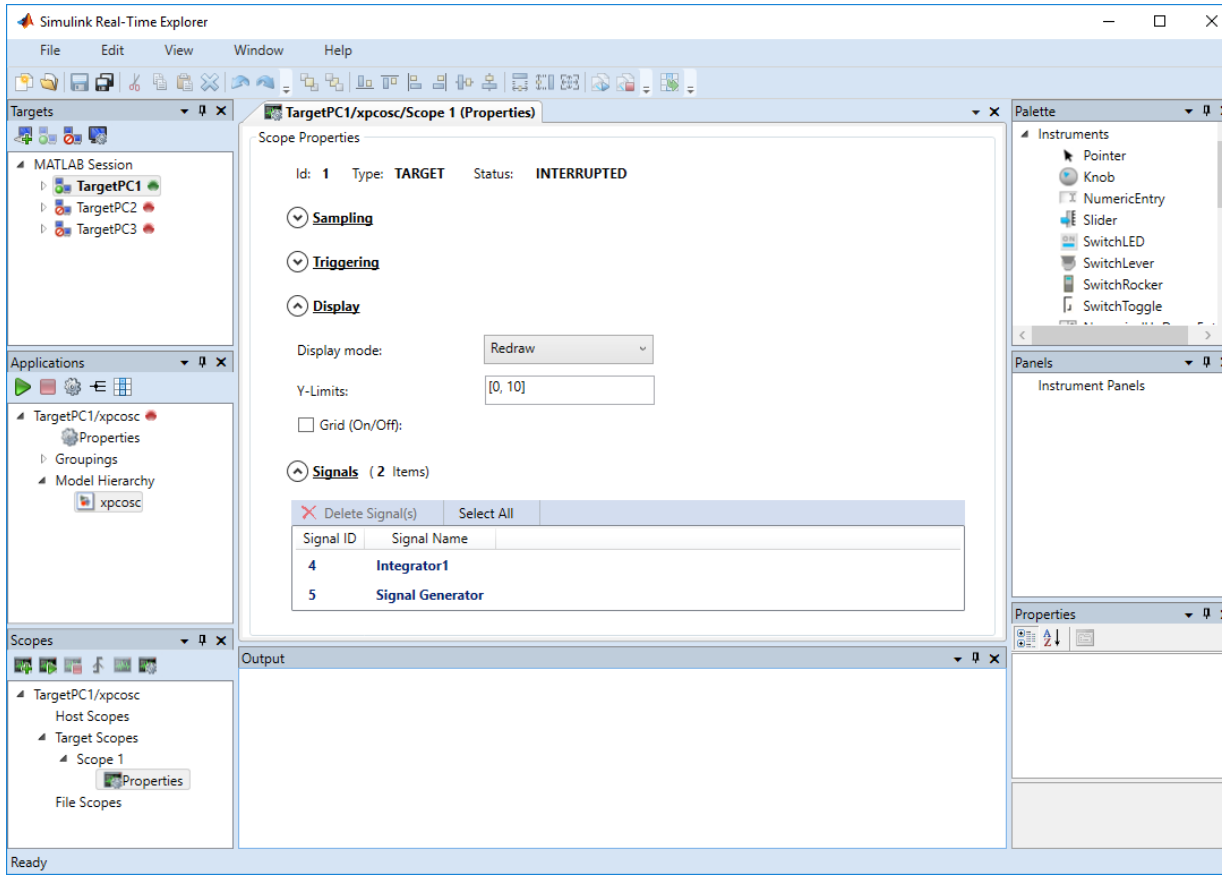
This procedure uses the model `xpcosc`. You must have already completed the procedure in “Create Target Scopes with Simulink Real-Time Explorer” on page 6-31. Target execution and scopes must be stopped.

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.

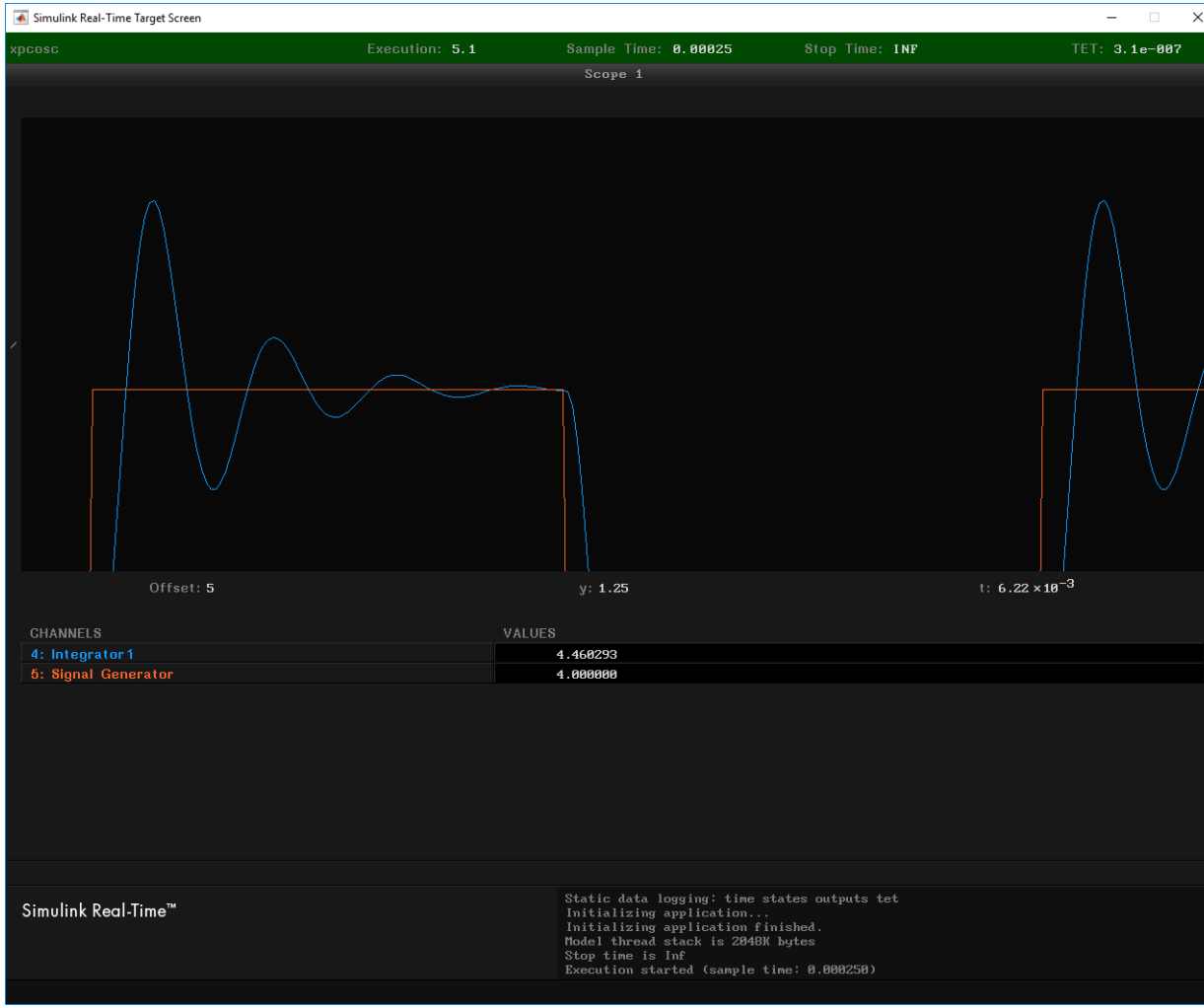
   This value is the default. It causes the scope display to redraw when it has acquired as many samples as specified in **Number of Samples**.
5. Start **Scope 1** (on the toolbar).

   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 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 log. New data enters the display from the right and then moves toward the left.

10 Stop **Scope 1** (on the toolbar).

11 Stop execution (on the **Applications** toolbar).

See Also

`SimulinkRealTime.target.viewTargetScreen`

Configure Target Scopes with 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`. To do this procedure, you must have already built the real-time application for`xpcosc` 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)
```

See Also

More About

- “Monitor Signals with MATLAB Language” on page 6-9

Create Signal Groups with 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`. 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**.

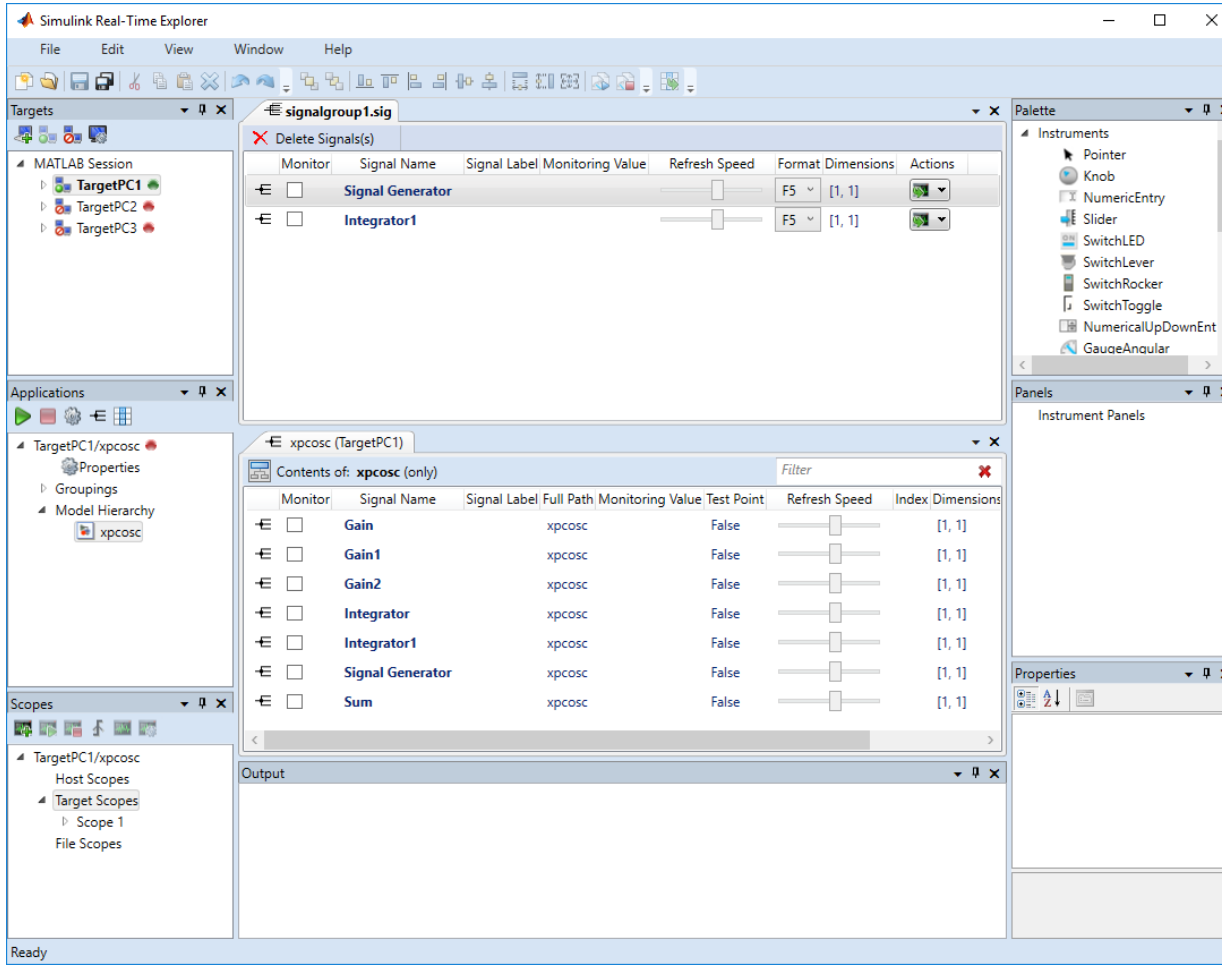
   The Add New Signal Group Item dialog box appears.
3. In the Add New Signal Group Item dialog box, enter a name in the **Name** text box, for example **signalgroup1.sig**. In the **Location** text box, enter a folder for the group file.
4. Click **OK**.

   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 6-16.



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 the workspace 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

More About

- “Monitor Signals with Simulink Real-Time Explorer” on page 6-6
- “Create Target Scopes with Simulink Real-Time Explorer” on page 6-31
- “Create Host Scopes with Simulink Real-Time Explorer” on page 6-66
- “Create File Scopes with Simulink Real-Time Explorer” on page 6-98
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179

Host Scope Usage

- 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.
- To clarify your model functionality, consider adding signal labels. If you define signal labels, the host scope displays the labels, highlighted with pointed brackets, instead of the signal names. If you do not define signal labels, the host scope displays the short name of the signal.
- 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. 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 Scope block dialog box by setting **Trigger Mode** to `Signal Triggering`, `Software Triggering`, or `Scope Triggering`.
- The target computer transfers data to the development computer for display in the host scope viewer. Because of this latency, the host scope display lags a target scope display during real-time application execution.

See Also

More About

- “Configure Real-Time Host Scope Blocks” on page 6-62
- “Create Host Scopes with Simulink Real-Time Explorer” on page 6-66
- “Simulink Real-Time Scope Usage” on page 6-21
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179

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.
- To clarify your model functionality, consider adding signal labels. If you define signal labels, the host scope displays the labels, highlighted with pointed brackets, instead of the signal names. If you do not define signal labels, the host scope displays the short name of the signal.

This procedure uses the example model `ex_slrt_rt_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_rt_osc')))`).

1. In the Command Window, open `ex_slrt_rt_osc`.
2. Double-click the block labeled **Scope**.

   The Scope block 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 one of the following:

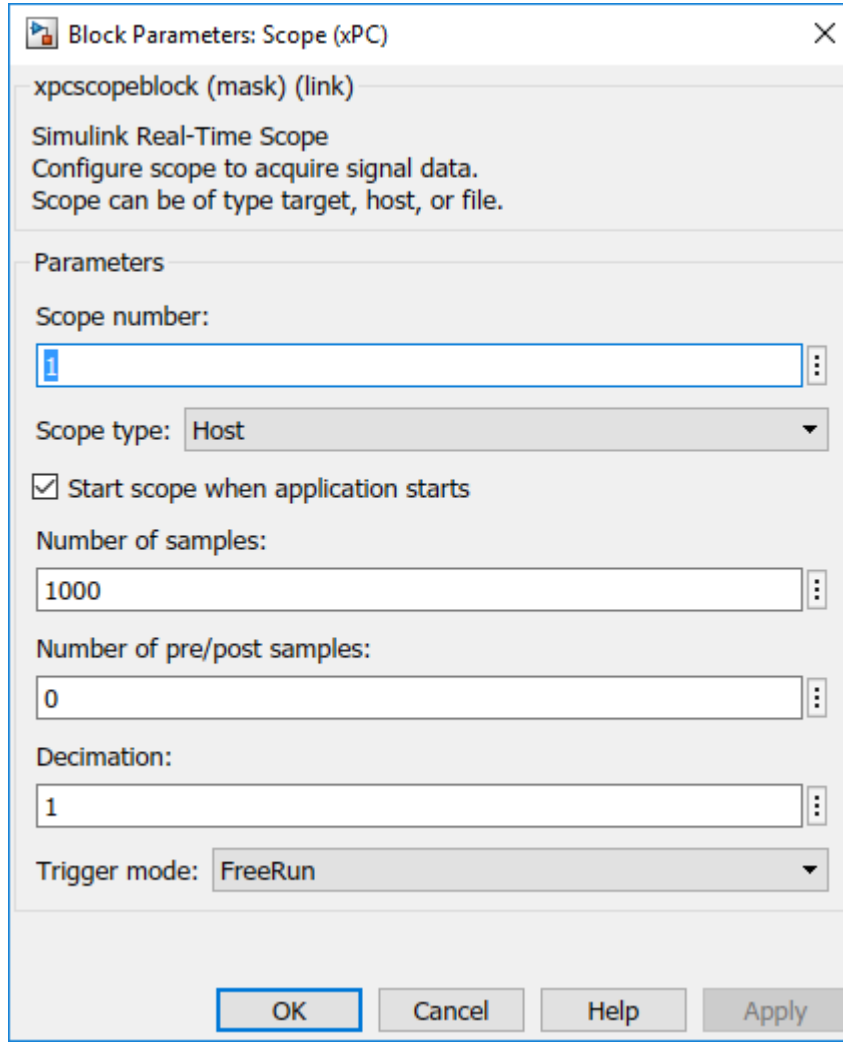
   - `FreeRun` or `Software Triggering` — No extra parameters.
   - `Signal Triggering` — enter additional parameters, as required:

     - 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`.
   - `Scope Triggering` — enter 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, `0`, indicates that the triggered scope starts on the same sample as the triggering scope.

   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` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_host_osc')))`).

See Also

Scope

More About

- “Simulink Real-Time Scope Usage” on page 6-21
- “Host Scope Usage” on page 6-61
- “Create Host Scopes with Simulink Real-Time Explorer” on page 6-66
- “Trigger One Scope with Another Scope” on page 11-19

Create Host Scopes with Simulink Real-Time Explorer

You can create a host scope on the target computer with 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.

For information on using host scope blocks, see “Configure Real-Time Host Scope Blocks” on page 6-62 and “Host Scope Usage” on page 6-61.

This procedure uses the model `ex_slrt_sf_car` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_sf_car')))`).

Set Up Model

Before creating a host scope, perform these steps:

1. Build and download `ex_slrt_sf_car` to the target computer with Simulink (on the toolbar).
2. Run Simulink Real-Time Explorer (**Tools** > **Simulink Real-Time**).
3. Connect to the target computer that is in the **Targets** pane (on the toolbar).
4. Set property **Stop time** to `inf` in the **Applications** pane (on the toolbar).

Configure Host Scope

1. In the **Scopes** pane, expand the **ex_slrt_sf_car** node.
2. To add a host scope, select **Host Scopes**, and then click the **Add Scope** button on the toolbar.
3. Expand **Scope 1**, and then click the **Properties** button on the toolbar.

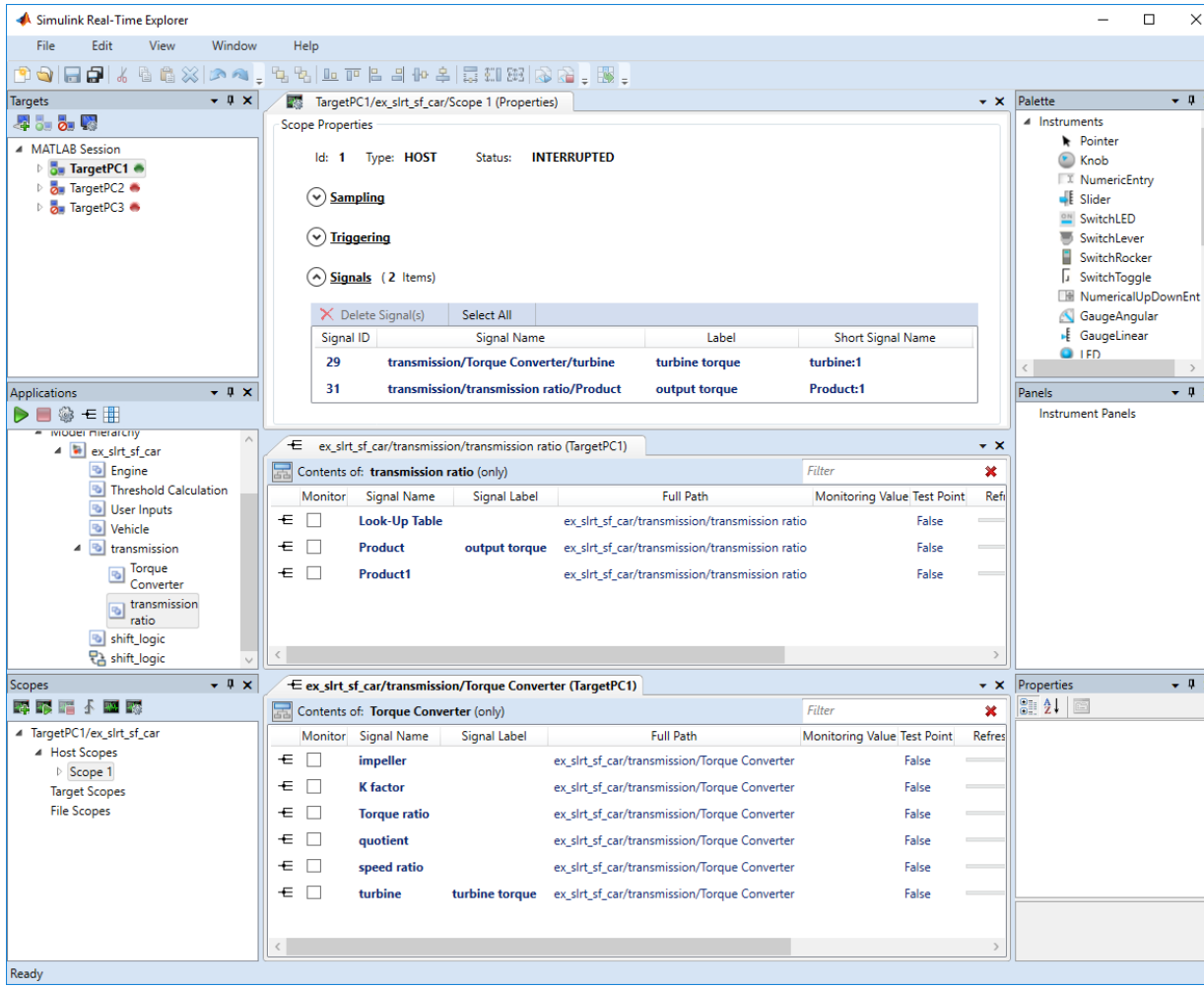
   To display the host scope signals, in the **Scope Properties** pane, click **Signals**.
4. In the **Applications** pane, expand the real-time application node, and then the node **Model Hierarchy**.
5. Double-click the **ex_slrt_sf_car** > **transmission** > **Torque Converter** node.

   The **Torque Converter** signal list opens.

6 To add signal `turbine` to **Scope1**, drag signal `turbine` from the **Torque Converter** signal list to the **Scope1** properties workspace.

   To make the **Scope1** properties visible below the **Torque Converter** signal list, drag the **Torque Converter** tab down until the icon appears.

7 Double-click **transmission ratio** and add signal `Product` to **Scope 1** in the same way as described in step 6.



View Host Scope

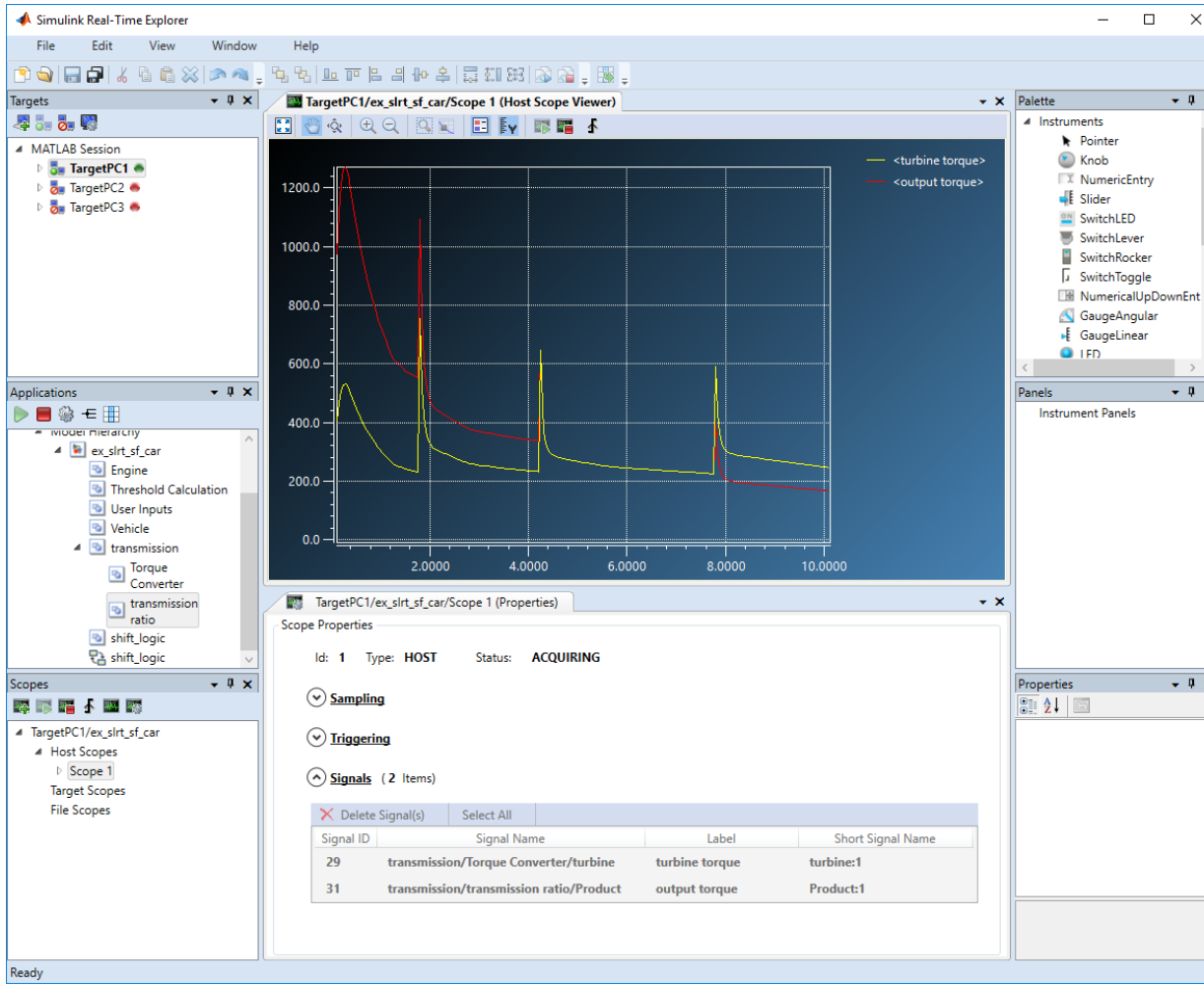
1. Select **Scope 1**, and then click the **View Scope** button on the toolbar.

   The host scope viewer opens as a separate tab. The signals that you add to the scope appear at the top right of the viewer. The labels appear in pointed brackets because these signals are labeled signals.

2. To start **Scope 1**, click **Scope 1** in the **Scopes** pane, and then click the **Start Scope** button on the toolbar.

3. To start execution, click the real-time application, and then click the **Start** button on the toolbar.

   The real-time application starts running. The host scope on the target computer transfers data to the host scope on the development computer.



4 To stop **Scope 1**, click **Scope 1** in the **Scopes** pane, and then click the **Stop Scope** button on the toolbar.

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

See Also

Scope

More About

- “Host Scope Usage” on page 6-61
- “Configure Real-Time Host Scope Blocks” on page 6-62
- “Configure the Host Scope Viewer” on page 6-71
- “Create Signal Groups with Simulink Real-Time Explorer” on page 6-58
- “Configure Scope Sampling with Simulink Real-Time Explorer” on page 6-36
- “Trigger Scopes with Simulink Real-Time Explorer” on page 6-40
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179

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`. You must have already completed the procedure in “Create Host Scopes with Simulink Real-Time Explorer” on page 6-66. 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.
- **Y-Axes Scale Display** — To show the scale of the Y-axis.

7 To stop **Scope 1**, click the **Stop** button on the Host Scope Viewer toolbar.

8 Stop execution (on the **Applications** toolbar).

See Also

More About

- “Trigger Scopes with Simulink Real-Time Explorer” on page 6-40

Trace Signals with Simulink External Mode

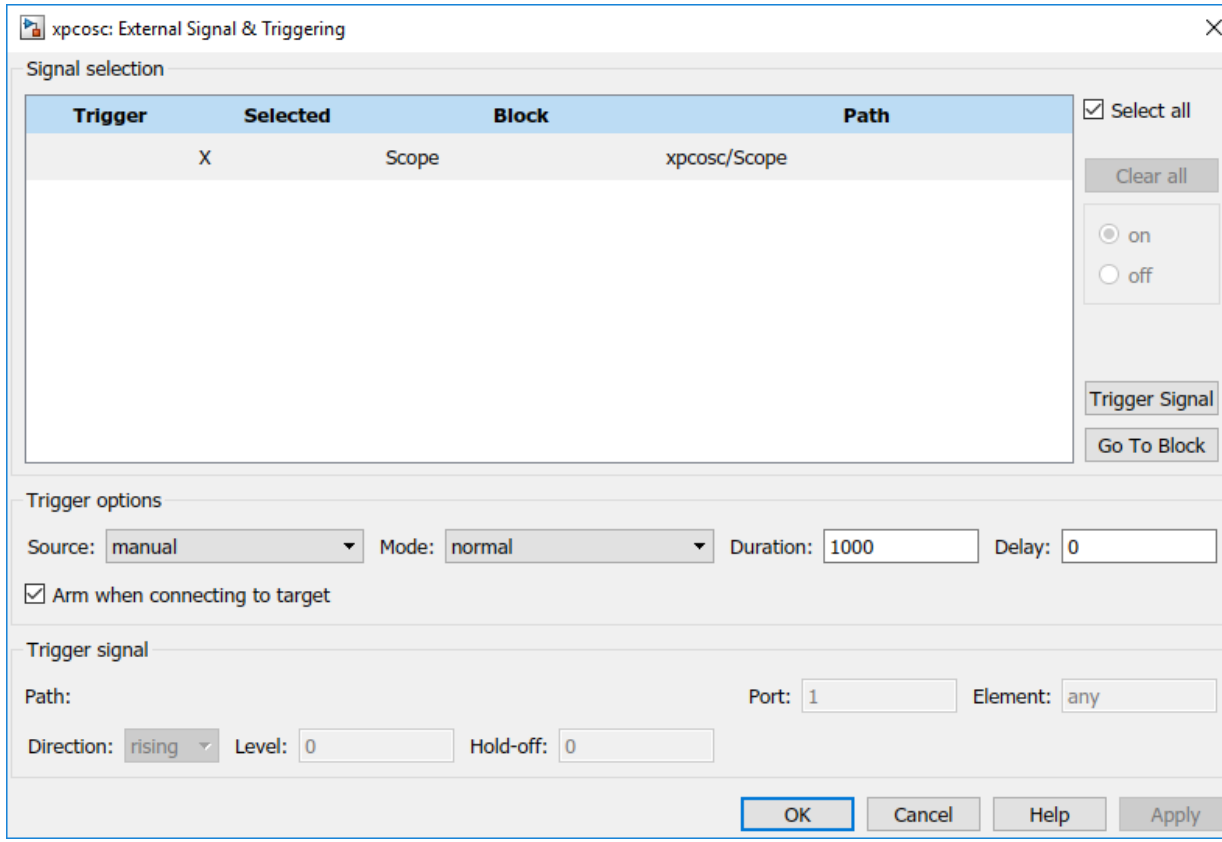
You can use Simulink external mode to establish a communication channel between your Simulink block diagram and your real-time application. The block diagram becomes a user interface to your real-time application. Simulink scopes can display signal data from the real-time application, including from 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" (Simulink Coder) and "Control External Mode Simulation Through External Mode Control Panel" (Simulink Coder)).

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`. `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.

See Also

Inspect Simulink® Real-Time™ Data with Simulation Data Inspector

This example shows how to use Simulation Data Inspector (SDI) to log signal and task execution time (TET) data from the real-time application. You can select signals for display from models referenced at arbitrary levels within a model hierarchy.

- Simulation Data Inspector (SDI) and the third-party calibration tools (Vector CANape® and ETAS® Inca) are mutually exclusive. If you use SDI to view signal data, you cannot use the calibration tools. If you use the calibration tools, you cannot use SDI to view signal data.
- 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, consider selecting buffered logging. You can also reduce the number of signals being inspected or increase the sample time.
- Simulink® Real-Time™ records signals inside enabled subsystems even when they are not running. In `while` and `for` iterator subsystems, Simulink® Real-Time™ records only the last data point.

This example uses the model `xpcosc` (`open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcosc'))`).

In this example, you control the model from Simulink® Real-Time™ Explorer. You can also access Simulation Data Inspector by using external mode.

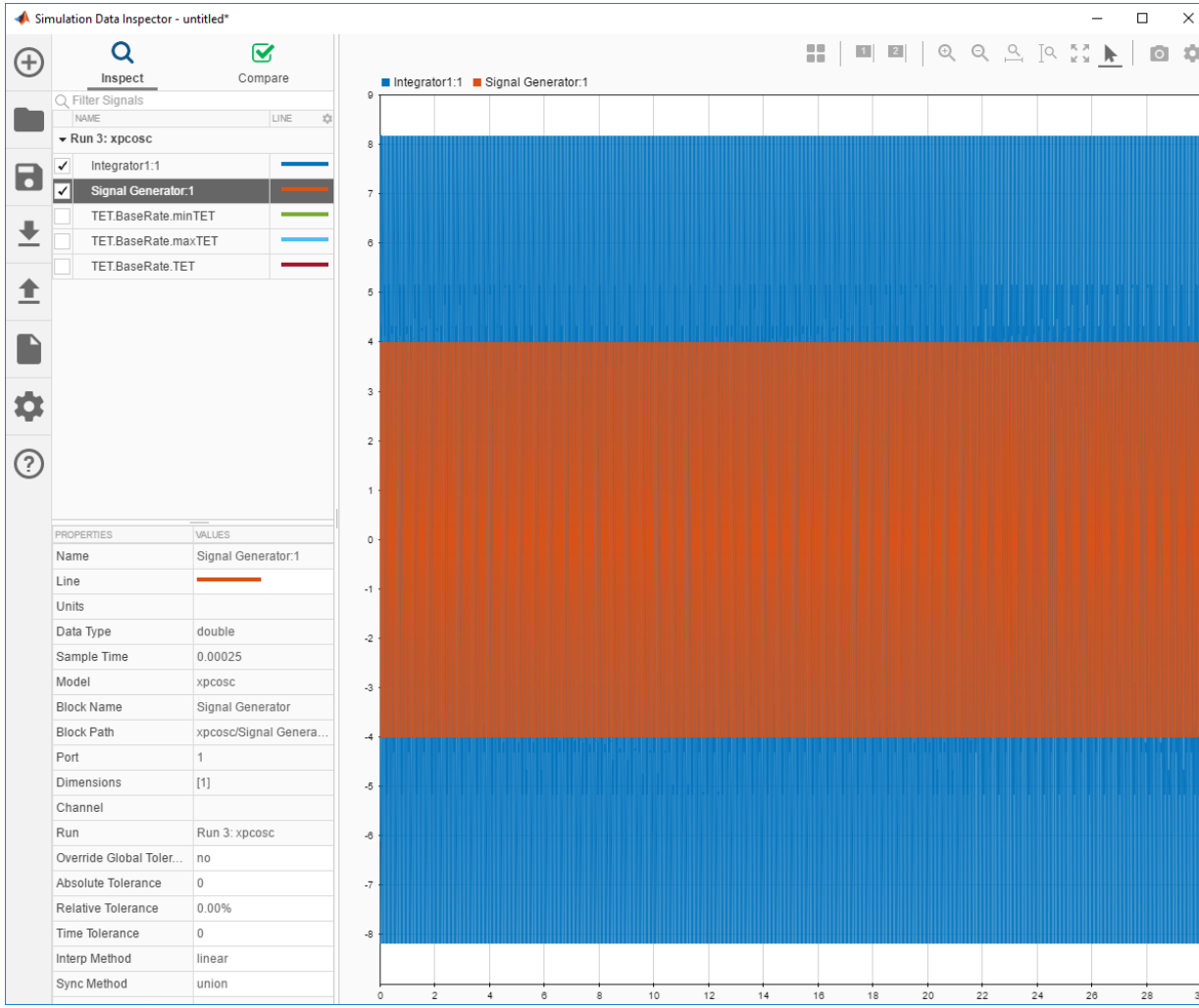
SDI Setup

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. To log signals with SDI, in the model, select and right-click the signals `Signal Generator` and `Integrator1`. Select **Log Selected Signals**. A faint **Simulation Data Inspector** icon appears next to each signal.
4. To log task execution time (TET), open the Configuration Parameters dialog box. In the **Simulink Real-Time Options** tab, select **Monitor Task Execution Time**.
5. Build the model and download it to the target computer.

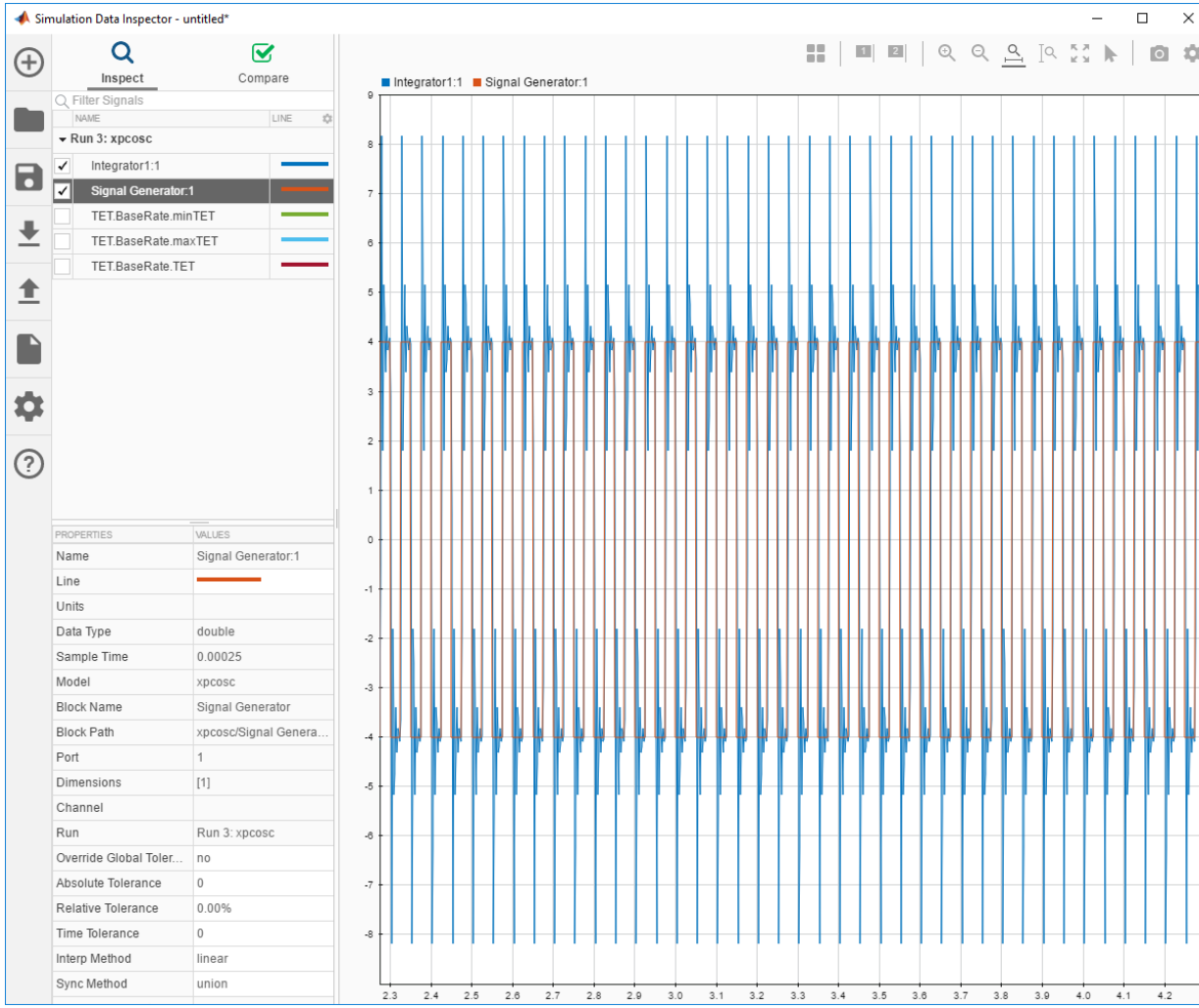
Inspect Signal Data

1. Click **Tools > Simulink Real-Time**.
2. In Simulink Real-Time Explorer, start the real-time application. The **Simulation Data Inspector** button glows in Simulink Editor, indicating that Simulation Data Inspector has data available for viewing.
3. Click the **Simulation Data Inspector** button.
4. In Simulation Data Inspector, select the signals `Integrator1:1` and `SignalGenerator:1`. Simulation Data Inspector displays plotted signal data.



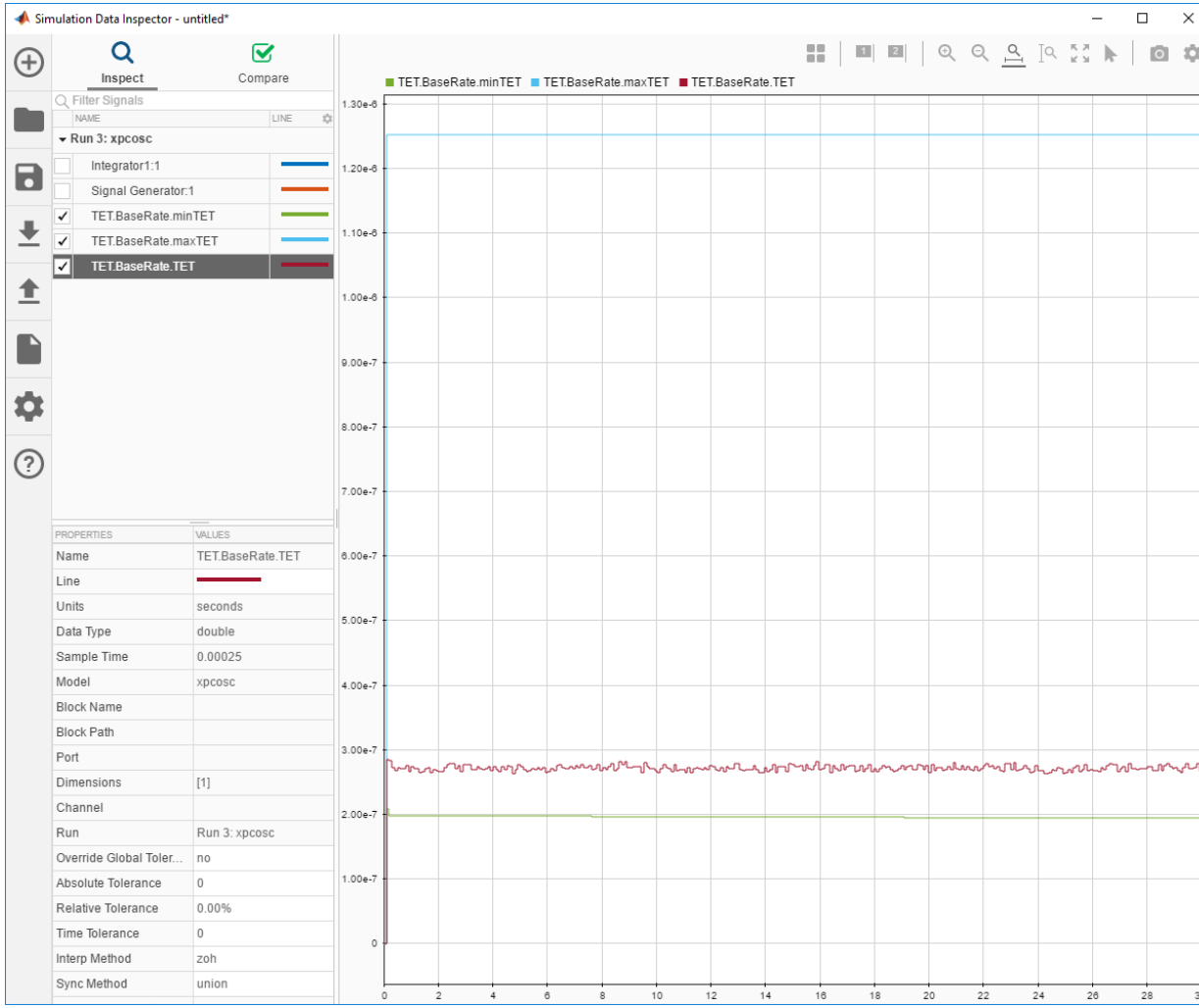
5. Stop the real-time application.

6. After the simulation, use the toolbar buttons to explore the data. For example, to view the simulation between seconds `0.02` and `0.04`, in Simulation Data Inspector, click the **Zoom in Time** button. Drag the cursor over the range from `0.02` to `0.04`.



Inspect TET Data

1. To view the TET data, clear `Integrator1:1` and `SignalGenerator:1`.
2. Select `TET.BaseRate.minTET`, `TET.BaseRate.maxTET`, and `TET.BaseRate.TET`.



3. To save the Simulation Data Inspector session as a `.mat` file, click **Save**.

See Also

`SimulinkRealTime.utils.TETMonitor.open`

More About

- “Trace or Log Data with Simulation Data Inspector” on page 6-82
- “Simulation Data Inspector in Your Workflow” (Simulink)

Trace or Log Data with Simulation Data Inspector

With Simulation Data Inspector (SDI) and Simulink Real-Time, you can trace signal data with data logging in immediate mode or log signal data with data logging in buffered mode. In immediate mode, you view the output in real time as the application produces it. The application can produce more data than the target computer can transmit in real time to the development computer. Data accumulates in the network buffer, and, if the buffer fills up, the kernel drops data points.

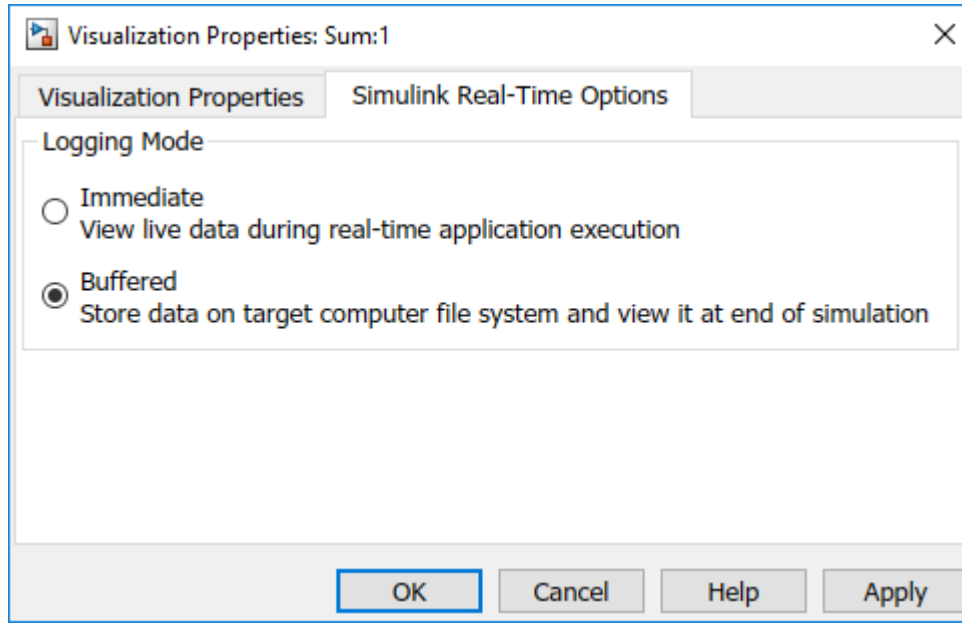
To avoid dropped data points caused by network buffer overruns, you can use buffered logging mode. In buffered mode, the kernel stores data for the buffered signals in a file on the target computer. At the end of execution, it transmits it to the development computer. You can then view the most important signals immediately and view the buffered signals afterward.

Buffered logging mode supports decimation and conditional block execution semantics. Some examples are logging buffered data by enabling data logging for a signal inside a for-iterator, function-call, or enabled/triggered subsystem. For more information about data logging features, see “Simulation Data Inspector in Your Workflow” (Simulink).

Set Up Model

1. Open `xpcosc`.
2. Right-click the `Mux` output signal and select **Log Selected Signals**.
3. Right-click the `Sum` output signal and select **Log Selected Signals**.
4. Right-click the `Sum` output badge ( ) and select **Properties**.

   Select **Logging Mode** to **Buffered**.



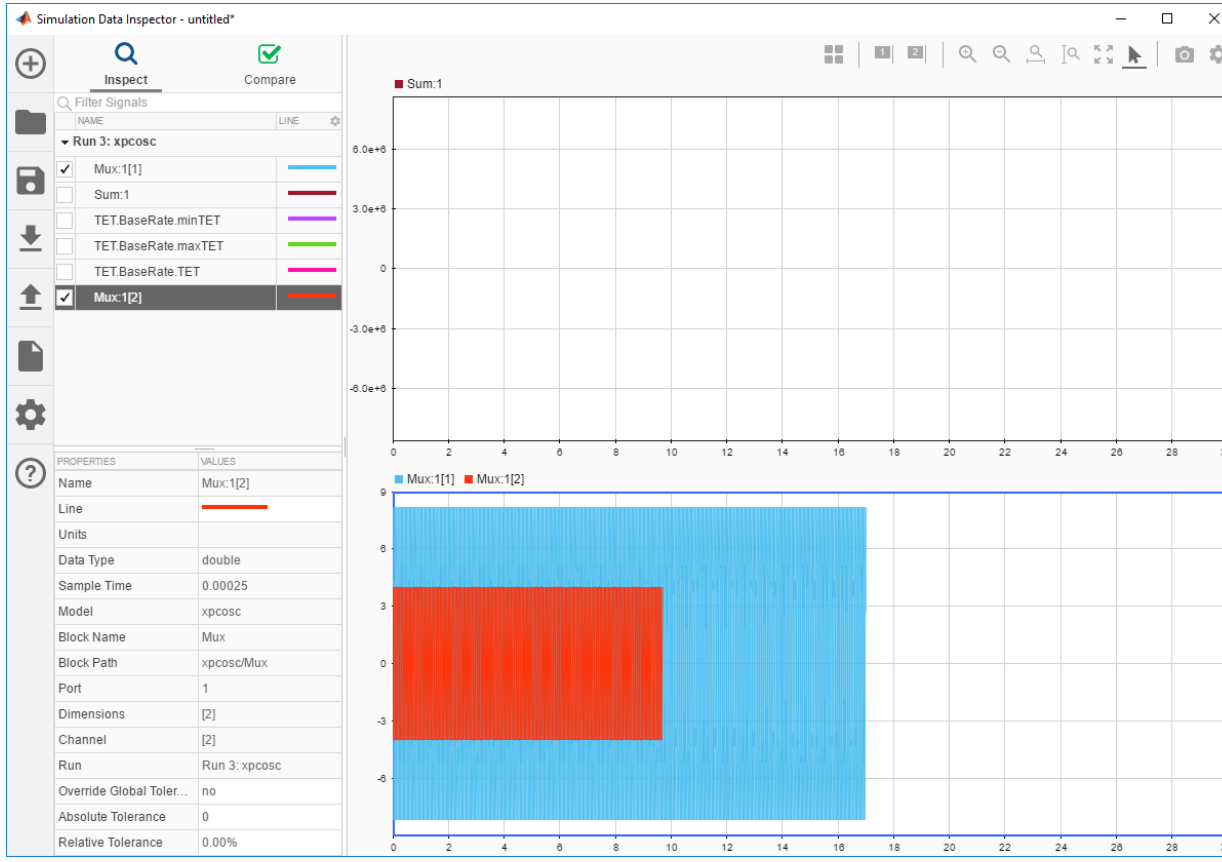
Set Up Simulation Data Inspector

1. Open Simulation Data Inspector ( ).
2. Click the **Layout** button ( ).
3. Select two horizontal displays.

View Simulation Data

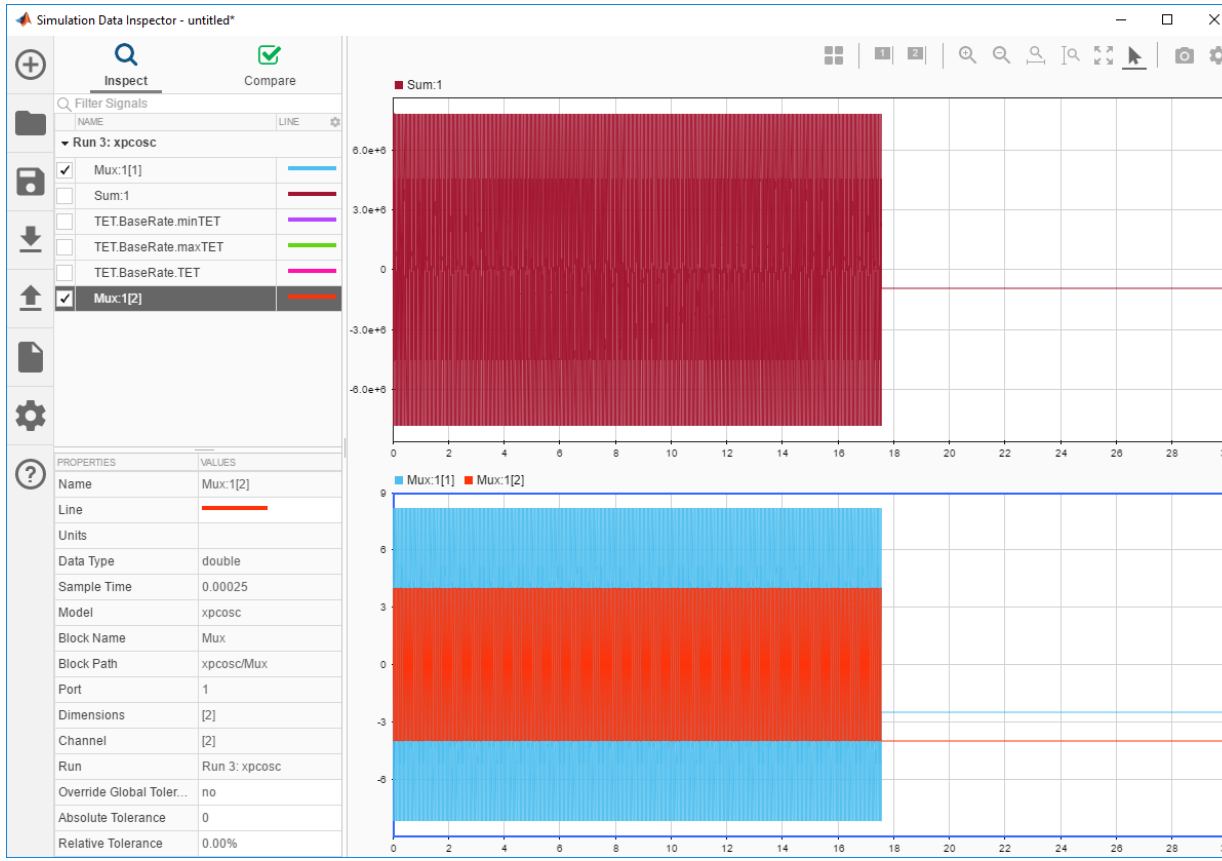
1. Build and download `xpcosc`.
2. Start real-time execution.
3. When the Simulation Data Inspector button changes to , click in the top display and select the `Sum` output signal.

   Click in the bottom display and select the `Mux` output signals.

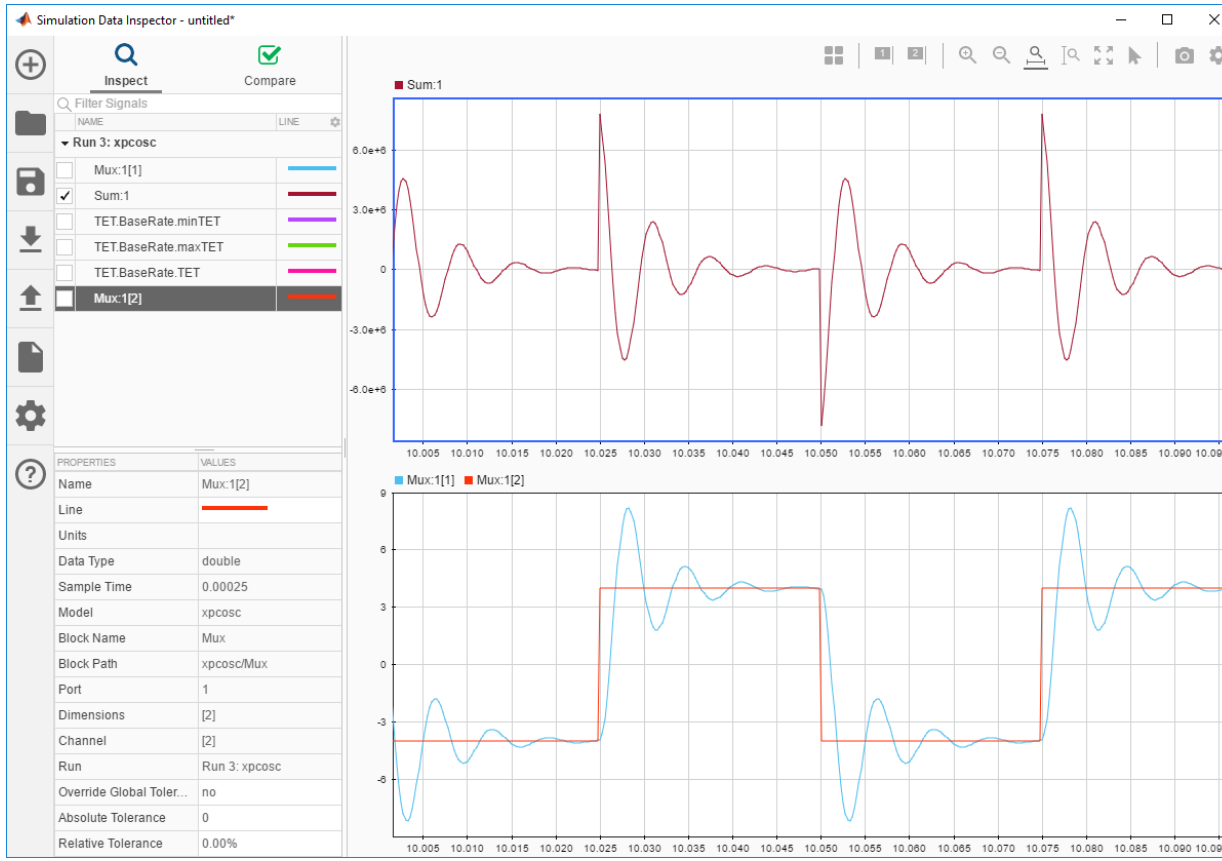


4 Stop real-time execution.

When the Sum output appears, click **Fit to View** ( ).



5 To zoom in on a time segment of interest, for example 10.0–10.1 s, click **Zoom in Time** ( ) and use the mouse and mouse wheel.



6 To save the Simulation Data Inspector session as a `.mat` file, click **Save**.

See Also

More About

- “Inspect Simulink® Real-Time™ Data with Simulation Data Inspector” on page 6-76
- “Simulation Data Inspector in Your Workflow” (Simulink)

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.

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. 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 scope blocks in the model
- File scopes created using Simulink Real-Time Explorer
- File scopes created using MATLAB language

See Also

`SimulinkRealTime.utils.bytes2file` | `SimulinkRealTime.utils.getFileScopeData`

More About

- “Configure File Scopes with Simulink Real-Time Explorer” on page 6-102
- “Log Signal Data with Outport Blocks and Simulink Real-Time Explorer” on page 6-110
- “Log Signal Data with Outport Block and MATLAB Language” on page 6-117
- “Troubleshoot Signals Not Accessible by Name” on page 6-184
- “Simulink Real-Time Scope Usage” on page 6-21

- “Target Scope Usage” on page 6-23
- “Host Scope Usage” on page 6-61
- “File Scope Usage” on page 6-90
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179

File Scope Usage

- Simulink Real-Time supports eight file scopes. Each file scope can contain as many signals as the target computer resources can support.
- You can have at most 128 files open on the target computer at the same time.
- The largest single file that you can create on the target computer 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 at most 12 characters: eight for the file name, one for the period, and at most three for the file extension. A file name longer than eight characters is truncated to six characters followed by `'~1'`.
- Do not write data to the `private` folder on your target computer. It is reserved for Simulink Real-Time internal use.
- 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 that is 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 starts collecting data into a memory buffer. A background task examines the buffer and writes data to the disk continuously, appending new data to the end of the file. When the scope reaches the number of samples that you specified, it starts collecting data again, overwriting the memory buffer. If the background task cannot keep pace with data collection, data can be lost.
  - Autorestart is off — When the scope triggers, the scope starts collecting data into a memory buffer. It stops when it has collected the number of samples that you specified. A background task examines the buffer and writes data to the disk continuously, appending the new data to the end of the file.
- 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 Scope block 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

`File System` | `SimulinkRealTime.fileSystem.mkdir` | `SimulinkRealTime.utils.bytes2file` | `SimulinkRealTime.utils.getFileScopeData`

More About

- “Configure Real-Time File Scope Blocks” on page 6-93
- “Create File Scopes with Simulink Real-Time Explorer” on page 6-98
- “Log Signal Data into Multiple Files” on page 6-106
- “Simulink Real-Time Scope Usage” on page 6-21
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179
- “Using SimulinkRealTime.fileSystem Objects” on page 12-5

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 in the target computer file system. 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 uses the example model `ex_slrt_rt_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_rt_osc')))`).

1. In the Command Window, open `ex_slrt_rt_osc`.
2. In the Simulink Editor, double-click the block labeled Scope.

   The Scope block 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 that is 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 starts collecting data into a memory buffer. A background task examines the buffer and writes data to the disk continuously, appending new data to the end of the file. When the scope reaches the number of samples that you specified, it starts collecting data again, overwriting the memory buffer. If the background task cannot keep pace with data collection, data can be lost.
- Autorestart is off — When the scope triggers, the scope starts collecting data into a memory buffer. It stops when it has collected the number of samples that you specified. A background task examines the buffer and writes data to the disk continuously, appending the new data to the end of the file.

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 how often data is collected, in units of sample time. The value `1` indicates that data is collected at each sample time. Values of `2` or more indicates that data is collected at less than every sample time.

9 From the **Trigger mode** list, select one of the following:

From the **Trigger mode** list, select one of the following:

- `FreeRun` or `Software Triggering` — No extra parameters.
- `Signal Triggering` — enter 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`.
- `Scope Triggering` — enter 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` causes the triggering scope and the triggered scope to 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 at most 12 characters: eight for the file name, one for the period, and at most three for the file extension. A file name longer than eight characters is truncated to six characters followed by `'~1'`.

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.

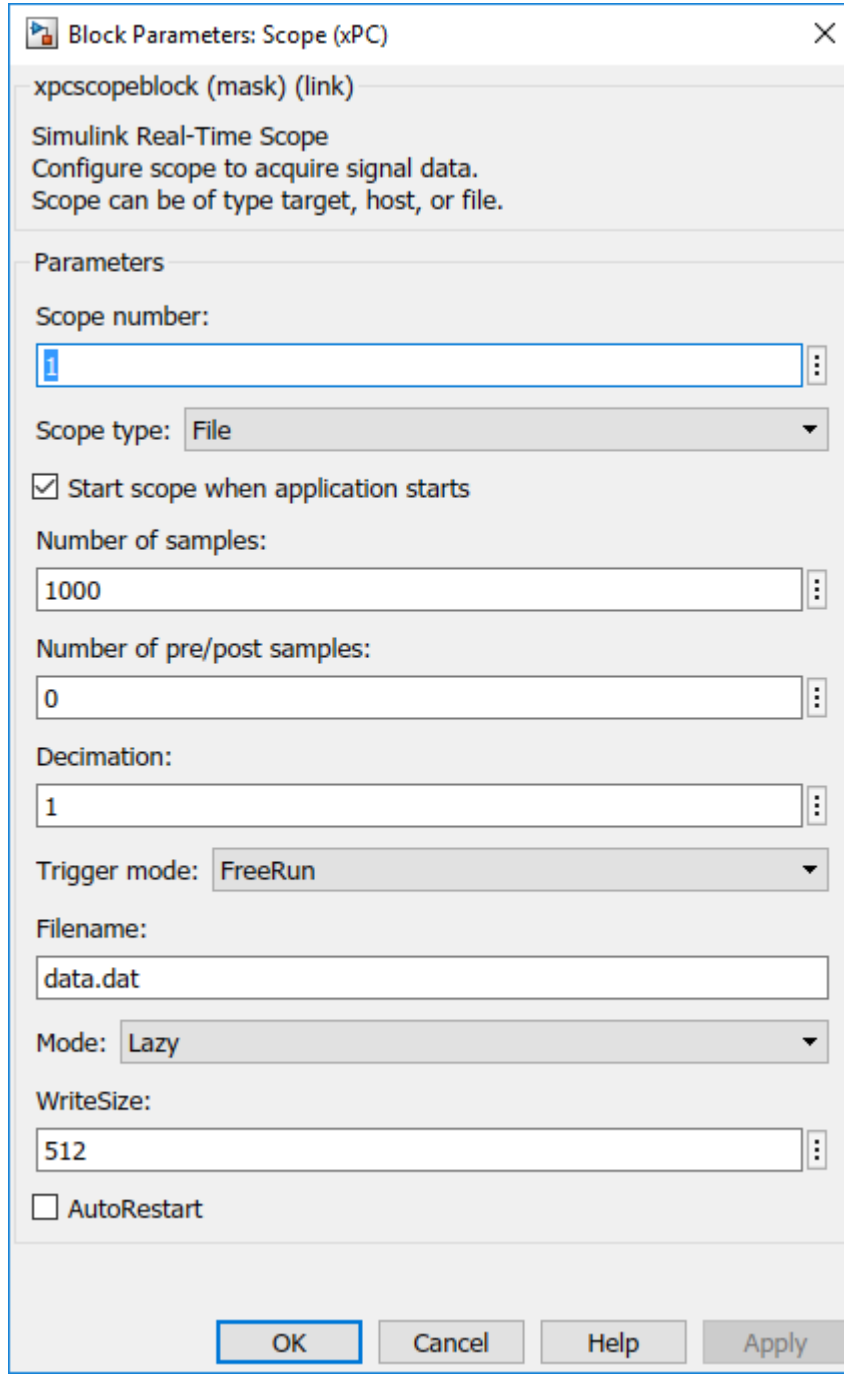
13 To have the file scope collect data up to **Number of samples** and then start over again reading new data, select the **AutoRestart** check box.

To have the file scope collect data up to **Number of samples** and then stop, clear the **AutoRestart** check box.

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.


Block Parameters: Scope (xPC)
xpcscopeblock (mask) (link)
Simulink Real-Time Scope
Configure scope to acquire signal data.
Scope can be of type target, host, or file.
Parameters
Scope number:
1
Scope type: File
Start scope when application starts
Number of samples:
1000
Number of pre/post samples:
0
Decimation:
1
Trigger mode: FreeRun
Filename:
data.dat
Mode: Lazy
WriteSize:
512
AutoRestart
OK
Cancel
Help
Apply

14 Click **OK**.

15 From the **File** menu, click **Save As**.

Save the model as `ex_slrt_file_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_file_osc')))`).

See Also

Scope | `SimulinkRealTime.utils.bytes2file` | `SimulinkRealTime.utils.getFileScopeData`

More About

- “Simulink Real-Time Scope Usage” on page 6-21
- “File Scope Usage” on page 6-90
- “Trigger One Scope with Another Scope” on page 11-19

Create File Scopes with Simulink Real-Time Explorer

You can create a 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 6-93 and “File Scope Usage” on page 6-90.

This procedure uses the model `xpcosc`. 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 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 To have the file scope collect data up to **Number of samples** and then start over again reading new data, select the **AutoRestart** check box.

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 that is 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.

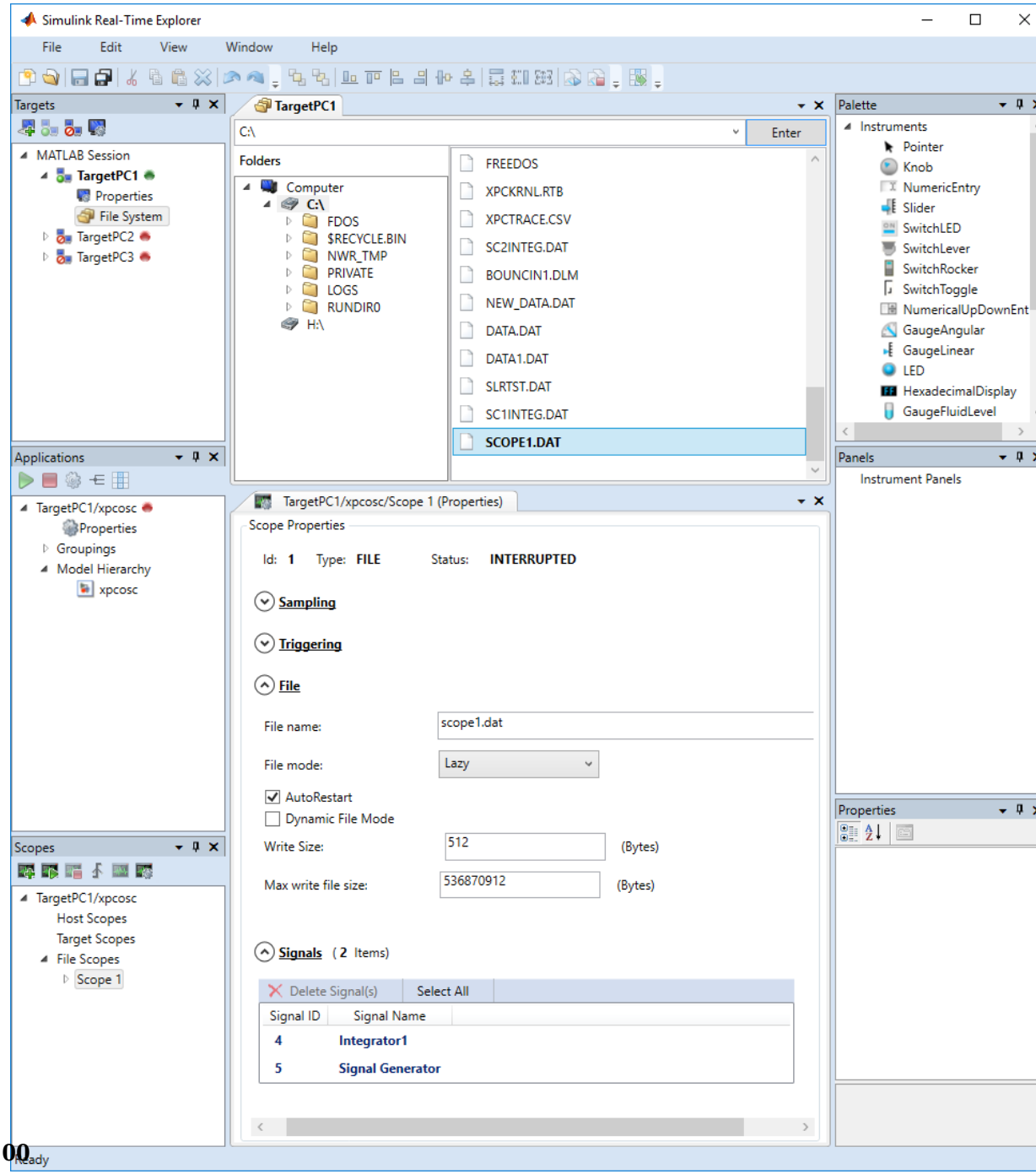
15 To stop **Scope 1**, click **Scope 1** in the **Scopes** pane, and then click the **Stop Scope** button on the toolbar.

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.


Simulink Real-Time Explorer
File
Edit
View
Window
Help
Targets
MATLAB Session
TargetPC1
Properties
File System
TargetPC2
TargetPC3
TargetPC1
C:\
Enter
Folders
Computer
C:\
FDOS
$RECYCLE.BIN
NWR_TMP
PRIVATE
LOGS
RUNDIR0
H:\
FREEDOS
XPCKRNL.RTB
XPCTRACE.CSV
SC2INTEG.DAT
BOUNCIN1.DLM
NEW_DATA.DAT
DATA.DAT
DATA1.DAT
SLRTST.DAT
SC1INTEG.DAT
SCOPE1.DAT
Palette
Instruments
Pointer
Knob
NumericEntry
Slider
SwitchLED
SwitchLever
SwitchRocker
SwitchToggle
NumericalUpDownEnt
GaugeAngular
GaugeLinear
LED
HexadecimalDisplay
GaugeFluidLevel
Applications
TargetPC1/xpcosc
Properties
Groupings
Model Hierarchy
xpcosc
Panels
Instrument Panels
TargetPC1/xpcosc/Scope 1 (Properties)
Scope Properties
Id: 1 Type: FILE Status: INTERRUPTED
Sampling
Triggering
File
File name:
scope1.dat
File mode:
Lazy
AutoRestart
Dynamic File Mode
Write Size:
512
(Bytes)
Max write file size:
536870912
(Bytes)
Signals (2 Items)
Delete Signal(s)
Select All
Signal ID
Signal Name
4
Integrator1
5
Signal Generator
Properties
Scopes
TargetPC1/xpcosc
Host Scopes
Target Scopes
File Scopes
Scope 1
Ready

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 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 the workspace 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.utils.bytes2file` | `SimulinkRealTime.utils.getFileScopeData`

More About

- “Log Signal Data into Multiple Files” on page 6-106
- “Configure File Scopes with Simulink Real-Time Explorer” on page 6-102
- “Using SimulinkRealTime.fileSystem Objects” on page 12-5
- “Create Signal Groups with Simulink Real-Time Explorer” on page 6-58
- “Configure Scope Sampling with Simulink Real-Time Explorer” on page 6-36
- “Trigger Scopes with Simulink Real-Time Explorer” on page 6-40
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179

Configure File Scopes with 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`. You must have already completed the procedure in “Create File Scopes with Simulink Real-Time Explorer” on page 6-98. 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 at most 12 characters: eight for the file name, one for the period, and at most three for the file extension. A file name longer than eight characters is truncated to six characters followed by `'~1'`.

   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.

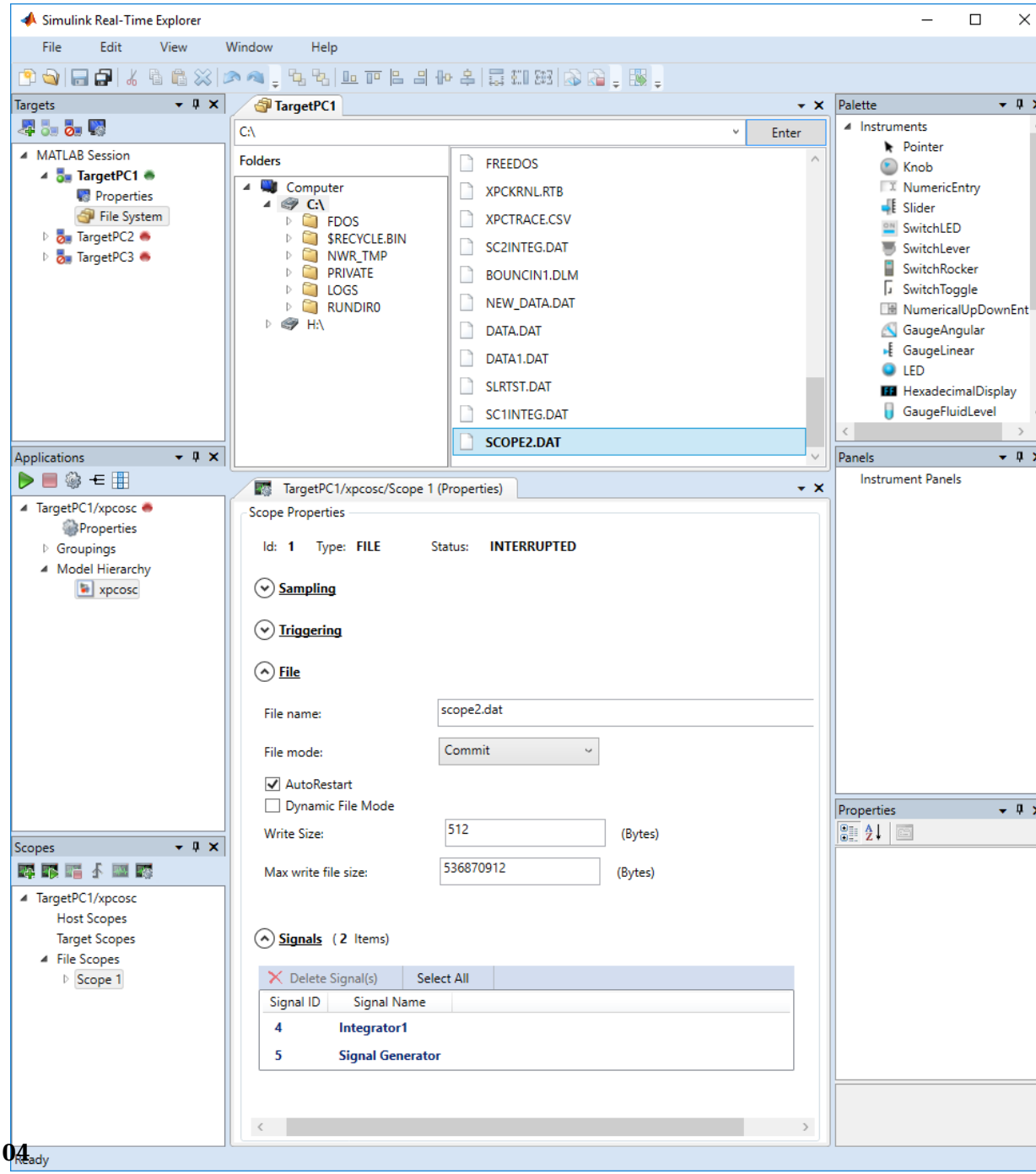
   - In `Commit` mode, the FAT entry and the actual file size are updated during each file write operation.
   - In `Lazy` mode, the FAT entry and the actual file size are updated only when the file is closed and not during each file write operation.

`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. To have the file scope collect data up to **Number of samples** and then start over again reading new data, select the **AutoRestart** check box.
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 that is 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).


Simulink Real-Time Explorer
File
Edit
View
Window
Help
Targets
MATLAB Session
TargetPC1
Properties
File System
TargetPC2
TargetPC3
TargetPC1
C:\
Enter
Folders
Computer
C:\
FDOS
$RECYCLE.BIN
NWR_TMP
PRIVATE
LOGS
RUNDIR0
H:\
FREEDOS
XPCKRNL.RTB
XPCTRACE.CSV
SC2INTEG.DAT
BOUNCIN1.DLM
NEW_DATA.DAT
DATA.DAT
DATA1.DAT
SLRTST.DAT
SC1INTEG.DAT
SCOPE2.DAT
Palette
Instruments
Pointer
Knob
NumericEntry
Slider
SwitchLED
SwitchLever
SwitchRocker
SwitchToggle
NumericalUpDownEnt
GaugeAngular
GaugeLinear
LED
HexadecimalDisplay
GaugeFluidLevel
Applications
TargetPC1/xpcosc
Properties
Groupings
Model Hierarchy
xpcosc
TargetPC1/xpcosc/Scope 1 (Properties)
Scope Properties
Id: 1 Type: FILE Status: INTERRUPTED
Sampling
Triggering
File
File name:
scope2.dat
File mode:
Commit
AutoRestart
Dynamic File Mode
Write Size:
512
(Bytes)
Max write file size:
536870912
(Bytes)
Signals (2 Items)
Delete Signal(s)
Select All
Signal ID
Signal Name
4
Integrator1
5
Signal Generator
Panels
Instrument Panels
Properties
Scopes
TargetPC1/xpcosc
Host Scopes
Target Scopes
File Scopes
Scope 1
Ready

13 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.

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` | `SimulinkRealTime.utils.bytes2file` | `SimulinkRealTime.utils.getFileScopeData`

More About

- “Log Signal Data into Multiple Files” on page 6-106
- “Using SimulinkRealTime.fileSystem Objects” on page 12-5
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179

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 with Simulink Real-Time Explorer” on page 6-98.

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 nine log files, `scope1_1.dat` to `scope1_9.dat`, on the target computer file system.

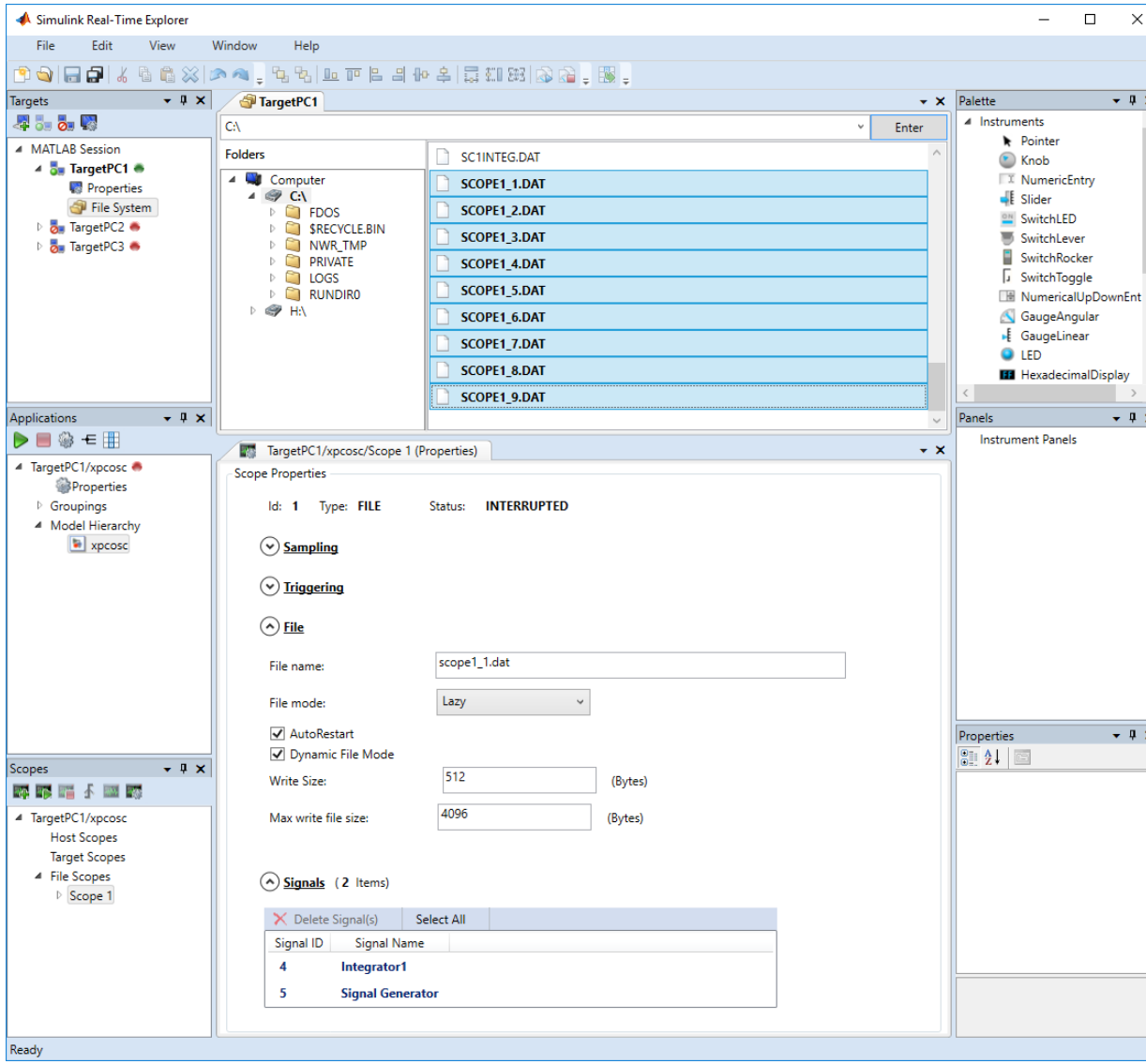
   You can configure the file scope to create up to 99999999 files (`<%%%%%%%%>.dat`). The length of a file name, including the specifier, cannot exceed eight characters.
8. In the **Max write file size** box, enter a value to limit the size of the signal log files. This value must be a multiple of the **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 that is 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

`File System` | `SimulinkRealTime.utils.bytes2file` | `SimulinkRealTime.utils.getFileScopeData`

More About

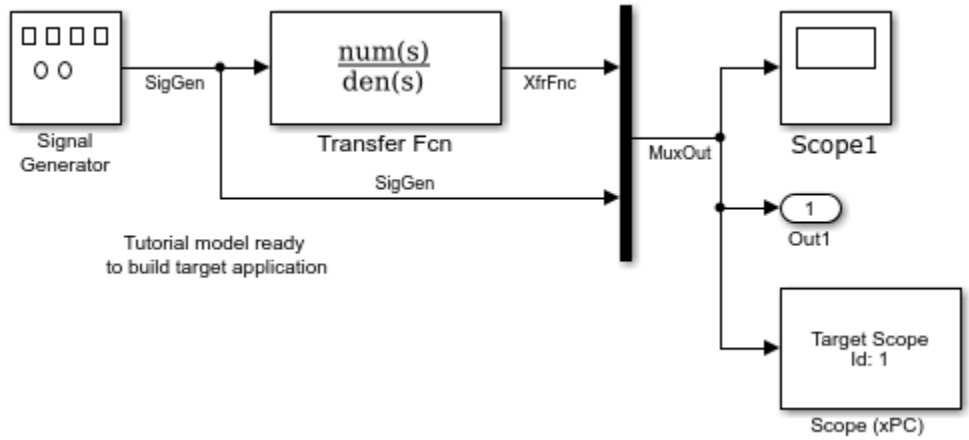
- “File Scope Usage” on page 6-90
- “Using SimulinkRealTime.fileSystem Objects” on page 12-5
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179

Log Signal Data with Outport Blocks 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 model `ex_slrt_rt_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_rt_osc')))`). The final configured model is `ex_slrt_outport_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', '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 6-111

In this section...
“Configure the Model for Data Logging” on page 6-112
“Log the Data” on page 6-112
“Download and Plot the Data” on page 6-113

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 **Monitor 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 Simulation 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, check 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, check that the **Monitor Task Execution Time** parameter 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` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_outport_osc')))`). Click **Save**.
6. Click **OK**.

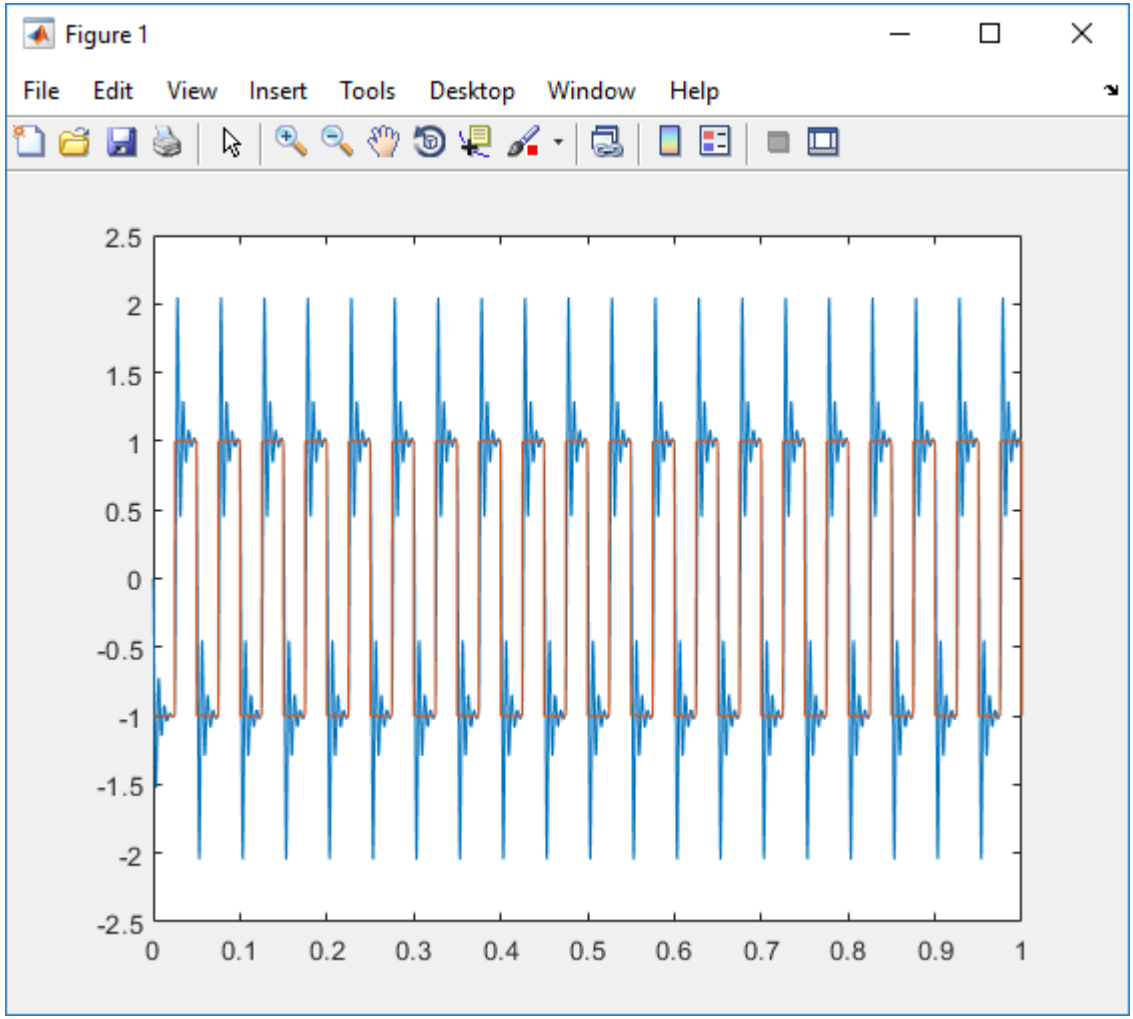
Log the Data

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

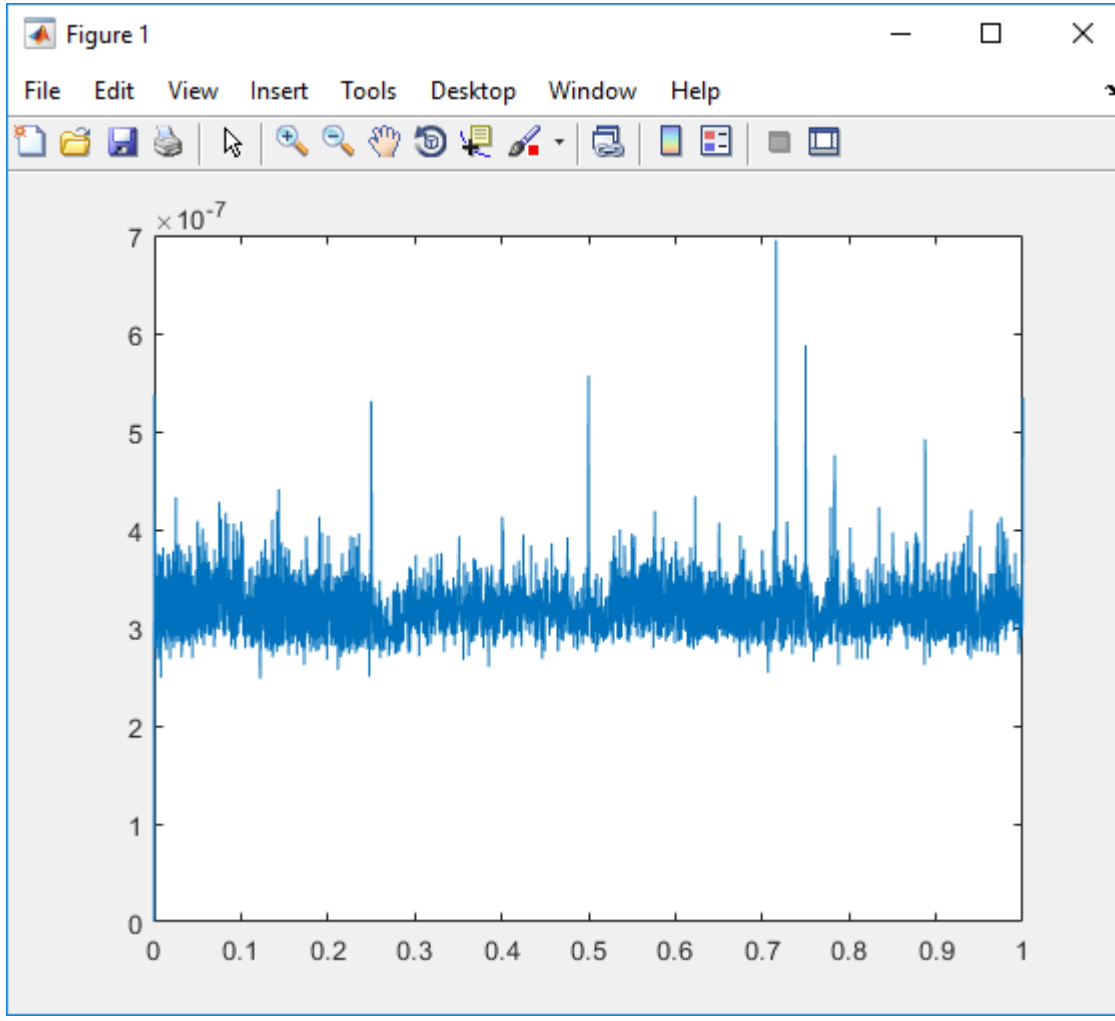
1 Download and plot the logged times and output values from the target computer. In the Command Window, type:

```
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.

Task execution time (TET) measures how long it takes the kernel to run for one base-rate time step. For a multirate model, use the profiler to find out what the execution time is for each rate.

See Also

Real-Time Application Properties | `SimulinkRealTime.target.getlog`

More About

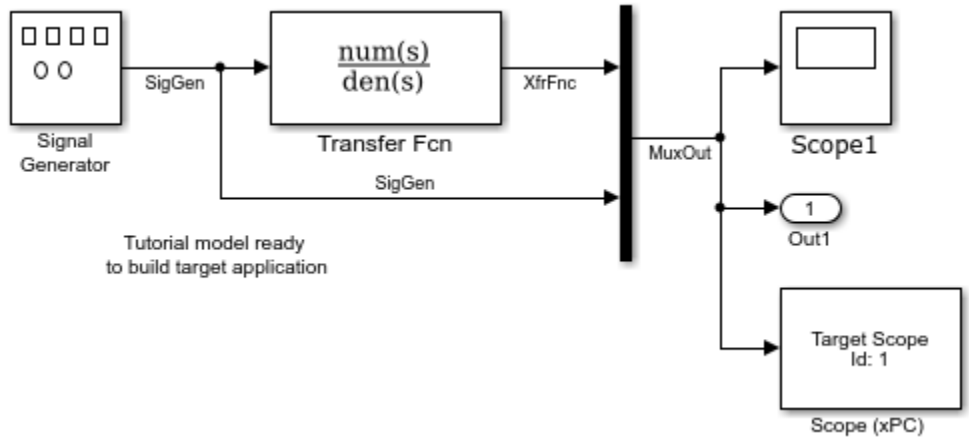
- “Simulate Simulink Model with MATLAB Language”
- “Log Signal Data with Outport Block and MATLAB Language” on page 6-117
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179
- “Signal Logging Buffer Size” on page 6-124

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 model `ex_slrt_rt_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_rt_osc')))`). The final configured model is `ex_slrt_outport_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', '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 6-118
“Configure the Model for Data Logging” on page 6-119

In this section...
“Log the Data” on page 6-120 “Download and Plot the Data” on page 6-120

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 Simulation 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. Check that signal data and task execution time are being logged.

```
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. Check 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);
   ```

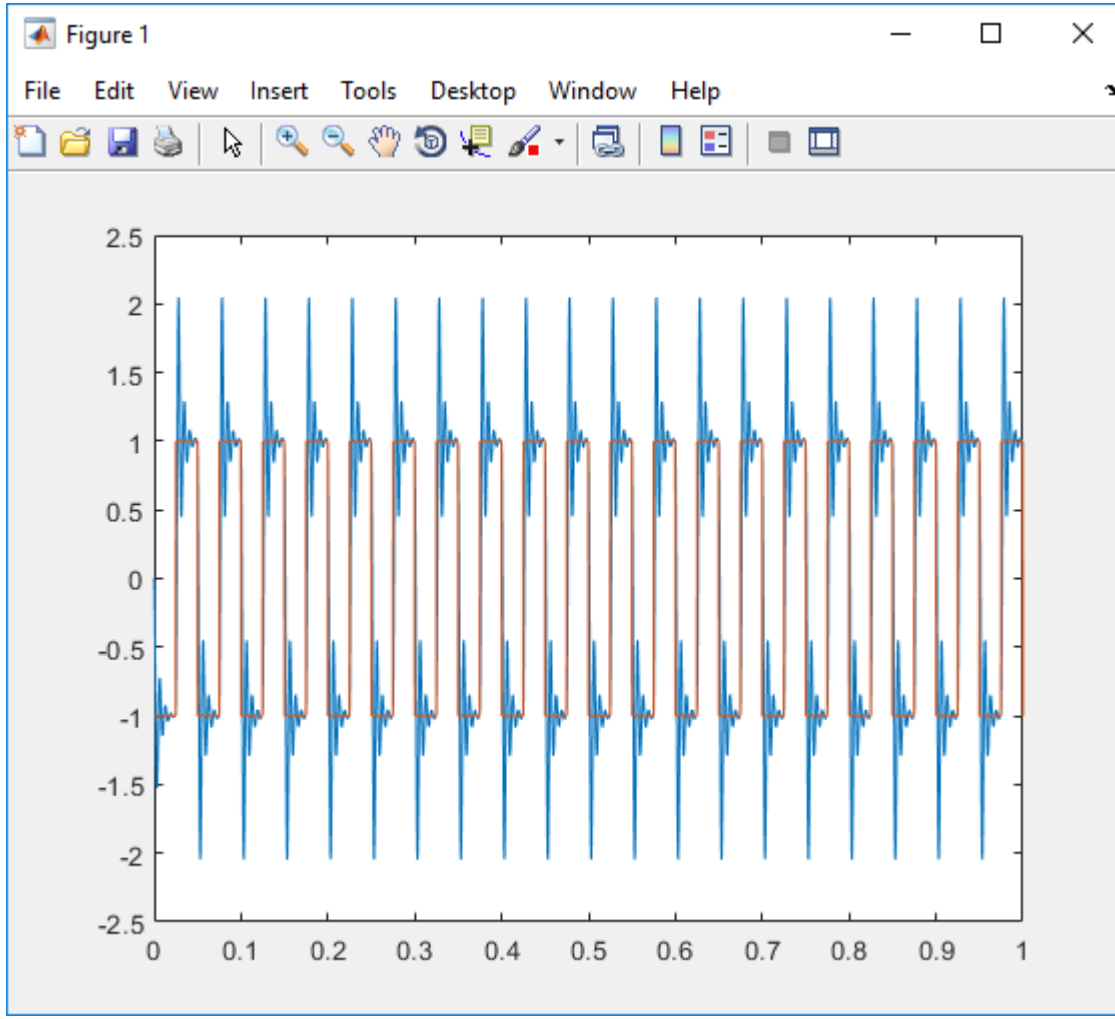
3. Stop execution.

   ```
   stop(tg);
   ```

Download and Plot the Data

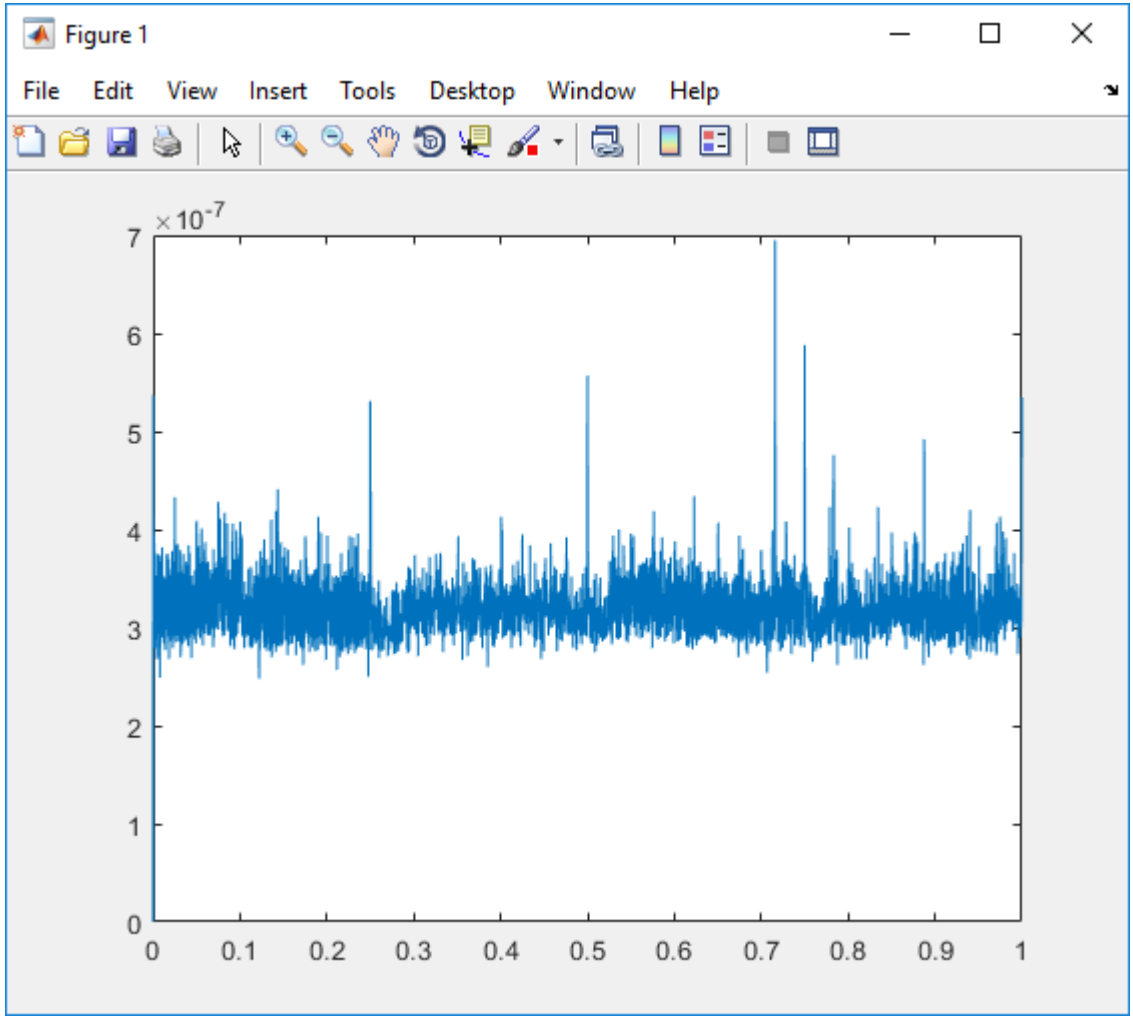
1. Download and plot the logged times and output values from the target computer. In the Command Window, type:

   ```
   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.

Task execution time (TET) measures how long it takes the kernel to run for one base-rate time step. For a multirate model, use the profiler to find out what the execution time is for each rate.

See Also

Real-Time Application Properties | `SimulinkRealTime.target.getlog`

More About

- “Simulate Simulink Model with MATLAB Language”
- “Log Signal Data with Outport Blocks and Simulink Real-Time Explorer” on page 6-110
- “Signal Logging Buffer Size” on page 6-124

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 with MATLAB Language

This procedure shows how to trace signals with file scopes using the Simulink model `xpcosc`. 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 that is 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

`SimulinkRealTime.fileSystem` | `SimulinkRealTime.utils.bytes2file` | `SimulinkRealTime.utils.getFileScopeData` | `plot`

More About

- “Using SimulinkRealTime.fileSystem Objects” on page 12-5
- “Monitor Signals with MATLAB Language” on page 6-9

Tune Parameters with 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.

This procedure uses the model `xpcosc`.

Set Up Host Scope

Before tuning parameters, do the following:

1. Build and download model `xpcosc` to the target computer with Simulink (on the toolbar).
2. Run Simulink Real-Time Explorer (**Tools > Simulink Real-Time**).
3. Connect to the target computer that is in the **Targets** pane (on the toolbar).
4. Set property **Stop time** to `inf` in the **Applications** pane (on the toolbar).
5. In the **Scopes** pane, expand the **xpcosc** node.
6. To add a host scope, select **Host Scopes**, and then click the **Add Scope** button on the toolbar.
7. In the **Applications** pane, expand the real-time application node, and then the node **Model Hierarchy**.
8. Double-click the **xpcosc** node.
9. To add signal `Signal Generator` to **Scope1**, drag signal `Signal Generator` from the **xpcosc** signal list to the **Scope1** properties workspace.

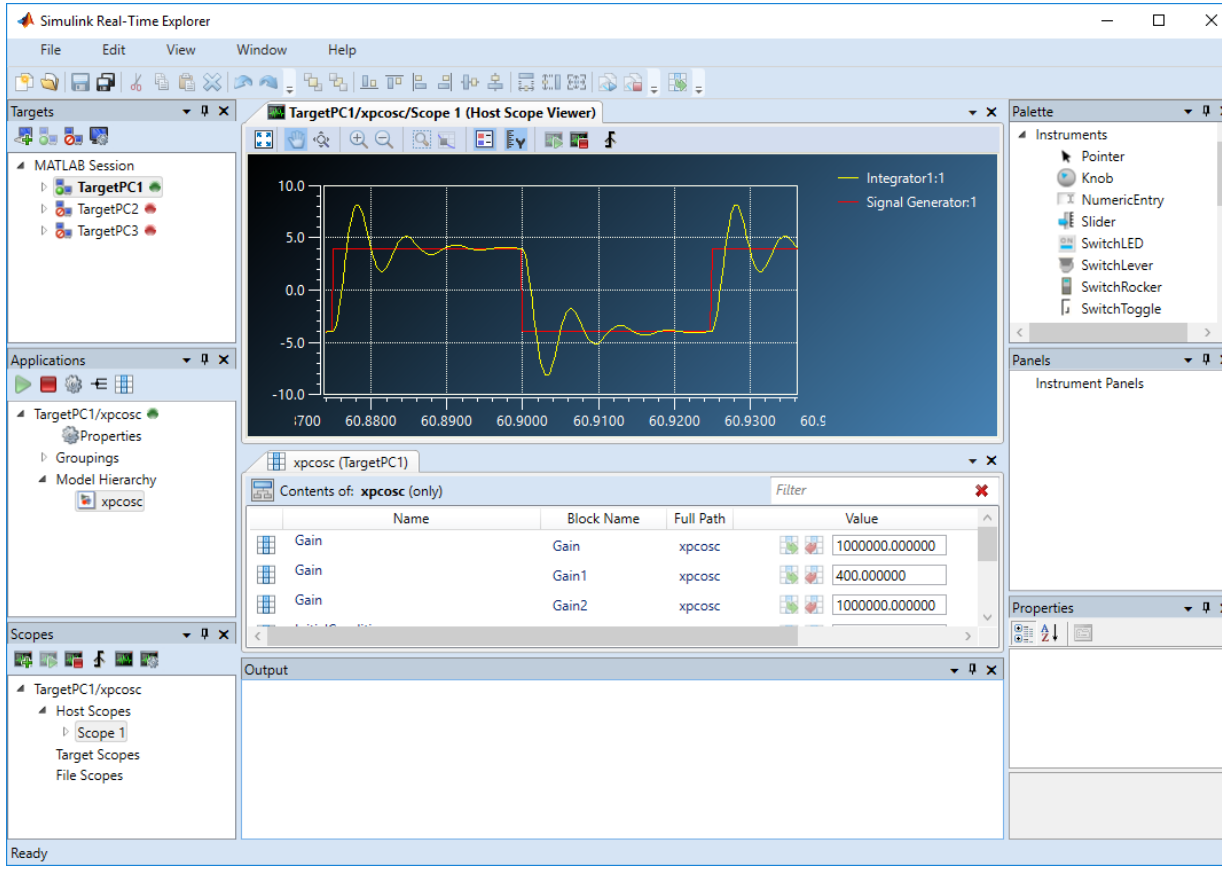
   Add signal `Integrator1` to **Scope1** in the same way.
10. Expand **Scope 1**, and then click the **Properties** button on the toolbar.

    To display the host scope signals, in the **Scope Properties** pane, click **Signals**.
11. To open the host scope display, select **Scope 1**, and then click the **View Scope** button on the toolbar.

See “Create Host Scopes with Simulink Real-Time Explorer” on page 6-66.

Initial Values

1. To view the initial parameter values, in the **Applications** pane, expand both the real-time application node and node **Model Hierarchy**.
2. Select the model node, and then click the **View Parameters** button on the toolbar.
3. Start **Scope 1**(on the toolbar).
4. Start execution (on the toolbar).



Updated Values

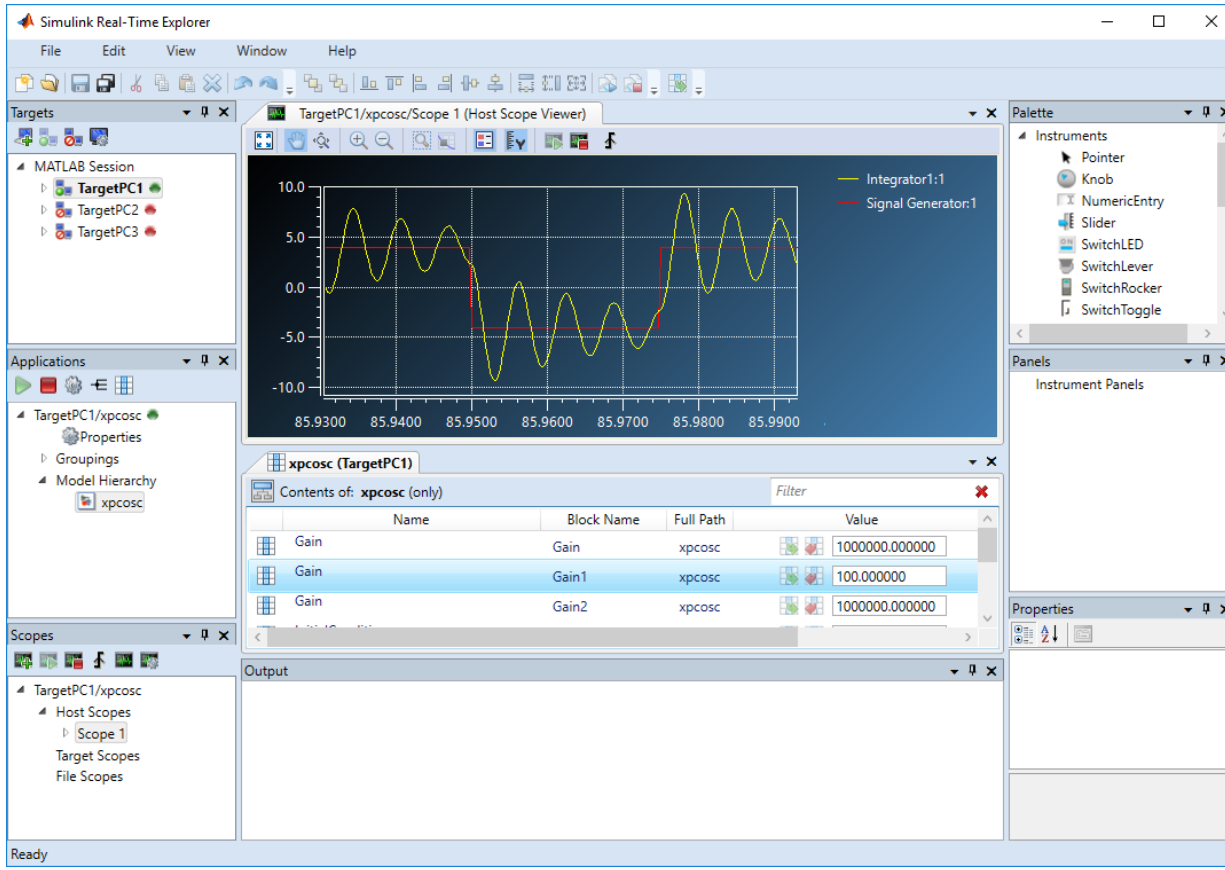
To update a parameter value:

1. In the **Applications** pane, expand both the real-time application node and node **Model Hierarchy**.
2. 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.
3. To change the value of the `Gain` for block `Gain1` to `100`, type `100` into the **Value** box, and then press **Enter**.

   To revert the `Gain` for block `Gain1` to its previous value, click the **Revert** button .
4. Click the **Apply parameter value(s) changes** button .

   Simulink Real-Time Explorer looks like this figure.



5 Stop **Scope 1** (on the toolbar).

6 Stop execution (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 appears in the middle of the dialog box. Continue to drag the workspace 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

More About

- “Create Host Scopes with Simulink Real-Time Explorer” on page 6-66
- “Configure Real-Time Host Scope Blocks” on page 6-62
- “Create Parameter Groups with Simulink Real-Time Explorer” on page 6-134
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179
- “Troubleshoot Parameters Not Accessible by Name” on page 6-186

Create Parameter Groups with 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.

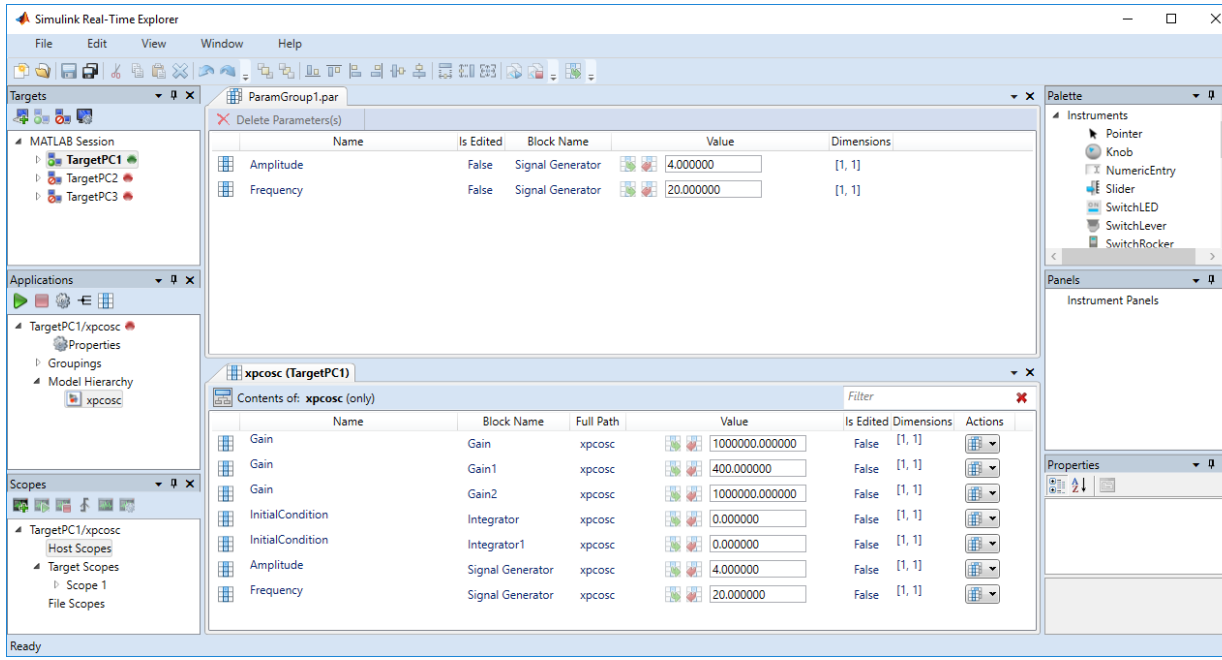
This procedure uses the model `xpcosc`. You must have already completed the following setup:

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 the workspace 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

More About

- “Tune Parameters with Simulink Real-Time Explorer” on page 6-129
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179

Tune Parameters with 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`. 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
```

See Also

More About

- “Troubleshoot Parameters Not Accessible by Name” on page 6-186

Tune Parameters with 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`. 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 MATLAB variable 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)
```

See Also

More About

- “Troubleshoot Parameters Not Accessible by Name” on page 6-186

Save and Reload Parameters with 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 6-142
“Load Saved Parameters to a Real-Time Application” on page 6-143
“List Parameter Values Stored in a File” on page 6-143

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` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', '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` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', '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 6-142).

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 Parameter Values 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 6-142).

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.

See Also

More About

- “Load a parameter set from a file on the designated target file system”
- “Tune Parameters with Simulink Real-Time Explorer” on page 6-129
- “Tune Parameters with MATLAB Language” on page 6-137
- “Tune Parameters with Simulink External Mode” on page 6-140

Tunable Block Parameters and Tunable Global Parameters

To change the behavior of a real-time application, you can tune Simulink Real-Time tunable parameters. In Simulink external mode, you can change the parameters directly in the block or indirectly by using MATLAB variables to create tunable global parameters. Simulink Real-Time Explorer and MATLAB language enable you to change parameter values and MATLAB variables as your real-time application is executing.

Note Simulink Real-Time does not support parameters of multiword data types.

Tunable Parameters

Simulink Coder defines two kinds of parameters that can be modified during execution: tunable block parameters and tunable global parameters.

Tunable Block Parameters

A tunable block parameter is a literal expression that you reference in a Simulink block dialog box.

Suppose that you assign the value `5/2` to the **Amplitude** parameter of a Signal Generator block. **Amplitude** is a tunable parameter.

Tunable Global Parameter

A tunable global parameter is a variable in the MATLAB workspace that you reference in a Simulink block dialog box.

Suppose that you enter `A` in the **Amplitude** parameter of a Signal Generator block. Variable `A` is a tunable parameter.

You can tune the values of MATLAB variables that are grouped in a parameter structure. For example:

1. Assign a parameter structure that contains the field `Ampl` to variable `A`.
2. Enter `A.Ampl` in the **Amplitude** parameter of a Signal Generator block.
3. Change the amplitude of the signal generator by tuning the value of `A.Ampl` in the MATLAB workspace during simulation.

Inlined Parameters

To optimize execution efficiency, you can change the **Default parameter behavior** option from `Tunable` to `Inlined` on the **Code GenerationOptimization** pane.

You cannot tune inlined block parameters. You can define a tunable global parameter or `Simulink.Parameter` object, enter it in the parameter in the block dialog box, and tune the MATLAB variable or object.

For more information about inlined parameters, see “Default parameter behavior” (Simulink Coder).

Tuning in External Mode

In external mode, Simulink Real-Time connects your Simulink model to your real-time application. The block diagram becomes a user interface for the real-time application.

You can change a block parameter value during execution in the block dialog box. When you click **OK**, Simulink transfers the new value to the real-time application.

You can also change a tunable global parameter during execution by assigning a new value to the MATLAB workspace. You must then explicitly command Simulink to transfer the data by pressing **Ctrl+D** or clicking **Simulation > Update Diagram**.

Tuning with Simulink Real-Time Explorer

During real-time execution, Simulink Real-Time Explorer becomes a user interface for the real-time application.

To access a block parameter value, navigate to the block in the Explorer model hierarchy. You can change the value in a text entry box in the parameter window. When you apply the new value, Simulink Real-Time transfers the new value to the real-time application.

You can access a tunable global parameter at the top level of the model hierarchy. Change it the same way as you would a tunable block parameter.

You can also use Simulink Real-Time Explorer instrument panels to tune block parameters and global parameters.

Tuning with MATLAB Language

To change the values of tunable block parameters and tunable global parameters during execution, use the Simulink Real-Time command `setparam`. The following code examples use the model `xpcosc`.

To change a block parameter value, use a nonempty block path and the parameter name. For example, to change the amplitude of the signal generator:

```
tg = slrt;
setparam(tg, 'Signal Generator', 'Amplitude', 4.57)
```

To change a tunable global parameter, use the variable name. For example, to change the amplitude of the signal generator via the parameter structure field `A.Ampl`:

```
tg = slrt;
setparam(tg, 'A.Ampl', 4.57)
```

See Also

`SimulinkRealTime.target.getparam` | `SimulinkRealTime.target.setparam`

More About

- “Tune Inlined Parameters with Simulink Real-Time Explorer” on page 6-148
- “Default parameter behavior” (Simulink Coder)
- “Specify Source for Data in Model Workspace” (Simulink)
- “Troubleshoot Parameters Not Accessible by Name” on page 6-186
- “Tune and Experiment with Block Parameter Values” (Simulink)
- “Share and Reuse Block Parameter Values by Creating Variables” (Simulink)
- “How Generated Code Stores Internal Signal, State, and Parameter Data” (Simulink Coder)
- “Configure Data Accessibility for Rapid Prototyping” (Simulink Coder)

Tune Inlined Parameters with 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.

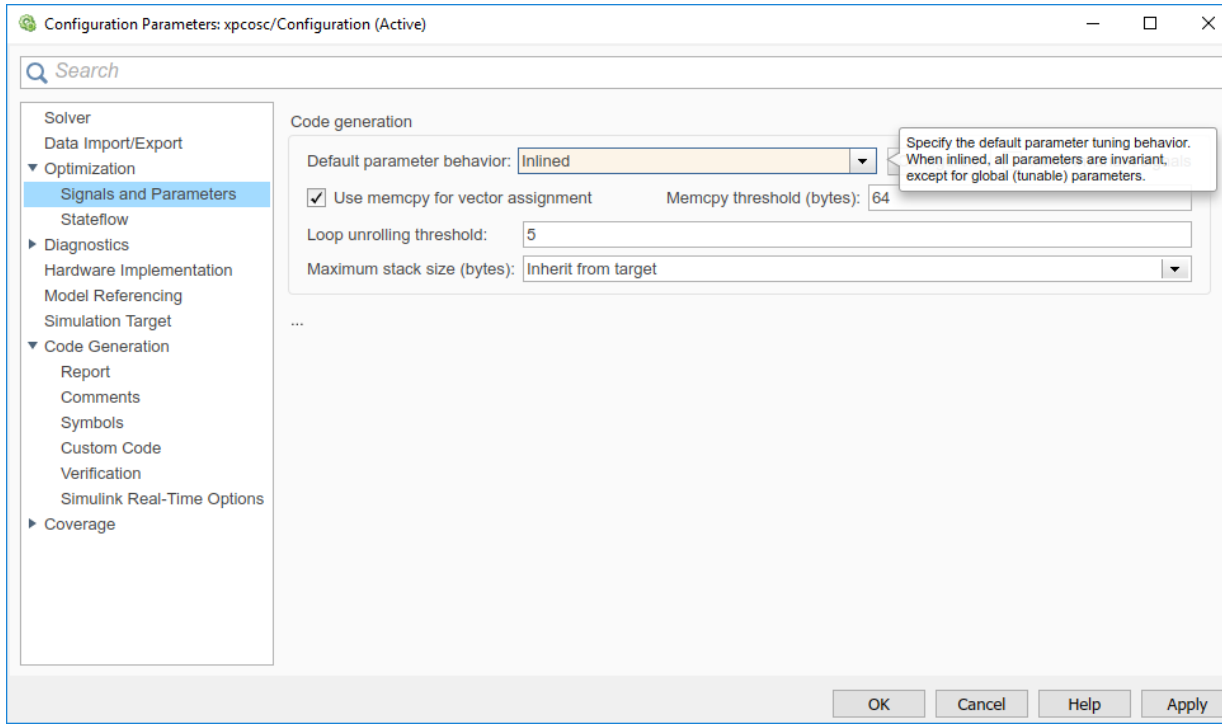
The following procedure starts with the Simulink model `xpcosc` and produces the model `ex_slrt_inlined_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_inlined_osc')))`).

Configure Model to Tune Inlined Parameters

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
   ```

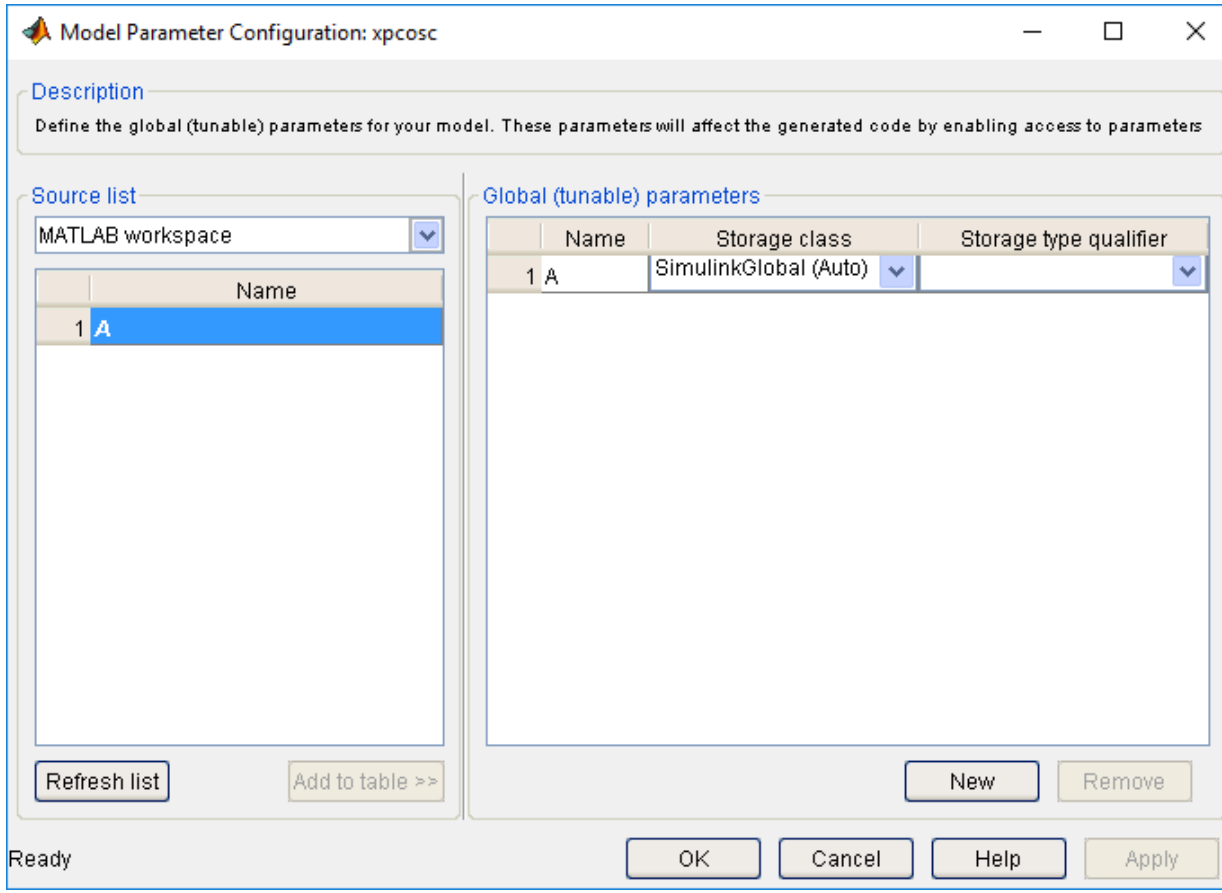
   The value is displayed in the MATLAB workspace.
5. From the Simulink Editor, click **Simulation** > **Model Configuration Parameters**.
6. In the Configuration Parameters dialog box, select the **Signals and Parameters** node under **Optimization**.
7. In the right pane, set **Default parameter behavior** 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**.

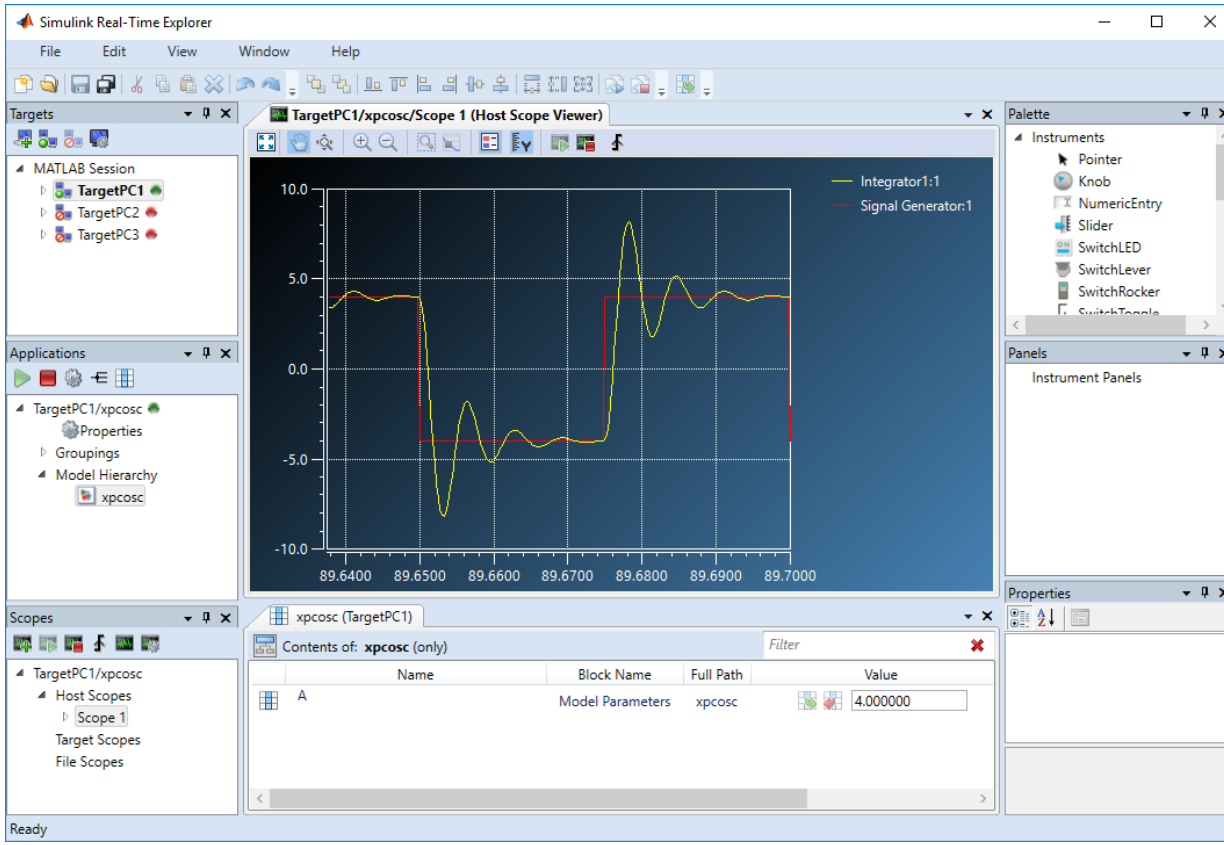
11 In the Configuration Parameters dialog box, click **Apply**, and then **OK**.

12 Save the model. For example, save it as `ex_slrt_inlined_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_inlined_osc')))`).

13 Build and download the model to your target computer.

Initial Value

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.



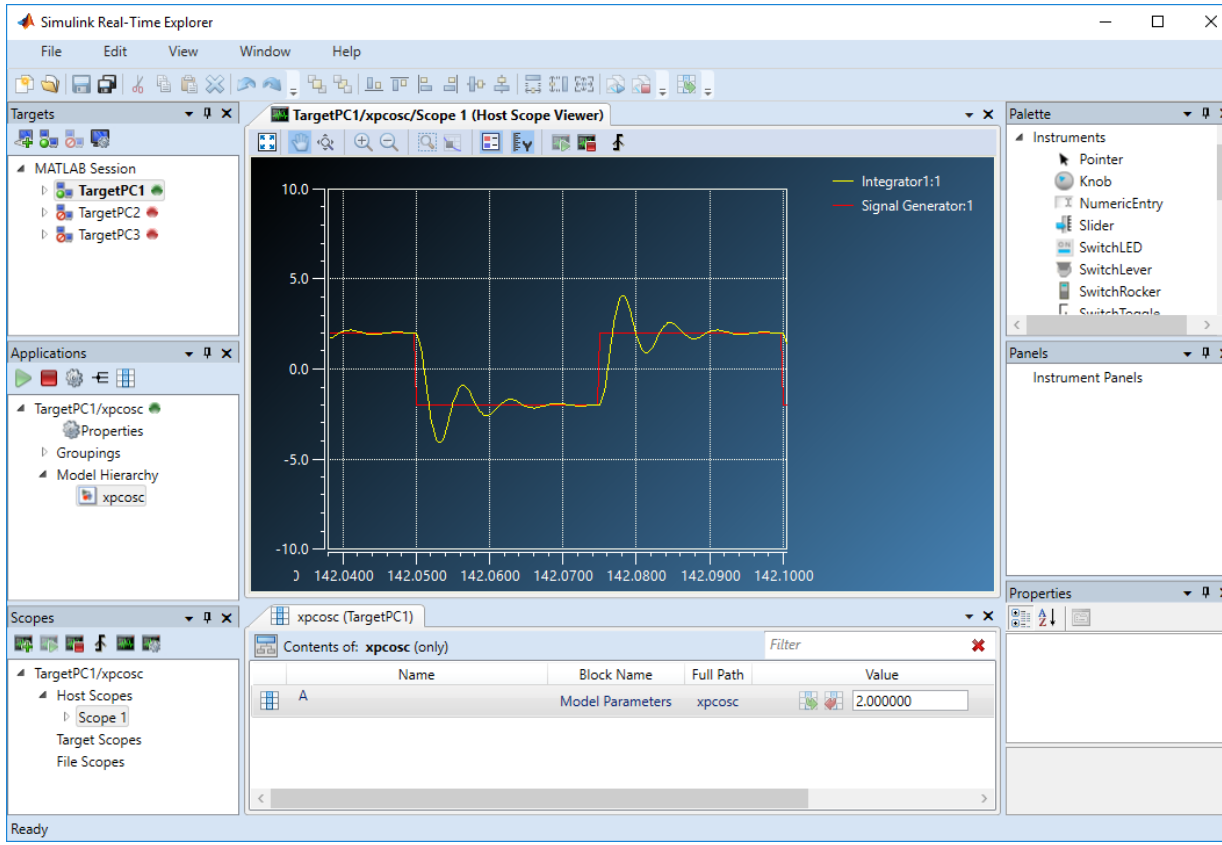
Updated Value

1 To change the value of MATLAB variable A to 2, type 2 into the **Value** box, and then press **Enter**.

   To revert the value of A to its previous value, click the **Revert** button .

2 Click the **Apply parameter value(s) changes** button .

   The dialog box looks like this figure.



3 To stop **Scope 1**, click **Scope 1** in the **Scopes** pane, and then click the **Stop Scope** button on the toolbar.

4 To stop execution, click the real-time application, and then click the **Stop** button on the toolbar.

See Also

More About

- “Tune Inlined Parameters with MATLAB Language” on page 6-155

- “Display and Filter Hierarchical Signals and Parameters” on page 6-179
- “Troubleshoot Parameters Not Accessible by Name” on page 6-186

Tune Inlined Parameters with 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` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_inlined_osc')))`). 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
```

See Also

More About

- “Troubleshoot Parameters Not Accessible by Name” on page 6-186

Tune Parameter Structures with Simulink Real-Time Explorer

In this section...
“Create Parameter Structure” on page 6-157
“Replace Block Parameters with Parameter Structure Fields” on page 6-159
“Tune Parameters in a Parameter Structure” on page 6-159
“Save and Load Parameter Structure” on page 6-161

To reduce the number of workspace variables you must maintain and avoid name conflicts, you can group closely related parameters into structures (see “Organize Related Block Parameter Definitions in Structures” (Simulink)).

In this example, the initial model `xpcosc` has four parameters that among them determine the shape of the output waveform.

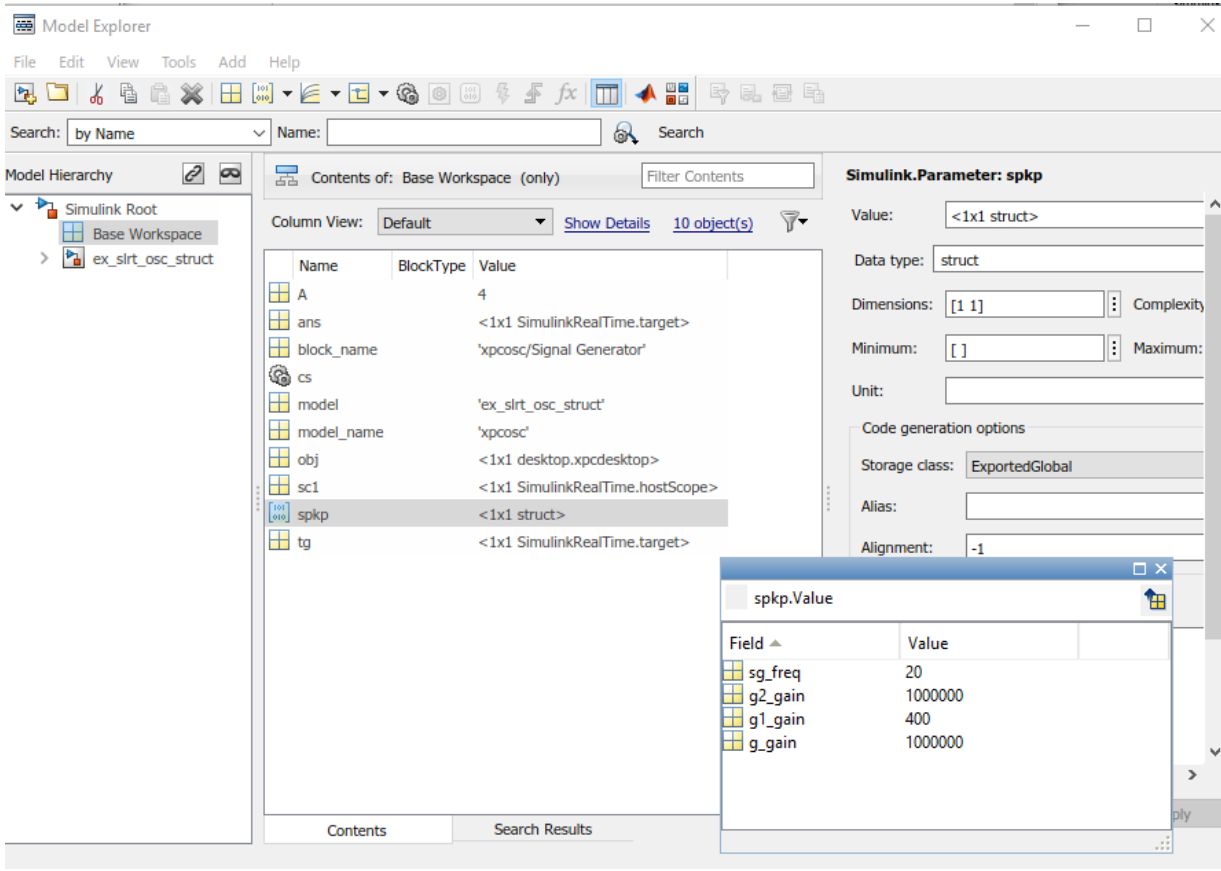
Block	Parameter	Structure Field Expression	Initial Value
`Signal Generator`	**Freq**	`spkp.sg_freq`	`20`
`Gain`	**Gain**	`spkp.g_gain`	`1000^2`
`Gain1`	**Gain**	`spkp.g1_gain`	`2*0.2*1000`
`Gain2`	**Gain**	`spkp.g2_gain`	`1000^2`

Create Parameter Structure

1. Open `xpcosc` in a working folder.
2. Open **Tools** > **Model Explorer**.
3. Select the **Base Workspace** node.
4. Click the **Add Simulink Parameter** button .
5. In the **Name** column, type the name `spkp`.
6. In the **Storage class** field, select `ExportedGlobal`.
7. In the **Value** field, type as one line:

```
struct('sg_freq',20, 'g2_gain',1000^2, ...
     'g1_gain',2*0.2*1000, 'g_gain',1000^2)
```

The field values duplicate the literal values in the dialog boxes. To change the field values, in row `spkp`, click the **Value** cell and click the **Edit** button .



8 Click **Apply**.

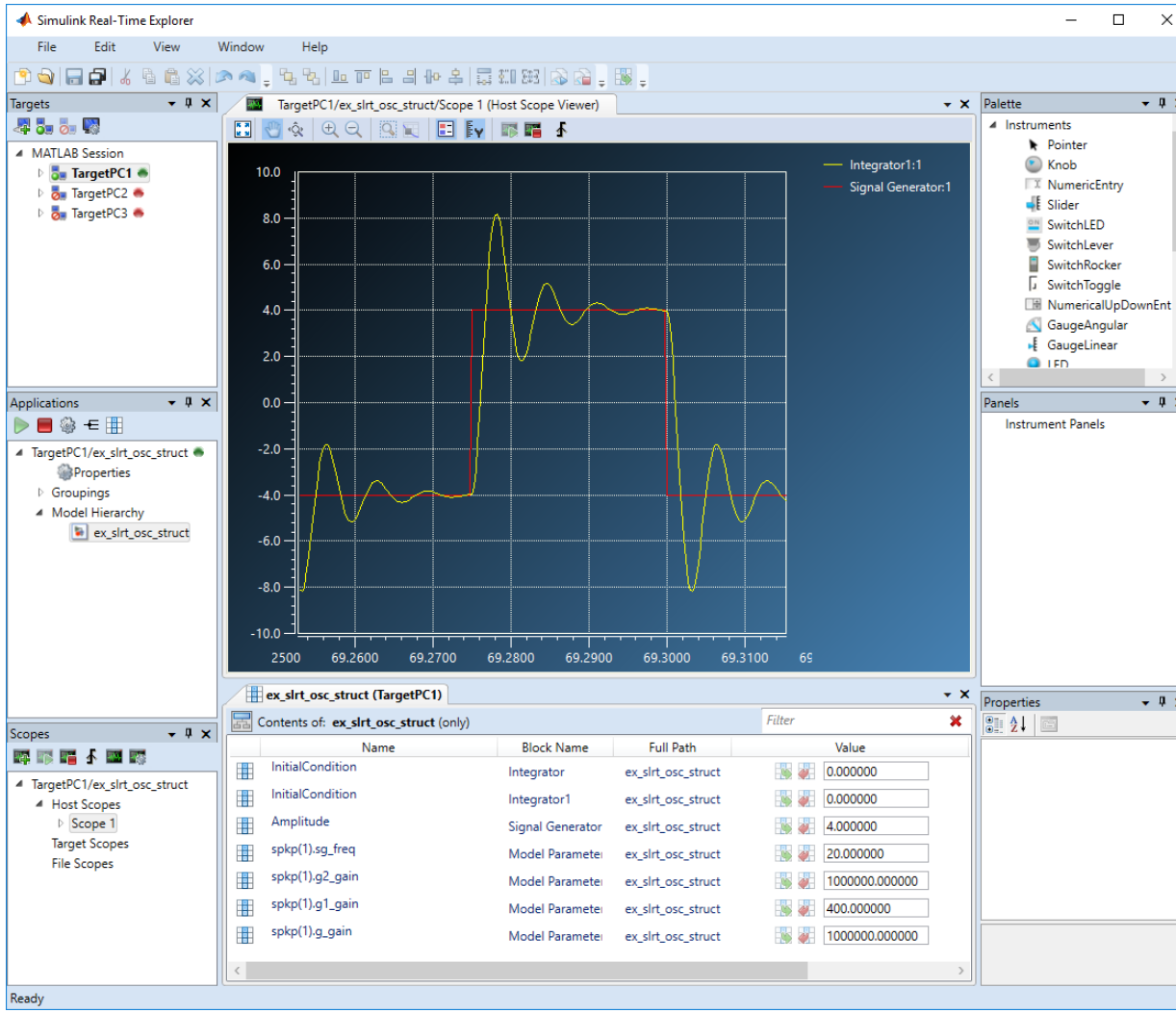
9 Save the model as `ex_slrt_osc_struct`.

Replace Block Parameters with Parameter Structure Fields

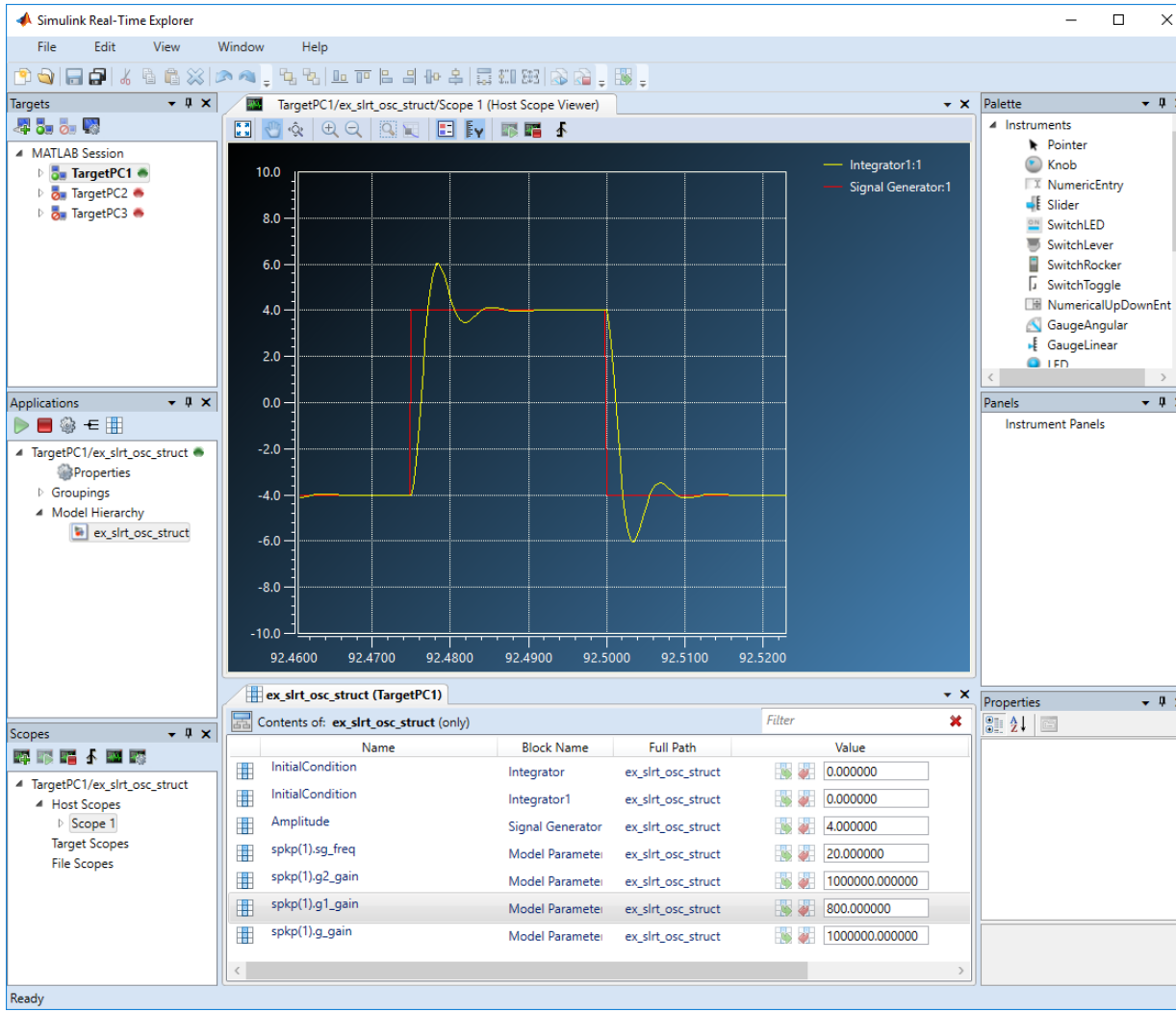
1. In the `Signal Generator` block, replace the value of parameter **Frequency** with `spkp.sg_freq`.
2. In the `Gain` block, replace the value of parameter **Gain** with `spkp.g_gain`.
3. In the `Gain1` block, replace the value of parameter **Gain** with `spkp.g1_gain`.
4. In the `Gain2` block, replace the value of parameter **Gain** with `spkp.g2_gain`.

Tune Parameters in a Parameter Structure

1. Build and download the model to your target computer.
2. Open **Tools** > **Simulink Real-Time**.
3. In the real-time application properties, set the **Stop Time** parameter to `Inf`.
4. Create and configure a host scope:
   a. In the Model Hierarchy node, right-click the model and open **View Signals**.
   b. Add a host scope ( ).
   c. Drag the signals `Integrator1` and `Signal Generator` to the scope.
   d. Start the scope ( ).
   e. View the scope ( ).
5. In the Model Hierarchy node, right-click the model and open **View Block Parameters**.
6. Open the **Values** text box for `spkp(1).g1_gain`.
7. Start the real-time application ( ).



8 In the **Values** text box for `spkp(1).g1_gain`, change the value to `800`, click outside the box, and click the **Apply parameter value(s) changes** button .



9 Stop the real-time application ( ).

Save and Load Parameter Structure

1 In Model Explorer, right-click row `spkp`.

2 Click **Export selected** and save the variable as `ex_slrt_osc_struct.mat`.

To load the parameter structure when you open the model, add a `load` command to the `PreLoadFcn` callback. To remove the parameter structure from the workspace when you close the model, add a `clear` command to the `CloseFcn` callback. For more information, see “Model Callbacks” (Simulink).

See Also

More About

- “Organize Related Block Parameter Definitions in Structures” (Simulink)
- “Display and Filter Hierarchical Signals and Parameters” on page 6-179
- “Model Callbacks” (Simulink)

Tune Parameter Structures with MATLAB Language

In this section...
“Create Parameter Structure” on page 6-163 “Replace Block Parameters with Parameter Structure Fields” on page 6-164 “Tune Parameters in a Parameter Structure” on page 6-164 “Save and Load Parameter Structure” on page 6-166

To reduce the number of workspace variables you must maintain and avoid name conflicts, you can group closely related parameters into structures (see “Organize Related Block Parameter Definitions in Structures” (Simulink)).

In this example, the initial model `xpcosc` has four parameters that among them determine the shape of the output waveform.

Block	Parameter	Structure Field Expression	Initial Value
`Signal Generator`	**Freq**	`spkp.sg_freq`	`20`
`Gain`	**Gain**	`spkp.g_gain`	`1000^2`
`Gain1`	**Gain**	`spkp.g1_gain`	`2*0.2*1000`
`Gain2`	**Gain**	`spkp.g2_gain`	`1000^2`

Create Parameter Structure

1. Open `xpcosc` in a working folder.
2. To create a parameter structure, in the MATLAB Command Window, enter:

```
kp = struct(...
    'sg_freq', 20, ...
    'g2_gain',1000^2, ...
    'g1_gain', 2*0.2*1000, ...
    'g_gain',1000^2)

kp =

  struct with fields:
```

```
    sg_freq: 20
    g2_gain: 1000000
    g1_gain: 400
     g_gain: 1000000
```

3 To make the parameter structure tunable on the target computer:

```
spkp = Simulink.Parameter(kp);
spkp.StorageClass = 'ExportedGlobal';
spkp.Value

ans =

  struct with fields:

    sg_freq: 20
    g2_gain: 1000000
    g1_gain: 400
     g_gain: 1000000
```

Replace Block Parameters with Parameter Structure Fields

1 In the `Signal Generator` block, replace the value of parameter **Frequency** with `spkp.sg_freq`.

2 In the `Gain` block, replace the value of parameter **Gain** with `spkp.g_gain`.

3 In the `Gain1` block, replace the value of parameter **Gain** with `spkp.g1_gain`.

4 In the `Gain2` block, replace the value of parameter **Gain** with `spkp.g2_gain`.

Tune Parameters in a Parameter Structure

1 Build and download the model to the target computer.

```
rtwbuild('xpcosc')
```

2 Set stop time to `inf`.

```
tg = slrt;
tg.StopTime = inf;
```

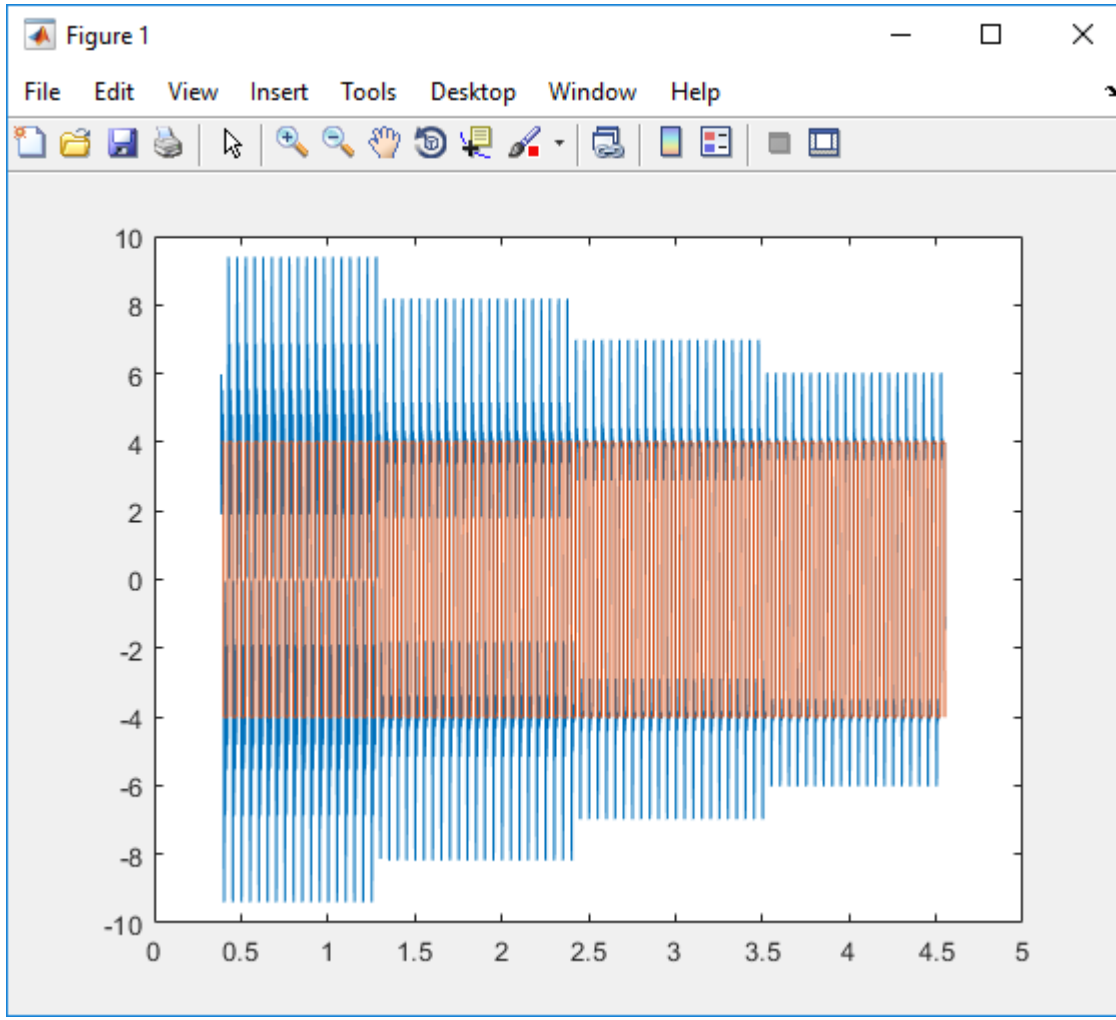
3 Sweep the `Gain` value of the `Gain1` block from `200` to `800`.

```
start(tg);
for g = 200 : 200 : 800
    setparam(tg, 'spkp.g1_gain', g);
```

```
    pause(1);
end
stop(tg);
```

4 Plot the results.

```
time = tg.TimeLog;
output = tg.OutputLog;
plot(time, output);
```



Save and Load Parameter Structure

To save the parameter structure `spkp` for later use, type:

```
save 'ex_slrt_osc_struct.mat', 'spkp'
```

To load the parameter structure when you open the model, add a `load` command to the `PreLoadFcn` callback. To remove the parameter structure from the workspace when you close the model, add a `clear` command to the `CloseFcn` callback. For more information, see “Model Callbacks” (Simulink).

See Also

More About

- “Organize Related Block Parameter Definitions in Structures” (Simulink)
- “Model Callbacks” (Simulink)

Define and Update Inport Data

In this section...
“File Dependencies” on page 6-167
“Map Inport to Use Square Wave” on page 6-167
“Update Inport to Use Sawtooth Wave” on page 6-170

You can create root-level input ports and use Root Inport Mapper to define input data. You can update the input data without rebuilding the model by using MATLAB language.

File Dependencies

This procedure depends on the following files:

- `ex_slrt_inport_osc` (open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_inport_osc')))) — Damped oscillator that takes its input data from input port `In1` and sends its muxed output to output port `Out1`.
- `ex_slrt_inport_square.mat` (load(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_inport_square.mat')))) — One second of output from a Signal Generator block that is configured to output a square wave.
- `ex_slrt_inport_sawtooth.mat` (load(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_inport_sawtooth.mat')))) — One second of output from a Signal Generator block that is configured to output a sawtooth wave.

Before starting this procedure, navigate to a working folder.

Map Inport to Use Square Wave

1. Open `ex_slrt_inport_osc` and save it to a working folder.
2. Load `ex_slrt_inport_square.mat` and assign `square` to a temporary workspace variable for use with Root Inport Mapper.

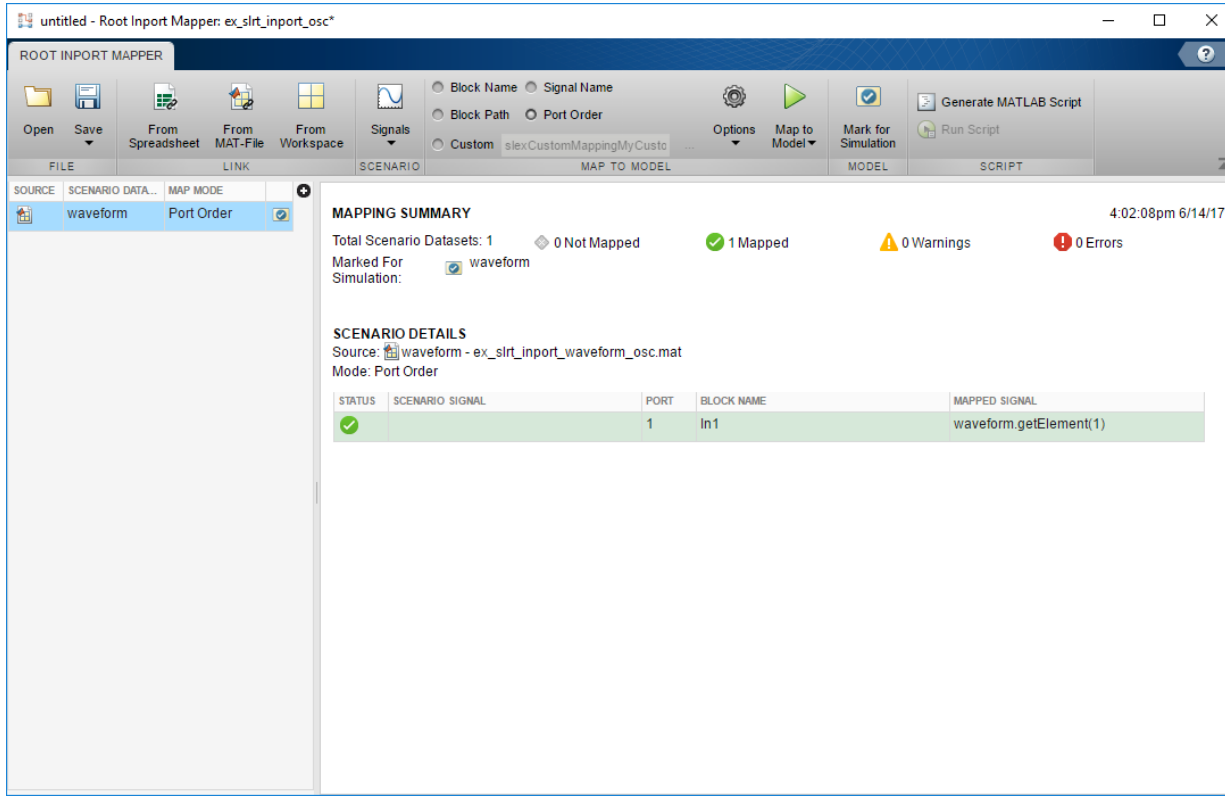
   ```
   waveform = square;
   ```
3. Double-click input port `In1`.
4. Clear **Interpolate data**, and then click **Connect Input**.
5. In Root Inport Mapper, click **From Workspace** and select variable `waveform`. Clear the other variables.

6 In the **Save to** text box, enter a name such as `ex_slrt_inport_waveform_osc.mat`, and then click **OK**.

7 Select map to model option **Port order** and, in the **Options** menu, select **Update Model**.

8 Click **Map to Model**.

9 To update the model with the mapped input data, select scenario `waveform`, and then click **Mark for Simulation**.



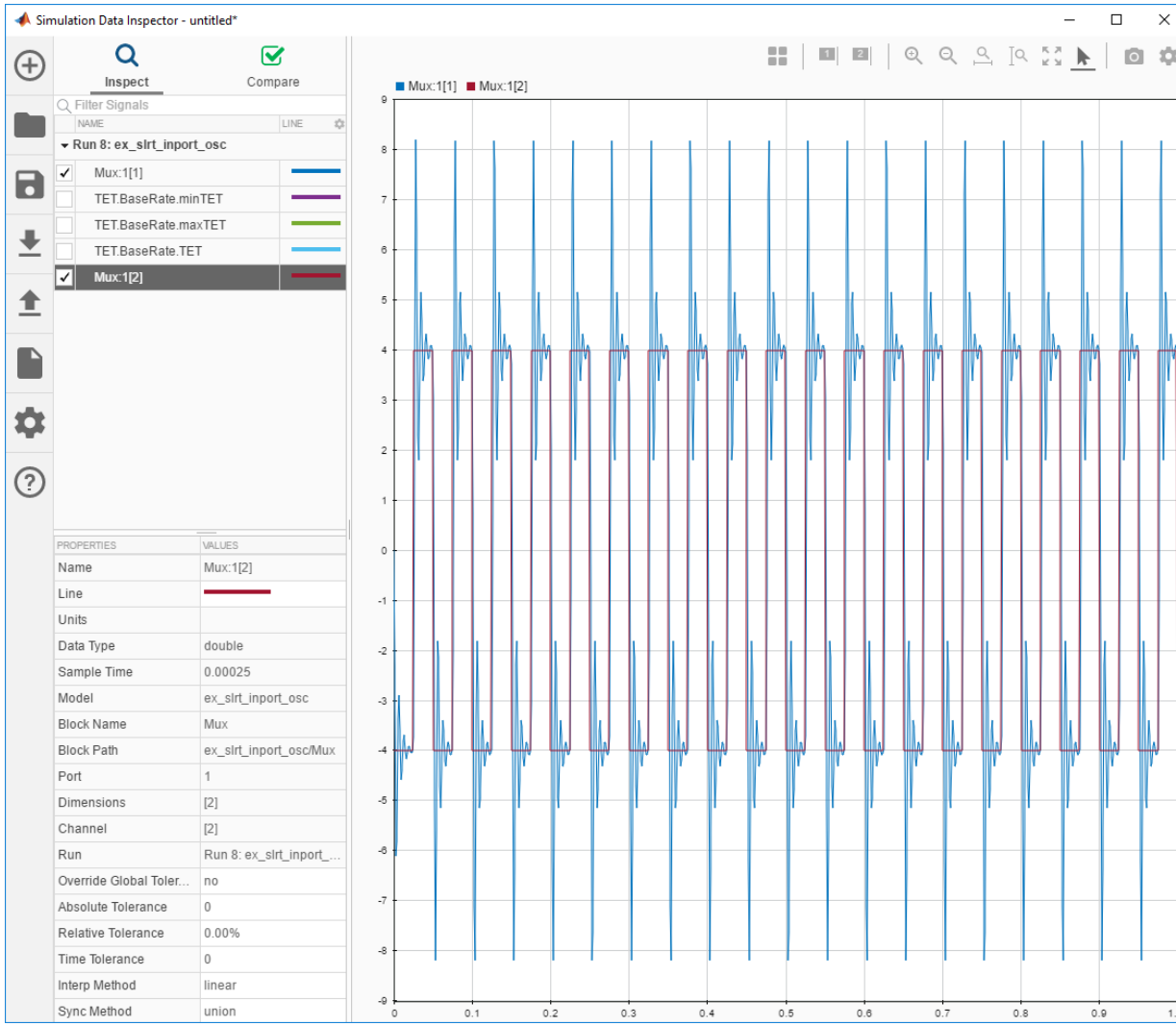
10 Click **Save**.

Save the scenario under a name such as `ex_slrt_inport_waveform_scenario.mldatx`.

11 Close the Root Inport Mapper. In the `In1` block parameters dialog box, click **OK**.

12 To display the output of the Mux block with Simulation Data Inspector, right-click the output signal and select **Log Selected Signals**.

You can now save, build, download, and execute the real-time application. Display the output with Simulation Data Inspector.



Update Inport to Use Sawtooth Wave

You can update the inport data to use a different data file without rebuilding the real-time application. The `ex_slrt_inport_osc.mldatx` file must be in the working folder.

1. Load `ex_slrt_inport_sawtooth.mat`, and then assign `sawtooth` to the temporary variable that you used with Root Inport Mapper.

   ```
   load(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', ...
       'ex_slrt_inport_sawtooth.mat')));
   waveform = sawtooth;
   ```

2. Create an application object.

   ```
   app_object = SimulinkRealTime.Application('ex_slrt_inport_osc');
   ```

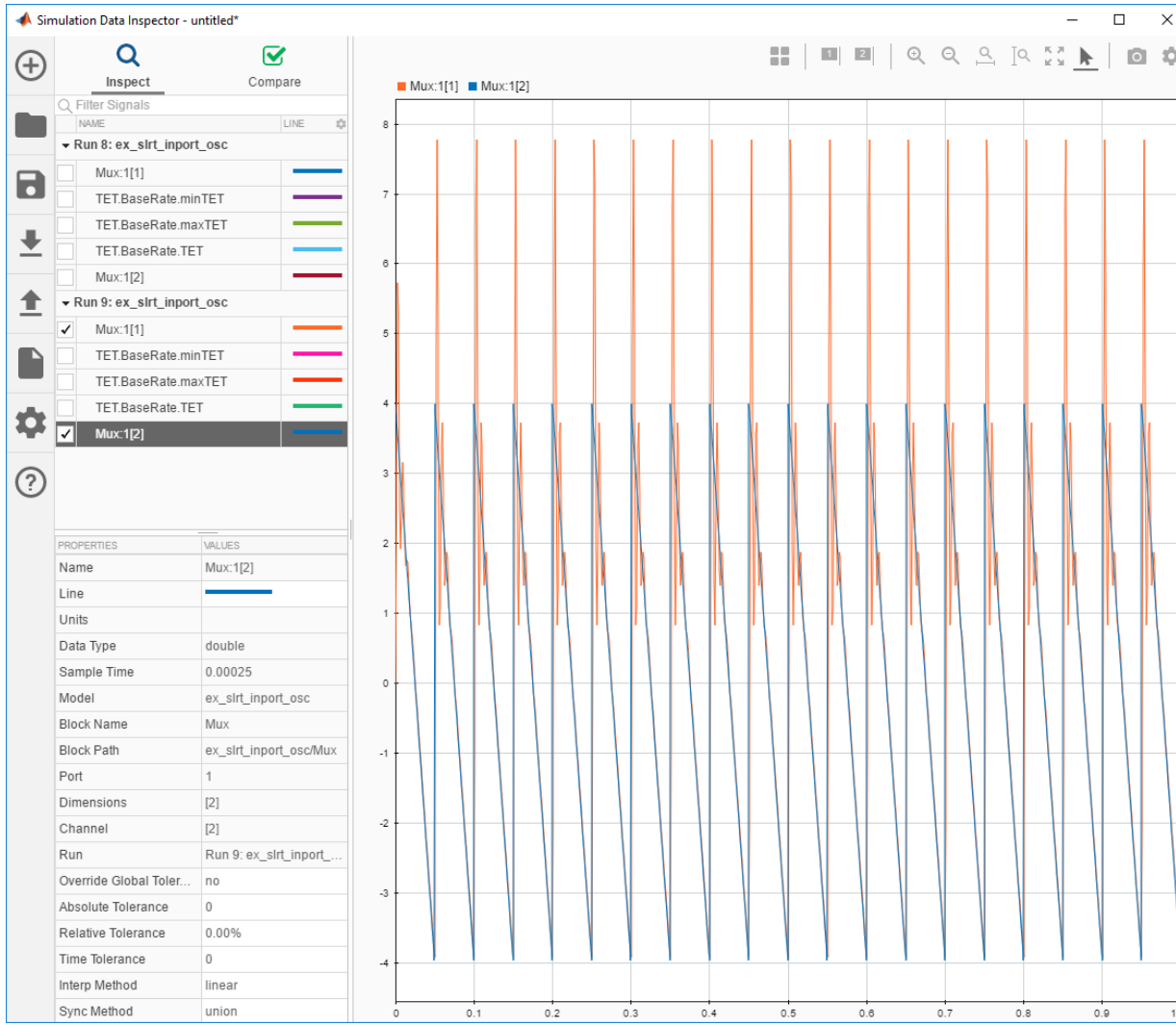
3. Update the application object.

   ```
   updateRootLevelInportData(app_object);
   ```

4. Load the updated object to the target computer and execute it.

   ```
   tg = slrt;
   load(tg, 'ex_slrt_inport_osc');
   start(tg);
   ```

5. Display the output with Simulation Data Inspector.


Simulation Data Inspector - untitled*
Inspect
Compare
Filter Signals
NAME
LINE
Run 8: ex_slrt_inport_osc
Mux:1[1]
TET.BaseRate.minTET
TET.BaseRate.maxTET
TET.BaseRate.TET
Mux:1[2]
Run 9: ex_slrt_inport_osc
Mux:1[1]
TET.BaseRate.minTET
TET.BaseRate.maxTET
TET.BaseRate.TET
Mux:1[2]
PROPERTIES
VALUES
Name
Mux:1[2]
Line
Units
Data Type
double
Sample Time
0.00025
Model
ex_slrt_inport_osc
Block Name
Mux
Block Path
ex_slrt_inport_osc/Mux
Port
1
Dimensions
[2]
Channel
[2]
Run
Run 9: ex_slrt_inport_...
Override Global Toler...
no
Absolute Tolerance
0
Relative Tolerance
0.00%
Time Tolerance
0
Interp Method
linear
Sync Method
union
Mux:1[1]
Mux:1[2]
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

See Also

More About

- “Define and Update Inport Data with MATLAB Language” on page 6-173
- “Load Data to Root-Level Input Ports” (Simulink)
- “Inport Data Mapping Limitations” on page 6-178
- “Inspect Simulink® Real-Time™ Data with Simulation Data Inspector” on page 6-76

Define and Update Inport Data with MATLAB Language

In this section...
“File Dependencies” on page 6-173
“Map Inport to Use Square Wave” on page 6-173
“Update Inport to Use Sawtooth Wave” on page 6-175

You can create root-level input ports and use the MATLAB language to define input data and to update the input data without rebuilding the model.

File Dependencies

This procedure depends on the following files:

- `ex_slrt_inport_osc` (open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_inport_osc')))) — Damped oscillator that takes its input data from input port `In1` and sends its muxed output to output port `Out1`.
- `ex_slrt_inport_square.mat` (load(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_inport_square.mat')))) — One second of output from a Signal Generator block that is configured to output a square wave.
- `ex_slrt_inport_sawtooth.mat` (load(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_inport_sawtooth.mat')))) — One second of output from a Signal Generator block that is configured to output a sawtooth wave.

Before starting this procedure, navigate to a working folder.

Map Inport to Use Square Wave

1. Open `ex_slrt_inport_osc`.

   ```
   model = docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples',...
       'ex_slrt_inport_osc'));
   open_system(model);
   save_system(model,'H:\workdir\ex_slrt_inport_osc.slx');
   ```

2. Load `ex_slrt_inport_square.mat`, and then assign `square` to a temporary workspace variable.

```
load(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', ...
    'ex_slrt_inport_square.mat')));
waveform = square;
```

3 Open `ex_slrt_inport_osc/In1`

```
inport = [model '/In1'];
load_system(inport);
```

4 Turn off inport data interpolation.

```
set_param(inport,'Interpolate','off');
```

5 Set external input variable.

```
set_param(model,'ExternalInput','waveform');
```

6 Load external input data.

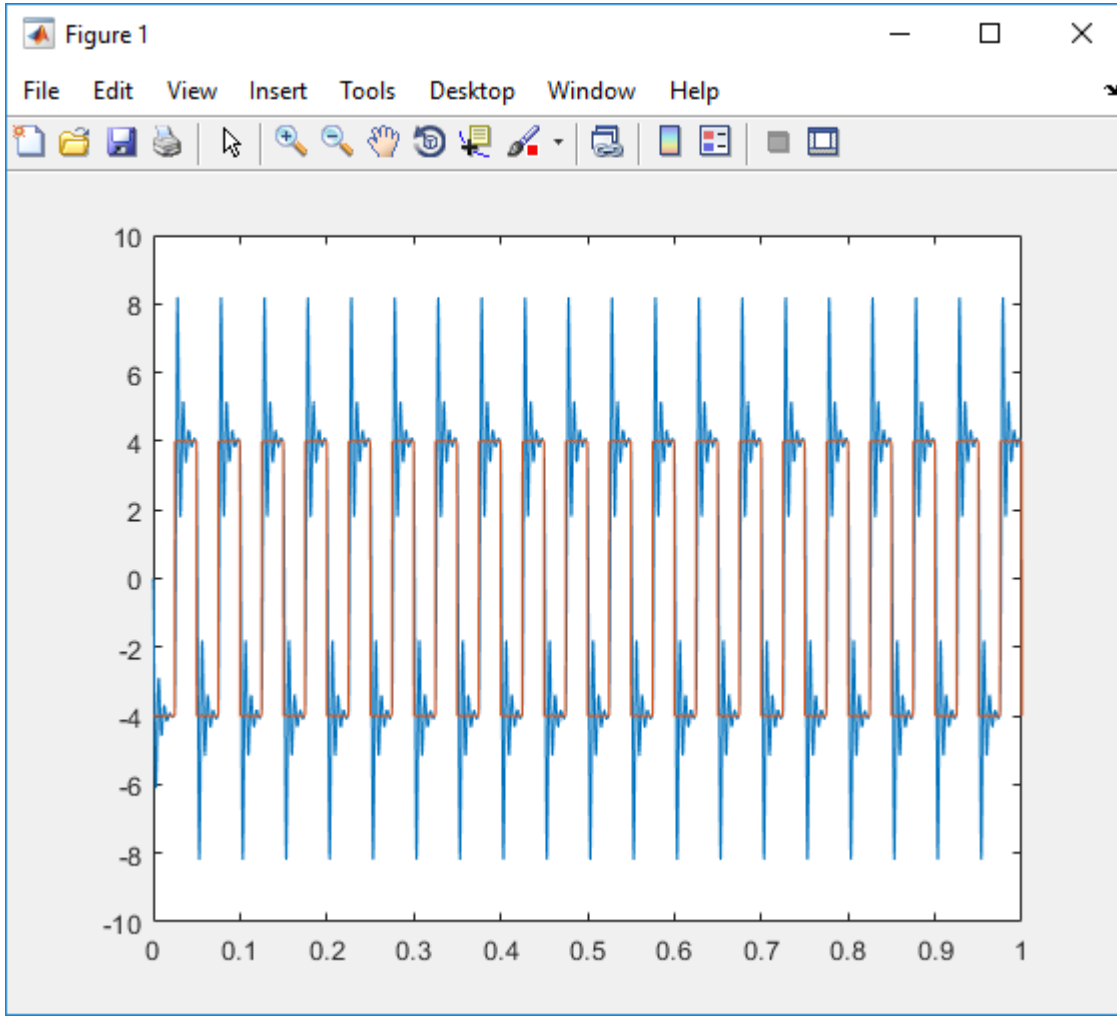
```
set_param(model,'LoadExternalInput','on');
```

7 You can now build, download, and execute the real-time application.

```
rtwbuild(model);
start(tg);
```

8 Plot the output.

```
plot(tg.TimeLog,tg.OutputLog);
```



Update Inport to Use Sawtooth Wave

You can update the inport data to use a different data file without rebuilding the real-time application. The `ex_slrt_inport_osc.mldatx` file must be in the working folder.

1 Load `ex_slrt_inport_sawtooth.mat`, and then assign `sawtooth` to the temporary variable that you used with Root Inport Mapper.

```
load(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', ...
    'ex_slrt_inport_sawtooth.mat')));
waveform = sawtooth;
```

2. Create an application object.

```
app_object = SimulinkRealTime.Application('ex_slrt_inport_osc');
```

3. Update the application object.

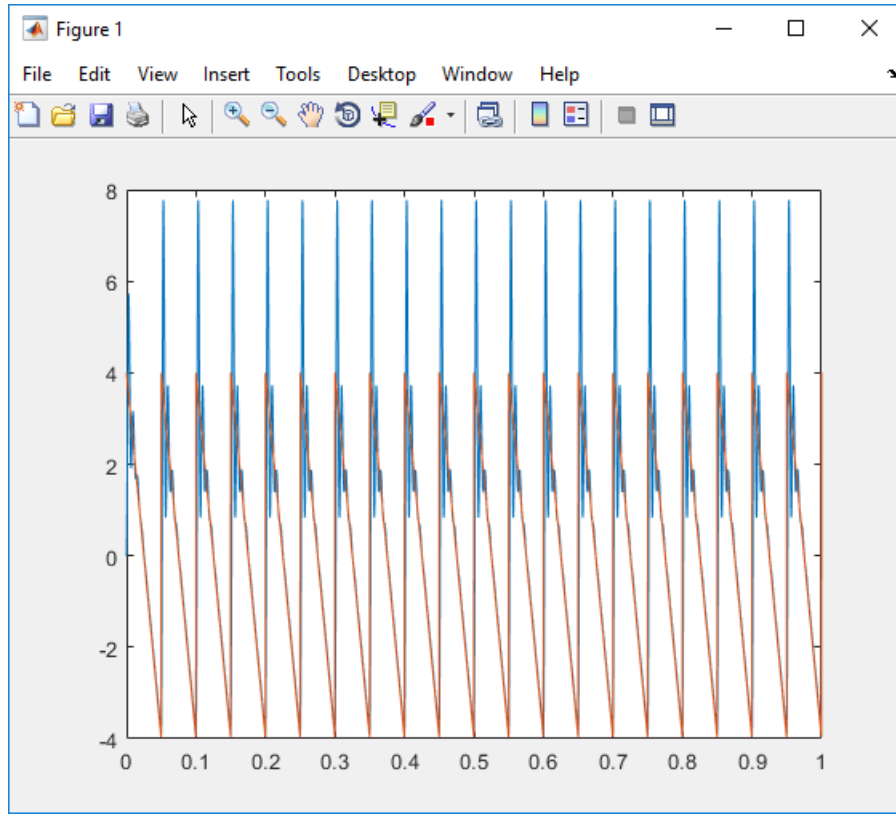
```
updateRootLevelInportData(app_object);
```

4. Download the updated object to the target computer and execute it.

```
tg = slrt;
load(tg, 'ex_slrt_inport_osc');
start(tg);
```

5. Plot the output.

```
plot(tg.TimeLog,tg.OutputLog);
```



See Also

More About

- “Define and Update Inport Data” on page 6-167
- “Load Data to Root-Level Input Ports” (Simulink)
- “Inport Data Mapping Limitations” on page 6-178
- “Inspect Simulink® Real-Time™ Data with Simulation Data Inspector” on page 6-76

Inport Data Mapping Limitations

In Simulink Real-Time, you cannot:

- Create data at run time for each time step by using the input `u = UT(t)` for MATLAB functions or expressions.
- Import complex values and asynchronous function-call signals into top-level input ports.
- Import signals of type `Stateflow.SimulationData.State` into top-level input ports.

See Also

More About

- “Define and Update Inport Data” on page 6-167
- “Load Data to Root-Level Input Ports” (Simulink)

Display and Filter Hierarchical Signals and Parameters

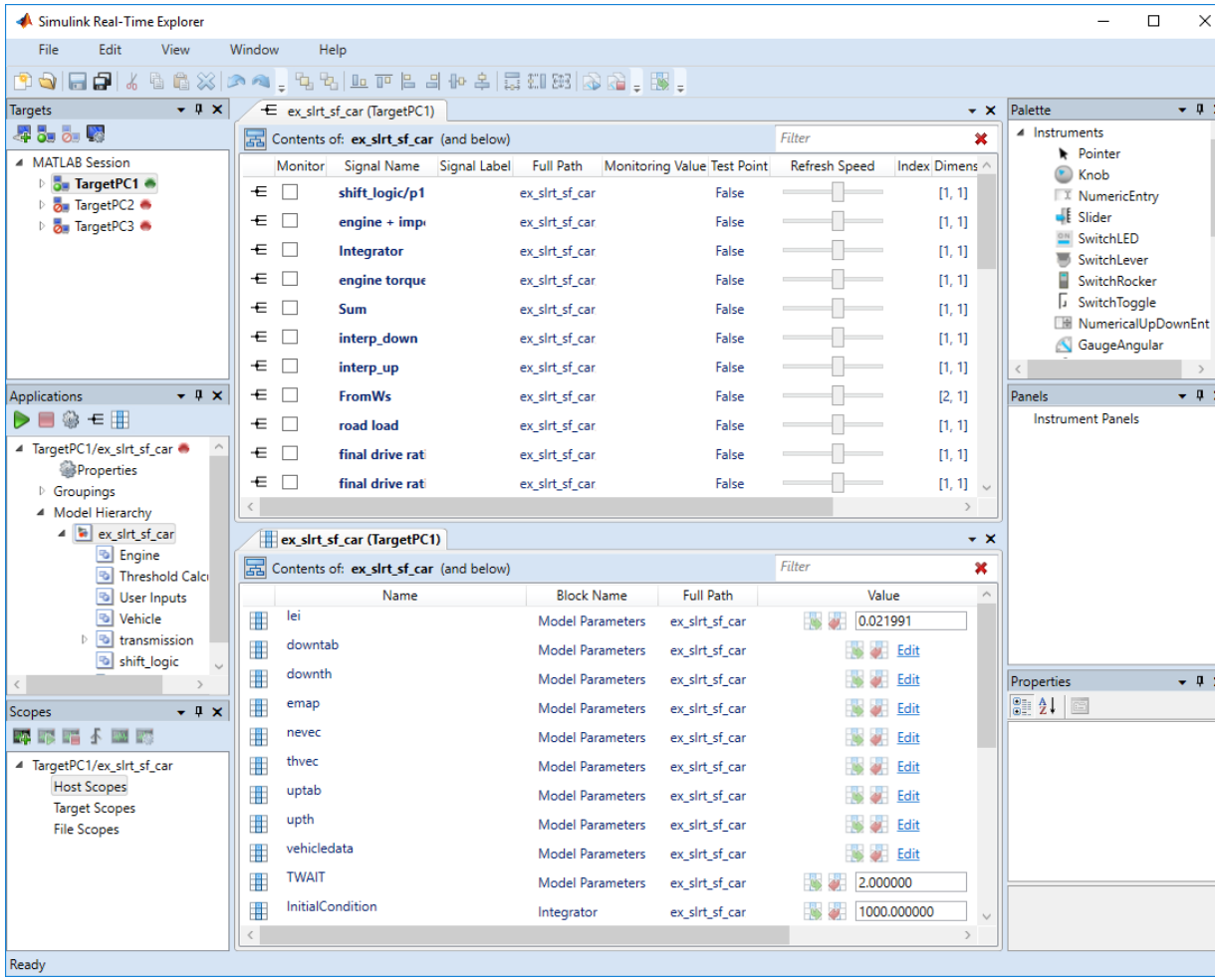
In this section...
“Hierarchical Display” on page 6-179 “Filtered Display” on page 6-180 “Grouped Display” on page 6-182

In Simulink Real-Time Explorer, the default view of the signal and parameter lists shows the signals and parameters only at the level that you selected. You can display signals and parameters for the current level and below and filter the display to show only the items that you are interested in.

Hierarchical Display

To show signals and parameters from the current level and below, navigate to the hierarchical level that you are interested in. Click the **Contents of** button (on the toolbar).

The contents of the top level of `ex_slrt_sf_car` are shown in the figure.



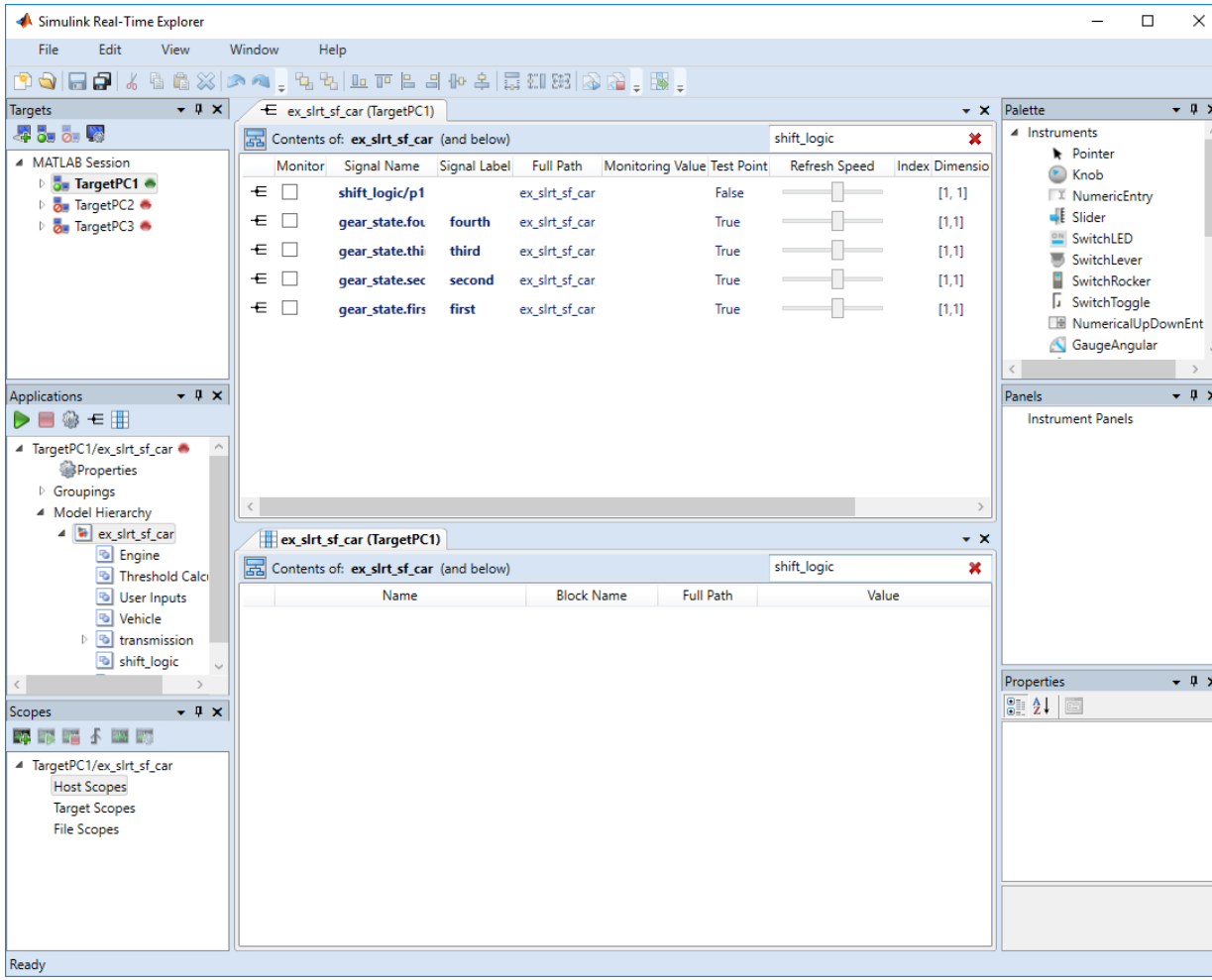
Filtered Display

To restrict display to signals or parameters with a particular characteristic, use the **Filter** text box that is on the toolbar. If you display only one level, the filter applies only to that level.

Explorer supports filtering by values in the following columns:

- Signals — **Signal Name**, **Signal Label**, **Full Path**
- Parameters — **Name**, **Block Name**, **Full Path**

For example, to restrict the display of signals and parameters to the `shift_logic` subsystem, select column **Signal Name**. Type `shift_logic` into the **Filter** text box.



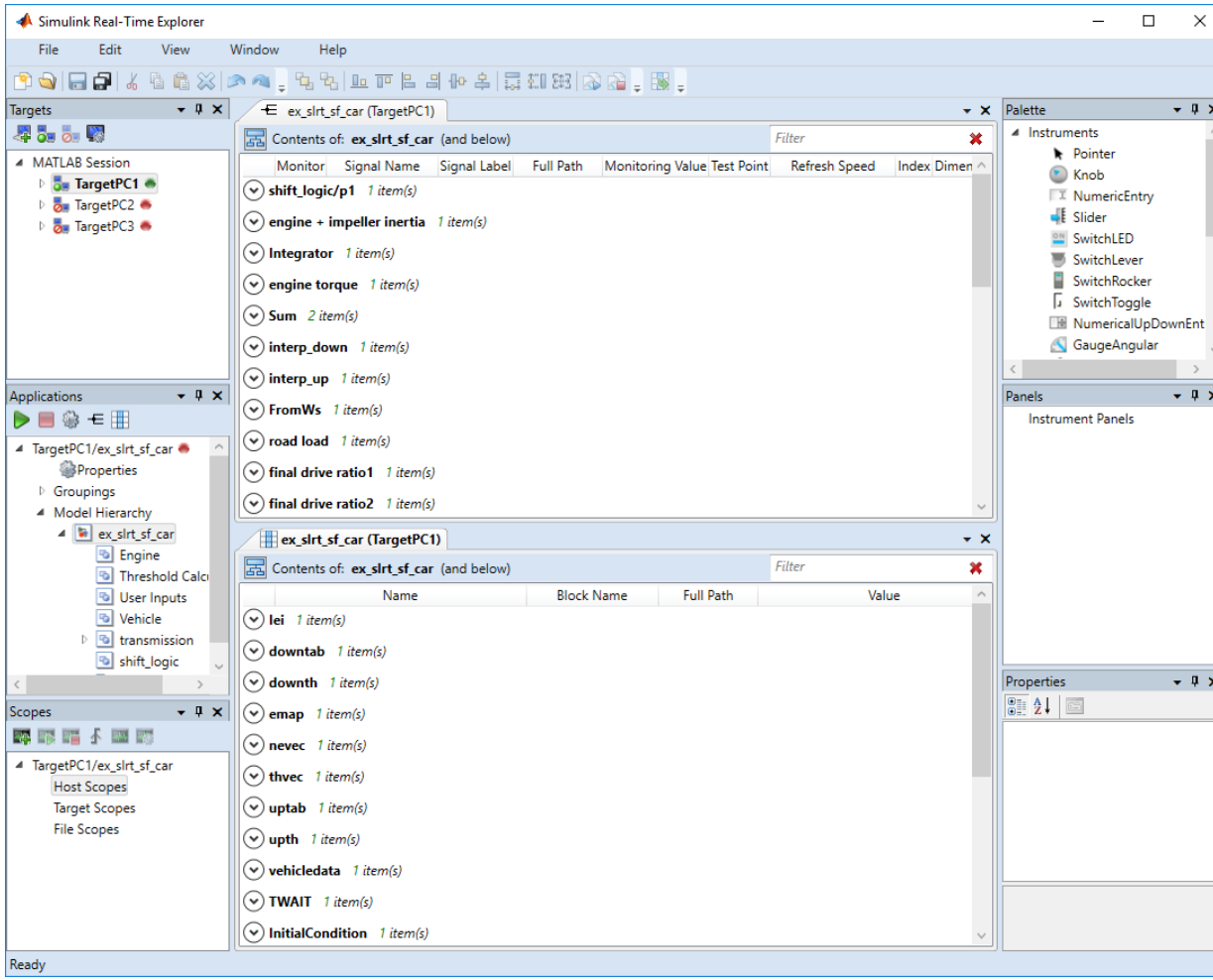
Grouped Display

To group signals and parameters by columns, right-click the column head and select **Group By This Column**. To remove the grouped display, right-click the column head and select **Remove Grouping**.

Explorer supports grouping by the following columns:

- Signals — **Signal Name**, **Signal Label**, **Full Path**, **Test Point**, **Dimensions**
- Parameters — **Name**, **Block Name**, **Full Path**, **Dimensions**

For example, to group signals by name, right-click the **Signal Name** column and select **Group By This Column**. To group parameters by name, right-click the **Name** column and select **Group By This Column**.


Simulink Real-Time Explorer
File
Edit
View
Window
Help
Targets
MATLAB Session
TargetPC1
TargetPC2
TargetPC3
Applications
TargetPC1/ex_slrt_sf_car
Properties
Groupings
Model Hierarchy
ex_slrt_sf_car
Engine
Threshold Calc
User Inputs
Vehicle
transmission
shift_logic
Scopes
TargetPC1/ex_slrt_sf_car
Host Scopes
Target Scopes
File Scopes
ex_slrt_sf_car (TargetPC1)
Contents of: ex_slrt_sf_car (and below)
Filter
Monitor
Signal Name
Signal Label
Full Path
Monitoring Value
Test Point
Refresh Speed
Index
Dimen
shift_logic/p1 1 item(s)
engine + impeller inertia 1 item(s)
Integrator 1 item(s)
engine torque 1 item(s)
Sum 2 item(s)
interp_down 1 item(s)
interp_up 1 item(s)
FromWs 1 item(s)
road load 1 item(s)
final drive ratio1 1 item(s)
final drive ratio2 1 item(s)
ex_slrt_sf_car (TargetPC1)
Contents of: ex_slrt_sf_car (and below)
Filter
Name
Block Name
Full Path
Value
lei 1 item(s)
downtab 1 item(s)
downth 1 item(s)
emap 1 item(s)
nevec 1 item(s)
thvec 1 item(s)
uptab 1 item(s)
upth 1 item(s)
vehicledata 1 item(s)
TWAIT 1 item(s)
InitialCondition 1 item(s)
Palette
Instruments
Pointer
Knob
NumericEntry
Slider
SwitchLED
SwitchLever
SwitchRocker
SwitchToggle
NumericalUpDownEnt
GaugeAngular
Panels
Instrument Panels
Properties
Ready

Troubleshoot Signals Not Accessible by Name

I cannot monitor, trace, or log some signal types in the real-time application.

What This Means

You cannot monitor, trace, or log by name the following types of signals from the development computer:

- Virtual or bus signals (including signals from bus creator blocks and virtual blocks). For example, assume that you connect the output of a Mux block (a virtual block) to a real-time Scope block. The Scope block displays the names of the Mux input signals rather than the names of the Mux output signals.
- Signals that Simulinkoptimizes away after you set the **Signal storage reuse** or **Block reduction** configuration parameters.

  The output of a block that was optimized away is replaced with the corresponding input signal to the block. To access these signals, make them test points.
- Blocks that buffer their input signals to make them contiguous. Examples include the To Workspace block and some S-function blocks. Such blocks generate a signal name associated with the generated block.

  If you connect a signal to the input port of such a block and to a real-time Scope block, the Scope block cannot access the signal. To access such a signal, add a unit Gain block (a Gain block with gain 1) 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.

Try This

Check the signal types for these issues.

See Also

Gain

More About

- “Nonvirtual and Virtual Blocks” (Simulink)
- “Virtual Signals” (Simulink)
- “Signal storage reuse” (Simulink)
- “Block reduction” (Simulink)
- “Troubleshoot Parameters Not Accessible by Name” on page 6-186
- “Internationalization Issues” on page 6-188

Troubleshoot Parameters Not Accessible by Name

I cannot observe or tune some parameters in the real-time application.

What This Means

Reasons that you cannot observe or tune some parameters in the real-time application are:

- Simulink Real-Time does not support parameters of multiword data types.
- You cannot tune parameters during execution that change the model structure, for example by adding a port. To change these parameters, you must stop execution, change the parameter, and rebuild the real-time application.

Try This

Check the parameters for these issues.

See Also

More About

- “Troubleshoot Signals Not Accessible by Name” on page 6-184
- “Internationalization Issues” on page 6-188

Troubleshoot Internationalization Issues

I seek to identify signal and parameter names through the target computer keyboard when the names contain Unicodecharacters from a non-English locale.

What This Means

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.

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 numeric ID, not by names.

Try This

For more information, see “Internationalization Issues” on page 6-188.

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 numeric ID, not by names.

When you use target scopes to observe signals, the kernel replaces a signal label that includes non-ASCII characters with the numeric 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 numeric portion of the screen contains this character vector for the English-language version:

  ```
  input1: block1/block2
  ```

  It contains this character vector for the Japanese-language version:

  ```
  1: ????1/????2
  ```

- In multiscope mode, the signal label above the scope contains this character vector for the English-language version:

  ```
  input1
  ```

  It contains this character vector for the Japanese-language version:

```
1
```

See Also

More About

- “Troubleshoot Signals Not Accessible by Name” on page 6-184
- “Troubleshoot Parameters Not Accessible by Name” on page 6-186

7

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.

- 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:

  1. Use multicore target computer hardware.
  2. In Simulink Real-Time Explorer, check that the **Multicore CPU** target setting is set to `'on'` for the target computer that you intend to use.
  3. On the **Simulink Real-Time Options** pane in the Configuration Parameters dialog box, set **Execution mode** to `Real-Time`.
  4. Enable polling by setting the `TLCOptions` setting `-axpcCPUClockPoll` to a nonzero value.

  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 interrupt mode. Because polling mode disables interrupts on the processor core where the model runs, it imposes restrictions on the model architecture and on target communication.

- Freerun mode — To use this non-real-time mode, in the Configuration Parameters dialog box:

  1. On the **Simulink Real-Time Options** pane, set **Execution mode** to `Freerun`.
  2. On the **Solver** pane, clear the check box for **Treat each discrete rate as a separate task**.

  In this mode, the real-time application thread does not wait for the timer. 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.

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). Also, 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 when the base sample time task completes operation. This capability gives a quasiparallel 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 an overhead, or latency, that reduces the minimal possible base sample time. The overhead is the sum of various factors related to the interrupt-driven execution scheme, such as interrupt controller latency, interrupt handler latency, and CPU latency. The overhead is referred to as overall interrupt latency.

The overall latency of interrupt mode is equivalent to a Simulink model containing a hundred nontrivial blocks. At least 5% of headroom is required because the CPU must also service lower priority tasks. 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 small sample times (Ts = 10–50 µs).
- DSP systems — Sample-based DSP system models of average model size and I/O complexity that are executed at high sample rates (Fs = 20–100 kHz). DSP models mainly process audio and speech data.

Polling mode for the kernel does not have the latency that interrupt mode does. It is sometimes seen as a “primitive” or “brute force” real-time execution scheme. When a real-time application executing in interrupt mode at a given sample time overloads the CPU, switching to polling mode is often the only alternative.

In interrupt mode, when a CPU has finished executing the real-time code, it cedes the rest of its execution time to the operating system. The operating system can use this time to execute other tasks, such as background or I/O tasks. When the next execution step is scheduled, the timer generates an interrupt, and Simulink Real-Time executes the next step.

In polling mode, however, when the CPU has finished executing the real-time code, the CPU does not cede time to other tasks. Instead, it does not do anything besides checking (*polling*) the time value to determine whether it is time to run the next execution step. Once this time arrives, it executes the next step and the process continues.

The latency associated with interrupts is not incurred because no timer interrupts are involved on this core. However, one core of the target computer is occupied with running the base rate, irrespective of how long it takes to run the actual real-time task.

The polling execution scheme does not depend on interrupt sources to notify the code to continue calculating the next model step. The base rate of the real-time code is executed on one core of the multicore processor, timed by the polling loop. Background tasks and model tasks other than the base rate task are executed on the other cores. For efficiency, put only the most critical code into the base rate task.

If you use Simulink Real-Time concurrent execution to execute your model, you can have multiple base rate tasks. The CPU scheduler arbitrarily selects one of these tasks to run in the polling loop.

Before considering polling mode, do the following:

- Optimize the model execution speed — To find possible speed optimizations using alternative blocks, use Performance Advisor or the profiler. 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 numeric 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 a PCI bus. Each register access introduces a comparably high latency time. Using the lowest latency I/O modules and drivers available is crucial.
- Consider running less critical code at a slower rate, taking advantage of the multitasking capabilities of Simulink Real-Time.

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 either mode without using another boot disk.

By default, the real-time application executes in interrupt mode. To switch to polling mode, you enable polling using a `TLCOptions` setting.

The following example uses `xpcosc`.

1. Open Simulink Real-Time Explorer.
2. Select the Properties pane for the target computer that you intend to use.
3. In the **Target settings** section, check that the **Multicore CPU** parameter is selected.
4. Open model `xpcosc`.
5. In the Configuration Parameters dialog box, on the **Simulink Real-Time Options** pane, set **Execution mode** to `Real-Time`.
6. In the Command Window, type:

   ```
   set_param('xpcosc','TLCOptions', '-axpcCPUClockPoll=1')
   ```

7. Build and download the real-time application.

After you have downloaded the real-time application, the target display shows the execution mode. If you want to execute the real-time application in interrupt mode again, either remove the setting or assign `0` to the setting:

```
set_param('xpcosc','TLCOptions','')
set_param('xpcosc','TLCOptions','-axpcCPUClockPoll=0')
```

Rebuild and download the real-time application.

Restrictions on Multicore Processors

Polling mode runs only on a multicore processor target computer with multicore processing enabled. For more information, see “Multicore Processor Configuration” on page 10-10.

Polling mode disables interrupts on the core where the base rate task is running. Background tasks and model tasks other than the base rate task are inactive on this core. Tasks for Ethernet communication, target display updates, and UDP transfers run on the other cores. Interrupts are still enabled on cores other than the one running the polling task.

See Also

“TLC Command-Line Options” | `SimulinkRealTime.utils.minimumSampleTime` | Target Settings

More About

- “Set Configuration Parameters”
- “Execution mode”
- “Performance Optimization”
- “Troubleshoot Overloaded CPU from Executing Real-Time Application” on page 24-5

Real-Time Application Execution

Execution with User Interface Models

You can use the Simulink interface to create a custom user interface (UI) for your real-time application. First, create a user interface model with the Simulink interface. Then, 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” on page 8-2
“Creating a Custom Graphical Interface” on page 8-3
“To Target Block” on page 8-4
“From Target Block” on page 8-6
“Creating a Real-Time Application Model” on page 8-8
“Marking Block Parameters” on page 8-8
“Marking Block Signals” on page 8-10

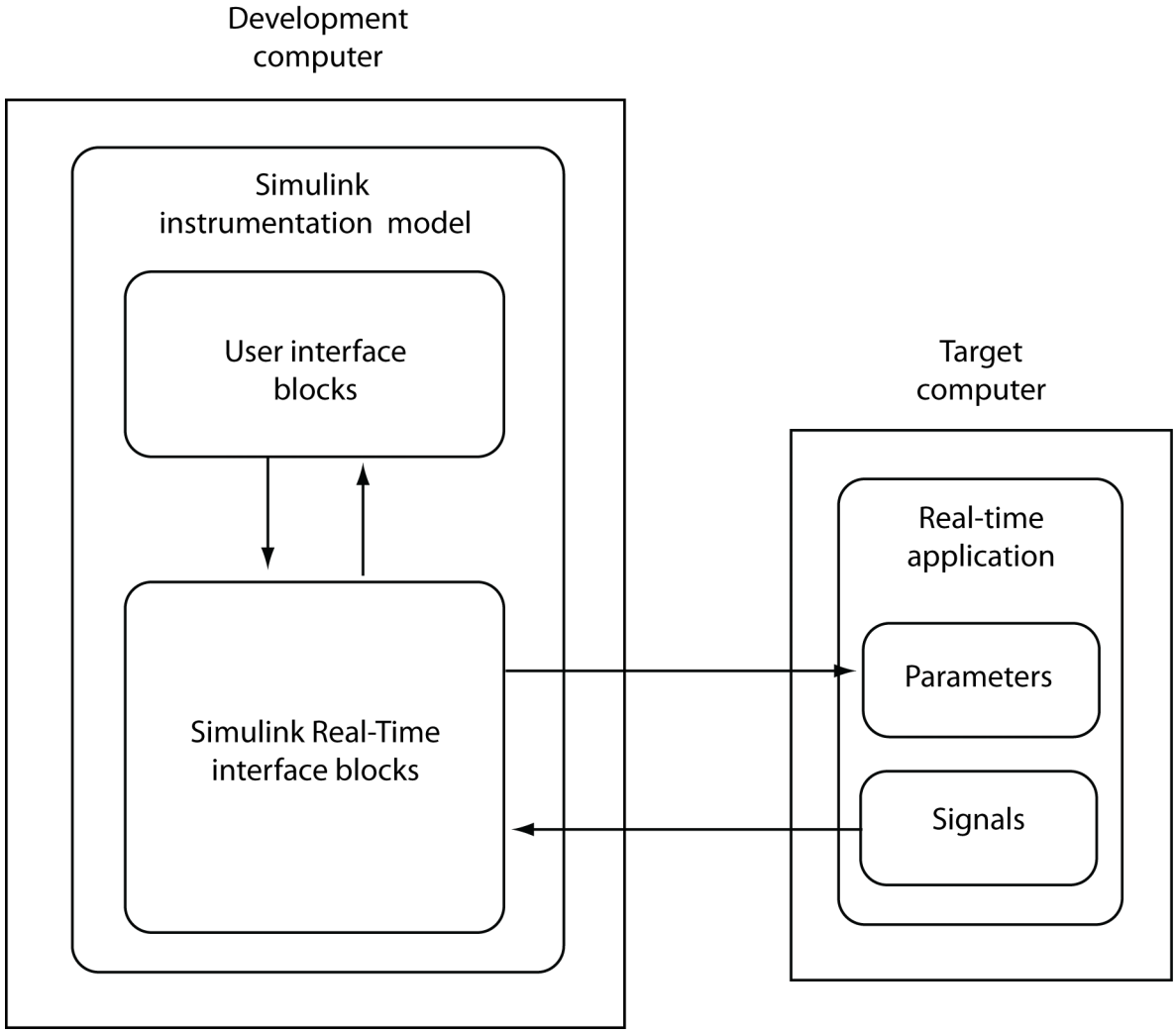
Simulink User Interface Model

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 plugin.

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 plugin.



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:

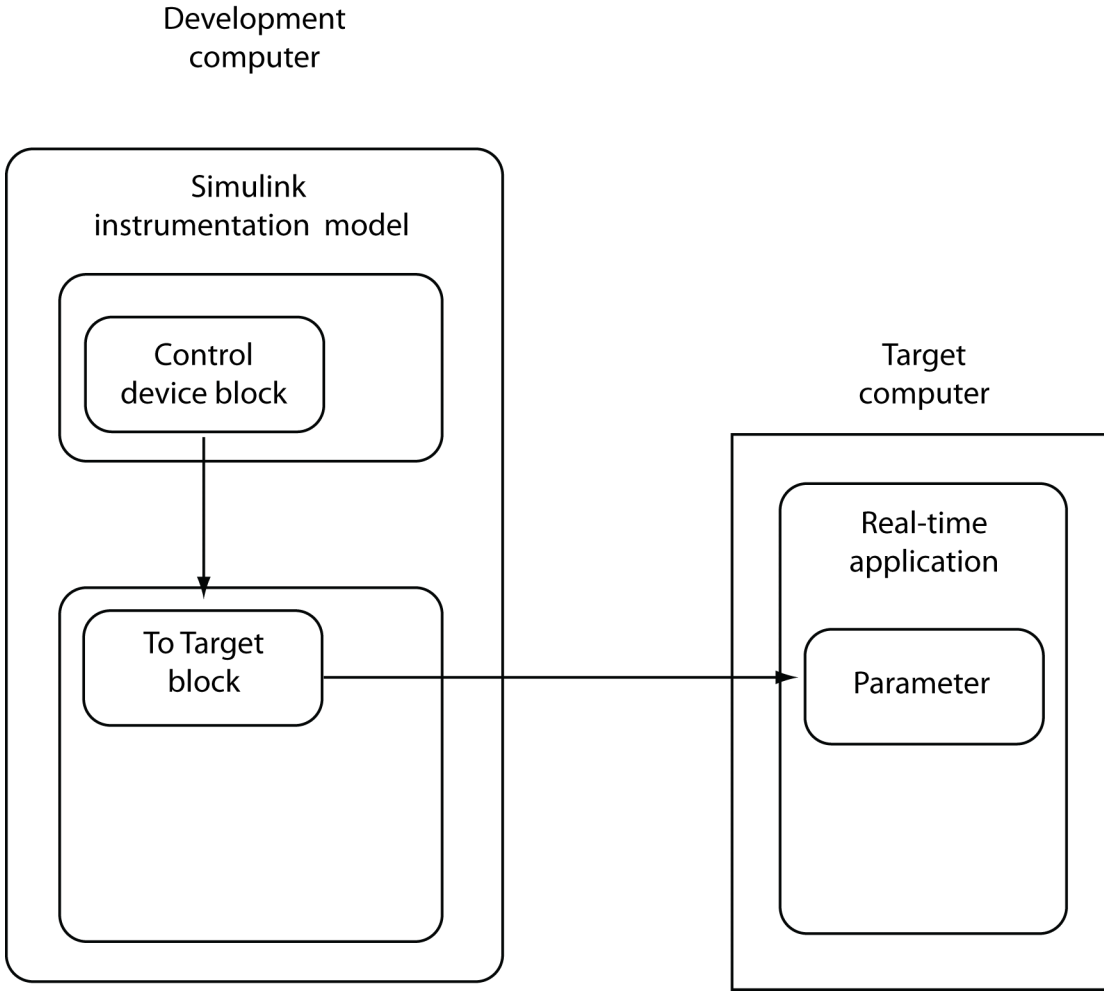
1. In the Simulink real-time application model, decide which block parameters and block signals that you want to access through user interface control and display devices.
2. Tag the block parameters in the Simulink model that you want to be connected to a control device. See “Marking Block Parameters” on page 8-8.
3. Tag the signals in Simulink model that you want to be connected to a display device. See “Marking Block Signals” on page 8-10.
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. Connect the To Target blocks on the left to control devices.
   - You can connect Altia and Simulink 3D Animation blocks to the Simulink Real-Time From Target interface blocks. Connect the From Target blocks on the right 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.

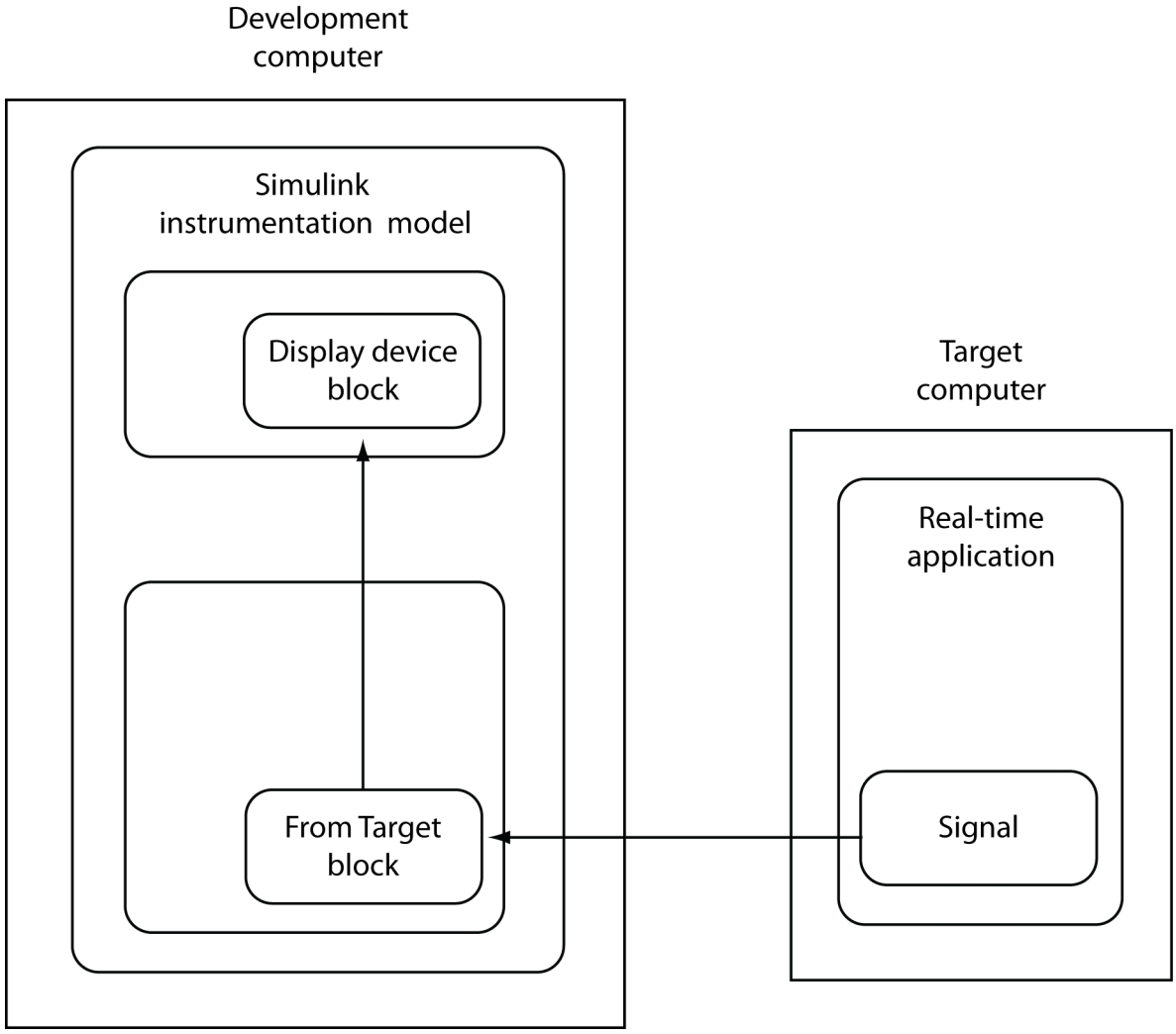


This block is implemented as a MATLAB S-function. The block only changes a parameter on the real-time application when the input value differs from the value that existed at the last time step. This block uses the parameter downloading feature of the Simulink Real-Time command-line interface. This block 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. Opening a model that contains these blocks or copying them to another model takes longer than normal without a connection between the development and target computers.

From Target Block

This block behaves like a source. Typically, you connect its output to the input of a display device.



Because only one numeric 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. Opening a model that contains these blocks or copying them to another model takes 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 to specify the parameters and signals that you want to connect to a custom graphical interface.

See “Marking Block Parameters” on page 8-8 and “Marking Block Signals” on page 8-10 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 contain placeholder tags illustrating the syntax. 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 specifies 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 is 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 C/C++ used 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 a 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 file name for your model. For example, enter

```
xpctank1
```

If you have not already marked block signals, your next task is to mark block signals, and then to create the user interface template model. See “Marking Block Signals” on page 8-10 and “Creating a Custom Graphical Interface” on page 8-3.

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 8-8.

Tip The `xpctank` model blocks and signals can contain placeholder tags illustrating the syntax. Replace these tags with your new tags or add the new tags using the multiple label syntax.

You cannot tag signals on the output ports of virtual blocks, such as Subsystem and Mux blocks. Also, you cannot tag 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 is 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 C/C++ uses to name functions, variables, and so forth. Do not use names like `-foo`.

To create the From Target blocks in a 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” (Simulink)). For example, if your model combines two signals into the inputs of a Mux block, do not tag the output signal from 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 file name for your model. For example, enter

```
xpc_tank1
```

If you have not already marked block parameters, your next task is to mark them. See “Marking Block Parameters” on page 8-8. If you have already marked block signals,

return to “Creating a Custom Graphical Interface” on page 8-3 for additional guidance on creating a user interface template model.

9

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. Using these commands, you can start and stop execution, configure and control scopes, and tune parameters.

These commands are 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”.

To read the target computer console log, call `SimulinkRealTime.utils.getConsoleLog`.

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 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 9-5.

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 9-5.

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. To run a command sequence, type the variable name. For example, to turn on the motor, 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`.

See Also

`SimulinkRealTime.utils.getConsoleLog`

Related Examples

- “Target Computer Commands”

Tuning Performance

- “Improve Performance of Multirate Model” on page 10-2
- “Multicore Processor Configuration” on page 10-10
- “Limits on Sample Time” on page 10-12
- “CPU Overload Options” on page 10-14
- “Execution Profiling for Real-Time Applications” on page 10-19
- “Building Referenced Models in Parallel” on page 10-26
- “Sample Time and Throughput in Real-Time Applications” on page 10-28

Improve Performance of Multirate Model

This example shows how to use Performance Advisor to detect blocks and parameter settings that can reduce performance. It determines the lower limit on sample time that does not produce a CPU overload.

Requirements

This example uses model `ex_slrt_perfadv`. To open this model, open the subsystem models first:

- open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_perfadv_ref1')))
- open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_perfadv_ref2')))
- open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', '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 with a MATLAB variable, `Ts`. You can change the base sample time by changing the value of `Ts`.

In addition to the MATLAB® software requirements, the following hardware is required:

- One Windows® development computer with an Ethernet card
- One target computer
- One crossover cable for communication between the development and target computers

Generate Baseline

Before you optimize model `ex_slrt_perfadv` using Performance Advisor, generate a baseline.

1. Open `ex_slrt_perfadv`.

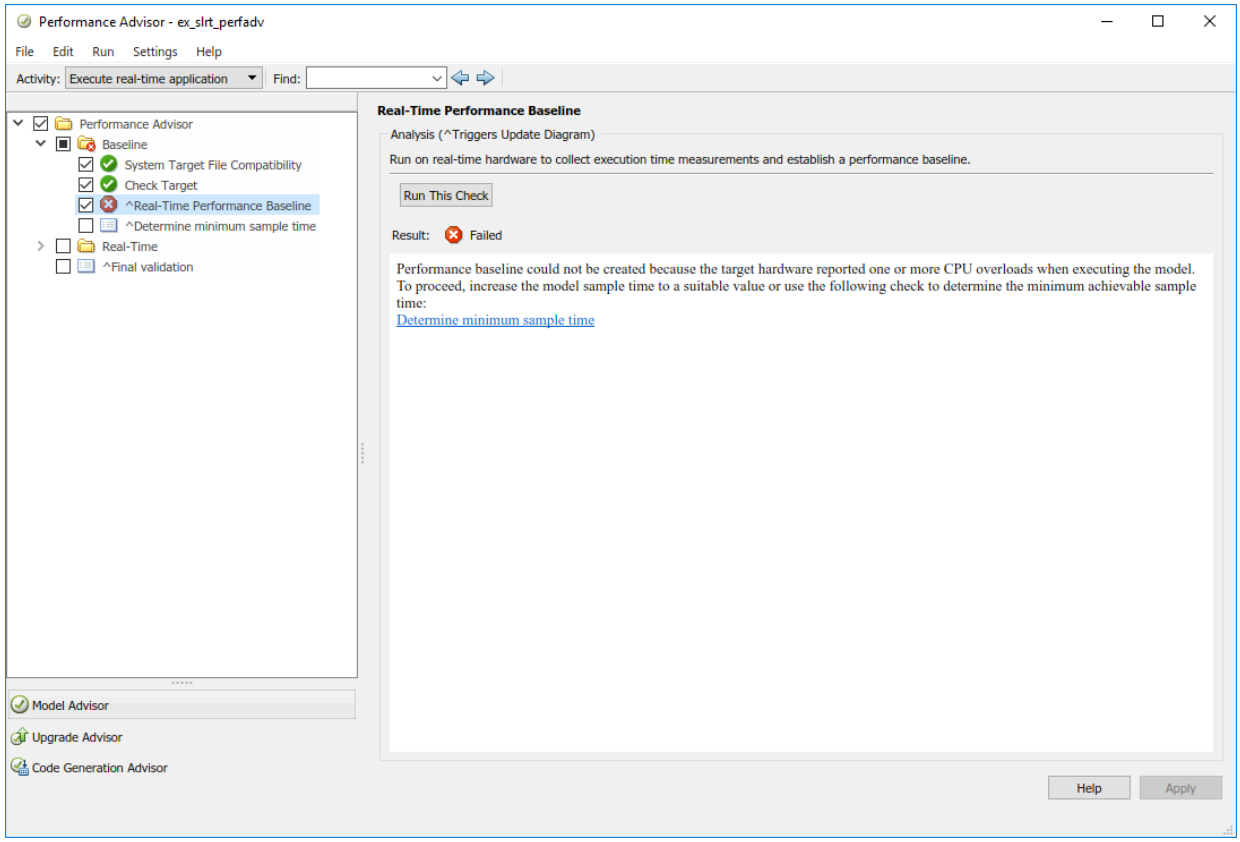
2. From the **Analysis** menu, click **Performance Tools >> Performance Advisor**.

3. Set **Activity** to `Execute real-time application`.

4. Under node **Performance Advisor**, select all of the **Baseline** checks except **Determine minimum sample time**.

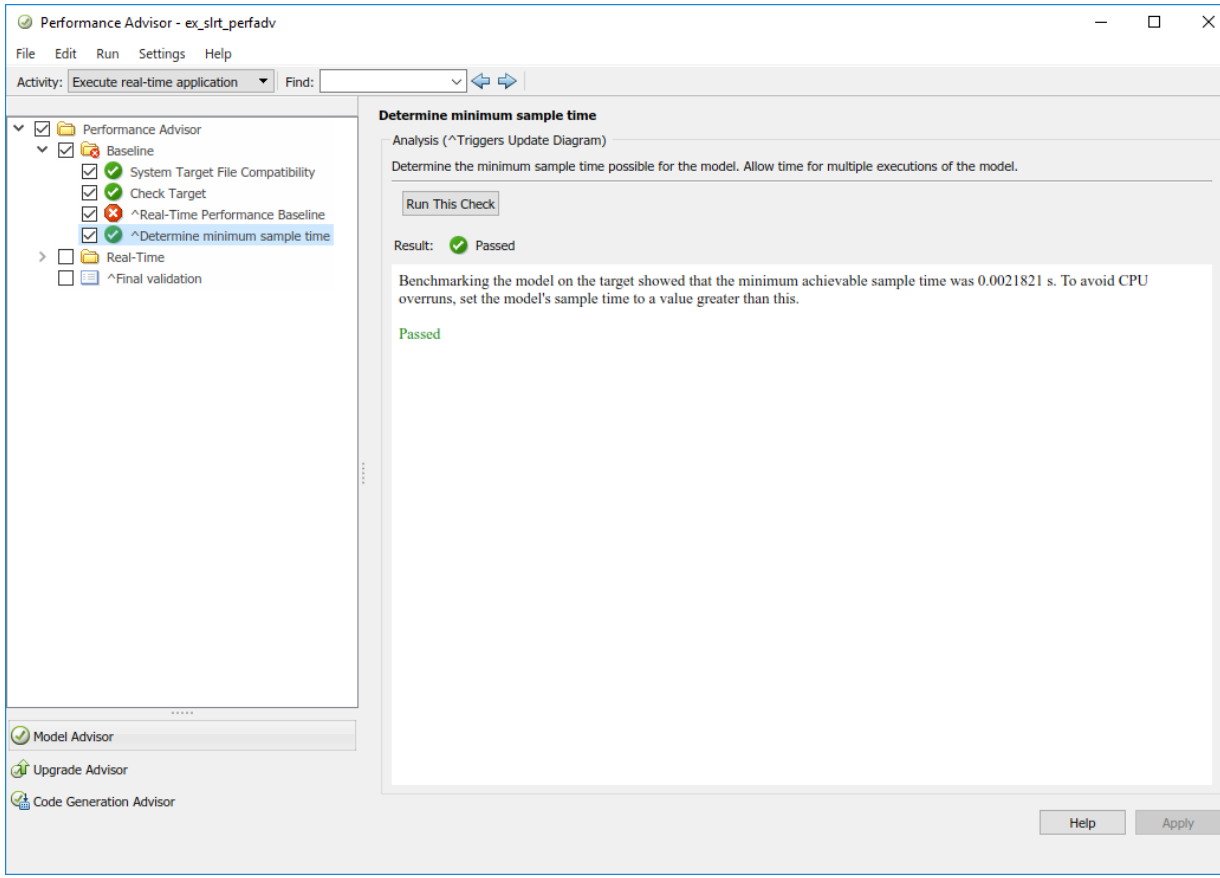
Determining the minimum sample time can be a lengthy process for a large model with a long execution time.

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**.



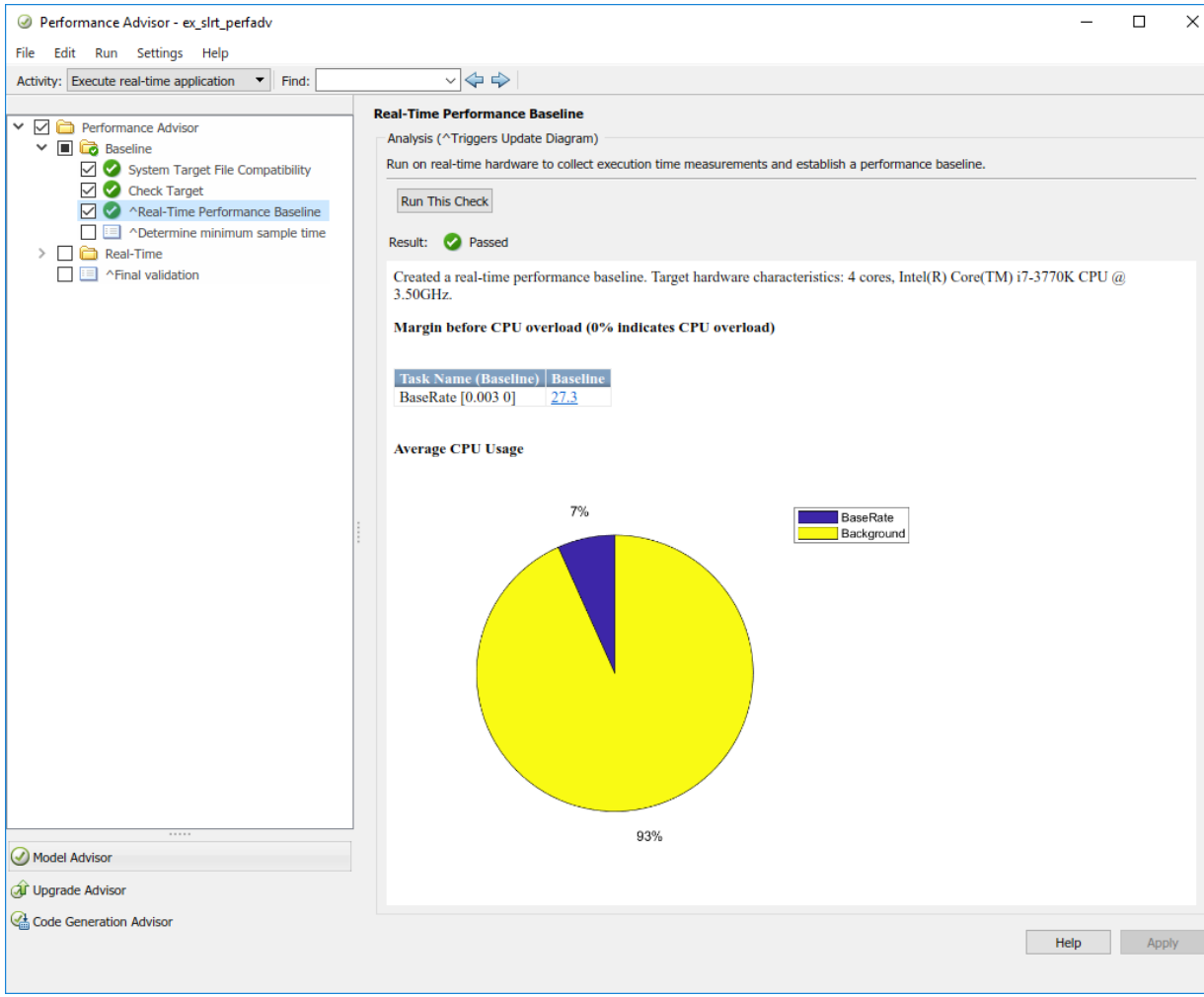
To avoid the overloads that random variations can cause, set `Ts` to a value above the lower limit. For example, set it to `0.003` s.

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**.



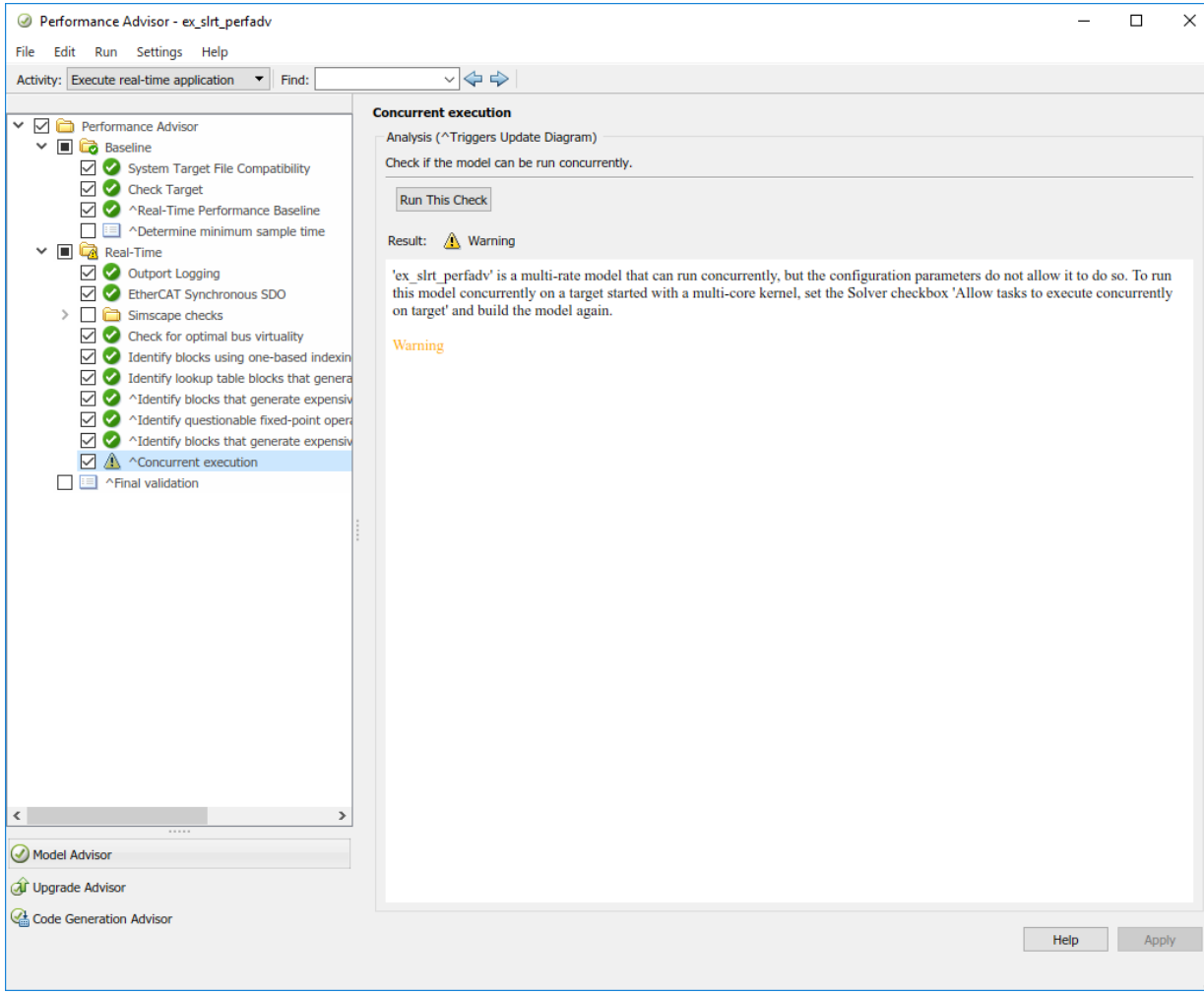
Perform Real-Time Checks

To perform the real-time performance checks on model `ex_slrt_perfadv`, first create a baseline. 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. `ex_slrt_perfadv` contains no Simscape blocks.

2. Select the **Real-Time** node, and then click **Run selected checks**.

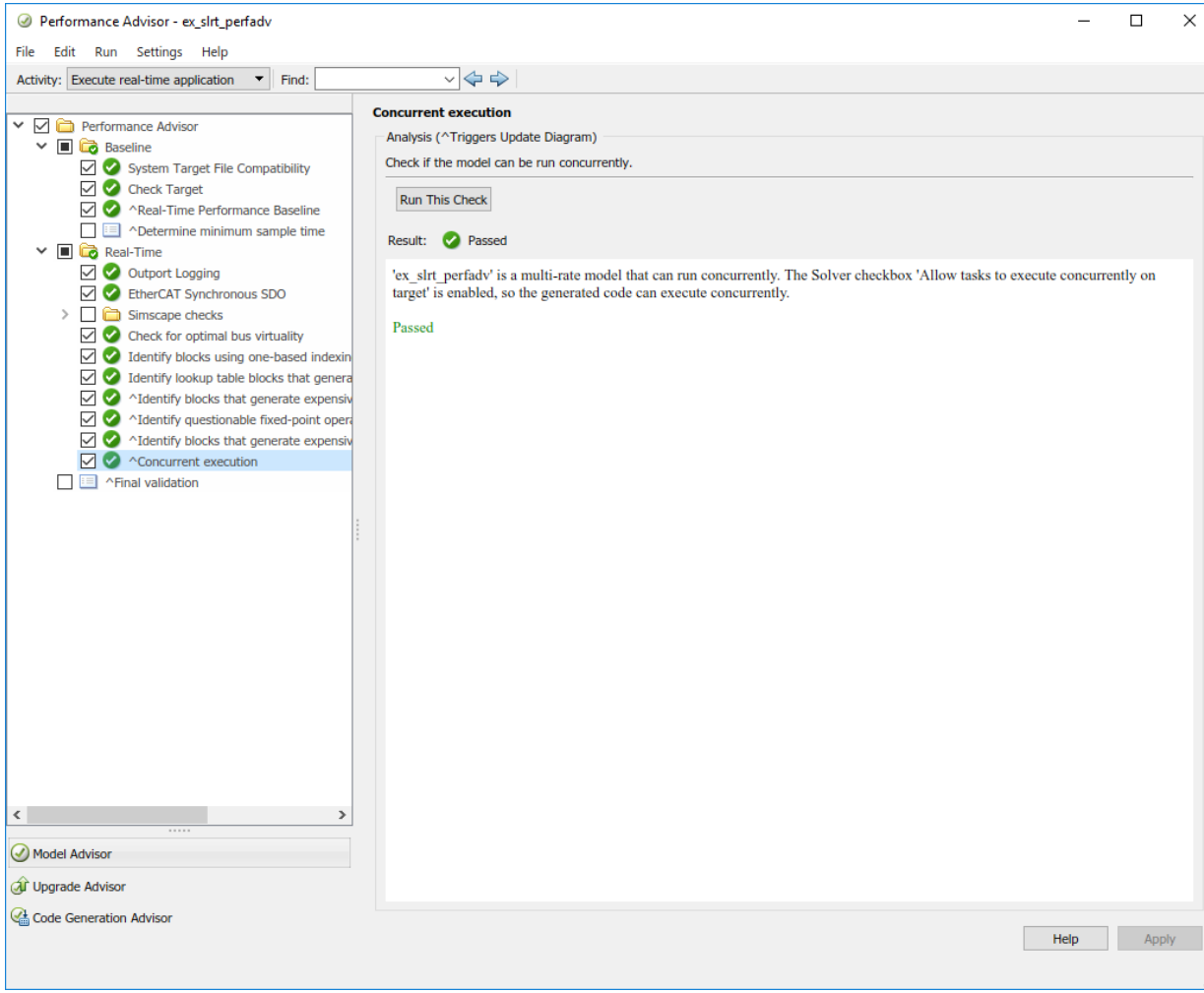


The model is a multirate model running on a multicore target computer, but it is not configured to use more than one core.

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:

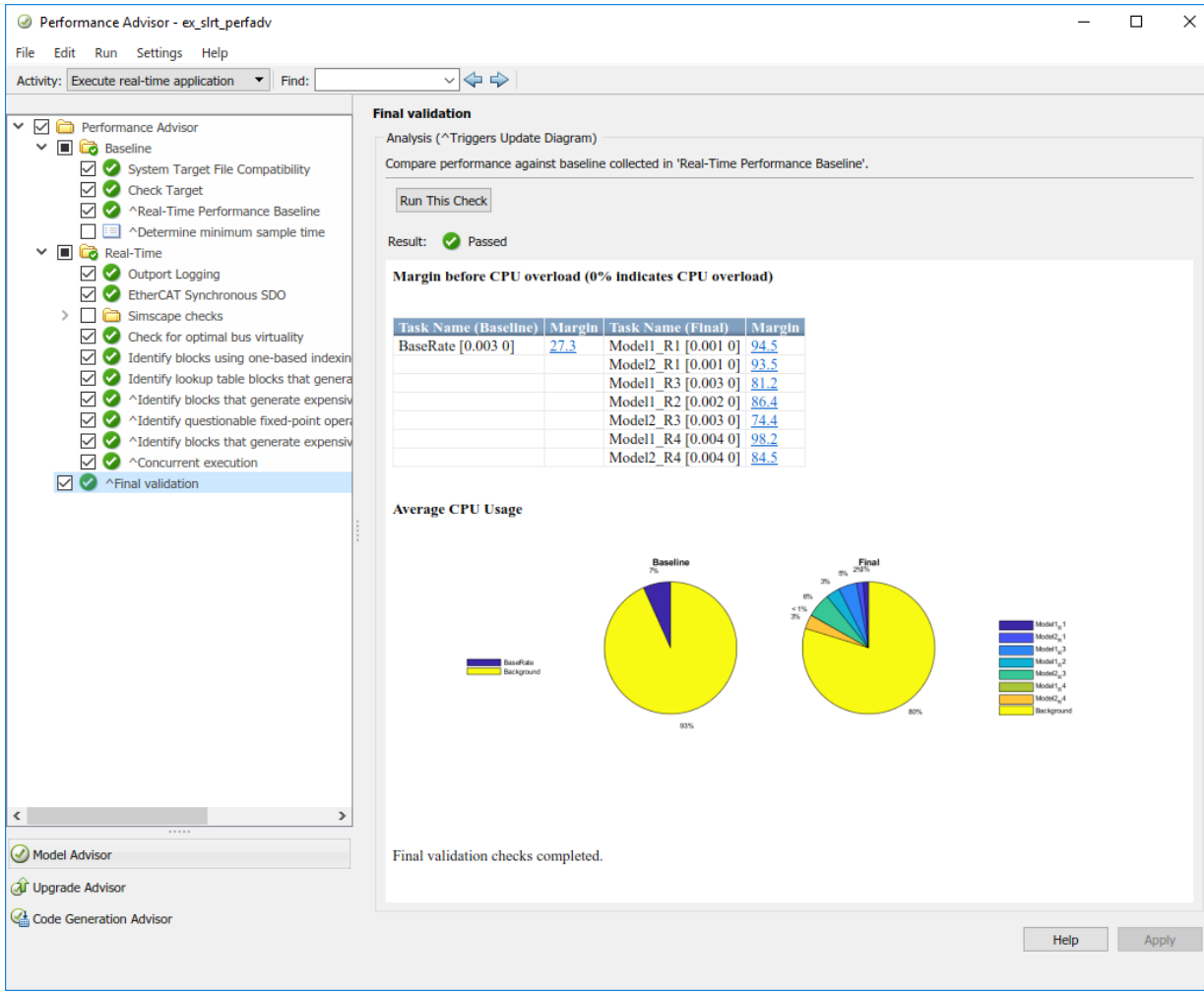
```
Ts = 0.001
```

8. Save ex_slrt_perfadv and its reference subsystems.

Final Validation

The final validation check tests whether model `ex_slrt_perfadv` works after you performed real-time performance optimizations.

1. Select the **Final validation** check box, and then click **Run this check**.



2. To investigate further improvements, see Execution Profiling for Real-Time Applications.

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.

The `MulticoreSupport` target setting is read-only and set to `'on'`. This property is obsolete. Attempting to set property to `'off'` generates and error. Single-core target computers are not supported.

To build and download multirate models on your multicore target computer:

1. Open your model in Simulink Editor.
2. 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.

3. Select the **Ensure data integrity during data transfer** check box of the Rate Transition block parameters.
4. 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.

5. In Simulink Editor, select **View > Model Explorer**.
6. In Simulink Model Explorer, right-click in the **Model Hierarchy** pane and select **Configuration > Add configuration for concurrent execution**
7. In the new configuration, select **Solver**.
8. Check **Enable concurrent tasking**.
9. Click **Configure Tasks**.

See Also

Rate Transition

More About

- “Multicore Programming with Simulink” (Simulink)

Limits on Sample Time

The sample time you can assign to your model is limited by the kernel and by the complexity of your model.

The kernel enforces lower and upper bounds on sample time:

Mode	Lower Bound	Upper Bound
Interrupt	`8e-6` s	`10` s
Polling	`5e-7` s	`10` s

In the **Solver** node in the Configuration Parameters dialog box, set **Fixed-step size** to a value within these bounds. If you set **Fixed-step size** to a value outside these bounds and attempt to build and download the real-time application, the application load fails with an error message.

At run time, if you attempt to set the sample time to a value outside these bounds, the kernel prints an error message.

Within these bounds, if you specify too short a sample time for the complexity of your model, the target computer can experience a CPU overload. To address this problem, use the following procedure:

1. To find the minimum sample time for your model, run `SimulinkRealTime.utils.minimumSampleTime` in the Command Window.
2. Change the value of **Fixed-step size** to a value slightly above the minimum sample time value.
3. Rebuild and download the model.

See Also

`SimulinkRealTime.utils.getConsoleLog` | `SimulinkRealTime.utils.minimumSampleTime`

More About

- “Troubleshoot Overloaded CPU from Executing Real-Time Application” on page 24-5

- “Execution Modes” on page 7-2

CPU Overload Options

In this section...
“Option Behavior” on page 10-14 “Violation of xPCMaxOverloads” on page 10-16 “Violation of xPCMaxOverloadLen” on page 10-17 “Violation of xPCStartupFlag” on page 10-17

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. An overload is registered every time an execution step cannot be executed because a previous step is running.

For example, assume that your model sample time is `1` ms, but running a particular model step takes `3.1` ms. This model step causes the kernel to skip three steps and causes three overloads.

Typically, the Simulink Real-Time kernel halts model execution when it encounters a CPU overload. However, some real-time applications can tolerate several CPU overloads without significant loss of data, for example during start up. For such applications, you can allow a specified number and configuration of CPU overloads. You do this using the `TLCOptions` settings `xPCMaxOverloads`, `xPCMaxOverloadLen`, and `xPCStartupFlag`.

Note Allowing the target computer CPU to overload can cause incorrect results, especially for multirate models. Use these TLC command-line options only for diagnosis. When your diagnosis is complete, turn off these options.

Option Behavior

If your real-time application causes a CPU overload, it finishes the current execution step and ignores timer interrupts. At the end of the execution step, the kernel compares the CPU overload count to the limits defined by `xPCMaxOverloads` and `xPCMaxOverloadLen`. If the count does not exceed the limits, the application executes at the next step. Otherwise it stops.

The limits are:

- `xPCMaxOverloads` — Number of acceptable overloads during a real-time application execution.

  When `xPCMaxOverloads` is set to a value, the Simulink Real-Time software stops execution with a CPU overload at the next overload within the same application execution. For example, if `xPCMaxOverloads` is set to `3`, the software stops with a CPU overload at the fourth overload in the same application execution.

  The default value of `0` means that overloads are registered on the first overload.
- `xPCMaxOverloadLen` — Number of acceptable overloads, in units of sample time, within the same execution step.

  When `xPCMaxOverloadLen` is set to a value, the software stops execution with a CPU overload at the next overload within the same execution step. For example, if `xPCMaxOverloadLen` is set to `2`, the software stops execution with a CPU overload at the third overload within the same execution step.

  The default value of `0` means that overloads are registered on the first overload within the same execution step.

  Specify a value that is less than or equal to the value for `xPCMaxOverloads`. If `xPCMaxOverload` is set to a value, for example `4`, and `xPCMaxOverloadLen` is not defined, the real-time application stops if one of following occurs:

  - The cumulative overloads since execution start is greater than `4`.
  - One execution step has two overloads.
- `xPCStartupFlag` — Number of executions of the model at start up.

  `xPCStartupFlag` temporarily disables CPU overload checking during the first few model execution steps. After the model finishes the first `xPCStartupFlag` steps, the software reenables CPU overload checking, which takes effect for the next execution of the model.

  The default value of `1` means that overloads are ignored on the first step. If `xPCMaxOverloads` and `xPCMaxOverloadLen` are not set, their default setting determines the software response to overloads.

`xPCMaxOverloads` and `xPCMaxOverloadLen` both count overloads, but over different time spans. `xPCMaxOverloads` counts the CPU overloads that were seen so far in the real-time application execution. `xPCMaxOverloadLen` counts the overloads that were seen within one execution step.

The three options interact. When the Simulink Real-Time kernel runs the model, it compares the number of CPU overloads to 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`.

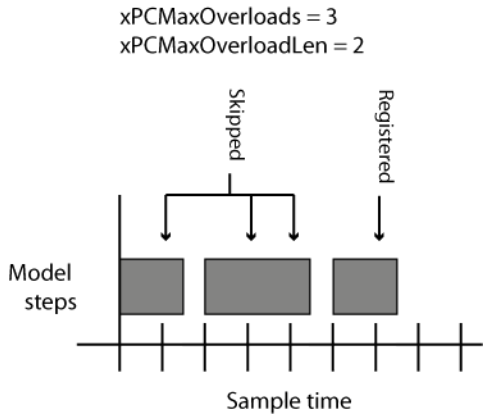
```
set_param('xpcosc','TLCOptions','-axPCMaxOverloads=30
          -axPCOverLoadLen=2 -axPCStartupFlag=5')
```

With these settings, 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 CPU overloads per model step.

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

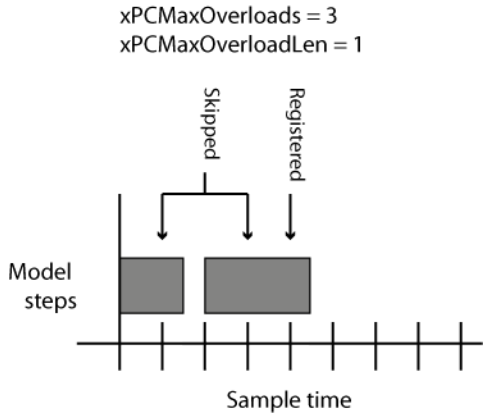
Violation of xPCMaxOverloads

Assume that `xPCMaxOverloads` is `3` and `xPCMaxOverloadLen` is `2`. The software tolerates the first three overloads and stops executing at the fourth. The number of overloads exceeds the maximum number allowed for real-time execution.



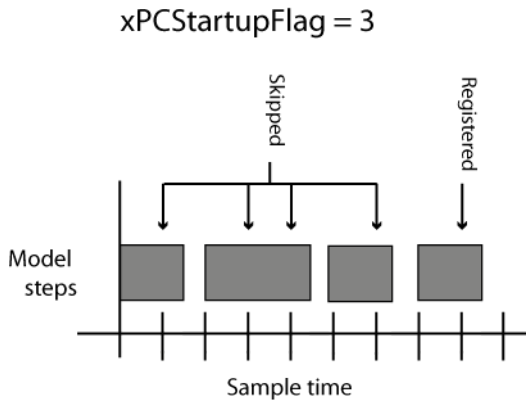
Violation of xPCMaxOverloadLen

Assume that `xPCMaxOverloads` is `3` and `xPCMaxOverloadLen` is `1`. The software tolerates the first two overloads and stops executing at the third. The second step execution is longer than the maximum allowed overload length of one sample time.



Violation of xPCStartupFlag

Assume that `xPCStartupFlag` is `3`. The kernel ignores CPU overloads for the first three time steps and stops executing on the first overload in the next time step.



See Also

“TLC Command-Line Options” | Get Overload Counter | Set Overload Counter

Execution Profiling for Real-Time Applications

This example shows how 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:

- In the Configuration Parameters for the model, enable the collection of function execution time data during execution.
- Build, download, and execute the model.
- Start and stop the profiler.
- Display the profiler data.

Profiling slightly increases the execution time of the real-time application.

Configure Real-Time Application for Function Execution Profiling

In this section, the model is `dxpcmds6t`. To open this model, open the subsystem models first:

- open_system(fullfile(matlabroot, '\toolbox', '\rtw', '\targets', '\xpc', '\xpcdemos', '\dxpcmds_ref1'))
- open_system(fullfile(matlabroot, '\toolbox', '\rtw', '\targets', '\xpc', '\xpcdemos', '\dxpcmds_ref2'))
- open_system(fullfile(matlabroot, '\toolbox', '\rtw', '\targets', '\xpc', '\xpcdemos', '\dxpcmds6t'))

1. Open model `dxpcmds6t`.
2. In the top model, open the Configuration Parameters dialog box. Select **Code Generation >> Verification**.
3. Select the **Measure function execution times** check box. The **Measure task execution time** check box is checked and locked.

- Value — 0
- DataType — double
- Storage Class — ExportedGlobal

5. In a similar manner, add Simulink parameters w_open and w_close. Because these parameters are in the `ex_slrt_slt_osc_Harness1` model workspace as model parameters, you access them by name directly, without model hierarchy.

6. Save the model.

Step 6. Prepare Test Assessment Steps

1. Open the Test Assessment block

2. Add these parameters to the Parameter symbol list:

- Criterion
- w_open
- w_close

3. To add a step, in the **Step** column, move the cursor to the top row, click **Add step after**, and type:

`CheckSetting`

4. Right-click step `CheckSetting` and set the **When decomposition** check box.

5. 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.

6. Right-click substep `Hi when` and set the **When decomposition** check box.

7. To substep `Hi when`, add substep:

```
HiCheck when ((et >= w_open) && (et <= w_close))
verify((abs(Int1) >= abs(SigGen) * (1.0 - Criterion)) && ...
   (abs(Int1) <= abs(SigGen) * (1.0 + Criterion)));
```

The when expression selects the time window for testing the acceptance criterion. The verify command tests the acceptance criterion.

8. In a similar manner, to step `CheckSetting`, add substep:

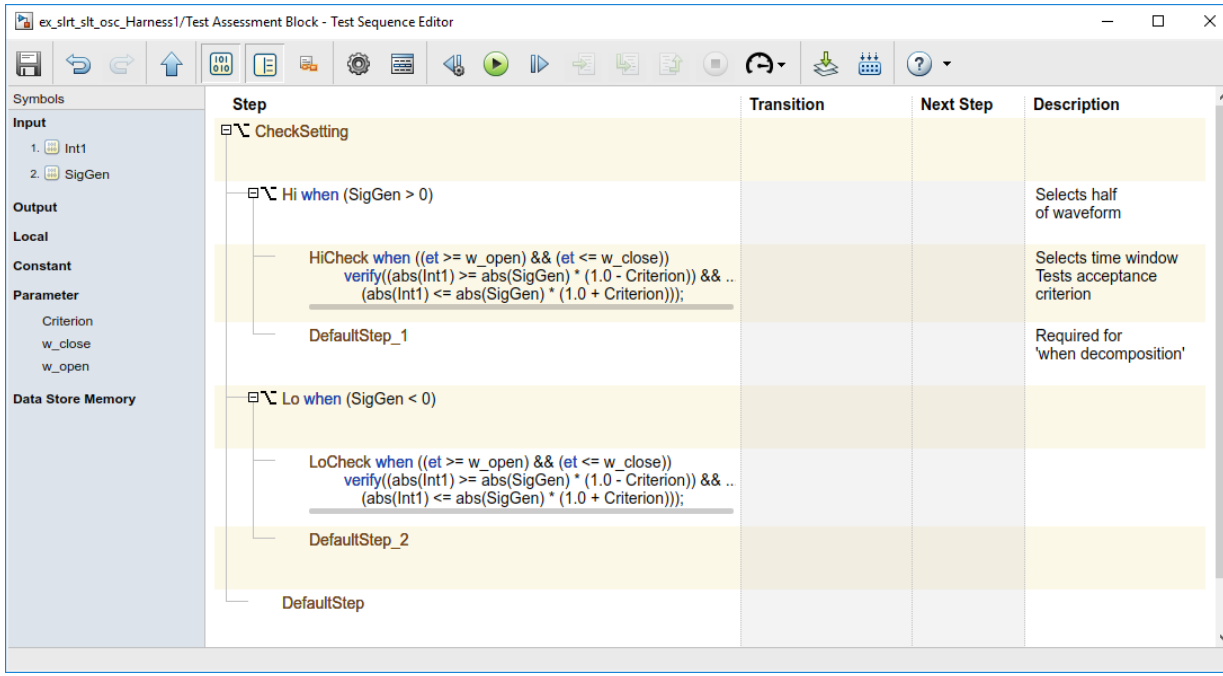
```
Lo when (SigGen < 0)
```

9. To substep `Lo when`, add substep:

```
LoCheck when ((et >= w_open) && (et <= w_close))
verify((abs(Int1) >= abs(SigGen) * (1.0 - Criterion)) && ...
   (abs(Int1) <= abs(SigGen) * (1.0 + Criterion)));
```

10. Right-click substep `Lo when` and set the **When decomposition** check box.

11. To satisfy the requirements of **When decomposition**, remove the default `Run` step and insert `DefaultStep` substeps after steps `CheckSetting`, `Hi when`, and `Lo when`. **When decomposition** requires at least two steps at each level of nesting, and one nondecomposed step at the end of each list of steps.



Step 7. Initialize Test Suite

1. Click on the `ex_slrt_slt_osc` subsystem.
2. Open **Analysis > Test Manager**.
3. Select **New > Test File**.
4. Name the test file `realtimetest`.
5. Right-click the test file and select **New > Real-Time Test**.
6. In the new real-time test dialog box, enter `Simulation` in the **Test Type** field.
7. Click **Create**.
8. Rename the new test suite to `realtimesuite`.
9. Rename the new test case to `frequencysweep`.

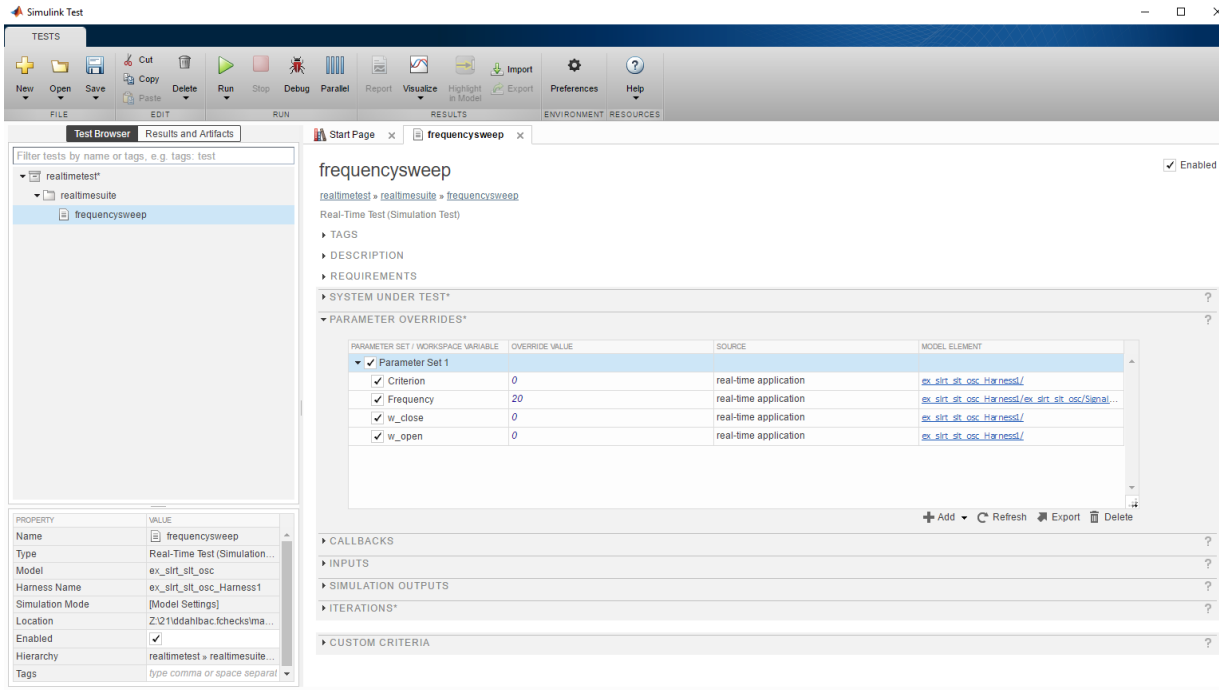
Step 8. Initialize System Under Test

1. In Test Manager, select node **frequencysweep**.
2. Select tab **System Under Test**.

3. Set **Load Application From** to `Model`.
4. Set **Model** to `ex_slrt_slt_osc`.
5. Set **Target Computer** to `TargetPC1`.
6. In tab **Test Harness**, set **Harness** to `ex_slrt_slt_osc_Harness1`.
7. In tab **Simulation Settings Overrides**, select the **Stop Time** check box.
8. Take the defaults for the other fields.

Step 9. Initialize Parameter Overrides

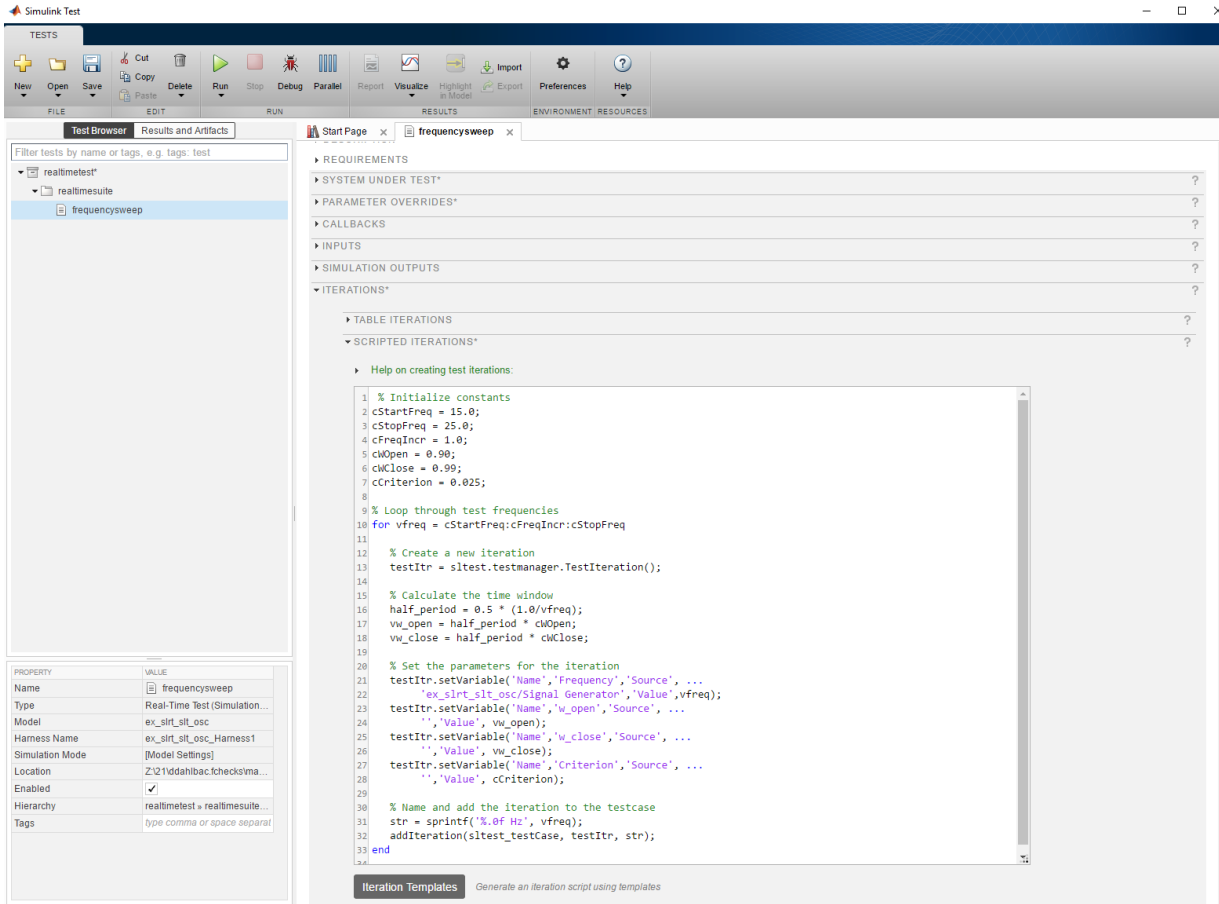
1. In Test Manager, select tab **Parameter Overrides**.
2. Click the **Add** button. A dialog box opens containing a list of parameters. If parameters are not visible, click the **Refresh** line at the top of the dialog box. The refresh builds the model and uploads the model and block parameters from `ex_slrt_slt_osc_Harness1` and `ex_slrt_slt_osc`.
3. Open **Parameter Set 1** and select the **Criterion**, **Frequency**, **w_close**, and **w_open** check boxes. Leave the other check boxes cleared.



Step 10. Create Scripted Iterations

1. In Test Manager, select tab **Iterations > Scripted Iterations**.
2. In the text box, enter the following code. To resize the **Scripted Iterations** text box, click and drag the lower-right corner of the box.

```
% Initialize constants
cStartFreq = 15.0;
cStopFreq = 25.0;
cFreqIncr = 1.0;
cWOpen = 0.90;
cWClose = 0.99;
cCriterion = 0.025;

% Loop through test frequencies
for vfreq = cStartFreq:cFreqIncr:cStopFreq

  % Create a new iteration
  testItr = sltest.testmanager.TestIteration();

  % Calculate the time window
  half_period = 0.5 * (1.0/vfreq);
  vw_open = half_period * cWOpen;
  vw_close = half_period * cWClose;

  % Set the parameters for the iteration
  testItr.setVariable('Name','Frequency','Source', ...
       'ex_slrt_slt_osc/Signal Generator','Value',vfreq);
  testItr.setVariable('Name','w_open','Source', ...
       '','Value', vw_open);
  testItr.setVariable('Name','w_close','Source', ...
       '','Value', vw_close);
  testItr.setVariable('Name','Criterion','Source', ...
       '','Value', cCriterion);

  % Name and add the iteration to the testcase
  str = sprintf('%.0f Hz', vfreq);
  addIteration(sltest_testCase, testItr, str);
end
```



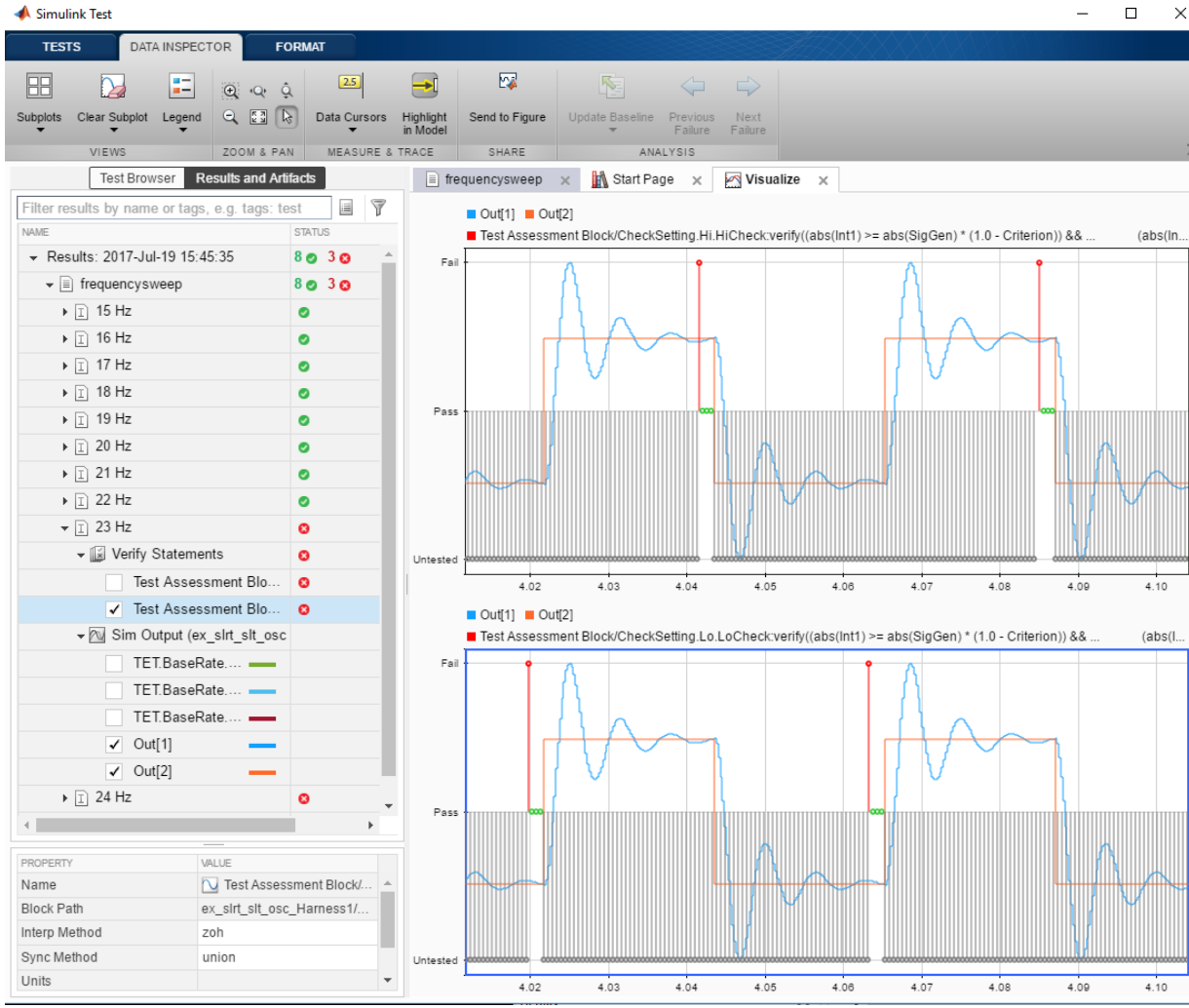
Step 11. Run Test

1. Build and download `ex_slrt_slt_osc` to the target computer.
2. In Test Manager, on the toolbar, click the **Run** button.
3. To view test results, in the left column, click **Results and Artifacts**. In this case, the test failed at iteration **23 Hz**.
4. To view the failing results, open nodes **23 Hz > Verify Statements** and **23 Hz > Sim Output (ex_slrt_slt_osc)**.

Step 12. Display Results

1. In the Simulation Data Inspector pane, select the **Layout** button.
2. Select two horizontal displays.
3. In the top Simulation Data Inspector display, select the two `Out` check boxes and the top **Test Assessment** check box. This assessment corresponds to the `HiCheck` substep.
4. In the bottom display, select the two `Out` check boxes and the bottom **Test Assessment** check box. This assessment corresponds to the `LoCheck` substep.
5. Click the **Zoom in Time** button and select the range `4.00-4.1`.

In the top display, the vertical red line near `4.04` followed by a horizontal green line shows that the `HiCheck` test failed briefly before succeeding. In the bottom display, the vertical red spike near `4.02` followed by a horizontal green line shows that the `LoCheck` test failed briefly before succeeding.



See Also

Test Assessment | Test Sequence

More About

- “Test Models in Real Time” (Simulink Test)
- open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_slt_osc')))
- “Real-Time Testing” (Simulink Test)

Troubleshooting

Troubleshooting Basics

For questions or issues about your installation of the Simulink Real-Time product, refer to these guidelines and tips. For more specific troubleshooting solutions, go to the MathWorks® Support website:

```
www.mathworks.com/support/search_results.html?q=product:"Simulink
+Real-Time"
```

Troubleshoot with Confidence Test

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. To address these issues, follow this process:

1. Run the confidence test. For more information, see “Run Confidence Test on Configuration”.

   Run the confidence test as the first step in troubleshooting, and in validating your initial product installation and configuration.
2. If one or more tests fail, see the following information about the specific test:

   - “Test 1: Ping Target Computer with System Ping” on page 16-2
   - “Test 2: Ping Target Computer with slrtpingtarget” on page 16-4
   - “Test 3: Software Restart Target Computer” on page 16-5
   - “Test 4: Build and Download slrttestmdl” on page 16-7
   - “Test 5: Check Communication with Target Computer” on page 16-9
   - “Test 6: Download Prebuilt Real-Time Application” on page 16-10
   - “Test 7: Execute Real-Time Application” on page 16-11
   - “Test 8: Upload Logged Data and Compare Results” on page 16-12
3. Investigate the categorized troubleshooting sections for clues to the root cause of the issue.

   - To get information about the PCI boards in the target computer, call `SimulinkRealTime.target.getPCIInfo`.
   - To read the target computer console log, call `SimulinkRealTime.utils.getConsoleLog`.
4. If the tests run, but task execution time is slow or the CPU becomes overloaded, see `Real-Time Application Performance` in “Troubleshooting in Simulink Real-Time”.
5. For more information, refer to the following sources:

   - MathWorks Tech Support: `www.mathworks.com/support/search_results.html?q=product:"Simulink+Real-Time"`
   - MATLAB Answers: `www.mathworks.com/matlabcentral/answers/?term=Simulink+Real-Time`

- MATLAB Central: `www.mathworks.com/matlabcentral`

For Speedgoat hardware issues, contact Speedgoat Tech Support: `www.speedgoat.com/support`.

6 If these sources do not solve your issue, contact MathWorks Technical Support. See “Find Simulink Real-Time Support” on page 25-2.

Confidence Test Failures

For questions or issues that you have while using the Simulink Real-Time product, see these guidelines and tips. For specific troubleshooting solutions, refer to the MathWorks Support website:

`www.mathworks.com/support/search_results.html?q=product:"Simulink +Real-Time"`.

- “Test 1: Ping Target Computer with System Ping” on page 16-2
- “Test 2: Ping Target Computer with slrtpingtarget” on page 16-4
- “Test 3: Software Restart Target Computer” on page 16-5
- “Test 4: Build and Download slrttestmdl” on page 16-7
- “Test 5: Check Communication with Target Computer” on page 16-9
- “Test 6: Download Prebuilt Real-Time Application” on page 16-10
- “Test 7: Execute Real-Time Application” on page 16-11
- “Test 8: Upload Logged Data and Compare Results” on page 16-12

Test 1: Ping Target Computer with System Ping

If you are using a network connection, this test is a standard system `ping` to your target computer.

1. At a Windows command prompt, type the IP address of the target computer:

   ```
   ping xxx.xxx.xxx.xxx
   ```

   Review the messages.

   If the window displays a message similar to this message, 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<10 ms TTL=59
   ```

   If the window displays this 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, determine whether you entered the required IP and gateway addresses in Simulink Real-Time Explorer:

   a. In the MATLAB Command Window, type `slrtexplr`.
   b. In the **Targets** pane, expand the target computer node.
   c. On the toolbar, click the **Target Properties** button .
   d. Select **Host-to-Target communication**.
   e. Check that the **IP address**, **Subnet mask**, and **Gateway** text boxes contain the required values.
   f. Select **Boot configuration**.
   g. Click **Create boot disk**.
   h. Restart the target computer with the new kernel.

3. **Ping fails — Cables OK?**

   If `ping` fails, look for a faulty network cable or, if you are using a coaxial cable, a missing terminator.

4 **Ping fails — Simulink Real-Time properties OK?**

Check that you entered the required properties in Simulink Real-Time Explorer:

a In the MATLAB Command Window, type `slrtexplr`.

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

c On the toolbar, click the **Target Properties** button .

d Select **Host-to-Target communication**.

e Check that the **IP address**, **Subnet mask**, and **Gateway** text boxes contain the required values.

f Check that **Bus type** is set to **PCI** or **USB**, depending on the Ethernet adapter that you are using.

g Select **Boot configuration**.

h Click **Create boot disk**.

i Restart the target computer with the new kernel.

5 **Ping fails — Ethernet interface operating?**

Check that your Ethernet protocol interface is operating. For example, when the cable is connected to the Ethernet card, make sure that the green “ready” light goes on.

6 **Ping fails — Interference from firewall or antivirus software?**

Check that the development computer is not running a firewall or antivirus software sensitive to the Ethernet port that you are using. For more information, consult your IT department.

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`.

If this procedure does not solve your issue, continue with the tests in “Troubleshoot with Confidence Test” on page 15-2. If you still cannot solve your issue, see “Find Simulink Real-Time Support” on page 25-2.

Test 2: Ping Target Computer with slrtpingtarget

This test is a Simulink Real-Time ping to your target computer.

1. In the MATLAB Command Window, type:

   ```
   tg = SimulinkRealTime.target('argument-list')
   ```

   `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.

   Review the messages in the Command Window.

   If the communication link is functioning, you see a message that looks like this message:

   ```
   Target: TargetPC1
      Connected            = Yes
      Application          = loader
   ```

2. **Not connected — Bad target boot kernel?**

   If you do not get the preceding message, it is possible that you have a bad target boot kernel. Recreate the target boot kernel and restart the target computer with the new kernel. See “Target Computer Boot Methods”.

3. **Not connected — Target settings?**

   Use Simulink Real-Time Explorer to check the target settings. In particular, if test 1 passes but test 2 fails, check the IP address that you entered in the target settings.

If this procedure does not solve your issue, continue with the tests in “Troubleshoot with Confidence Test” on page 15-2. If you still cannot solve your issue, see “Find Simulink Real-Time Support” on page 25-2.

Test 3: Software Restart Target Computer

This test is a Simulink Real-Time command that attempts to restart your target computer. This error is not necessarily fatal because some otherwise functional target computers do not support software restart.

You must have already configured the target settings with Simulink Real-Time Explorer. See “PCI Bus Ethernet Setup” or “USB-to-Ethernet Setup”.

Note Do not modify the files installed with the Simulink Real-Time software. If you want to modify one of these files for your own purposes, copy the file and modify the copy.

1. In the MATLAB Command Window, type:

   ```
   slrttest('-noreboot')
   ```

   This command reruns the test without using the `reboot` command, and then displays the message:

   ```
   ### Test 3, Software reboot the target computer: ... SKIPPED
   ```

2. **Build Succeeded — Software restart supported?**

   Review the results of `Test 4, Build and download a Simulink Real-Time application using model slrttestmdl` performed without a software restart. If `slrttest` builds and loads the real-time application without producing an error message, it is possible that the target computer does not support the `reboot` command. In this case, restart by using a physical reset button.

3. **Build Failed — Example model modified?**

   To determine the cause of failure, in the Diagnostics Viewer and in the Command Window, review the error messages. You can also open `slrttestmdl` and build and download it manually.

   If you directly or indirectly modify the `slrttestmdl` example model supplied with the product, test 3 is likely to fail. 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.

If this procedure does not solve your issue, continue with the tests in “Troubleshoot with Confidence Test” on page 15-2. If you still cannot solve your issue, see “Find Simulink Real-Time Support” on page 25-2.

Test 4: Build and Download slrttestmdl

This test attempts to build and download the model `slrttestmdl`.

1. To determine the cause of failure, in the Diagnostics Viewer and in the Command Window, review the error messages. You can also open `slrttestmdl` and build and download it manually.
2. **Build Failed — Compiler not supported?**

   Using `slrtgetCC`, check that you are using a supported compiler. Check that you can compile the blocks in the model with the given compiler and compiler version.

   If you did not explicitly specify a compiler by using `slrtsetCC`, the build procedure uses the compiler that you specified by using `mex -setup`. If the MEX compiler is not a supported Microsoft Visual C++ compiler, the build procedure halts with an error.
3. **Build Failed — Compiler path?**

   After installation, the Microsoft Visual C++ compiler components must be in the Microsoft Visual Studio folder. If you do not install the compiler at the required location, you can 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 this error:

   ```
   NMAKE: fatal error U1064: MAKEFILE not found and no target
   specified
   Stop.
   ```

   Check your compiler setup:

   a. In the Command Window, type:

      ```
      slrtsetCC('setup')
      ```

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

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

[1] Microsoft Visual C++ Compilers 2010 Professional in
    C:\Program Files (x86)\Microsoft Visual Studio 10.0
[2] Microsoft Visual C++ Compilers 2013 Team Explorer
    Language Pack in C:\Program Files (x86)\Microsoft Visual Studio 12.0


[0] None

Compiler:
```

b At the `Compiler` prompt, enter the number for the compiler that you want to use. For example, `1`.

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**.

If this procedure does not solve your issue, continue with the tests in “Troubleshoot with Confidence Test” on page 15-2. If you still cannot solve your issue, see “Find Simulink Real-Time Support” on page 25-2.

Test 5: Check Communication with Target Computer

This error occurs only when the target computer settings are out of date.

1. In the MATLAB Command Window, type `slrtexplr`.
2. In the **Targets** pane, expand the target computer node.
3. On the toolbar, click the **Target Properties** button .
4. Select **Host-to-Target communication** and make the required changes to the communication properties.
5. Select **Boot configuration**.
6. Set the required **Boot mode**.

   For information on boot options, see “Target Computer Boot Methods”.
7. Click **Create boot disk**
8. Restart 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`.

If this procedure does not solve your issue, continue with the tests in “Troubleshoot with Confidence Test” on page 15-2. If you still cannot solve the issue, see “Find Simulink Real-Time Support” on page 25-2.

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. Check that tests 1–5 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”).

If this procedure does not solve your issue, continue with the tests in “Troubleshoot with Confidence Test” on page 15-2. If you still cannot solve your issue, see “Find Simulink Real-Time Support” on page 25-2.

Test 7: Execute Real-Time Application

This test executes a real-time application (`slrttestmdl`) on the target computer. If you change the `slrttestmdl` model start time to something other than `0`, such as `0.001`, this test fails. 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.

If this procedure does not solve your issue, continue with the tests in “Troubleshoot with Confidence Test” on page 15-2. If you still cannot solve your issue, see “Find Simulink Real-Time Support” on page 25-2.

Test 8: Upload Logged Data and Compare Results

This test executes a real-time application (`slrttestmdl`) on the target computer. If you change the `slrttestmdl` model (for example, if you remove the Outport block), this test can fail.

Note Do not modify the files installed with the Simulink Real-Time software. If you want to modify one of these files for your own purposes, copy the file and modify the copy.

1. 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, check that you have updated the target boot kernel for this release. See “Install Simulink Real-Time Software Updates” on page 25-3.

If this procedure does not solve your issue, continue with the tests in “Troubleshoot with Confidence Test” on page 15-2. If all tests are successful but these do not solve your issue, try the solutions available in “Troubleshooting in Simulink Real-Time”. If you still cannot solve your issue, see “Find Simulink Real-Time Support” on page 25-2.

17

Development Computer Configuration

Troubleshoot Halted Boot Drive Creation

My development computer MATLAB interface halts while creating a Simulink Real-Time boot disk or network boot image.

What This Means

This issue occurs when the location for writing the boot disk or image is not accessible or you do not have write access to the location.

Try This

Try these workarounds to identify the issue that has halted boot drive creation.

Try Another Development Computer Drive

Use another development computer drive to recreate the Simulink Real-Time boot drive or network boot image.

Try Command as Administrator

Log in on the development computer as administrator to recreate the Simulink Real-Time boot drive or network boot image.

Try Checking the Target Computer Drive

Check that the development computer drive is accessible. If it is not accessible, contact Speedgoat support about replacing the development computer drive.

See Also

External Websites

- http://www.speedgoat.com/support

Target Computer Configuration

- “Troubleshoot Target Computer Stack Size” on page 18-2
- “Troubleshoot Target Computer Ethernet and MAC Address Information” on page 18-4

Troubleshoot Target Computer Stack Size

I seek to learn the best amount of target computer memory to use for the stack.

What This Means

The stack size on the target computer affects real-time application performance. For some applications, it is important to analyze the current available stack size and minimum available stack size, so you can adjust the stack size to make it appropriate for the application.

Try This

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. Use Minimum Available Stack Size 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.

   Target computer memory for the real-time application executable, the kernel, and other uses is limited to a maximum of 4 GB.

4. Adjust the stack size of the real-time threads by using a `TLCOptions` setting.

   For example, to set the stack size for real-time application `xpcosc` to `4096` kBytes, in the MATLAB Command Window, type:

   ```
   set_param('xpcosc','TLCOptions','-axPCModelStackSizeKB=4096')
   ```

See Also

“TLC Command-Line Options” | Current Available Stack Size | Minimum Available Stack Size

Troubleshoot Target Computer Ethernet and MAC Address Information

I seek to get Ethernet address information and MAC address information from the target computer for configuring blocks in models.

What This Means

To use Ethernet blocks in Simulink Real-Time models, you configure the blocks with Ethernet address information and MAC address information. Typically, you set block parameters to configure this information.

Try This

To configure a two-target-computer Ethernet network, collect the Ethernet address information that is listed in this table. This information applies to Ethernet, EtherCAT®, PTP, TCP, and UDP.

EtherCAT and PTP have protocol-specific requirements. For information about the protocols that the card supports, see the network card documentation.

Connector	Card Name	Boot	Comm	Bus	Slot	Function	PTP	EtherCAT	MAC Address
TargetPC1 A									
TargetPC1 B									
TargetPC2 A									
TargetPC2 B									

Set Up Hardware and Software for Information Gathering

Assemble these components:

- One Windows development computer with an Ethernet card.

- Two Speedgoat target machines, each with two Ethernet cards installed that support network booting.
- One Ethernet switch.
- Three crossover Ethernet cables.

Starting with target computer `TargetPC1`, perform these steps:

1. On the development computer, run MATLAB, and then Simulink Real-Time Explorer.
2. Connect the development computer to the Ethernet switch.
3. Connect the target computer to the Ethernet switch by using a randomly chosen Ethernet connector on the target computer.
4. In the Explorer **Targets** pane, create a `SimulinkRealTime.target` object for the target computer.
5. In the **Properties** pane for the target computer, assign an **IP address** value.
6. Set **Target driver** to `auto`.
7. Set **Boot mode** to `Network`.

Repeat steps 3–7 for the other target computer,`TargetPC2`.

Collect PCI Address and MAC Address Information

Before collecting PCI address and MAC address information, perform the steps in “Set Up Hardware and Software for Information Gathering” on page 18-4.

To perform the steps in this section, MATLAB and Simulink Real-Time Explorer must be running.

Starting with target computer `TargetPC1`, fill in the table.

1. Click the **Properties** node representing the target computer.
2. Click the **Reset** button next to the **MAC address** field, and then click **Create boot disk**.
3. Start the target computer and select the target computer name in the Simulink Real-Time Network Boot dialog box.

   If the target computer fails to start, replace the target computer Ethernet card with an Ethernet card that supports network booting.
4. In Simulink Real-Time Explorer, from the **Properties** pane for the target computer, copy down the MAC address that is displayed in the **MAC address** field.

This MAC address is the address of the card that caused the target computer to start. For this MAC address, in the **Boot** column, enter Y.

5 To verify that the development computer can communicate with the target computer by using this Ethernet card, type:

```
tg = slrt('TargetPC1')
```

If `slrt` returns something like:

```
Target: TargetPC1
   Connected            = No
```

Enter N in the **Comm** column for this MAC address.

Stop the target computer, switch the Ethernet switch cable to another Ethernet connector on the target computer, and start again from step 2.

6 If `slrt` returns something like:

```
Target: TargetPC1
   Connected            = Yes
   Application          = loader
```

Enter Y in the **Comm** column next to the MAC address. Enter the PCI bus, slot, and function numbers displayed in the output log area in the **Bus**, **Slot**, and **Function** columns for this MAC address.

This Ethernet connector is the connector that you use to start the target computer and to communicate with it from the development computer.

7 In the Command Window, type:

```
getPCIInfo(tg, 'ethernet')
```

```
List of installed PCI devices:

Intel                    82579LM
     Bus 0, Slot 25, Function 0, IRQ 3
     Ethernet controller
     VendorID 0x8086, DeviceID 0x1502, SubVendorID 0x15bd,
          SubDeviceID 0x100a
     Released in: R2012b
     Notes: Intel 8254x Gigabit Ethernet series

Intel                    82574L
```

```
Bus 5, Slot 0, Function 0, IRQ 10
Ethernet controller
VendorID 0x8086, DeviceID 0x10d3, SubVendorID 0x15bd,
     SubDeviceID 0x100a
Released in: R2010a
Notes: Intel 8254x Gigabit Ethernet series
```

Record the device name (for example, `Intel 82574L`) for each PCI bus, slot, and function triplet.

If the device supports the PTP protocol, enter `Y` in the **PTP** column. Otherwise, enter `N`. For more information, see “Prerequisites, Limitations, and Unsupported Features”.

If the device supports the EtherCAT protocol, enter `Y` in the **EtherCAT** column. Otherwise, enter `N`. For more information, see “Ethernet Chip Sets Compatible with EtherCAT”.

Repeat steps 1–7 for the other target computer, `TargetPC2`. At the end of the process, the table looks something like this table.

Connector	Device Name	Boot	Comm	Bus	Slot	Function	PTP	EtherCAT	MAC Address
TargetPC1 A	Intel 82579LM	Y	Y	0	25	0	N	N	00:01:29:55:3C:BB
TargetPC1 B	Intel 82574L	Y	N	5	0	0	Y	N	00:01:29:55:3C:BA
TargetPC2 A	Intel 82574L	Y	N	52	0	0	Y	N	68:05:CA:31:B9:EF
TargetPC2 B	Intel 82541GI_LF	Y	Y	16	4	0	N	N	90:E2:BA:17:5D:15

See Also

Simulink Real-Time Explorer | `SimulinkRealTime.target.getPCIInfo` | `slrt`

More About

- “Ethernet”

- “EtherCAT”
- “IEEE 1588 Precision Time Protocol”
- “TCP”
- “Real-Time UDP”

Link Between Development and Target Computers

- “Troubleshoot Communication Failure with Target Computers” on page 19-2
- “Troubleshoot Communication Timeout with Target Computers” on page 19-4
- “Troubleshoot Communication Timeout with Target Computers and Multiple Ethernet Cards” on page 19-6

Troubleshoot Communication Failure with Target Computers

Some issue is causing the communications link to fail between the development computer and the target computer.

What This Means

The development computer communicates with the target computer using an Ethernet communications link. Various issue can interfere with this link.

Try This

To test the communication link between the development and target computers, use these MATLAB commands from the development computer

- `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.

Boot Kernel Mismatch

Link errors can occur if the target computer is running an old Simulink Real-Time boot kernel that is not synchronized with the Simulink Real-Time release installed on the development computer. Recreate the target boot kernel for each new release.

Firewall Interference

Link errors can occur if you have an active firewall in your system. To work around this issue, add the MATLAB interface to the firewall exception list.

Multiple Ethernet Card Configurations

Link errors can occur if multiple Ethernet cards or chips are installed in the target computer, see “Troubleshoot Communication Timeout with Target Computers and Multiple Ethernet Cards” on page 19-6.

Troubleshoot Communication Timeout with Target Computers

Some issue is causing communications between the development computer and target computer to timeout.

What This Means

If the communication link between the development and target computers is broken or misconfigured, the link times out after about 5 seconds. Before continuing, check that you have followed the instructions outlined in “System Configuration”.

Try This

To identify timeout issues, follow these steps:

1. In the MATLAB Command Window, type `slrtexplr`.
2. In the **Targets** pane, expand the target computer node.
3. On the toolbar, click the **Target Properties** button .
4. Select **Host-to-Target communication** and make the required changes to the communication properties.
5. Select **Boot configuration** and click **Create boot disk**.
6. Restart the target computer and try downloading the real-time application again.
7. Sometimes, the download is complete even though you get a timeout error. To detect this condition, wait until the target display shows:

   ```
   System:initializing application finished.
   ```
8. In the MATLAB Command Window, type `slrtpingtarget`.

   If `slrtpingtarget` finds a working connection between the development and target computers, the response is something like:

   ```
   ans =

   success
   ```
9. To set the connection between host and target, in the Command Window, type:

```
tg=slrt;
ping(tg,'reset')
```

10 Right-click the target computer and select **Connect**.

If the connection resumes, the connection is working. If the connection times out consistently for a particular model, increase the amount of time allowed before time out.

Read More

By default, the development computer times out after about 5 seconds if the target computer does not respond after you establish a connection. 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, to increase the value to 20 s, enter `20`, and then build and download the model.
- At the real-time application level, set the `CommunicationTimeOut` property to the timeout value that you want. For example, to increase the value to 20 seconds:

  ```
  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. The development computer waits the full 20 seconds only if a download actually fails.

Troubleshoot Communication Timeout with Target Computers and Multiple Ethernet Cards

Some issue related to multiple Ethernet cards in the target computer is causing communications between the development computer and target computer to timeout.

What This Means

The Simulink Real-Time product supports multiple Ethernet cards and chips. If your target computer has more than one of these cards or chips installed, it is possible to experience timeout problems. For example, suppose that you are using the network boot option to start target computer A. When the development computer starts the target computer, it associates the target computer IP address with the Media Access Control (MAC) address of Ethernet adapter A. Then, suppose that the target computer BIOS connects the target computer to Ethernet adapter B. In this case, the Simulink Real-Time software cannot connect the development and target computers because they are connected to different Ethernet controllers.

Try This

If your Speedgoat target computer has multiple Ethernet adapters of the same type, use one of the following procedures to determine which Ethernet adapter the software found. For support with resolving this possible issue with your Speedgoat target machine, contact Speedgoat support:

`www.speedgoat.com/support`.

Check Communications for Network Boot

If you are using the network boot option to start the target computer, follow the procedure in “Ethernet Card Selection by Index” on page 5-15.

Check Communications for Non-Network Boot

If you are not using the network boot option to start the target computer, do the following:

1. Switch the network cable to the other Ethernet port and restart the target computer. Try to communicate with the target computer from the development computer.

2 If you can communicate using this Ethernet port, use this port to connect the development computer to the target computer.

Target Computer Boot Process

Model Compilation

- “Troubleshoot Model Links to DLLs” on page 21-2
- “Troubleshoot Build Error for Accelerator Mode” on page 21-3

Troubleshoot Model Links to DLLs

Some model build issues are caused by linking to dynamic link libraries (.dll).

What This Means

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. Building a real-time application from a model with links to one or more DLLs produces a build error.

Try This

When you build your models, check that you link only to static link libraries. When you compile with Simulink Real-Time S-functions, linking to static libraries is not an issue. Each static library must be self contained; not dependent on another external library.

See Also

More About

- “Build Support for S-Functions” (Simulink Coder)

Troubleshoot Build Error for Accelerator Mode

I get a build error when building a model in accelerator mode or rapid accelerator mode when the model contains Simulink Real-Time blocks (for example, model blocks that represent hardware).

What This Means

Simulink Real-Time does not support accelerator mode or rapid accelerator mode simulation of models with blocks that represent hardware. For example, if you open the `xpcEnetDemo1Rx` model, change the Simulink mode to rapid accelerator, and run the model, Simulink displays the following error:

```
### Build procedure for model: 'xpcEnetDemo1Rx' aborted due to an error.
Unable to build a standalone executable to simulate the model
'xpcEnetDemo1Rx' in rapid accelerator mode.
```

This error occurs because accelerator mode and rapid accelerator mode produce compiled code that runs on the development computer, not on Simulink Real-Time target computer. Any blocks that access hardware report a build error if compiled for accelerator mode or rapid accelerator mode.

Try This

To eliminate the error, change the simulation mode to normal mode or external mode.

See Also

More About

- “How Acceleration Modes Work” (Simulink)
- “Simulink Real-Time Options Pane”

Real-Time Application Execution

- “Troubleshoot Missing or Unreadable Crash Information” on page 22-2
- “Troubleshoot Unexpected Measured Sample Time Value” on page 22-4
- “Troubleshoot Changed Sample Time at Run Time Affects Results” on page 22-6
- “Troubleshoot Unexpected Measured Stop Time Value” on page 22-7

Troubleshoot Missing or Unreadable Crash Information

I get missing or unreadable development computer crash information and errors for the `SimulinkRealTime.crashInfo` function.

What This Means

Target computers save crash data to their hard disk after a fatal error. Use the `SimulinkRealTime.crashInfo` function to view this information.

Caution After a fatal error, do not restart the computer manually by using the boot or power switch. A manual restart prevents the computer from saving the crash data.

Twenty seconds after a fatal error, the target computer restarts itself and saves the crash data on the target computer hard drive. When the computer is running again, you can call the `SimulinkRealTime.crashInfo` function from the development computer to retrieve the crash data.

If an error occurs when you call the `SimulinkRealTime.crashInfo` function, the target computer can display error:

```
Error: -9:file not found
```

And, the development computer can display the error:

```
Could not open target file c:\SLRTCRB.bin
```

Try This

If you see one of the messages in the examples, look for one of these causes and try the related workaround.

Try Waiting for the Target Computer Restart

If you restarted the target computer manually by using the boot or power switch, the manual restart prevented the target computer from generating crash information. Try waiting for the target computer to restart itself instead if another crash occurs.

Try Checking the Boot Kernel

If the target computer restarted with a different kernel from the one that it was running when it experienced the fatal error, the different kernel can prevent the target computer from generating crash information. For example, suppose that you install DOS Loader on the target computer. If you start the computer with a USB drive that you remove afterward, and the computer has a fatal error, the restart uses DOS Loader. Try to make sure that the target computer can boot from the same kernel (not a different kernel) if another crash occurs.

Try Checking the Crash Message

If the target computer restarts itself after a fatal error but does not print a message referring to `SimulinkRealTime.crashInfo`, the target computer does not retain information in memory. The target computer does not retain information in memory from before a software restart. Try to make sure that you see a message referring to `SimulinkRealTime.crashInfo` after a crash occurs.

Try Checking the Target Computer Drive

If the target computer does not have a functioning hard drive (for example, it uses a RAM drive instead), the target computer does not retain crash information. Try checking whether the target computer hard drive is functional.

Try Checking the Crash Info File

If the target computer wrote data into a crash file, the `SimulinkRealTime.crashInfo` function fails if the file is unreadable. Try checking whether the crash information file is readable.

See Also

`Crash Info` | `SimulinkRealTime.utils.getConsoleLog`

Troubleshoot Unexpected Measured Sample Time Value

Some issue is causing the measured sample time from the model to deviate from the requested sample time in the model.

What This Means

Sometimes the sample time that you measure from your model is not equal to the sample time that 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.

Some amount of error is common for most computers. The margin of error varies from machine to machine.

Most high-level operating systems, like Microsoft Windows or Linux®, occasionally insert extra long intervals to compensate for errors in the timer. 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, sometimes chips have inherent designs that produce residual jitters that can potentially change your system behavior. For example, some Intel Pentium chips produce residual jitters on the order of 0.5 microseconds from interrupt to interrupt.

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:

$$100 \times 10^{-6} \text{ s} \times 1.193 \times 10^{6} \text{ ticks/s} = 119.3 \text{ ticks}$$

The Simulink Real-Time software rounds this number to the nearest whole number, 119 ticks. The actual sample time is then:

$$119 \text{ ticks}/(1.193 \times 10^{6} \text{ ticks/s}) = 99.75 \times 10^{-6} \text{ s}$$
(99.75 microseconds)

Compared to the requested original sample time of 100 microseconds, this value is 0.25% faster.

Try This

You can use the calculated value for the number of ticks in the requested sample time to derive the expected measured sample time 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 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`). When the sample time is 1/10000, the deviation is 0.04%.

See Also

More About

- “Limits on Sample Time” on page 10-12

Troubleshoot Changed Sample Time at Run Time Affects Results

Some blocks produce incorrect results when their sample time is changed at run time.

What This Means

Some blocks produce incorrect results when you change their sample time at run time. If you include such blocks in your model, the software displays a warning message during model build.

Try This

To avoid incorrect results, change the sample time in the original model. Then, rebuild and download the model.

See Also

More About

- “Limits on Sample Time” on page 10-12

Troubleshoot Unexpected Measured Stop Time Value

Some issue is causing the measured stop time from the model to deviate from the requested stop time in the model.

What This Means

If you change the step size of a real-time application at run time, the real-time application sometimes executes 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 the 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 to an integral multiple. If you change the step size without rebuilding the model, Simulink Real-Time uses the new step size and the previously adjusted stop time. The resulting model sometimes executes for fewer steps than you expect.

See Examples

Suppose that a model has a stop time of `2.4` and a step size of `1`. At compilation time, Simulink Coder adjusts the stop time of the model to `2`. If you change the step size to `0.6` at run time but do not recompile the application, the expected number of steps is `4`. The actual number of steps is `3` because Simulink Real-Time uses the previously adjusted stop time of `2`.

Try This

To avoid unexpected stop time values, check that the original stop time (as specified in the model) is an integral multiple of the original step size.

See Also

More About

- “Control Real-Time Application at Target Computer Command Line” on page 9-2

Real-Time Application Signals

- “Troubleshoot Invalid File IDs on Target Computer” on page 23-2
- “Troubleshoot Missing Mux Block Output on Scope” on page 23-3

Troubleshoot Invalid File IDs on Target Computer

I get invalid file ID errors on the target computer.

What This Means

If you acquire signal data with a file scope, you can get `Error -10: Invalid File ID` on the target computer. This error occurs when the size of the signal data file exceeds the available space on the disk. The signal data is most likely corrupt and irretrievable. Delete the signal data file and restart the Simulink Real-Time system.

Try This

To prevent occurrences of invalid file IDs on the target computer, monitor the size of the signal data file as the scope acquires data. Stop data acquisition before the file size exceeds the available disk space.

For additional information, refer to the MathWorks Support website:

`www.mathworks.com/support/search_results.html?q=product:"Simulink +Real-Time"`.

See Also

Gain

External Websites

- www.mathworks.com/support/search_results.html?q=product:"Simulink+Real-Time"

Troubleshoot Missing Mux Block Output on Scope

Some issue is causing the Mux block output not to display on a real-time scope.

What This Means

When you connect Mux block output to a real-time Scope block, sometimes you cannot view the output from the Mux block. This issue occurs because the Mux block produces a virtual signal. The code optimizer removes the virtual signal during the build process for the real-time executable.

Try This

To address the virtual signal removal issue, insert a Gain block with the **Gain** parameter set to `1.0` at the input to the real-time Scope block.

See Also

Gain

More About

- “Virtual Signals” (Simulink)

Real-Time Application Performance

- “Troubleshoot Unsatisfactory Real-Time Performance” on page 24-2
- “Troubleshoot Overloaded CPU from Executing Real-Time Application” on page 24-5
- “Troubleshoot Task Execution Time” on page 24-7
- “Troubleshoot Failed Read of Profiling Data” on page 24-8
- “Troubleshoot Timeout During File System Access” on page 24-9

Troubleshoot Unsatisfactory Real-Time Performance

I am seeking recommended methods to improve unsatisfactory real-time application performance.

What This Means

Run-time performance and reduce the task execution time (TET) of a model depend on model design, target computer capacity, and target computer utilization.

Try This

You can improve run-time performance and reduce the task execution time (TET) of a model with various methods.

Try Running Performance Tools

Use the following performance tools:

- Run Performance Advisor from the Simulink **Analysis > Performance Tools** menu and apply the advice that it provides. See “Improve Performance of Multirate Model” on page 10-2 and “Sample Time and Throughput in Real-Time Applications” on page 10-28.
- Configure a real-time application for profiling, run it, and call `profile_slrt` to retrieve the results. Evaluate the results for potential improvements in the task and core distribution of the model. See “Execution Profiling for Real-Time Applications” on page 10-19.

Try Using Multicore Target Computer

You can improve run-time performance by configuring your model to take advantage of your multicore target computer:

1. Partition the model into subsystems according to the physical requirements of the system that you are modeling. Set the block sample rates within each subsystem to the slowest rate that meets the physical requirements of the system.
2. In the Configuration Parameters dialog box, on the **Solver** pane, select the check box for **Treat each discrete rate as a separate task**.
3. Select the `Allow tasks to execute concurrently on target` check box.

4. Click **Configure Tasks**, and then select the **Enable explicit model partitioning for concurrent behavior** check box.
5. Create tasks and triggers, and then explicitly assign subsystem partitions to the tasks. See “Partition Your Model Using Explicit Partitioning” (Simulink) and “Multicore Programming with Simulink” (Simulink).
6. In Simulink Real-Time Explorer, on the **Target settings** pane, check that you selected the **Multicore CPU** check box.
7. Run the real-time application.

Try Minimizing Model

You can improve run-time performance by minimizing your model to make more memory and CPU cycles available for the real-time application:

1. If the model contains many states (for example, more than 20 states), clear the **States** check box in the Configuration Parameters dialog box, on the **Data Import/Export** pane. You have now disabled state logging, making more memory available for the real-time application.
2. On the **Data Import/Export** pane, clear the **Time**, **States**, **Output**, **Final states**, and **Signal logging** parameters. You have now turned off data logging, making more CPU cycles available for calculating the model.
3. On the **Simulink Real-Time Options** pane, clear the **Monitor Task Execution Time** check box. You have now disabled TET logging for the real-time application.
4. On the **Solver** pane, increase **Fixed-step size (fundamental sample time)**. Executing with a short sample time can overload the CPU.
5. Use polling mode. See “Polling Mode” on page 7-3.
6. In Simulink Real-Time Explorer, on the **Target settings** pane, clear the **Graphics mode** check box to disable the target scope display.
7. Remove scopes from the model.
8. Eliminate or minimize target computer disk I/O in your model.
9. Reduce the number of I/O channels in the model.

Try Contacting Technical Support

For additional guidance, refer to the following sources:

- MathWorks Tech Support: `www.mathworks.com/support/search_results.html?q=product:"Simulink+Real-Time"`

- MATLAB Answers: `www.mathworks.com/matlabcentral/answers/?term=Simulink+Real-Time`
- MATLAB Central: `www.mathworks.com/matlabcentral`

For Speedgoat hardware issues, contact Speedgoat Tech Support: `www.speedgoat.com/support`.

Read More

"Find Simulink Real-Time Support" on page 25-2

See Also

More About

- "Sample Time and Throughput in Real-Time Applications" on page 10-28
- "Improve Performance of Multirate Model" on page 10-2
- "Sample Time and Throughput in Real-Time Applications" on page 10-28
- "Execution Profiling for Real-Time Applications" on page 10-19
- "Partition Your Model Using Explicit Partitioning" (Simulink)
- "Polling Mode" on page 7-3
- "Find Simulink Real-Time Support" on page 25-2
- "Multicore Programming with Simulink" (Simulink)

External Websites

- www.speedgoat.com/products
- http://www.speedgoat.com/support

Troubleshoot Overloaded CPU from Executing Real-Time Application

Some issue is producing a CPU overload when executing a real-time application.

What This Means

A CPU overload indicates that the CPU is unable to complete processing a model time step before restarting for the next time step. When this error occurs, the target object property `CPUoverload` changes from `none` to `detected`. One of the following can happen:

- The Simulink Real-Time kernel halts model execution.
- If you allow the overload, model execution continues until a predefined event occurs (see “Try Permitting CPU Overloads for Diagnosis” on page 24-6). If the model continues to run after a CPU overload, the time step lasts as long as the time required to finish the execution. This behavior delays the next time step.

For more information and test models, see `www.mathworks.com/matlabcentral/fileexchange/23507`.

Model design or target computer resources cause CPU overloads. Possible reasons are:

- The target computer is too slow or the model sample time is too small (see “Limits on Sample Time” on page 10-12).
- The model is too complex (algorithmic complexity).
- The model does disk I/O on the target computer hard drive.
- I/O latency, where each I/O channel used introduces latency into the system. I/O latency can cause the execution time to exceed the model time step.

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

Try This

The Simulink Real-Time kernel usually halts model execution when it encounters a CPU overload. You can configure the Simulink Real-Time model to allow CPU overloads. Use this capability to support long initializations and for overload diagnosis.

Try Permitting Long Initialization Time

For some real-time applications, normal initialization can extend beyond the first sample time. For such an application, use the `TLCOptions` property `xPCStartupFlag` with the smallest effective value, up to approximately `5`.

Try Permitting CPU Overloads for Diagnosis

During execution, hardware-specific factors can cause the real-time application to process data beyond the sample time. Use the `TLCOptions` properties `xPCMaxOverloads` and `xPCMaxOverloadLen` to diagnose and address this issue.

Note Allowing the target computer CPU to overload can cause incorrect results, especially for multirate models. Use these TLC command-line options only for diagnosis. When your diagnosis is complete, turn off these options.

See Also

`SimulinkRealTime.utils.getConsoleLog`

More About

- “CPU Overload Options” on page 10-14
- “TLC Command-Line Options”

Troubleshoot Task Execution Time

I seek to find out the task execution time (TET) for one sample step.

What This Means

Task execution time (TET) measures how long it takes the kernel to run for one base-rate time step.

Try This

For a multirate model, use the profiler to find out what the execution time is for each rate.

See Also

Related Examples

- “Open TET Monitor and View Status”

Troubleshoot Failed Read of Profiling Data

I get an error from the `SimulinkRealTime.target.getProfileData` function.

What This Means

A call of the `SimulinkRealTime.target.getProfileData` function produces an error for these reasons:

- The file does not exist in the expected location.
- The attempt to read the file causes an error.

Try This

To address this issue, try the following procedure:

1. Check that you have set the profiling options in **Code Generation** > **Verification**.
2. Build and download the real-time application.
3. Enter these commands in the Command Window:

```
tg = slrt;
startProfiler(tg);
start(tg);
pause(1);
stop(tg);
profiler_data = getProfilerData(tg)
```

See Also

Related Examples

- “Execution Profiling for Real-Time Applications” on page 10-19
- “Troubleshoot Timeout During File System Access” on page 24-9

Troubleshoot Timeout During File System Access

Some issue is causing a timeout during access to the target computer file system.

What This Means

While you are accessing the target computer file system to read or write a large data or log file, the connection between the development and target computer systems times out.

Try This

To address this problem, try increasing the communication timeout value. If using file scopes, try using simulation data inspector (SDI) instead to reduce file system I/O. If the file system access is failing during large file transfers, try using an ftp client for the transfer.

See Also

Related Examples

- “Troubleshoot Communication Timeout with Target Computers” on page 19-4
- “Troubleshoot Failed Read of Profiling Data” on page 24-8

Simulink Real-Time Support

- “Find Simulink Real-Time Support” on page 25-2
- “Install Simulink Real-Time Software Updates” on page 25-3

Find Simulink Real-Time Support

For support with Speedgoat target machines or I/O modules, contact Speedgoat support:

`www.speedgoat.com/support`.

For support on general MATLAB or Simulink issues, see MathWorks Support:

`www.mathworks.com/support`.

For support on Simulink Real-Time issues, see:

- Simulink Real-Time Support:

  `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`

After searching these resources, if you still cannot solve your issue:

1. Call function `SimulinkRealTime.getSupportInfo` to retrieve diagnostic information for your Simulink Real-Time configuration.

   `SimulinkRealTime.getSupportInfo` can record information that is sensitive to your organization. Review this information before disclosing it to MathWorks.
2. For online or phone support, contact the Simulink Real-Time Technical Team directly:

   `www.mathworks.com/products/simulink-real-time/expert-contact.html`.

Install Simulink Real-Time Software Updates

The general procedure for updating Simulink Real-Time is:

1. Navigate to the MathWorks download page:

   `www.mathworks.com/downloads`.
2. Navigate to the page for the Simulink Real-Time software version that you want. Download it to your development computer.
3. Install and integrate the new release software.

After updating Simulink Real-Time, to recreate your Simulink Real-Time target settings:

1. In the MATLAB Command Window, type `slrtexplr`.
2. On the **Targets** pane, expand the target computer node.
3. On the toolbar, click the **Target Properties** button .
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”).
5. Select **Boot configuration** and click **Create boot disk**.
6. Restart the target computer.
7. For each model that you want to execute, in Simulink Editor, from the **Code** menu, click **C/C++ Code > Build Model**.