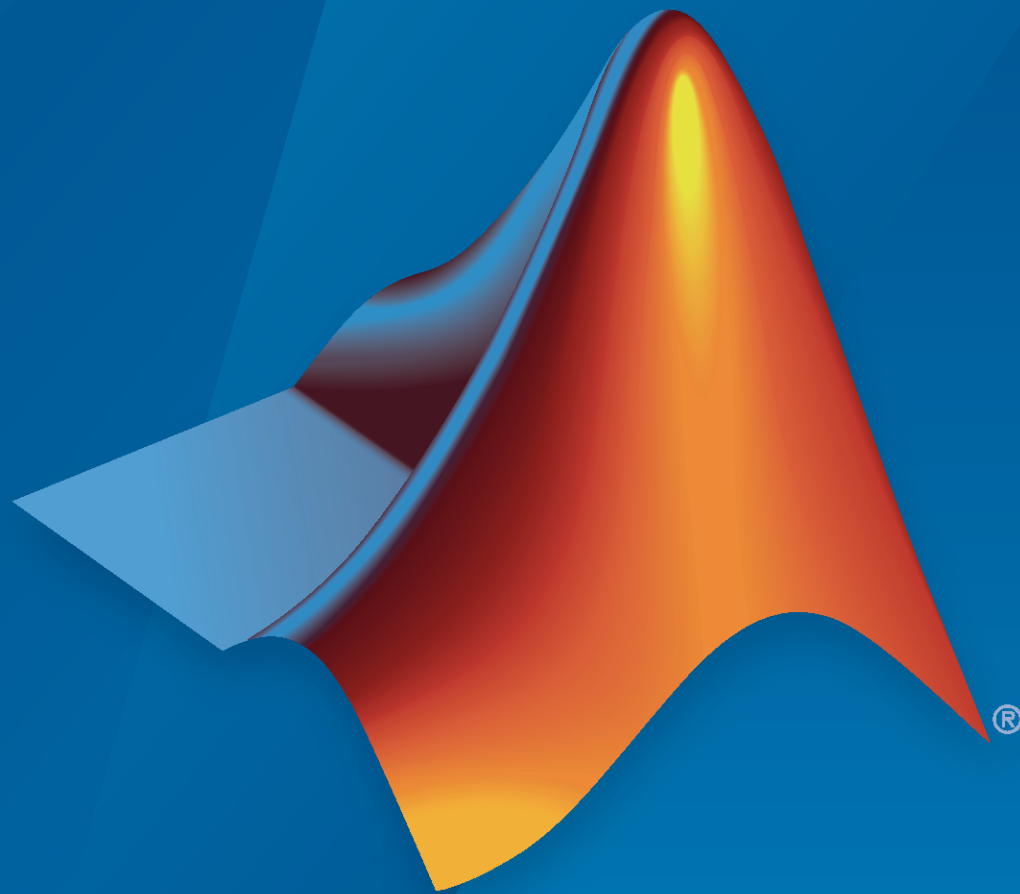


Simulink® Real-Time™

User's Guide



MATLAB® & SIMULINK®

R2020a



How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us



Phone: 508-647-7000



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

Simulink® Real-Time™ User's Guide

© COPYRIGHT 1999–2020 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) |
| September 2018 | Online only | Revised for Version 6.9 (Release 2018b) |
| March 2019 | Online only | Revised for Version 6.10 (Release 2019a) |
| September 2019 | Online only | Revised for Version 6.11 (Release 2019b) |
| March 2020 | Online only | Revised for Version 6.12 (Release 2020a) |

Model Architectures

| | |
|----------|--|
| | FPGA Models |
| 1 | |
| | Speedgoat FPGA Support with HDL Workflow Advisor 1-2 |
| | Introduction 1-2 |
| | Speedgoat FPGA IO Module Support 1-2 |
| | Workflow 1-3 |
| | FPGA Programming and Configuration 1-4 |
| | Interrupt Configuration 1-14 |
| | FPGA Domain Model 1-14 |
| | Simulink Real-Time Domain Model 1-14 |
| | FPGA Subsystem Plan 1-16 |
| | Target Device 1-16 |
| | FPGA Synchronization Mode 1-16 |
| | FPGA Inports and Outports 1-16 |
| | FPGA Clock Frequency 1-17 |
| | FPGA Deployment 1-17 |
| | FPGA Synchronization Modes 1-19 |
| | |
| | Functional Mockup Units and Simulink Real-Time |
| 2 | |
| | Apply Functional Mockup Units with Simulink Real-Time 2-2 |
| | Build Considerations 2-2 |
| | |
| | Third-Party Calibration Support |
| 3 | |
| | Calibrate Real-Time Application 3-2 |
| | Prepare ASAP2 Data Description File 3-3 |
| | Initial Setup 3-4 |

| | |
|--|-------------|
| Set Up Parameters | 3-5 |
| Set Up Signals | 3-5 |
| Set Up Lookup Tables | 3-6 |
| Generate Data Description File | 3-6 |
| Calibrate Parameters with Vector CANape | 3-8 |
| Prepare Project | 3-8 |
| Prepare Device | 3-8 |
| Configure Signals and Parameters | 3-8 |
| Perform Signal Measurement and Parameter Calibration | 3-9 |
| Vector CANape Limitations | 3-10 |
| Troubleshoot Vector CANape Operation | 3-11 |
| What This Issue Means | 3-11 |
| Try This Workaround | 3-11 |
| Calibrate Parameters with ETAS Inca | 3-12 |
| Prepare Database | 3-12 |
| Prepare Project | 3-12 |
| Prepare Workspace | 3-12 |
| Prepare Experiment | 3-12 |
| Configure Signals and Parameters | 3-12 |
| Perform Signal Measurement and Parameter Calibration | 3-13 |
| ETAS Inca Limitations | 3-14 |
| Troubleshoot ETAS Inca Operation | 3-15 |
| What This Issue Means | 3-15 |
| Try This Workaround | 3-15 |

Incorporating Fortran S-Functions

4

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

Real-Time Application Setup

Real-Time Application Environment

5

| | |
|---------------------------------------|------------|
| Default Target Computers | 5-2 |
|---------------------------------------|------------|

| | |
|---|-------------|
| Command-Line C Compiler Configuration | 5-3 |
| Command-Line Setup | 5-4 |
| Command-Line PCI Bus Ethernet Setup | 5-5 |
| PCI Bus Ethernet Protocol Hardware | 5-5 |
| Command-Line PCI Bus Ethernet Settings | 5-5 |
| Ethernet Card Selection by Index | 5-8 |
| Command-Line Ethernet Card Selection by Index | 5-9 |
| Command-Line Target Computer Settings | 5-11 |
| Command-Line Target Computer Boot Methods | 5-12 |
| Command-Line Network Boot Method | 5-13 |
| Command-Line Standalone Boot Method | 5-15 |
| Command-Line Standalone Settings | 5-15 |
| Real-Time Application Build | 5-15 |
| Real-Time Application Transfer and Boot Configuration | 5-16 |

Signals and Parameters

6

| | |
|--|-------------|
| Signal Monitoring Basics | 6-3 |
| Monitor Signals with Simulink Real-Time Explorer | 6-4 |
| Monitor Signals with MATLAB Language | 6-7 |
| Instrument a Stateflow Subsystem | 6-9 |
| Configure Stateflow States as Test Points | 6-9 |
| Monitor Stateflow States with Simulink Real-Time Explorer | 6-10 |
| Signal Group Monitoring Formats | 6-12 |
| Monitor Stateflow States with MATLAB Language | 6-13 |
| Animate Stateflow Charts with Simulink External Mode | 6-14 |
| Signal Tracing Basics | 6-16 |
| Simulink Real-Time Scope Usage | 6-17 |
| Target Scope Usage | 6-18 |
| Configure Real-Time Target Scope Blocks | 6-19 |
| Create Target Scopes with Simulink Real-Time Explorer | 6-24 |

| | |
|---|-------------|
| Configure Scope Sampling with Simulink Real-Time Explorer | 6-29 |
| Trigger Scopes with Simulink Real-Time Explorer | 6-32 |
| Freerun Triggering | 6-32 |
| Software Triggering | 6-32 |
| Signal Triggering | 6-34 |
| Scope Triggering | 6-36 |
| Configure Target Scopes with Simulink Real-Time Explorer | 6-39 |
| Configure Target Scopes with MATLAB Language | 6-42 |
| Create Signal Groups with Simulink Real-Time Explorer | 6-45 |
| Host Scope Usage | 6-48 |
| Configure Real-Time Host Scope Blocks | 6-49 |
| Create Host Scopes with Simulink Real-Time Explorer | 6-52 |
| Set Up Model | 6-52 |
| Configure Host Scope | 6-52 |
| View Host Scope | 6-53 |
| Configure the Host Scope Viewer | 6-56 |
| Trace Signals with Simulink External Mode | 6-58 |
| Inspect Simulink® Real-Time™ Data with Simulation Data Inspector | 6-61 |
| Stream Signal Data from Target Computer to Simulation Data Inspector | 6-65 |
| Trace or Log Data with the Simulation Data Inspector | 6-68 |
| Set Up Model | 6-68 |
| Set Up the Simulation Data Inspector | 6-68 |
| View Simulation Data | 6-69 |
| External Mode Usage | 6-72 |
| Signal Logging Basics | 6-73 |
| File Scope Usage | 6-74 |
| Configure Real-Time File Scope Blocks | 6-76 |
| Create File Scopes with Simulink Real-Time Explorer | 6-81 |
| Configure File Scopes with Simulink Real-Time Explorer | 6-85 |
| Log Signal Data into Multiple Files | 6-89 |

| | |
|--|--------------|
| Log Signal Data with Outport Blocks and Simulink Real-Time Explorer | 6-92 |
| Data Logs | 6-92 |
| Configure the Model for Data Logging | 6-93 |
| Log the Data | 6-93 |
| Download and Plot the Data | 6-93 |
| Log Signal Data with Outport Block and MATLAB Language | 6-97 |
| Data Logs with SimulinkRealTime.target Properties | 6-97 |
| Data Logs with the Simulation Data Inspector and Data Profiler ... | 6-98 |
| Configure the Model for Data Logging | 6-98 |
| Log the Data | 6-99 |
| Download and Plot the Data | 6-99 |
| Signal Logging Buffer Size | 6-103 |
| Configure File Scopes with MATLAB Language | 6-104 |
| Tune Parameters with Simulink Real-Time Explorer | 6-107 |
| Set Up Host Scope | 6-107 |
| Initial Values | 6-107 |
| Updated Values | 6-108 |
| Create Parameter Groups with Simulink Real-Time Explorer | 6-111 |
| Tune Parameters with MATLAB Language | 6-113 |
| Tune Parameters with Simulink External Mode | 6-115 |
| Tuning with Batch Mode and Update All Parameters | 6-115 |
| Save and Reload Parameters with MATLAB Language | 6-117 |
| Save the Current Set of Real-Time Application Parameters | 6-117 |
| Load Saved Parameters to a Real-Time Application | 6-117 |
| List Parameter Values Stored in a File | 6-118 |
| Tunable Block Parameters and Tunable Global Parameters | 6-119 |
| Tunable Parameters | 6-119 |
| Inlined Parameters | 6-119 |
| Tuning in External Mode | 6-120 |
| Tuning with Simulink Real-Time Explorer | 6-120 |
| Tuning with MATLAB Language | 6-120 |
| Tune Inlined Parameters with Simulink Real-Time Explorer | 6-122 |
| Configure Model to Tune Inlined Parameters | 6-122 |
| Initial Value | 6-124 |
| Updated Value | 6-126 |
| Tune Inlined Parameters with MATLAB Language | 6-128 |
| Tune Parameter Structures with Simulink Real-Time Explorer ... | 6-129 |
| Create Parameter Structure | 6-129 |
| Replace Block Parameters with Parameter Structure Fields | 6-130 |
| Tune Parameters in a Parameter Structure | 6-130 |
| Save and Load Parameter Structure | 6-132 |

| | |
|--|--------------|
| Tune Parameter Structures with MATLAB Language | 6-134 |
| Create Parameter Structure | 6-134 |
| Replace Block Parameters with Parameter Structure Fields | 6-135 |
| Tune Parameters in a Parameter Structure | 6-135 |
| Save and Load Parameter Structure | 6-136 |
| Define and Update Inport Data | 6-137 |
| File Dependencies | 6-137 |
| Map Inport to Use Square Wave | 6-137 |
| Update Inport to Use Sawtooth Wave | 6-139 |
| Define and Update Inport Data with MATLAB Language | 6-142 |
| File Dependencies | 6-142 |
| Map Inport to Use Square Wave | 6-142 |
| Update Inport to Use Sawtooth Wave | 6-143 |
| Inport Data Mapping Limitations | 6-146 |
| Display and Filter Hierarchical Signals and Parameters | 6-147 |
| Hierarchical Display | 6-147 |
| Filtered Display | 6-148 |
| Grouped Display | 6-149 |
| Display and Filter Hierarchical Signals and Parameters (tech preview) | 6-151 |
| Hierarchical Display | 6-151 |
| Filtered Display | 6-152 |
| Grouped Display | 6-153 |
| Troubleshoot Signals Not Accessible by Name | 6-155 |
| What This Issue Means | 6-155 |
| Try This Workaround | 6-155 |
| Troubleshoot Parameters Not Accessible by Name | 6-156 |
| What This Issue Means | 6-156 |
| Try This Workaround | 6-156 |
| Troubleshoot Instance-Specific Parameters Not Saved | 6-157 |
| What This Issue Means | 6-157 |
| Try This Workaround | 6-157 |
| Troubleshoot Instrument Label Not Present | 6-158 |
| What This Issue Means | 6-158 |
| Try This Workaround | 6-158 |
| Troubleshoot Internationalization Issues | 6-159 |
| What This Issue Means | 6-159 |
| Try This Workaround | 6-159 |
| Internationalization Issues | 6-160 |

7

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

Real-Time Application Execution

Execution with User Interface Models

8

| | |
|---|-----|
| 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 | 8-5 |
| Creating a Real-Time Application Model | 8-6 |
| Marking Block Parameters | 8-6 |
| Marking Block Signals | 8-7 |

Execution Using the Target Computer Command Line

9

| | |
|--|-----|
| 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 | 9-3 |
| Alias Commands at Target Computer Command Line | 9-4 |
| Find Signal and Parameter Indexes | 9-4 |

Tuning Performance

10

| | |
|---|-------|
| Improve Performance of Multirate Model | 10-2 |
| Multicore Processor Configuration | 10-9 |
| Limits on Sample Time | 10-10 |
| CPU Overload Options | 10-11 |
| Option Behavior | 10-11 |

| | |
|--|--------------|
| Violation of xPCMaxOverloads | 10-12 |
| Violation of xPCMaxOverloadLen | 10-13 |
| Violation of xPCStartupFlag | 10-13 |
| Execution Profiling for Real-Time Applications | 10-15 |
| Reduce Build Time for Simulink Real-Time Referenced Models .. | 10-20 |
| Sample Time and Throughput in Real-Time Applications | 10-22 |
| Real-Time Performance Factors | 10-22 |
| Resources | 10-22 |
| Improving Performance with Concurrency | 10-23 |
| Additional Optimizations | 10-38 |

Execution with MATLAB Scripts

Real-Time Applications and Scopes in the MATLAB Interface

11

| | |
|---|--------------|
| Real-Time Application Objects | 11-2 |
| Create Real-Time Application Objects | 11-2 |
| Display Application Object Properties | 11-3 |
| Set Real-Time Application Object Property Values | 11-3 |
| Get Real-Time Application Object Property Values | 11-4 |
| Use Real-Time Application Object Functions | 11-4 |
| Real-Time Scope Objects | 11-5 |
| Display Scope Object Properties for One Scope | 11-6 |
| Display Scope Object Properties for Multiple Scopes | 11-6 |
| Set Scope Property Values | 11-6 |
| Get Scope Property Values | 11-7 |
| Use Scope Object Functions | 11-8 |
| Acquire Signal Data with File Scopes | 11-9 |
| Acquire Signal Data into Dynamically Named Files | 11-10 |
| Scope Trigger Configuration | 11-12 |
| Pretriggering and Posttriggering of Scopes | 11-13 |
| Trigger One Scope with Another Scope | 11-15 |
| Scope-Triggered Data Acquisition | 11-15 |
| Trigger Sample Setting | 11-18 |
| Minimize Data Gaps with Two Scopes | 11-22 |

Logging Signal Data with File System Objects

12

| | |
|---|-------------|
| File System Basics | 12-2 |
| Using SimulinkRealTime.fileSystem Objects | 12-4 |
| Copying Files from the Target Computer to the Development Computer | 12-5 |
| Copying Files from the Development Computer to the Target Computer | 12-5 |
| Accessing File Systems on a Specific Target Computer | 12-6 |
| Reading the Contents of a File on the Target Computer | 12-6 |
| Removing a File from the Target Computer | 12-8 |
| Getting a List of Open Files on the Target Computer | 12-8 |
| Getting Information About a File on the Target Computer | 12-9 |
| Getting Information About a Disk on the Target Computer | 12-9 |

Deploy the MATLAB Application as a Standalone Executable

13

| | |
|---|-------------|
| MATLAB Runtime Setup | 13-2 |
| Deploy MATLAB Application to Control Real-Time Application | 13-3 |
| Prerequisites | 13-3 |
| Package the MATLAB Application | 13-3 |
| Run the MATLAB Application | 13-4 |

Automated Test with Simulink Test

14

| | |
|---|-------------|
| Test Real-Time Application | 14-2 |
|---|-------------|

Troubleshooting

Simulink Real-Time Examples

15

| | |
|---|-------------|
| Parameter Tuning and Data Logging | 15-2 |
| Signal Tracing With a Host Scope in Freerun Mode | 15-6 |

| | |
|---|---------------|
| Signal Tracing Using Software Triggering | 15-10 |
| Signal Tracing Using Signal Triggering | 15-14 |
| Signal Tracing Using Scope Triggering | 15-18 |
| Signal Tracing With a Target Scope | 15-23 |
| Pre- and Post-Triggering of a Host Scope | 15-27 |
| Time- and Value-Equidistant Data Logging | 15-31 |
| Frame Signal Processing | 15-37 |
| Spectrum Analyzer | 15-38 |
| Simple Client Application With the .NET API | 15-43 |
| Concurrent Execution on Simulink® Real-Time™ | 15-49 |
| Standalone User Interface using the MATLAB® Compiler™ | 15-56 |
| Add App Designer Instrument Panel App to Tank Model | 15-59 |
| Add m-Script Instrument Panel App to Tank Model | 15-67 |
| Add App Designer App to Inverted Pendulum Model | 15-75 |
| Triggering Scope Instruments | 15-79 |
| Asynchronous Events | 15-80 |
| EtherCAT® Communication with Beckhoff® Analog IO Slave Devices EL3062 and EL4002 | 15-81 |
| EtherCAT® Communication with Beckhoff® Digital IO Slave Devices EL1004 and EL2004 | 15-86 |
| EtherCAT® Communication - Motor Velocity Control with Accelnet™ Drive | 15-91 |
| EtherCAT® Communication - Motor Position Control with an Accelnet™ Drive and Beckhoff® Analog IO Devices | 15-96 |
| Generate ENI Files for EtherCAT Devices | 15-101 |
| Digital I/O with Speedgoat FPGA Board | 15-110 |
| PLL-Based Interrupt Generation from FPGA Input | 15-117 |
| IEEE® 1588™ Precision Time Protocol - Execution Synchronization | 15-127 |
| IEEE® 1588™ Precision Time Protocol - Clock Synchronization | 15-134 |

| | |
|--|---------------|
| Real-Time Transmit and Receive over Ethernet | 15-141 |
| Filtering on MAC Address | 15-144 |
| Filtering on EtherType | 15-148 |
| Ethernet Rx Block Filtering | 15-152 |
| Simple ASCII Encoding/Decoding Loopback Test (With Baseboard Blocks) | 15-157 |
| ASCII Encoding/Decoding Loopback Test | 15-158 |
| ASCII Encoding/Decoding Loopback Test (With Baseboard Blocks) | 15-160 |
| ASCII Encoding/Decoding Resync Loopback Test | 15-162 |
| ASCII Encoding/Decoding Resync Loopback Test (With Baseboard Blocks) | 15-164 |
| Binary Encoding/Decoding Loopback Test | 15-166 |
| Binary Encoding/Decoding Loopback Test (With Baseboard Blocks) | 15-167 |
| Binary Encoding/Decoding Resync Loopback Test | 15-168 |
| Binary Encoding/Decoding Resync Loopback Test (With Baseboard Blocks) | 15-169 |
| Read CPU Temperature on Simulink® Real-Time™ | 15-170 |
| Target to Target communication using TCP | 15-171 |
| Target to Host Transmission using UDP | 15-177 |
| Target to Target Transmission using UDP | 15-182 |
| Apply Simulink Real-Time Model Template to Create Real-Time Application | 15-188 |
| Data Logging With Simulation Data Inspector (SDI) | 15-190 |
| Basic J1939 Communication over CAN | 15-194 |

Troubleshooting

Troubleshooting Basics

16

| | |
|---|------|
| Troubleshoot with Confidence Test | 16-2 |
|---|------|

Confidence Test Failures

17

| | |
|--|-------|
| Test 1: Ping Target Computer with System Ping | 17-2 |
| Test 2: Ping Target Computer with slrtpingtarget | 17-4 |
| Test 3: Software Restart Target Computer | 17-5 |
| Test 4: Build and Download slrttestmdl | 17-6 |
| Test 5: Check Communication with Target Computer | 17-7 |
| Test 6: Download Prebuilt Real-Time Application | 17-8 |
| Test 7: Execute Real-Time Application | 17-9 |
| Test 8: Upload Logged Data and Compare Results | 17-10 |

Development Computer Configuration

18

| | |
|---|------|
| Troubleshoot Halted Boot Drive Creation | 18-2 |
| What This Issue Means | 18-2 |
| Try This Workaround | 18-2 |

Target Computer Configuration

19

| | |
|---|------|
| Troubleshoot Target Computer Stack Size | 19-2 |
| What This Issue Means | 19-2 |
| Try This Workaround | 19-2 |

| | |
|--|-------------|
| Troubleshoot Target Computer Ethernet and MAC Address Information | 19-3 |
| What This Issue Means | 19-3 |
| Try This Workaround | 19-3 |

20 | **Link Between Development and Target Computers**

| | |
|---|-------------|
| Troubleshoot Communication Failure with Target Computers | 20-2 |
| What This Issue Means | 20-2 |
| Try This Workaround | 20-2 |
| Troubleshoot Communication Timeout with Target Computers . . . | 20-4 |
| What This Issue Means | 20-4 |
| Try This Workaround | 20-4 |
| Troubleshoot Communication Timeout with Target Computers and Multiple Ethernet Cards | 20-6 |
| What This Issue Means | 20-6 |
| Try This Workaround | 20-6 |
| Troubleshoot Communication Failure Through Firewall | 20-7 |
| What This Issue Means | 20-7 |
| Try This Workaround | 20-7 |
| Troubleshoot Network Boot Failure Through Firewall | 20-8 |
| What This Issue Means | 20-8 |
| Try This Workaround | 20-8 |

21 | **Target Computer Boot Process**

22 | **Model Compilation**

| | |
|--|-------------|
| Troubleshoot Model Links to DLLs | 22-2 |
| What This Issue Means | 22-2 |
| Try This Workaround | 22-2 |
| Troubleshoot Build Error for Accelerator Mode | 22-3 |
| What This Issue Means | 22-3 |
| Try This Workaround | 22-3 |
| Troubleshoot Referenced Model with Global Data Stores | 22-4 |
| What This Issue Means | 22-4 |
| Try This Workaround | 22-4 |

Real-Time Application Execution

23

| | |
|--|------|
| Troubleshoot Missing or Unreadable Crash Information | 23-2 |
| What This Issue Means | 23-2 |
| Try This Workaround | 23-2 |
| Troubleshoot Unexpected Measured Sample Time Value | 23-4 |
| What This Issue Means | 23-4 |
| Try This Workaround | 23-4 |
| Troubleshoot Changed Sample Time at Run Time That Affects Results | 23-6 |
| What This Issue Means | 23-6 |
| Try This Workaround | 23-6 |
| Troubleshoot Unexpected Measured Stop Time Value | 23-7 |
| What This Issue Means | 23-7 |
| Try This Workaround | 23-7 |

Real-Time Application Signals

24

| | |
|---|------|
| Troubleshoot Invalid File IDs on Target Computer | 24-2 |
| What This Issue Means | 24-2 |
| Try This Workaround | 24-2 |
| Troubleshoot Missing Mux Block Output on Scope | 24-3 |
| What This Issue Means | 24-3 |
| Try This Workaround | 24-3 |

Real-Time Application Performance

25

| | |
|---|------|
| Troubleshoot Unsatisfactory Real-Time Performance | 25-2 |
| What This Issue Means | 25-2 |
| Try This Workaround | 25-2 |
| Troubleshoot Overloaded CPU from Executing Real-Time Application | 25-5 |
| What This Issue Means | 25-5 |
| Try This Workaround | 25-5 |
| Troubleshoot Task Execution Time | 25-7 |
| What This Issue Means | 25-7 |
| Try This Workaround | 25-7 |

| | |
|---|-------------|
| Troubleshoot Failed Read of Profiling Data | 25-8 |
| What This Issue Means | 25-8 |
| Try This Workaround | 25-8 |
| Troubleshoot Timeout During File System Access | 25-9 |
| What This Issue Means | 25-9 |
| Try This Workaround | 25-9 |

Simulink Real-Time Support

26

| | |
|--|-------------|
| Find Simulink Real-Time Support | 26-2 |
| Install Simulink Real-Time Software Updates | 26-3 |

Model Architectures

FPGA Models

- “Speedgoat FPGA Support with HDL Workflow Advisor” on page 1-2
- “FPGA Programming and Configuration” on page 1-4
- “Interrupt Configuration” on page 1-14
- “FPGA Subsystem Plan” on page 1-16
- “FPGA Synchronization Modes” on page 1-19

Speedgoat FPGA Support with HDL Workflow Advisor

Introduction

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 1-4. 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 1-4.

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 FPGA IO Module Support

Speedgoat I/O FPGA boards are sold as part of Speedgoat target computer systems. Simulink Real-Time supports the following Speedgoat (www.speedgoat.com) FPGA IO modules.

| IO Module Name | Description | Remarks |
|-----------------|---|---|
| Speedgoat IO331 | <p>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.</p> <p>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.</p> | <p>Warning Speedgoat IO331 and its variant Speedgoat IO331-6 will no longer be supported in R2020b with the Simulink Real-Time FPGA I/O workflow. You can still run the workflow for these IO modules in R2020a. In R2020b, use Speedgoat IO333-325K or a later Speedgoat IO module that is based on Xilinx Vivado®.</p> |
| Speedgoat IO333 | <p>The Speedgoat IO333 is a field-programmable gate array (FPGA) board based on a Xilinx Kintex® 7 chip with 325k logic cells.</p> <p>HDL Coder HDL Workflow Advisor supports the Speedgoat IO333-325K-06 configuration. For more information, see <i>Speedgoat HDL Coder Integration Package for the IO333-325K</i> at www.speedgoat.com/help.</p> | - |

Workflow

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

- “HDL Language Support and Supported Third-Party Tools and Hardware” (HDL Coder)
- “Tool Setup” (HDL Coder)
- “FPGA Programming and Configuration” on page 1-4
- “Digital I/O with Speedgoat FPGA Board” on page 15-110
- “PLL-Based Interrupt Generation from FPGA Input” on page 15-117

External Websites

- www.speedgoat.com/help
- www.speedgoat.com

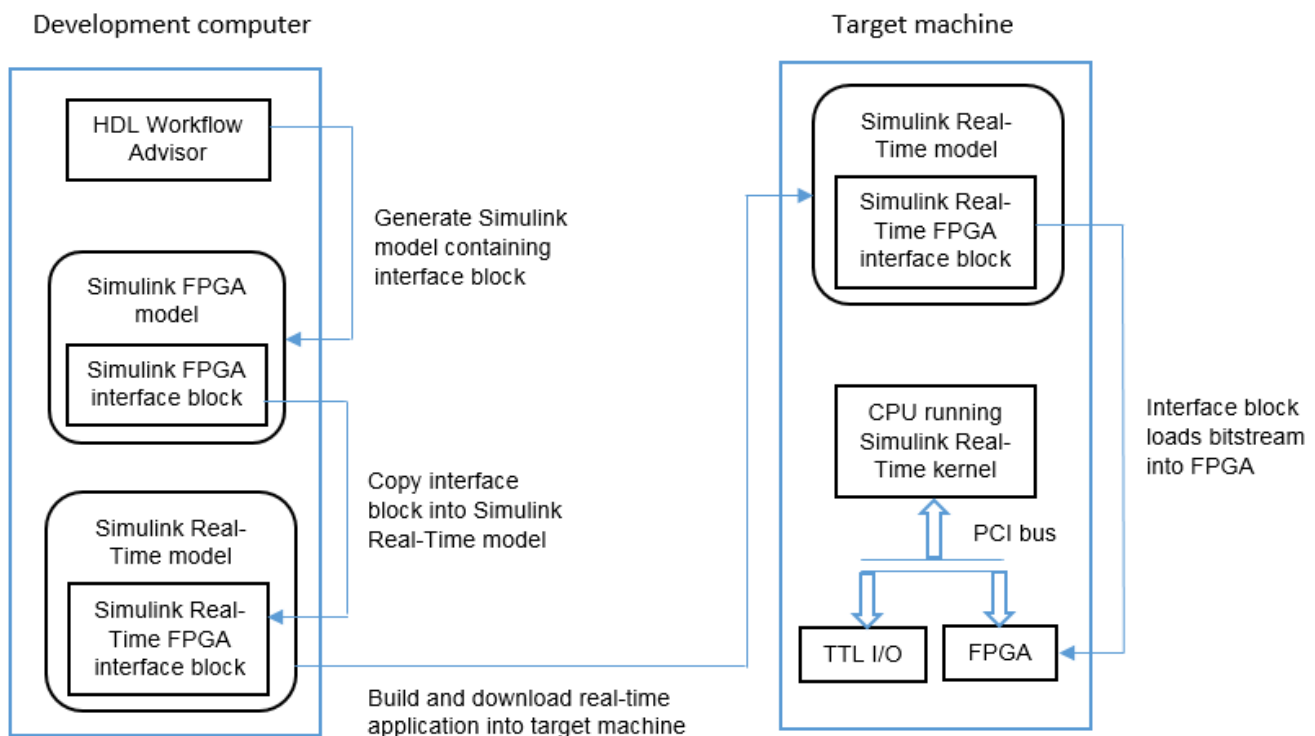
FPGA Programming and Configuration

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).

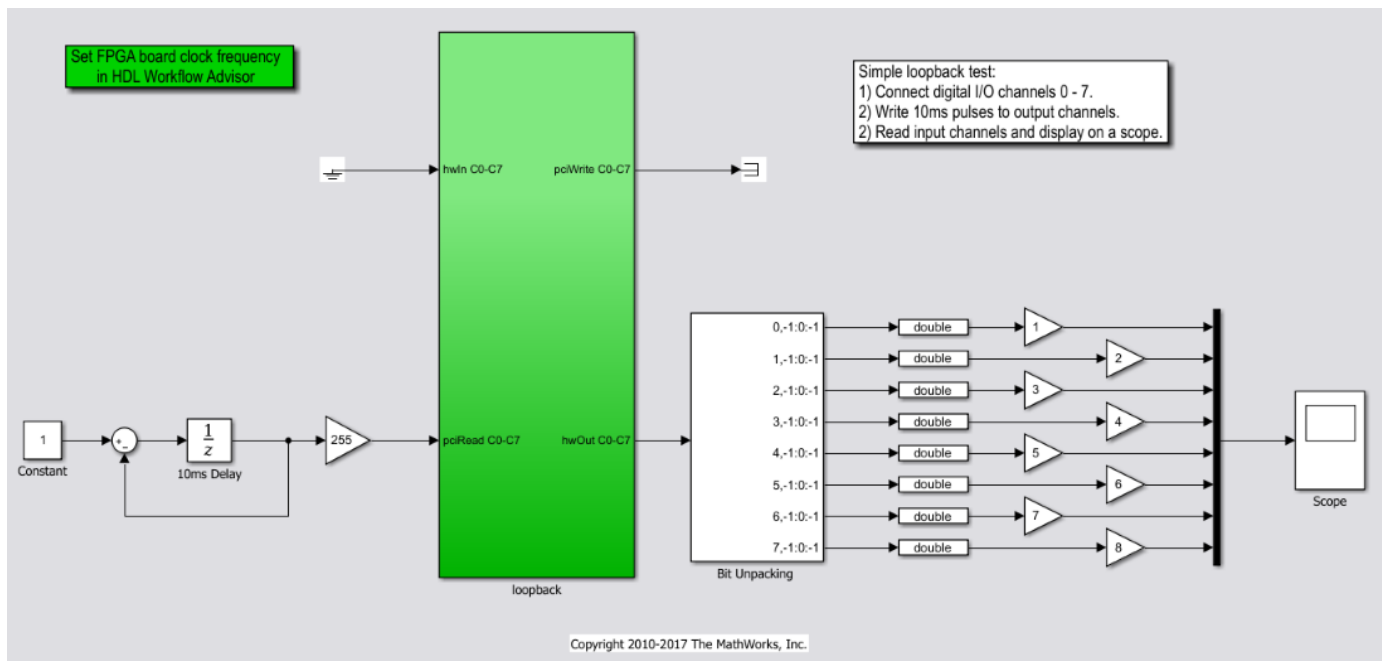
Step 1. Simulink Domain Model

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.

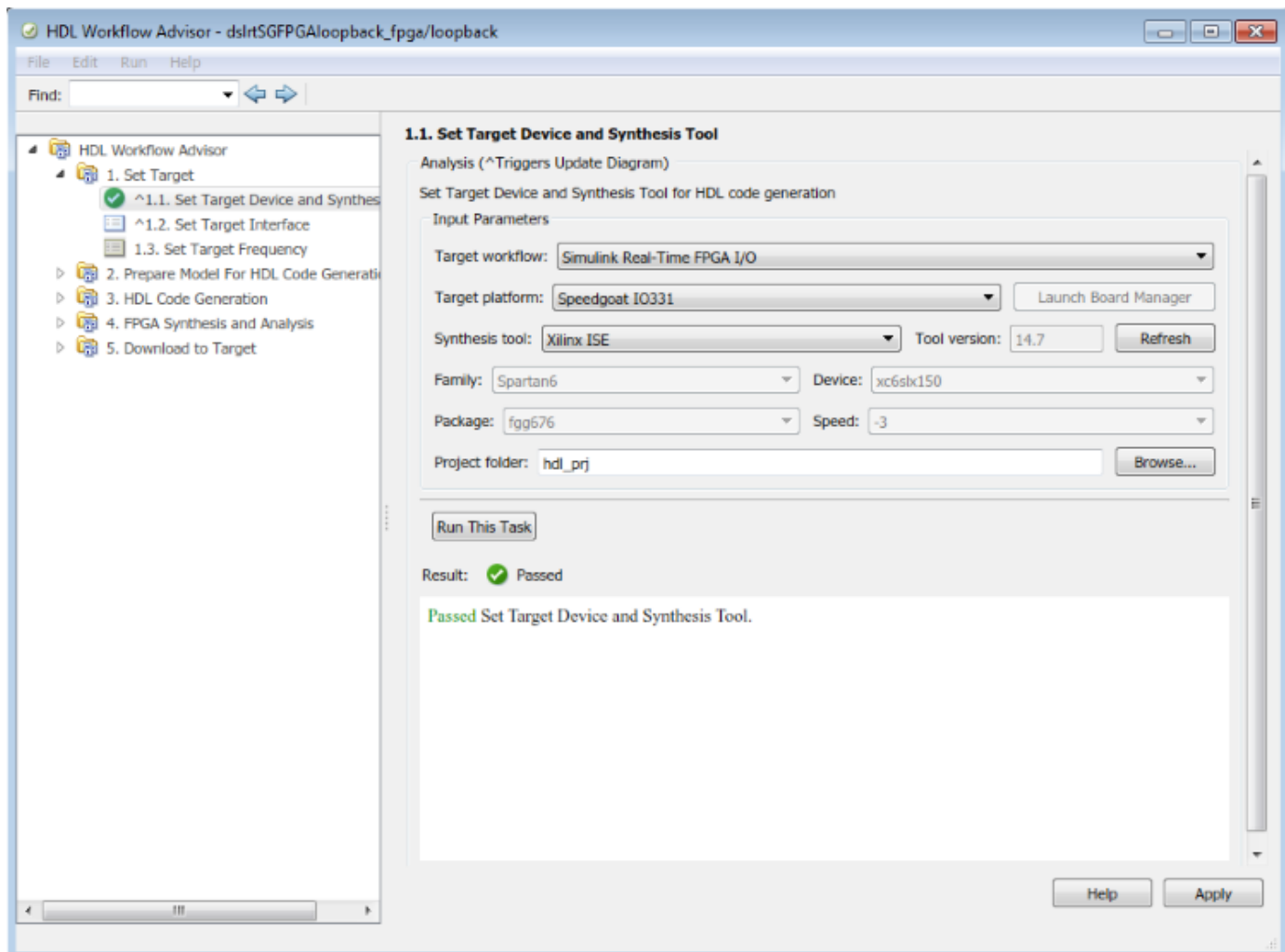
For an example of an FPGA domain model, see `dslrtSGFPGAloopback_fpga`. The `ServoSystem` subsystem contains the FPGA algorithm.



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.

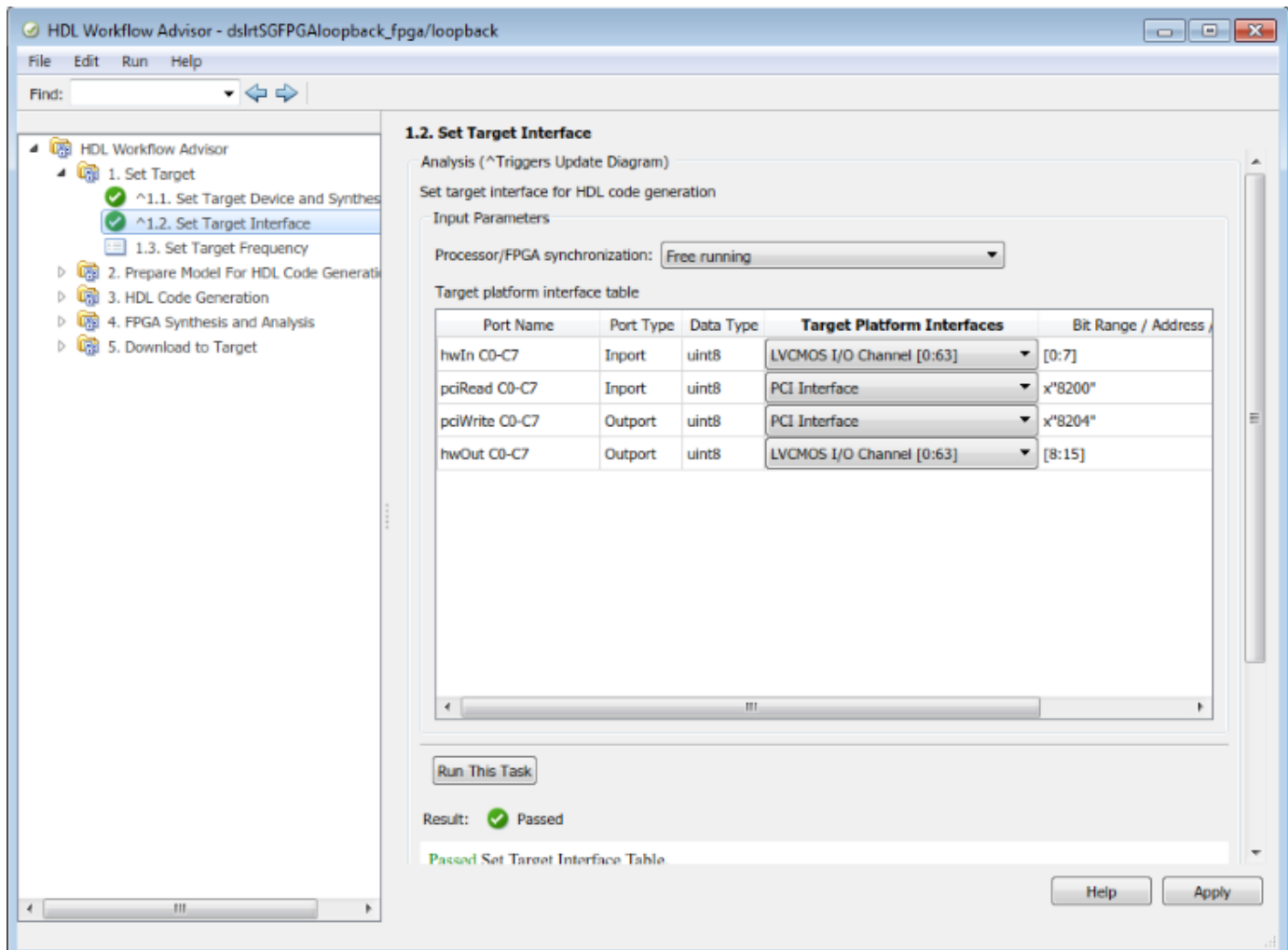
- 1 Open the FPGA domain model `dsLrtSGFPGAloopback_fpga`.
- 2 In the FPGA model, right-click the FPGA subsystem (ServoSystem). From the context menu, select **HDL Code > HDL Workflow Advisor**. The HDL Workflow Advisor dialog box displays several tasks for the subsystem. Address only your required subset 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**.

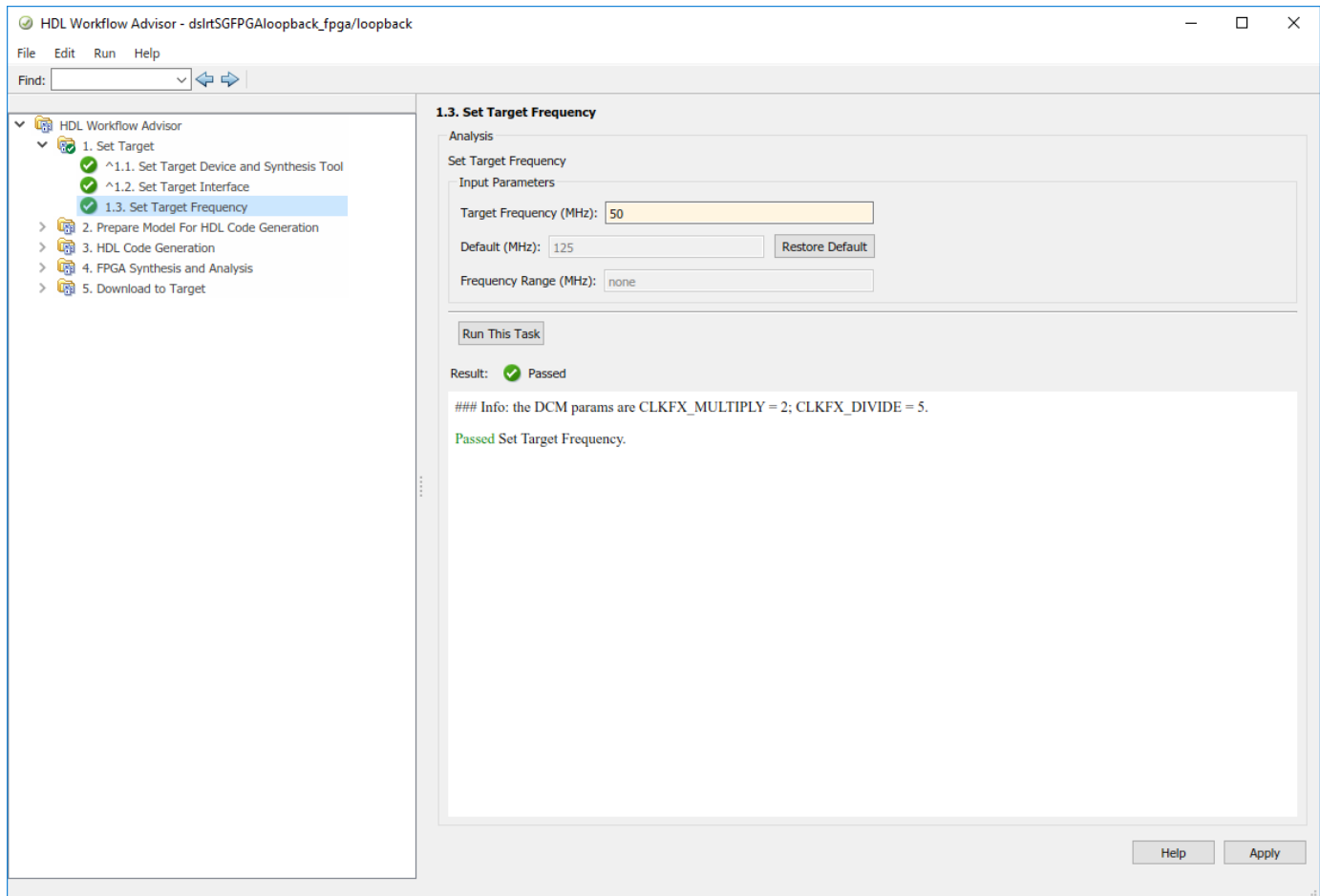


Step 4. FPGA Target Frequency Configuration

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.

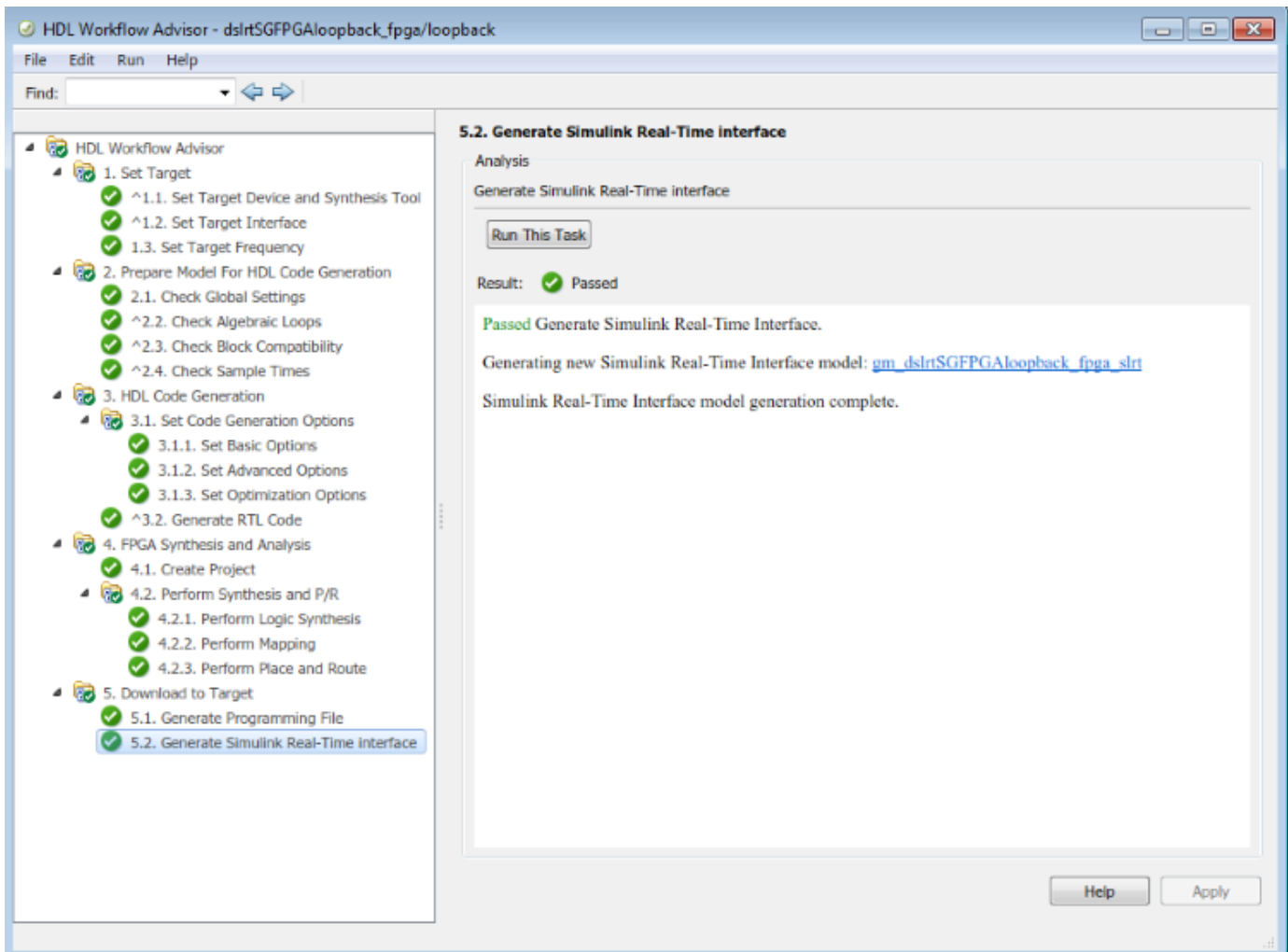
You must have already configured the FPGA target interface and the required target frequency. If you have specified vector inports or outports, you must have already selected the **Scalarize vector ports** check box. This check box is on the **Coding style** tab of node **Global Settings**, under node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

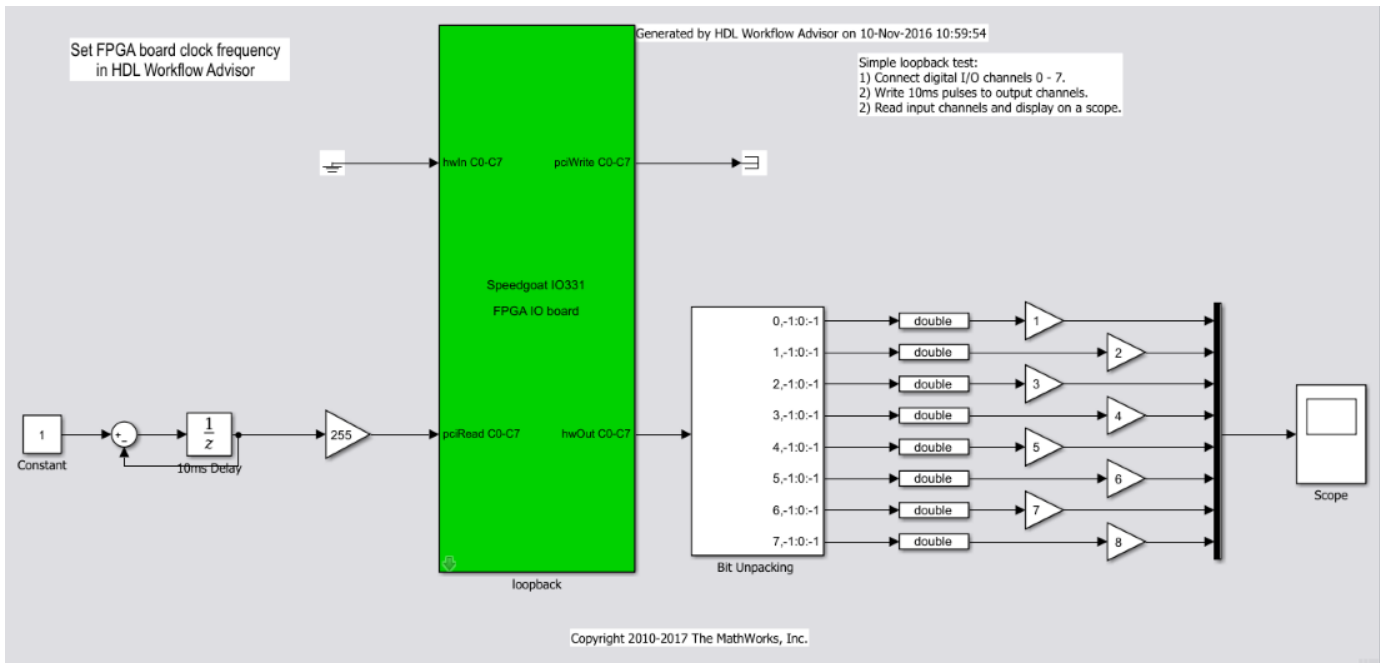
This action:

- 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.



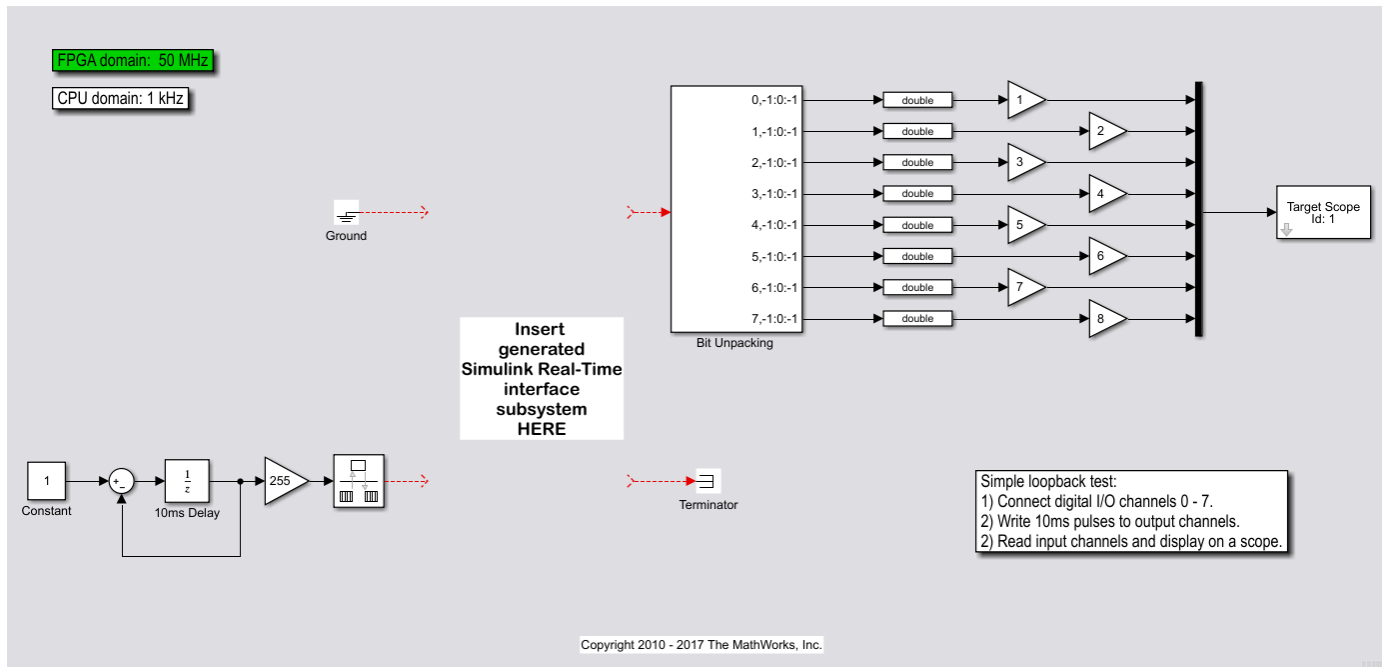
This generated model contains a masked subsystem with the same name as the subsystem in the Simulink FPGA domain model. Although the appearance is similar, this subsystem does not contain the Simulink algorithm. Instead, the algorithm is implemented in an FPGA bitstream. You reference and load this algorithm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

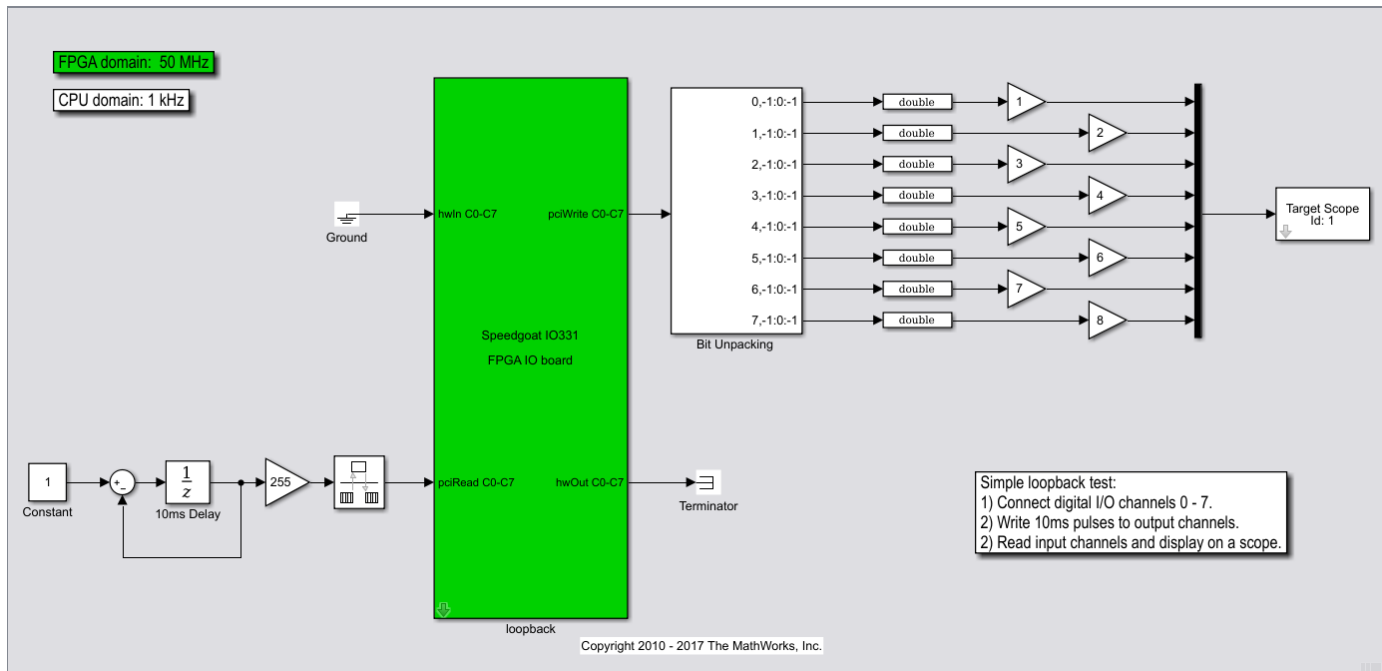
In the Simulink Real-Time interface subsystem mask, set three parameters:

- 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

Subsystem | getPCIInfo

More About

- “HDL Workflow Advisor” (HDL Coder)
- “IP Core Generation Workflow for Speedgoat I/O Modules” (HDL Coder)
- “FPGA Subsystem Plan” on page 1-16
- “FPGA Synchronization Modes” on page 1-19
- “FPGA Clock Frequency” on page 1-17
- “Digital I/O with Speedgoat FPGA Board” on page 15-110

- “PLL-Based Interrupt Generation from FPGA Input” on page 15-117

Interrupt Configuration


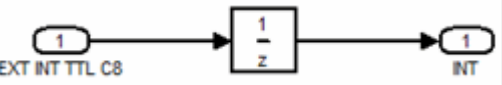
Simulink Real-Time software schedules the real-time application using either the internal timer of the Speedgoat target machine (default) or an interrupt from an I/O board. You can use your Speedgoat FPGA board to generate an interrupt, which allows you to:

- 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.

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 | <p>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 output labeled INT.</p>  |
| External | <p>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 output labeled INT.</p>  |

In both cases, wire the interrupt source to an output in the FPGA subsystem. Assign the output 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, on the **Real-Time** tab, click **Hardware Settings**.

- 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_I0331.
- 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” on page 15-117

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.

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 “HDL Language Support and 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 1-19.

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 1-14.

- **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_topiospeedgoat#.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 (*blkorreftmdlname_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

- “HDL Language Support and Supported Third-Party Tools and Hardware” (HDL Coder)
- “IP Core User Guide” (HDL Coder)
- “FPGA Synchronization Modes” on page 1-19
- “Interrupt Configuration” on page 1-14

FPGA Synchronization Modes

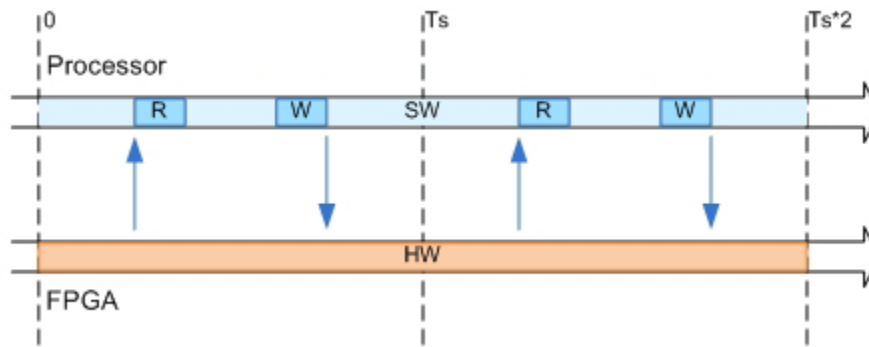
In Simulink Real-Time, an FPGA operates in three synchronization modes:

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

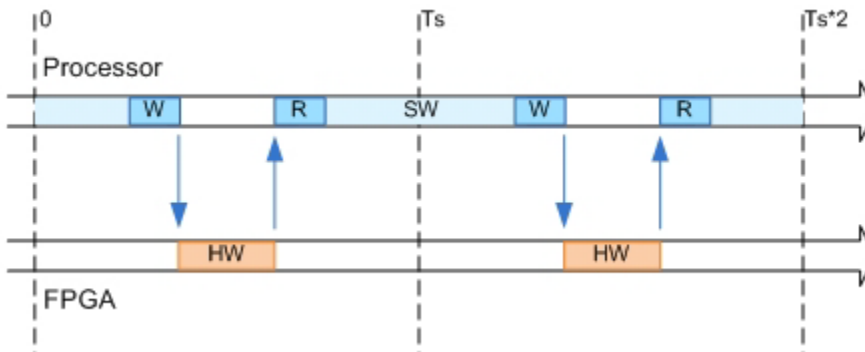
The shaded areas indicate that the processor and FPGA are running continuously.



- 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:

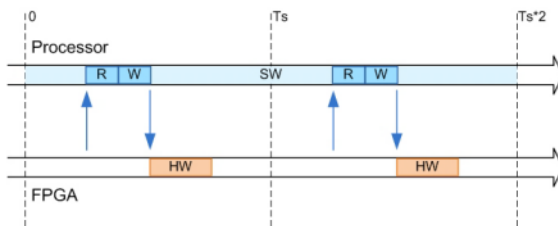
- 1 Writes data to the FPGA inputs.
- 2 Strokes the data into the FPGA.
- 3 Waits for the FPGA to finish executing.
- 4 Reads results from the FPGA outputs.



- 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.



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.

Build Considerations

When you build an FMU, dependencies on external DLLs are an important consideration. For example, if the compiler command line does not provide the MT flag, the linker links the `.obj` file with `MSVCRT.lib`. This library depends on many DLLs that are unavailable on a Simulink Real-Time target computer.

The MT flag on the compiler command line makes the real-time application use the multithread, static version of the run-time library. With MT, the compiler places `LIBCMT.lib` into the `.obj` file, so the linker uses `LIBCMT.lib` to resolve external symbols.

To build a FMU for a Simulink Real-Time real-time application, in the makefile to build a FMU, change from:

```
cl /wd4090 /nologo %DEF% ..\%1\%1.c /I ..\ /I ..\..\%FMI_DIR%\include
```

Change to:

```
cl /MT /wd4090 /nologo %DEF% ..\%1\%1.c /I ..\ /I ..\..\%FMI_DIR%\include
```

Note Note: Simulink Real-Time supports FMU blocks that comply with FMU v1.0. Blocks complying with FMU v2.0 are not supported.

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 3-3
- “Calibrate Parameters with Vector CANape” on page 3-8
- “Vector CANape Limitations” on page 3-10
- “Troubleshoot Vector CANape Operation” on page 3-11
- “Calibrate Parameters with ETAS Inca” on page 3-12
- “ETAS Inca Limitations” on page 3-14
- “Troubleshoot ETAS Inca Operation” on page 3-15

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 3-3
- “Calibrate Parameters with Vector CANape” on page 3-8
- “Calibrate Parameters with ETAS Inca” on page 3-12
- “XCP Master Mode”

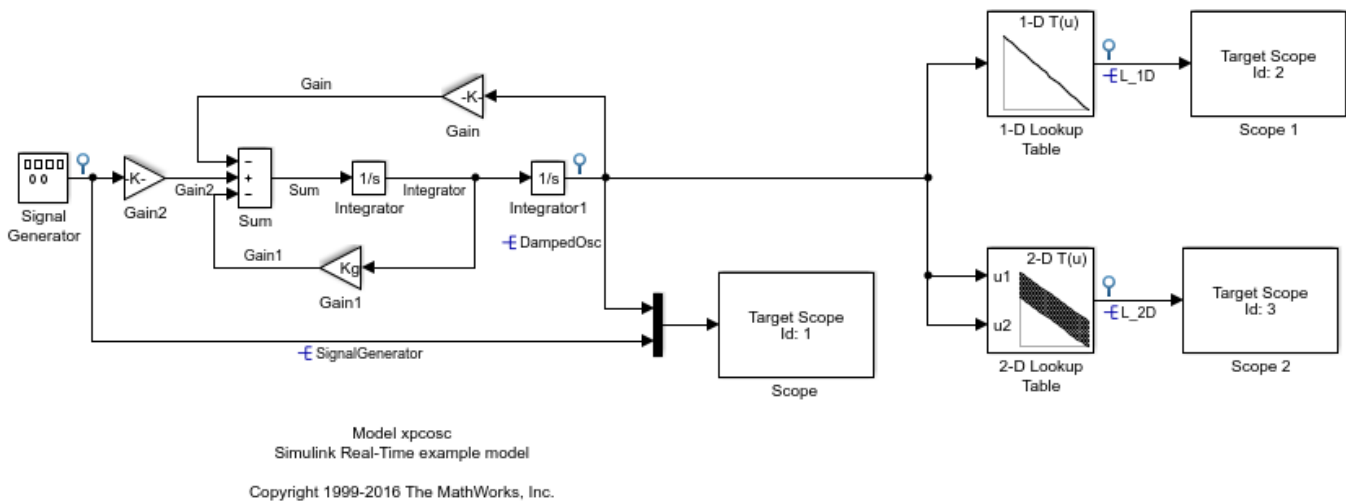
External Websites

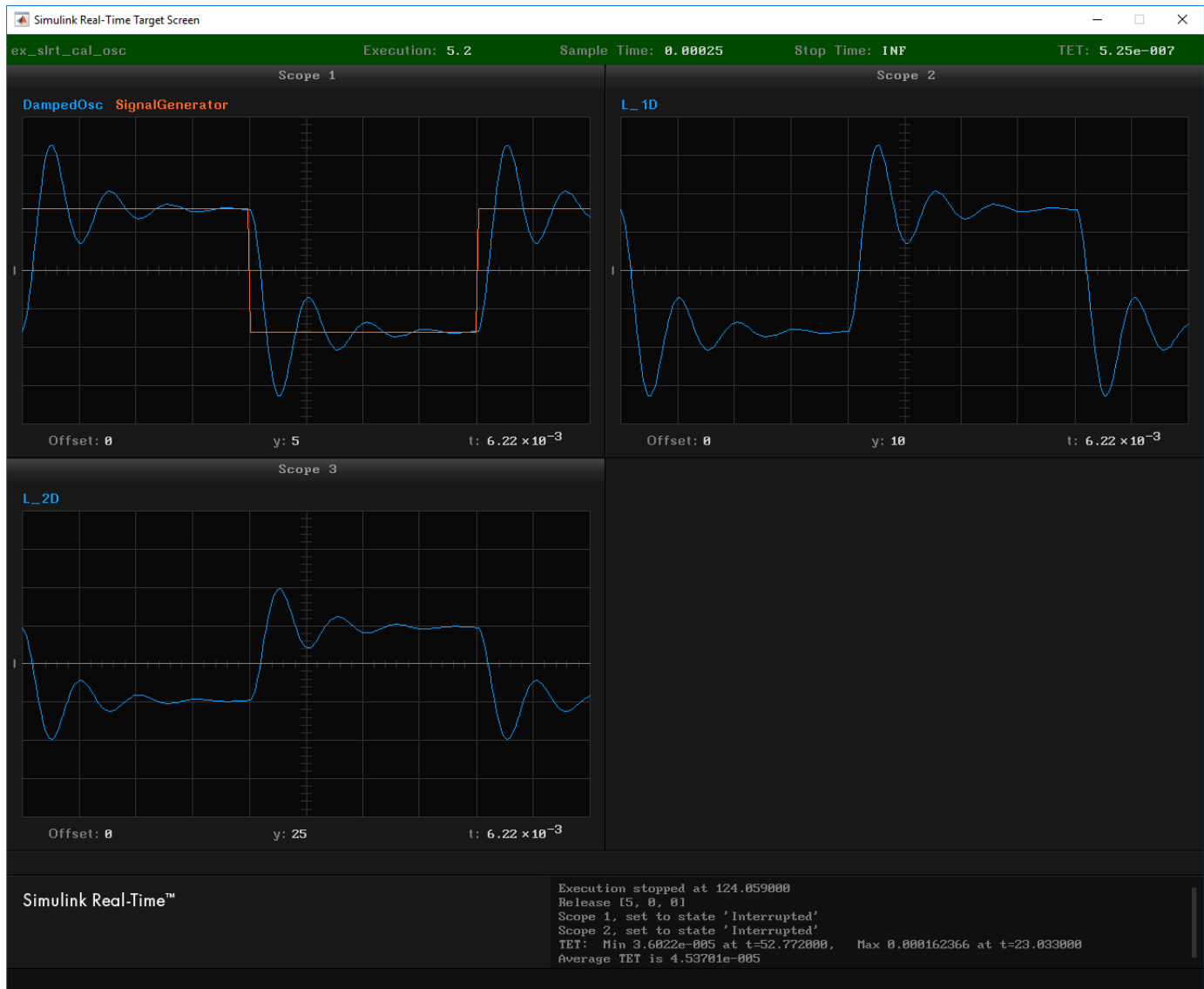
- www.vector.com
- www.etas.com

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.

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 **Modeling** tab, click **Design > Model Explorer** .
- 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 **Modeling** tab, click **Design > Model Explorer** .
- 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 the **Configuration Parameters**. On the **Real-Time** tab, select **Prepare > Hardware Settings**.
- 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 the 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. You can now connect to the target with a third-party calibration tool.

See Also

“Generate INCA/CANape extensions (disables the Simulation Data Inspector and Dashboard blocks)”
| n-D Lookup Table

More About

- “Calibrate Parameters with Vector CANape” on page 3-8
- “Calibrate Parameters with ETAS Inca” on page 3-12

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 3-3.

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

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 time for a signal, open the measurement properties for the signal. Select the required sample time from the measurement mode list.

The default sample time is the base sample time.

- 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 3-3
- “Vector CANape Limitations” on page 3-10
- “Troubleshoot Vector CANape Operation” on page 3-11

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 Issue 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 3-3. In addition to the limitations listed in “Vector CANape Limitations” on page 3-10, there are various issues that can prevent operation of this tool.

Try This Workaround

For Vector CANape tool issues, try these workarounds.

Simulation Data Inspector in Use

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

Master Cannot Connect

Check the IP address of the target computer associated with the model and compare it to 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 that you build the application.

See Also

More About

- “Prepare ASAP2 Data Description File” on page 3-3
- “Vector CANape Limitations” on page 3-10

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 3-3.

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

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 sample time to the base sample time of the real-time application (250 μ s).

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 3-3
- “ETAS Inca Limitations” on page 3-14
- “Troubleshoot ETAS Inca Operation” on page 3-15

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 Issue 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 3-3. In addition to the limitations listed in “ETAS Inca Limitations” on page 3-14, there are various issues that can prevent operation of this tool.

Try This Workaround

For ETAS Inca tool issues, try these workarounds.

Simulation Data Inspector in Use

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

Master Cannot Connect

Check the IP address of the target computer associated with the model and compare it to 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 that 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 3-3
- “ETAS Inca Limitations” on page 3-14
- “Troubleshoot ETAS Inca Operation” on page 3-15

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).

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 5-4
- “Command-Line PCI Bus Ethernet Setup” on page 5-5
- “Ethernet Card Selection by Index” on page 5-8
- “Command-Line Ethernet Card Selection by Index” on page 5-9
- “Command-Line Target Computer Settings” on page 5-11
- “Command-Line Target Computer Boot Methods” on page 5-12
- “Command-Line Network Boot Method” on page 5-13
- “Command-Line Standalone Boot Method” on page 5-15

Default Target Computers

When you start Simulink Real-Time Target Computer Manager 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.

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, select the node in the Target Computer manager and select the **Default** check box next to the target computer **Name** box. The default target computer node has a **(default)** label.

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 5-4.

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

When you call the `slrt` function without arguments, the constructor uses the link properties of the default target computer to communicate with the target computer.

```
tg = slrt;  
% create a target object tg for the default target computer TargetPC1
```

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` only if you require different compilers for MEX and Simulink Real-Time.

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 have 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 ('VisualC', '')  
mex -setup
```

The function queries the development computer for C compilers that the Simulink Real-Time environment supports.

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.

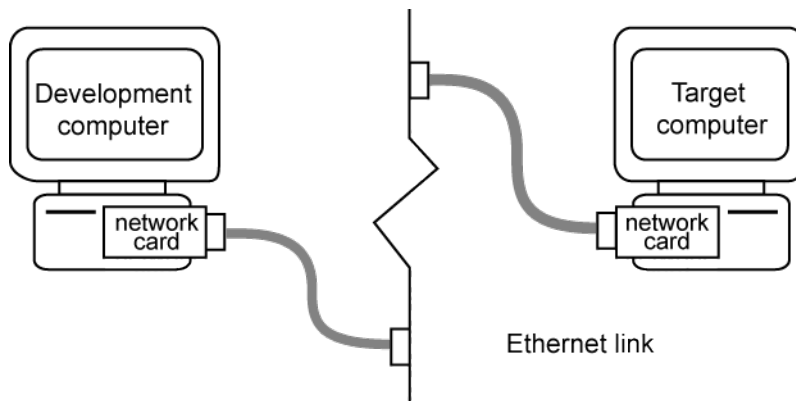
- 1** “Command-Line PCI Bus Ethernet Setup” on page 5-5
- 2** “Command-Line Target Computer Settings” on page 5-11
- 3** “Command-Line Target Computer Boot Methods” on page 5-12

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.

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** Create a target object for this target computer and make it the default target. In the Command Window, type:

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

You apply other settings to this target object.

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

```
tg.TcpIpTargetAddress = '10.10.10.15';
```

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

```
tg.TcpIpSubNetMask = '255.255.255.0';
```

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

```
tg.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:

```
tg.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'.

```
tg.TcpIpTargetBusType = 'PCI';
```

- 7** Set the target driver to one of 'I210', 'I217', 'I8254x', 'I82559', 'X540', 'R8139', 'R8168', or 'Auto' (the default).

```
tg.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 5-9.

Repeat this procedure as required for each target computer.

The next task is “Command-Line Target Computer Settings” on page 5-11.

See Also

getPCIInfo

More About


- “Ethernet Card Selection by Index” on page 5-8
- “Command-Line Ethernet Card Selection by Index” on page 5-9

Ethernet Card Selection by Index

If the Speedgoat target machine has multiple Ethernet cards, you could be guided by Speedgoat support to specify which card to use for the Ethernet link.

In the Speedgoat documentation, see information about the Ethernet index of the PCI cards in the Speedgoat target machine.

Use the following procedure for target TargetPC1:

- 1 Open Simulink Real-Time Explorer. In the Command Window, type: `slrtexplr`.
- 2 In the **Targets** pane, expand the target computer node.
- 3 In the toolbar, click the **Target Properties** button .
- 4 From the Speedgoat target machine documentation, note the index of the Ethernet card that you want to use for the Ethernet link, for example, 2.
- 5 In the Command Window, type:

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

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

- 6 In `slrtexplr`, set **Boot mode** to Network.
- 7 Click **Create boot disk**.
- 8 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 Get the target object for this target computer and make it the default target. In the Command Window, type:

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

You apply other settings to this object.

- 2 In the Command Window, type:

```
tg.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:

```
tg.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:

```
tg.TargetBoot='CDBoot';
```

- 5 Set the target monitor to print text only:

```
tg.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 In the Command Window, type:

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

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

10 Set the boot method back to network boot:

```
tg.TargetBoot= 'NetworkBoot';
```

11 Set the target monitor to graphics mode:

```
tg.TargetScope = 'Enabled' ;
```

12 Type `SimulinkRealTime.createBootImage`.

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

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

Repeat this procedure as required for each target computer.

Command-Line Target Computer Settings

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

Use the following procedure for target TargetPC1:

- 1 Get the target object for this target computer and make it the default target. In the Command Window, type:

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

You apply other settings to this object.

- 2 Assign the following target computer settings as required:

- **Target scope display**

- `tg.TargetScope='Enabled'` (the default) — Use to display information, such as a target scope, in graphic format.
- `tg.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**

- `tg.USBSupport='on'` (the default) — Use to enable USB ports on the target computer; for example, to connect a USB keyboard.
- `tg.USBSupport='off'` — Otherwise.

- **Target RAM size**

`tg.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 5-12.

Command-Line Target Computer Boot Methods

Speedgoat target machines come with DOS Loader software installed. You can set up the DOS Loader boot method on your development computer or configure another boot method. For more information about booting and kernel transfer, see your Speedgoat target machine documentation or follow the link from “Speedgoat Target Computers and Support”.

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

- 1** Select one of the following methods:
 - “Command-Line Network Boot Method” on page 5-13
 - “Command-Line Standalone Boot Method” on page 5-15
- 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

Speedgoat target machines come with DOS Loader software installed. You can set up the DOS Loader boot method on your development computer or configure another boot method. For more information about booting and kernel transfer, see your Speedgoat target machine documentation or follow the link from “Speedgoat Target Computers and Support”.

After you have configured the target computer network 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.

The network boot method has some 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 5-5.

Use the following procedure for target TargetPC1:

- 1 Get the target object for this target computer and make it the default target. In the Command Window, type:

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

The contents of target object `tg` are printed in the Command Window. Some properties can already have the required values.

- 2 Set network boot method:

```
tg.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:

```
tg.TcpIpTargetAddress='192.168.7.1'
```

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

```
tg.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 Standalone Boot Method

Speedgoat target machines come with DOS Loader software installed. You can set up the DOS Loader boot method on your development computer or configure another boot method. For more information about booting and kernel transfer, see your Speedgoat target machine documentation or follow the link from “Speedgoat Target Computers and Support”.

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.

Command-Line Standalone Settings

Speedgoat target machines come with DOS Loader software installed. You can set up the DOS Loader boot method on your development computer or configure another boot method. For more information about booting and kernel transfer, see your Speedgoat target machine documentation or follow the link from “Speedgoat Target Computers and Support”.

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 Get the target object for this target computer and make it the default target. In the Command Window, type:

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

You make other settings to this object.

- 2 Set network boot method:

```
tg.TargetBoot='StandAlone';
```

Repeat this procedure as required for each target computer.

Real-Time Application Build

After you set the 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')))).
- 2 Build the model. In the Simulink Editor, on the **Real-Time** tab, click **Run on Target > Build Application**.

The Simulink Coder and Simulink Real-Time software create a folder, `ex_slrt_rt_osc_slrt_emb`, containing the files that boot the target computer and run the real-time application.

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` object 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 USB 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 Start the target computer by using network boot mode.
- 3 In MATLAB, change to the folder that contains the kernel and real-time application files.
- 4 Copy these files to the root folder of the target computer C:\ drive:

```
tg = slrt;  
SimulinkRealTime.copyFileToTarget(tg, 'xpckrnl.rtb')  
SimulinkRealTime.copyFileToTarget(tg, 'ex_slrt_rt_osc.mldatx')
```

- 5 Restart the target computer.

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

```
C:\autoexec.bat  
C:\rttboot.com  
C:\xpckrnl.rtb  
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 6-3
- “Monitor Signals with Simulink Real-Time Explorer” on page 6-4
- “Monitor Signals with MATLAB Language” on page 6-7
- “Instrument a Stateflow Subsystem” on page 6-9
- “Signal Group Monitoring Formats” on page 6-12
- “Monitor Stateflow States with MATLAB Language” on page 6-13
- “Animate Stateflow Charts with Simulink External Mode” on page 6-14
- “Signal Tracing Basics” on page 6-16
- “Simulink Real-Time Scope Usage” on page 6-17
- “Target Scope Usage” on page 6-18
- “Configure Real-Time Target Scope Blocks” on page 6-19
- “Create Target Scopes with Simulink Real-Time Explorer” on page 6-24
- “Configure Scope Sampling with Simulink Real-Time Explorer” on page 6-29
- “Trigger Scopes with Simulink Real-Time Explorer” on page 6-32
- “Configure Target Scopes with Simulink Real-Time Explorer” on page 6-39
- “Configure Target Scopes with MATLAB Language” on page 6-42
- “Create Signal Groups with Simulink Real-Time Explorer” on page 6-45
- “Host Scope Usage” on page 6-48
- “Configure Real-Time Host Scope Blocks” on page 6-49
- “Create Host Scopes with Simulink Real-Time Explorer” on page 6-52
- “Configure the Host Scope Viewer” on page 6-56
- “Trace Signals with Simulink External Mode” on page 6-58
- “Inspect Simulink® Real-Time™ Data with Simulation Data Inspector” on page 6-61
- “Stream Signal Data from Target Computer to Simulation Data Inspector” on page 6-65
- “Trace or Log Data with the Simulation Data Inspector” on page 6-68
- “External Mode Usage” on page 6-72
- “Signal Logging Basics” on page 6-73
- “File Scope Usage” on page 6-74
- “Configure Real-Time File Scope Blocks” on page 6-76
- “Create File Scopes with Simulink Real-Time Explorer” on page 6-81
- “Configure File Scopes with Simulink Real-Time Explorer” on page 6-85
- “Log Signal Data into Multiple Files” on page 6-89

- “Log Signal Data with Outport Blocks and Simulink Real-Time Explorer” on page 6-92
- “Log Signal Data with Outport Block and MATLAB Language” on page 6-97
- “Signal Logging Buffer Size” on page 6-103
- “Configure File Scopes with MATLAB Language” on page 6-104
- “Tune Parameters with Simulink Real-Time Explorer” on page 6-107
- “Create Parameter Groups with Simulink Real-Time Explorer” on page 6-111
- “Tune Parameters with MATLAB Language” on page 6-113
- “Tune Parameters with Simulink External Mode” on page 6-115
- “Save and Reload Parameters with MATLAB Language” on page 6-117
- “Tunable Block Parameters and Tunable Global Parameters” on page 6-119
- “Tune Inlined Parameters with Simulink Real-Time Explorer” on page 6-122
- “Tune Inlined Parameters with MATLAB Language” on page 6-128
- “Tune Parameter Structures with Simulink Real-Time Explorer” on page 6-129
- “Tune Parameter Structures with MATLAB Language” on page 6-134
- “Define and Update Inport Data” on page 6-137
- “Define and Update Inport Data with MATLAB Language” on page 6-142
- “Inport Data Mapping Limitations” on page 6-146
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147
- “Display and Filter Hierarchical Signals and Parameters (tech preview)” on page 6-151
- “Troubleshoot Signals Not Accessible by Name” on page 6-155
- “Troubleshoot Parameters Not Accessible by Name” on page 6-156
- “Troubleshoot Instance-Specific Parameters Not Saved” on page 6-157
- “Troubleshoot Instrument Label Not Present” on page 6-158
- “Troubleshoot Internationalization Issues” on page 6-159
- “Internationalization Issues” on page 6-160

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 6-155
- “Simulink Real-Time Scope Usage” on page 6-17
- “Target Scope Usage” on page 6-18
- “Host Scope Usage” on page 6-48
- “File Scope Usage” on page 6-74
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147

Monitor Signals with Simulink Real-Time Explorer

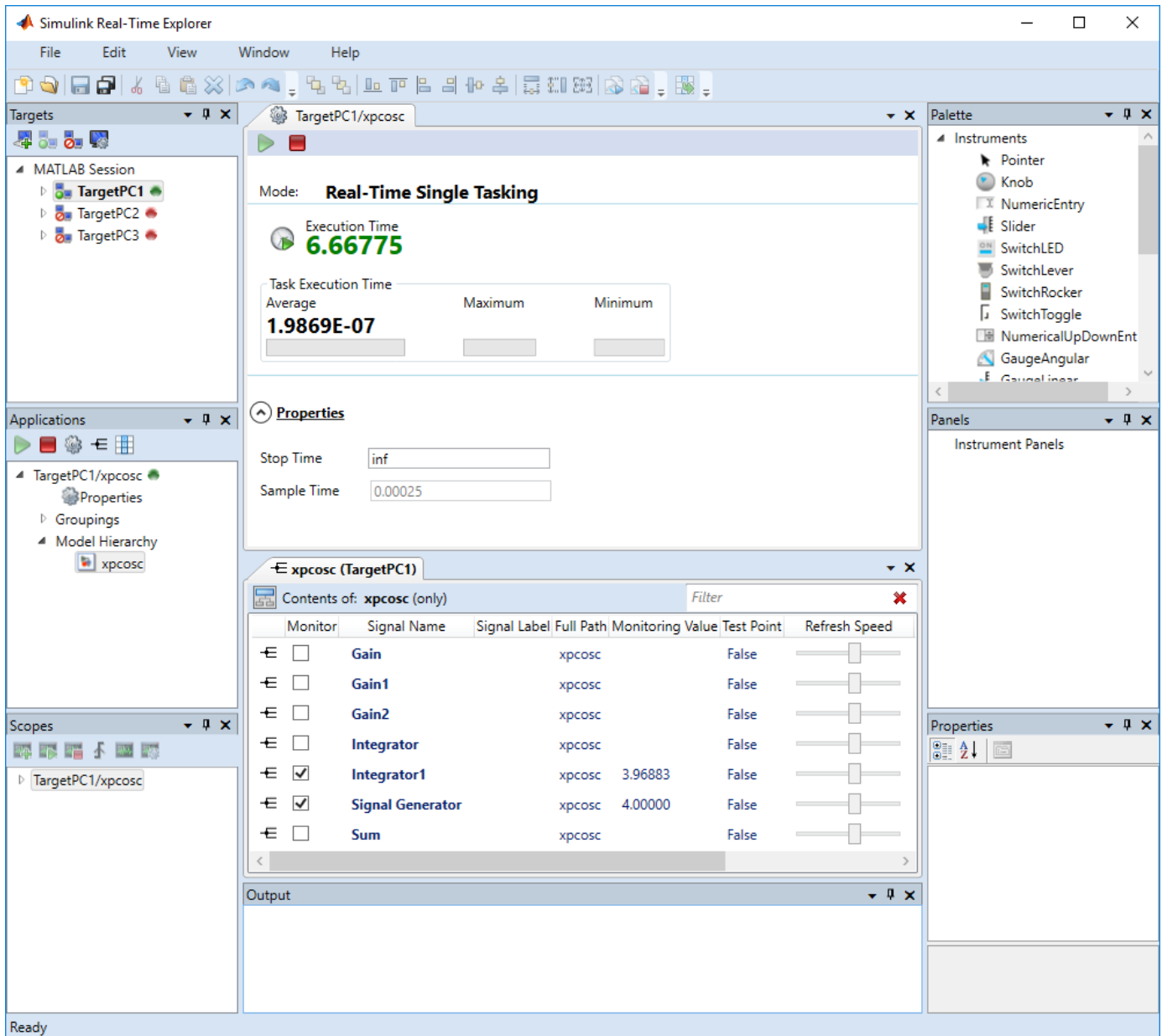
This procedure uses the model `xpcosc`. You must have already completed the following setup:


- 1 Open model `xpcosc`. Set property **Stop time** to `inf`. On the **Real-Time** tab, select **Run on Target > Stop Time** and set the **Stop Time** to `inf`.
- 2 Connect to the target computer. On the **Real-Time** tab, toggle the **Disconnected** indicator to **Connected**.
- 3 Build and download the real-time application to the target computer. On the **Real-Time** tab, click **Run on Target**.
- 4 Open Simulink Real-Time Explorer. On the **Real-Time** tab, click **Prepare > SLRT Explorer**.

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 6-45
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147
- “Signal Group Monitoring Formats” on page 6-12
- “Troubleshoot Signals Not Accessible by Name” on page 6-155

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 by Using MATLAB Language”.

Note

- Signal access by signal index will be removed in a future release. Access signals by signal name instead.
 - The Simulink Real-Time software lists referenced model signals with their full block path. For example, `ex_slrt_rt_osc/childmodel/gain`.
-

- 1 To get a list of signals, type:

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

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

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

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

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

```
ans =
```

```
-3.8771
```

See Also

More About

- “Configure Target Scopes with MATLAB Language” on page 6-42
- “Troubleshoot Signals Not Accessible by Name” on page 6-155

Instrument a Stateflow Subsystem

In this section...

“Configure Stateflow States as Test Points” on page 6-9

“Monitor Stateflow States with Simulink Real-Time Explorer” on page 6-10

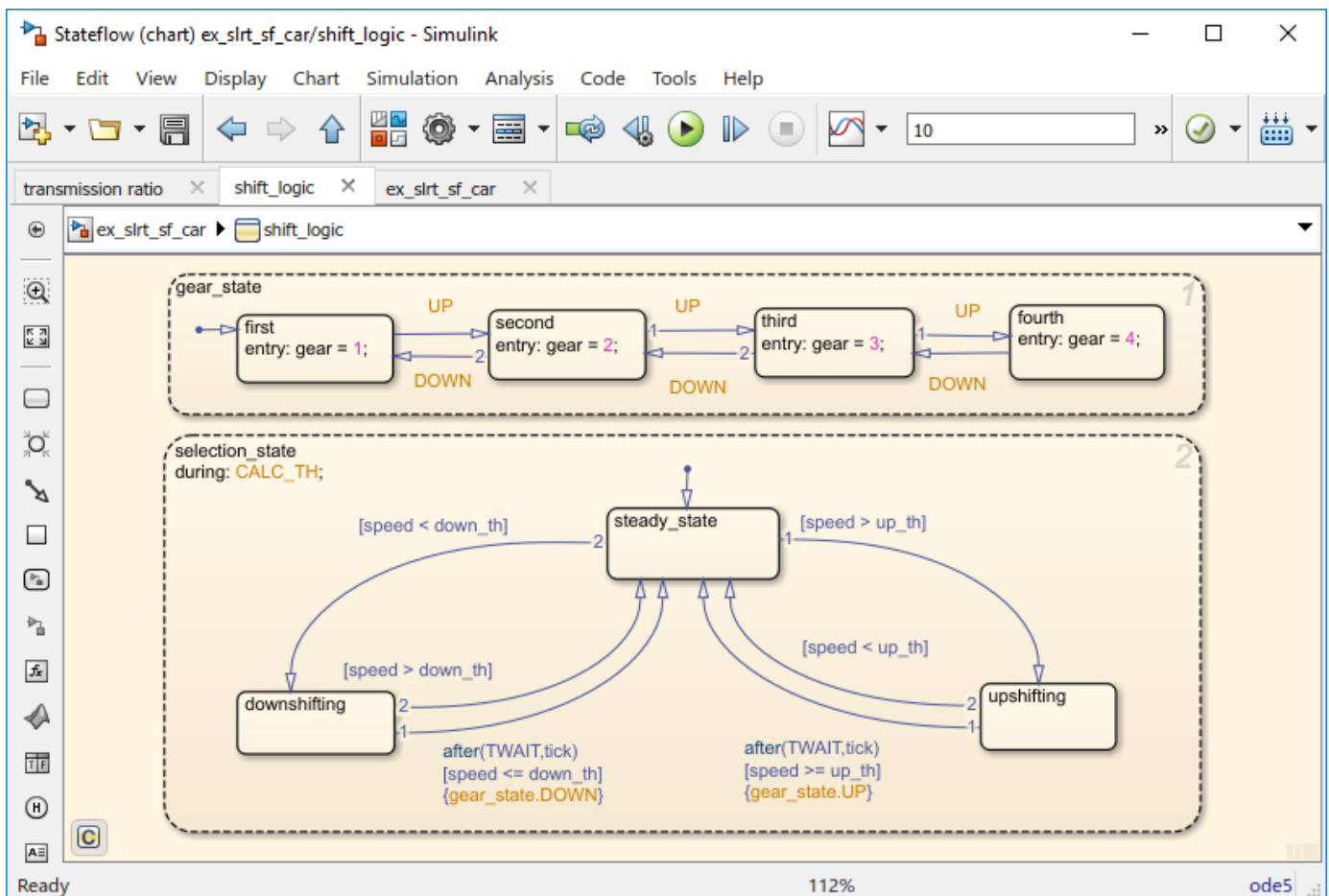
A Simulink Real-Time model that uses Stateflow blocks can present special circumstances. For example, if the model implements a control algorithm as a Stateflow subsystem, the Stateflow signals are not visible to Simulink Real-Time by default.

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 the `ex_slrt_sf_car` model.
- 2 Double-click the `shift_logic` chart.

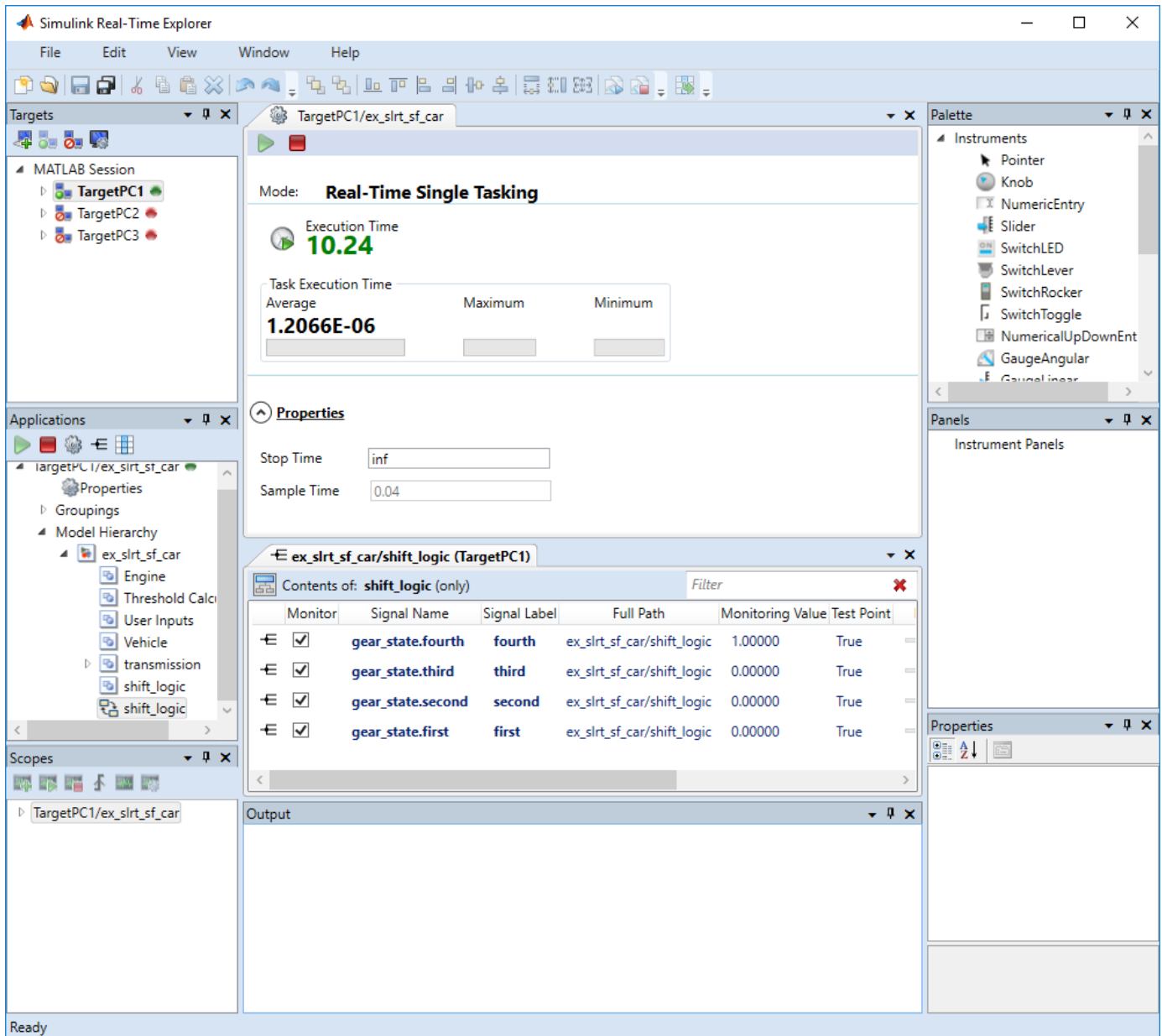


- 3 In the **Modeling** tab, click **Model Explorer**.

- 4 In the Model Explorer, expand **ex_slrt_sf_car**, then expand **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 real-time application to the target computer. On the **Real-Time** tab, click **Run on Target**.

Monitor Stateflow States with Simulink Real-Time Explorer

- 1 Open Simulink Real-Time Explorer. On the **Real-Time** tab, click **Prepare > SLRT Explorer**.
- 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 .



See Also

More About

- “Monitor Stateflow States with MATLAB Language” on page 6-13
- “Animate Stateflow Charts with Simulink External Mode” on page 6-14
- “Create Signal Groups with Simulink Real-Time Explorer” on page 6-45
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147
- “Signal Group Monitoring Formats” on page 6-12
- “Troubleshoot Signals Not Accessible by Name” on page 6-155

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 `sprintf` 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 6-45

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 6-9.

- 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 6-155

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 6-9.

- 1 Open the external mode control panel. In the Simulink Editor, in the **Real-Time** tab, click **Prepare > Control Panel**.
- 2 Select **Signal & Triggering**.
- 3 In the **Trigger** section of the **External Signal & Triggering** window:
 - To direct the trigger to re-arm after the trigger event completes, set **Mode** to **normal**.
 - To select the number of base rate steps for which external mode uploads data after a trigger event, in the **Duration** box, enter 5.
 - To direct data upload to begin immediately after the trigger event, select the **Arm when connecting to target** check box.
- 4 Click **Apply**. For more information about signal and triggering options, see “Configure Host Monitoring of Target Application Signal Data” (Simulink Coder).
- 5 Open Configuration Parameters. On the **Real-Time** tab, click **Hardware Settings**.
- 6 Select **Simulink Real-Time Options > Miscellaneous options > Enable Stateflow animation**.
- 7 Click **Apply**.
- 8 Verify that Stateflow animation is enabled for Simulink Real-Time. In the MATLAB Command Window, type:

```
get_param('ex_slrt_sf_car', 'xPCEnableSFAnimation')
```

```
ans =
```

```
    'on'
```

- 9 Connect to the target computer. On the **Real-Time** tab, toggle the **Disconnected** indicator to **Connected**.
- 10 Build and download the model to the target computer. On the **Real-Time** tab, click **Run on Target**.

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

- 11 To stop the simulation, on the **Real-Time** tab, click **Stop**.

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 6-155

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 6-155
- “Simulink Real-Time Scope Usage” on page 6-17
- “Target Scope Usage” on page 6-18
- “Host Scope Usage” on page 6-48
- “File Scope Usage” on page 6-74
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147

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 `getscope`. You can use `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 | `getscope`

More About

- “Troubleshoot Signals Not Accessible by Name” on page 6-155
- “Target Scope Usage” on page 6-18
- “Host Scope Usage” on page 6-48
- “File Scope Usage” on page 6-74
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147

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 `viewTargetScreen`.

To save the target scope image to a file, right-click in the display window and then click **Save as image**.

See Also

[Video Display](#) | [getscope](#) | [viewTargetScreen](#)

More About

- “Configure Real-Time Target Scope Blocks” on page 6-19
- “Create Target Scopes with Simulink Real-Time Explorer” on page 6-24
- “Simulink Real-Time Scope Usage” on page 6-17
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147

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 N th 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.

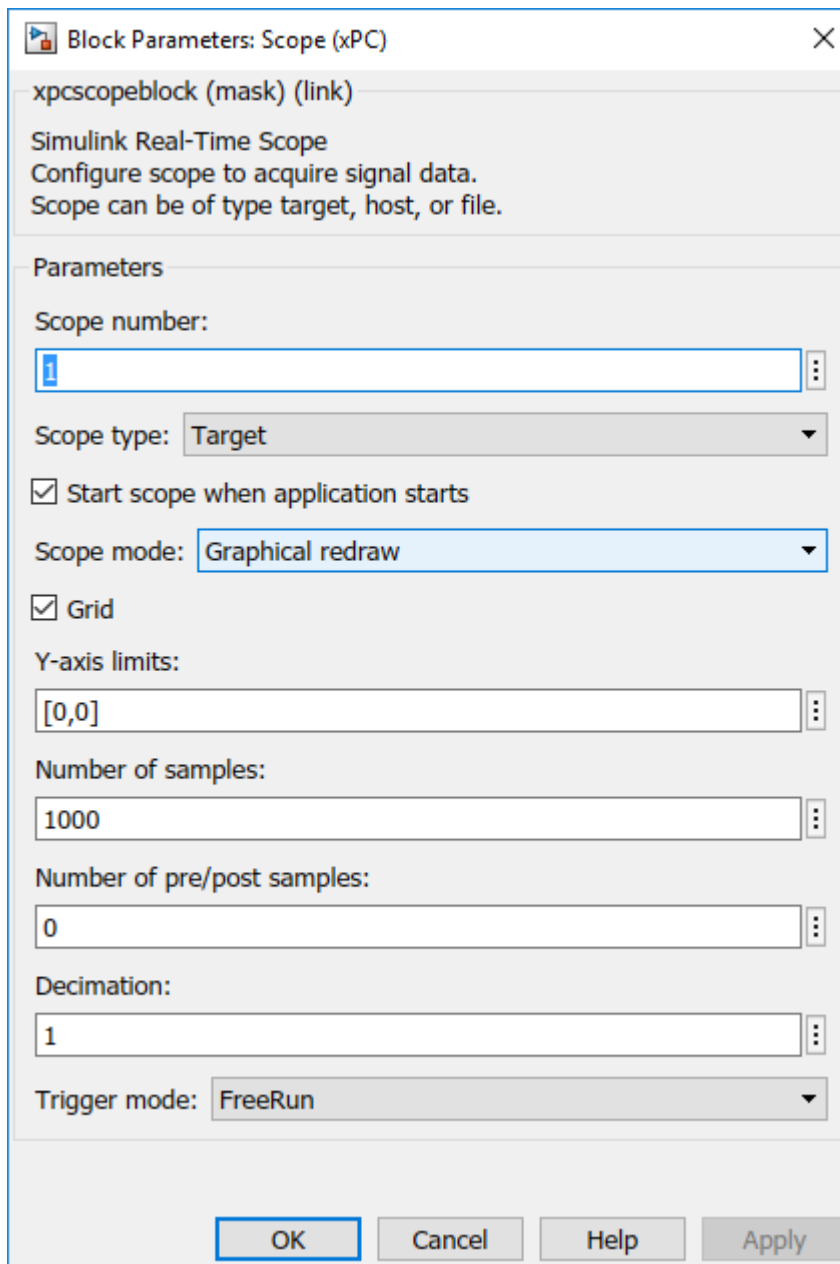
- 8 To display grid lines on the scope, select the **Grid** check box. This parameter is applicable only for target scopes with scope modes of type `Graphical redraw` or `Graphical rolling`.
- 9 In the **Y-Axis limits** box, enter a row vector with two elements. The first element is the lower limit of 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.



- 14 Click **OK**.
- 15 Save the model as `ex_slrt_target_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_target_osc')))`). On the **Simulation** tab, from **Save**, click **Save As**.

See Also

Scope

More About

- “Simulink Real-Time Scope Usage” on page 6-17

- “Target Scope Usage” on page 6-18
- “Trigger One Scope with Another Scope” on page 11-15

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 6-19 and “Target Scope Usage” on page 6-18.



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

- 1 Open model `xpcosc`. Set property **Stop time** to `inf`. On the **Real-Time** tab, select **Run on Target > Stop Time** and set **Stop Time** to `inf`.
- 2 Connect to the target computer. On the **Real-Time** tab, toggle the **Disconnected** indicator to **Connected**.
- 3 Build and download the real-time application to the target computer. On the **Real-Time** tab, click **Run on Target**.
- 4 Open Simulink Real-Time Explorer. On the **Real-Time** tab, click **Prepare > SLRT Explorer**.

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.

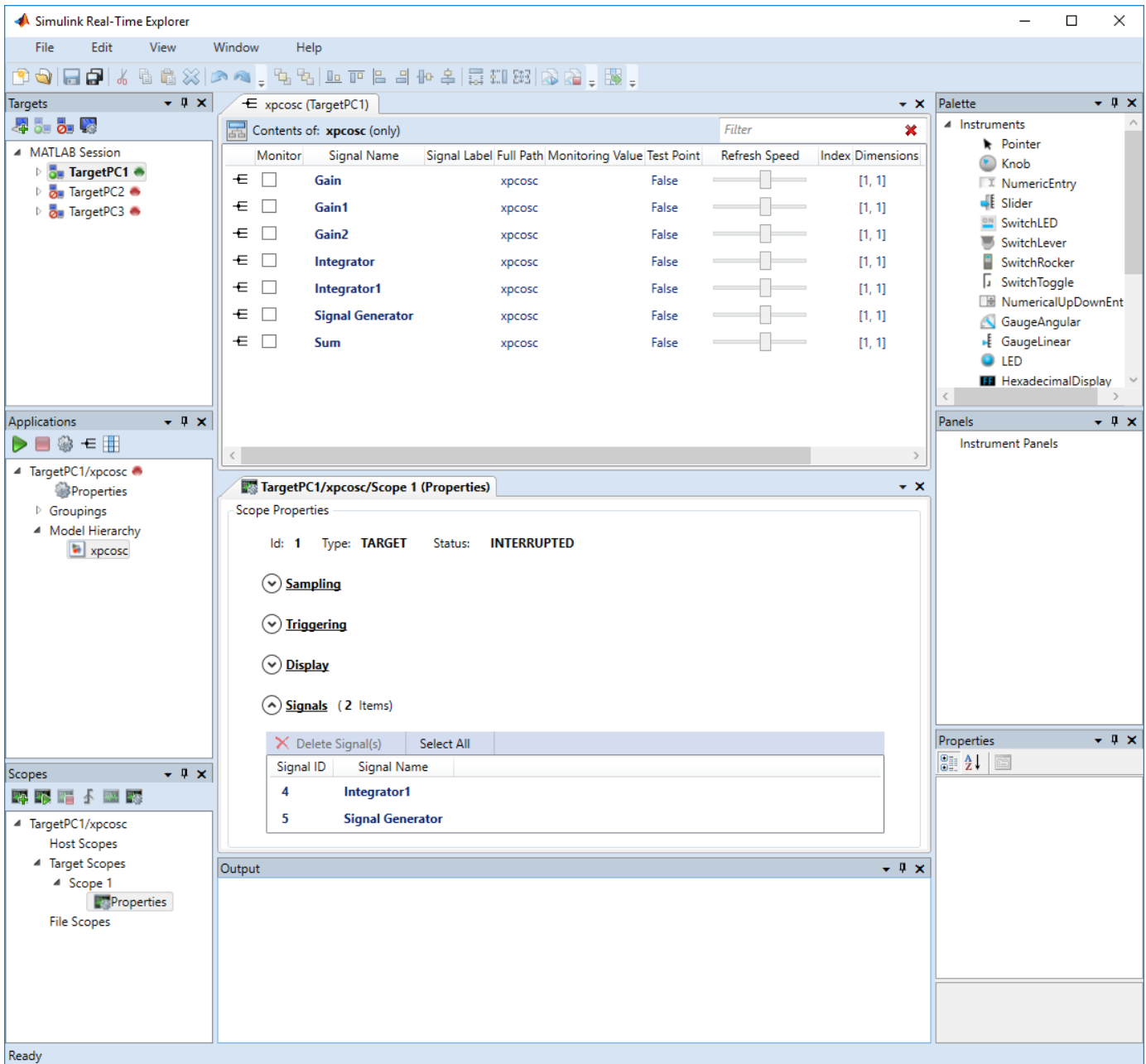
The new scope appears under node **Target Scopes**, for example **Scope 1**.


- 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.




- 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.


- 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'
```

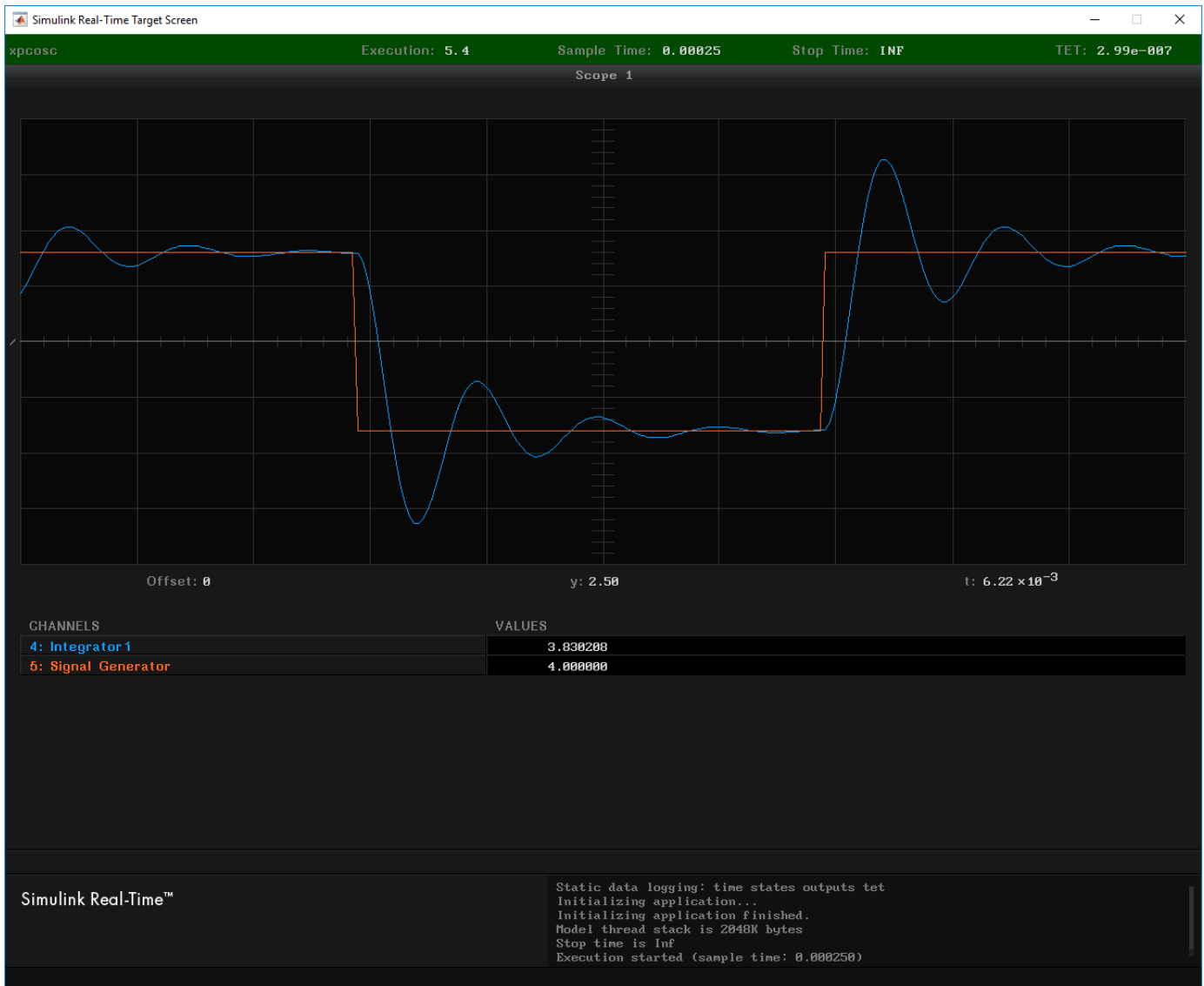
- 12** To stop execution, click the real-time application and then click the **Stop** button  on the toolbar.

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

viewTargetScreen


More About

- “Create Signal Groups with Simulink Real-Time Explorer” on page 6-45
- “Configure Target Scopes with Simulink Real-Time Explorer” on page 6-39
- “Configure Scope Sampling with Simulink Real-Time Explorer” on page 6-29
- “Trigger Scopes with Simulink Real-Time Explorer” on page 6-32
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147

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 6-24. 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.

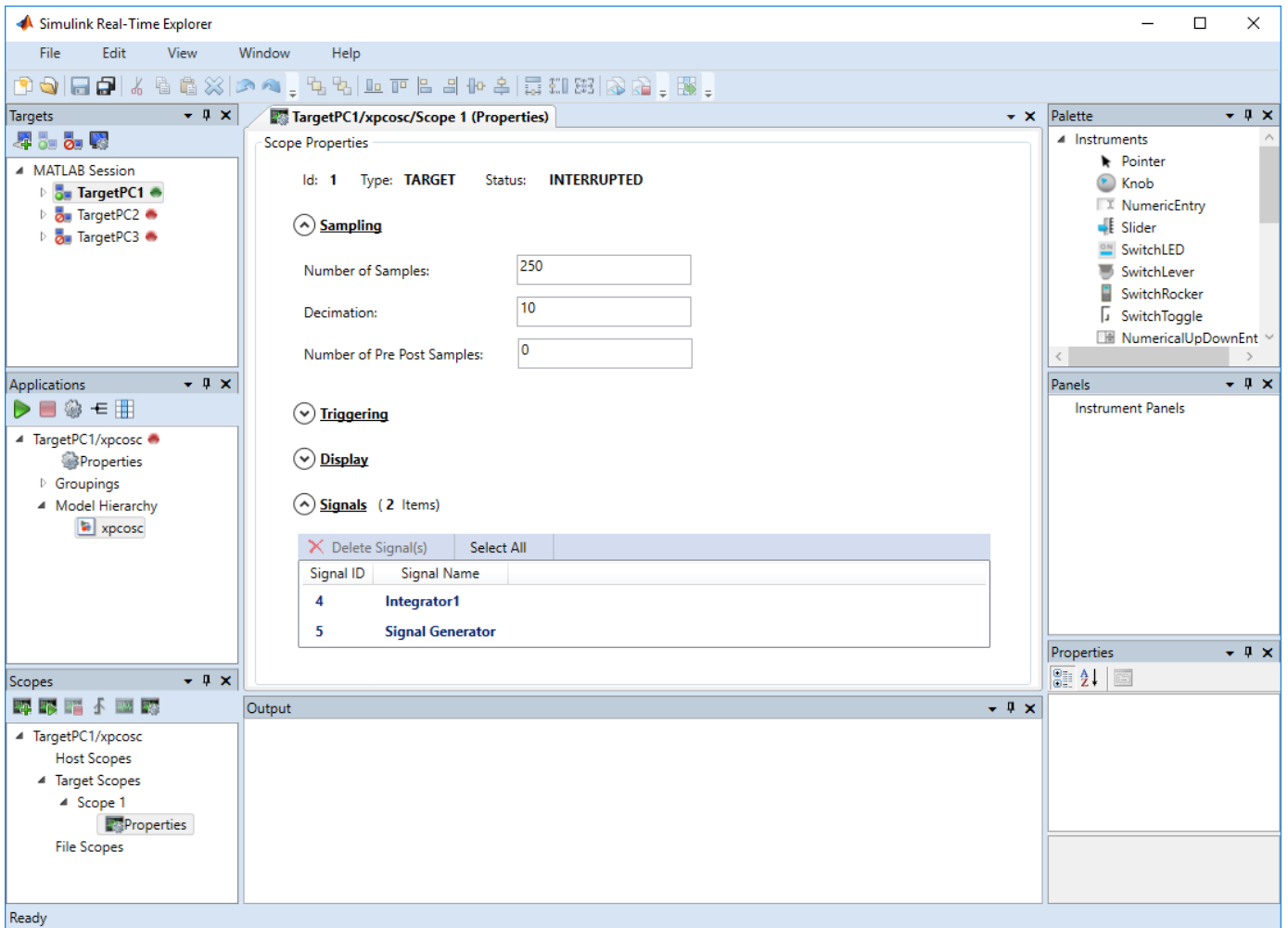
If you select a **Display mode** of `Graphical redraw`, the display redraws the graph every Number of Samples.

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.

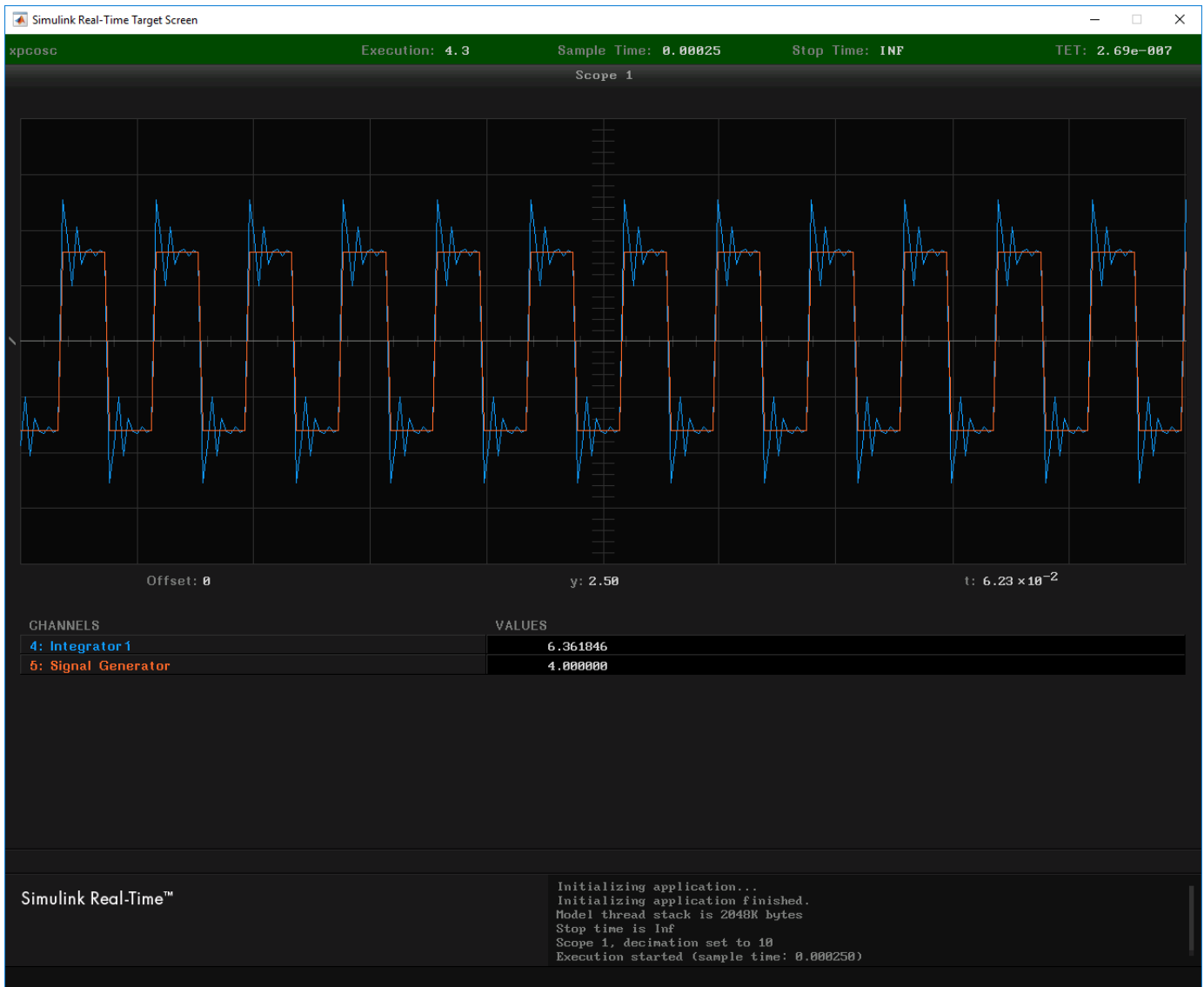
- 4 In the **Decimation** box, enter 10 to indicate that data must be collected at every 10th sample time. The default is 1, to collect data at every sample time.
- 5 In the **Number of pre/post samples** box, enter the number of samples to save or skip. To save N samples before a trigger event, specify 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 6-32





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 6-24. Target execution and scopes must be stopped.

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).

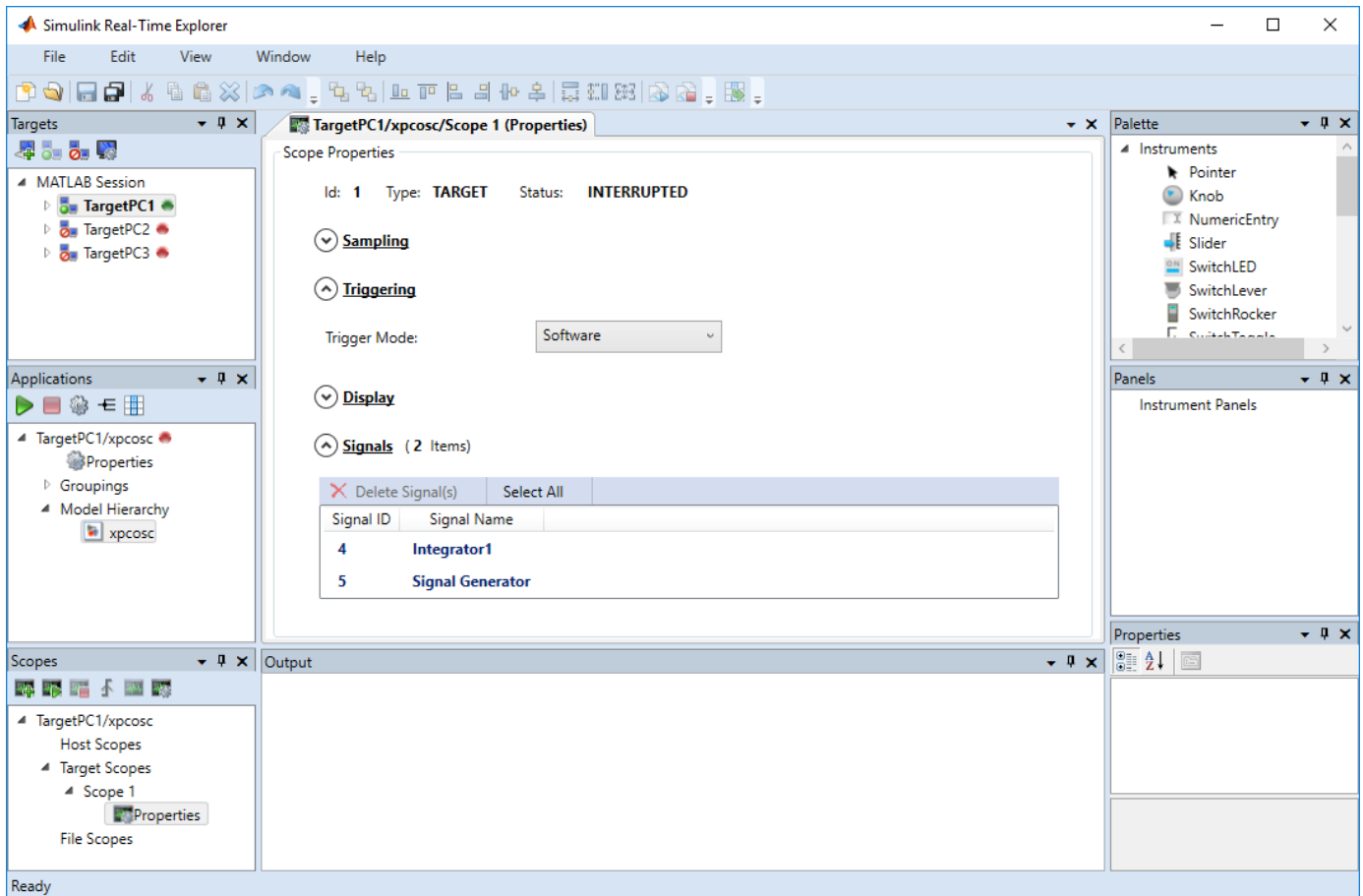
The **Trigger** button  is enabled on the toolbar.

- 4 Click the **Trigger**  button on the **Scopes** toolbar.

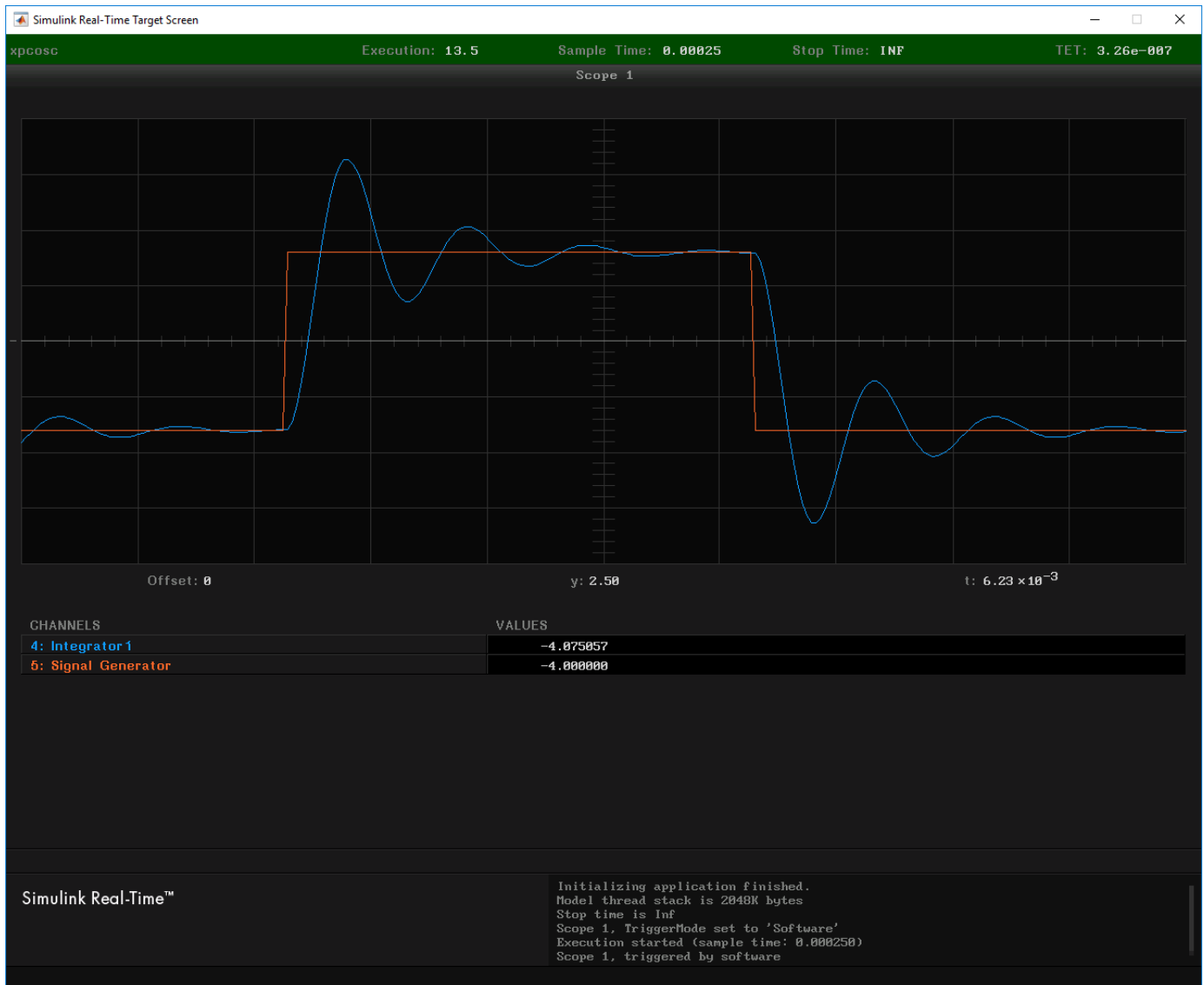
The current signal data is displayed on the target computer monitor when you click the button.

- 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

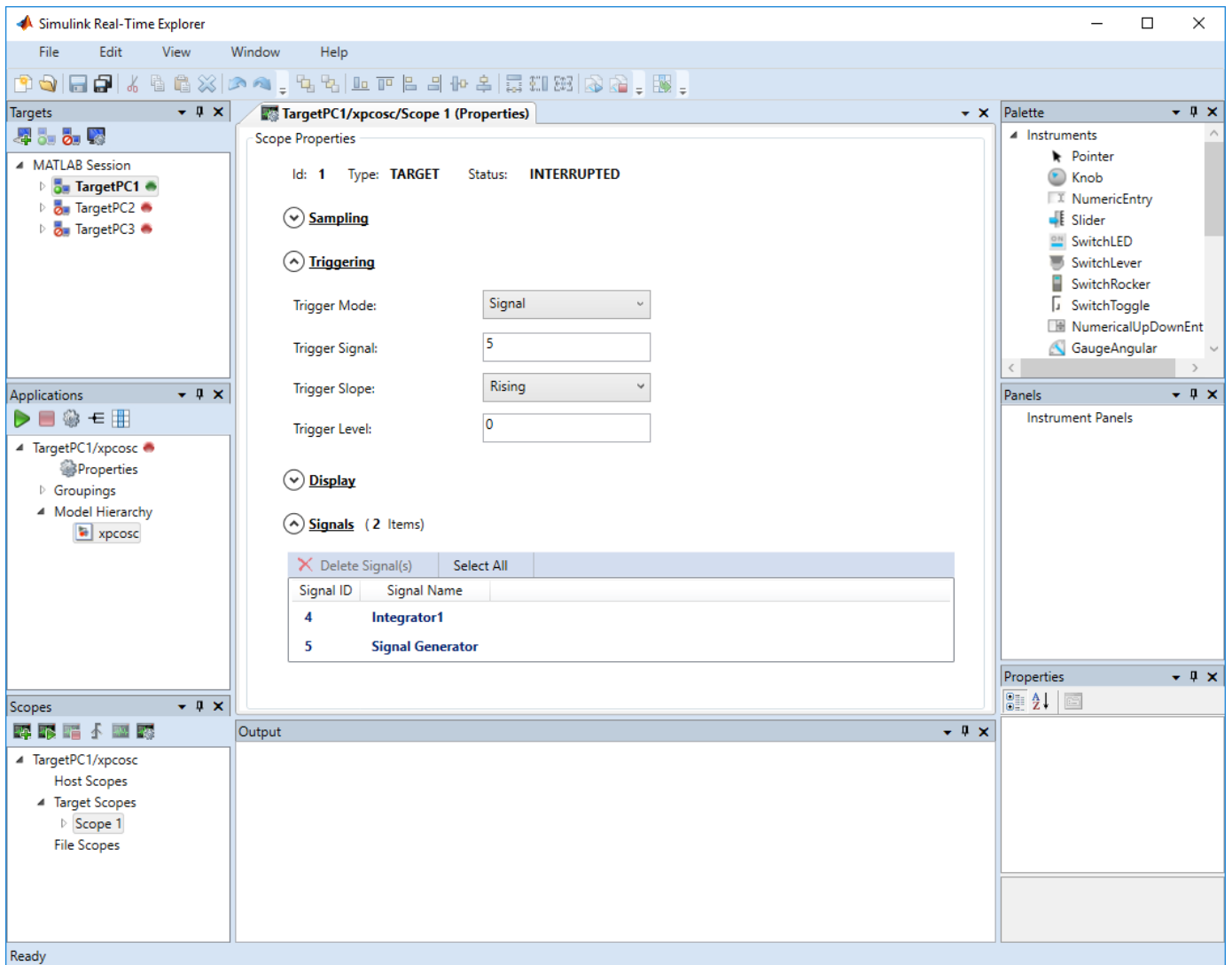
In **Trigger Mode Signal**, the scope triggers when a signal rises or falls through a specified level.

- 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.

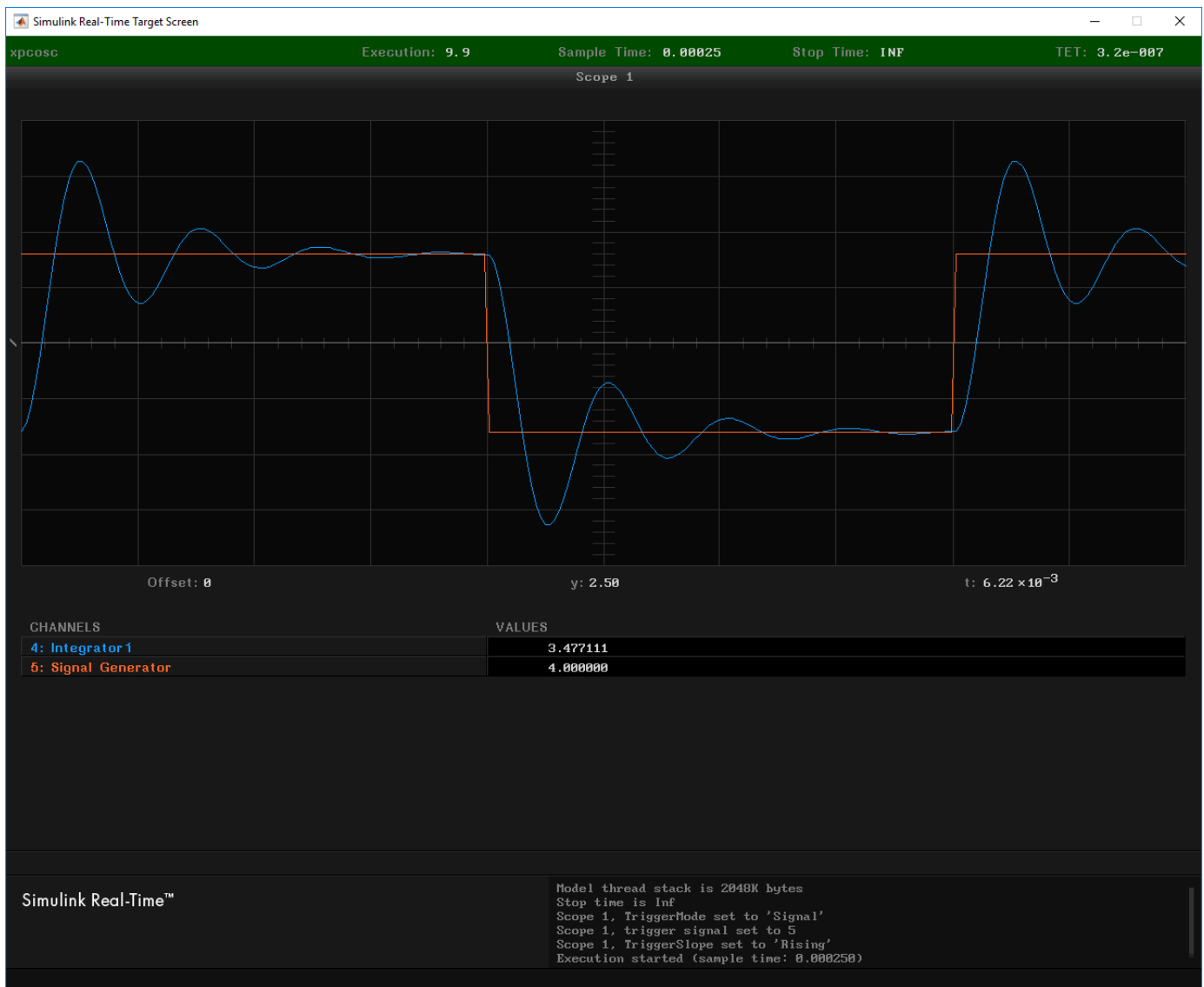
- 5 Type the number displayed on the target computer screen for **Signal Generator** (here, 5) in the **Trigger Signal** text box.

- 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

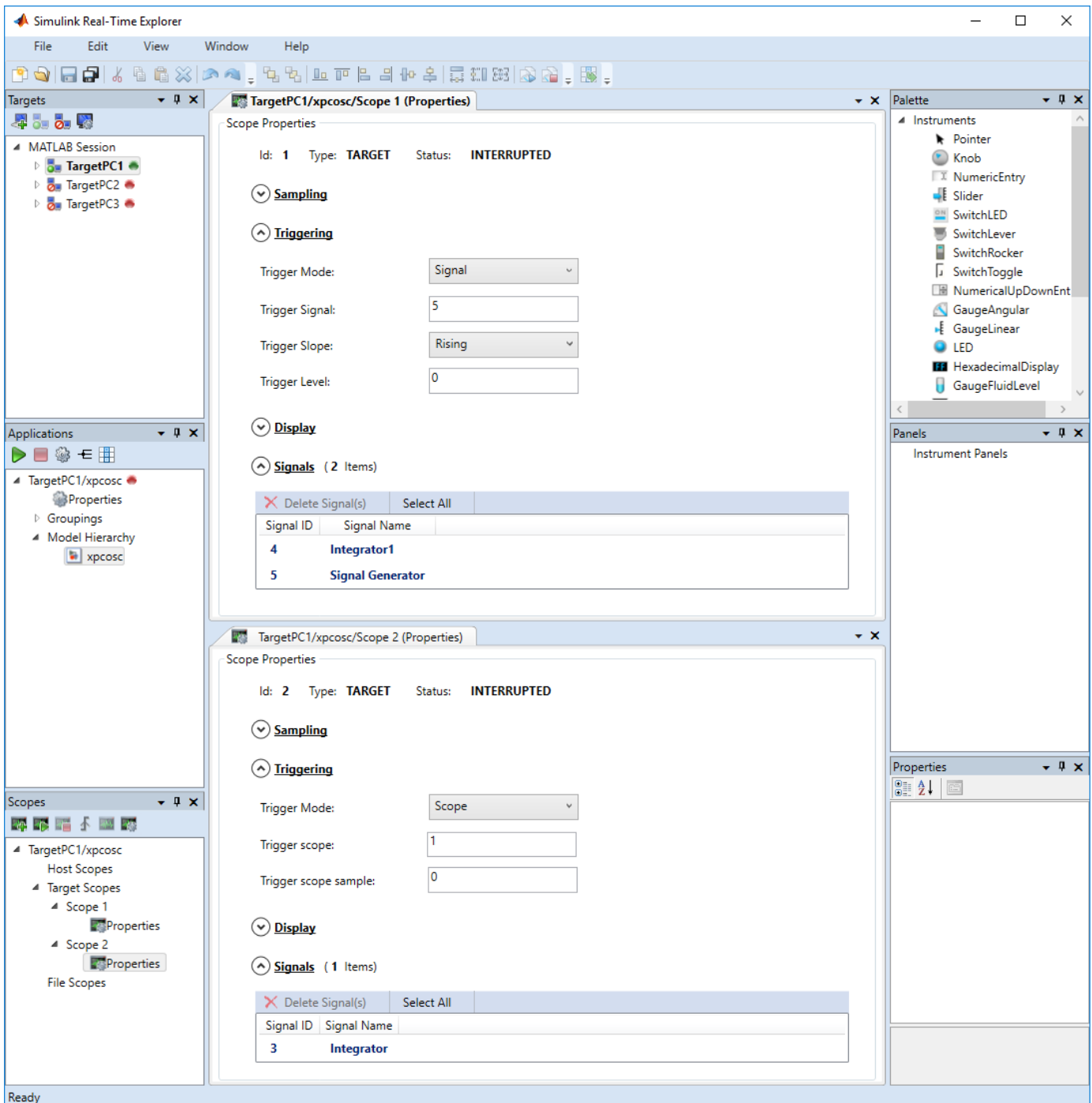
In **Trigger Mode** Scope, the scope triggers when another scope triggers. In this example, Scope 2 triggers when signal-triggered Scope 1 triggers.


- 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**.



- 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 6-147

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 6-24. 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**.

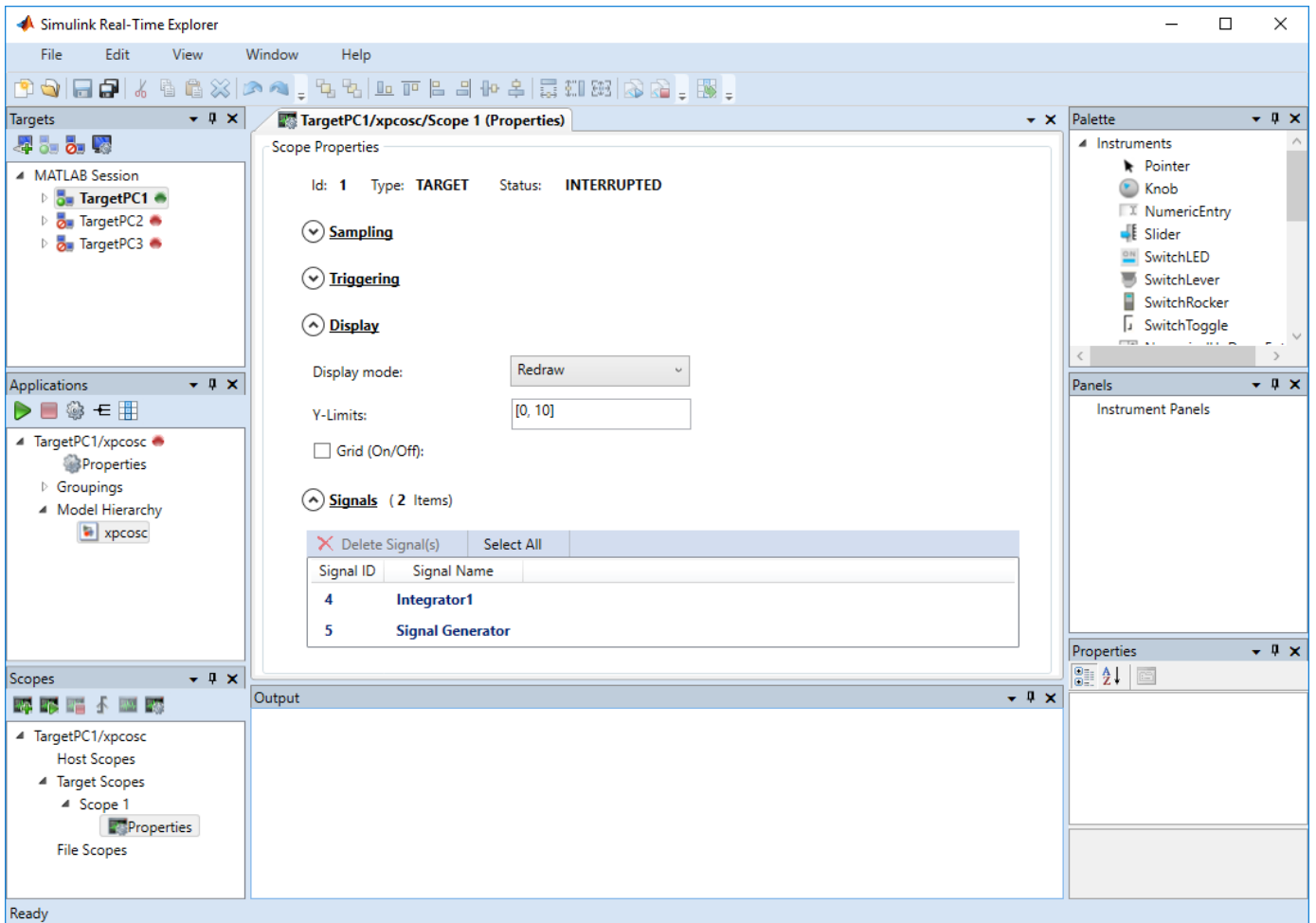
- 5 Start **Scope 1** ( on the toolbar).

Signal data is displayed on the target computer monitor, appearing to move to the left.

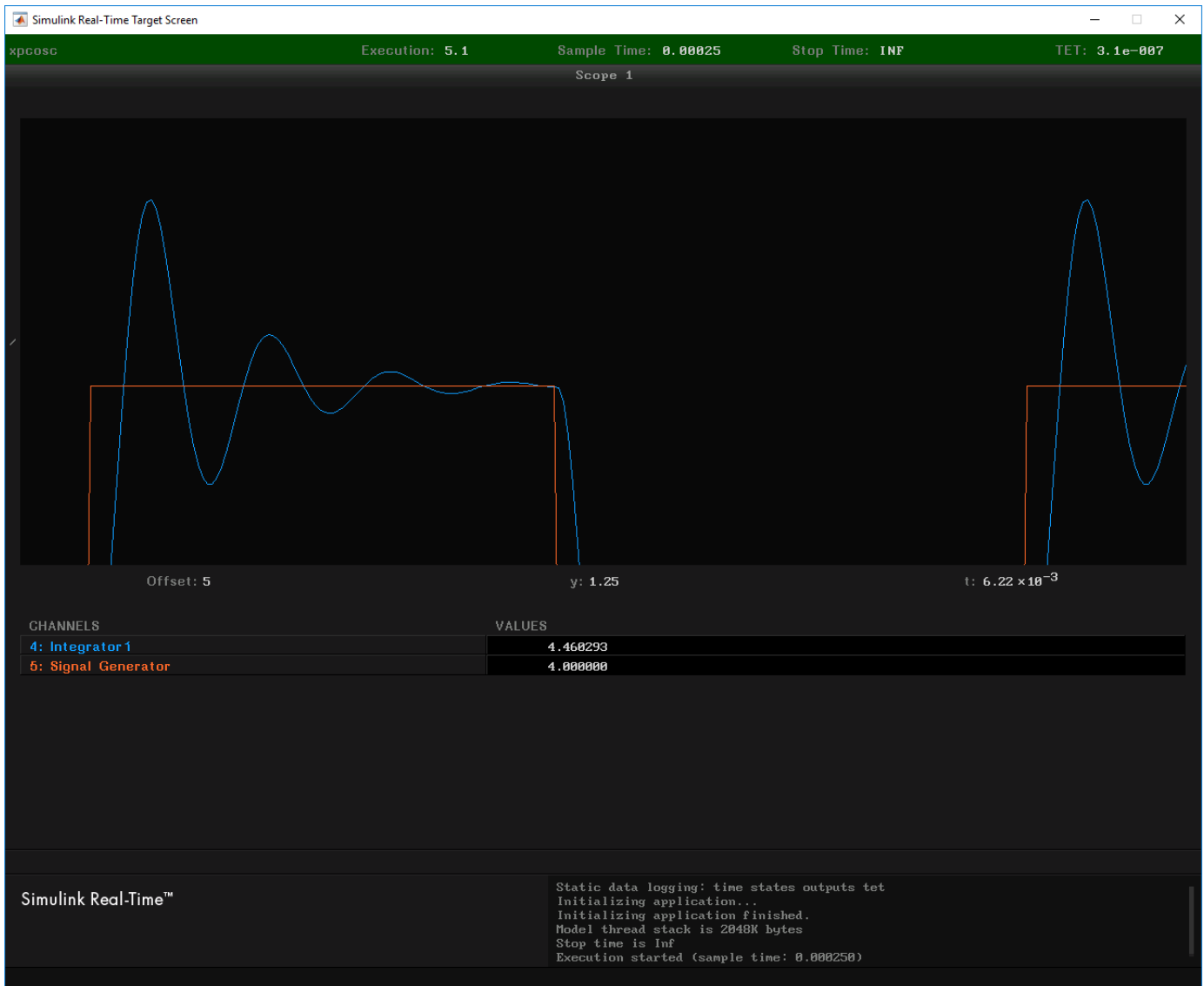
- 6 Enter `[0, 10]` in the **Y-Limits** box and then press **Enter**. The default setting is `[0, 0]`, which automatically scales the output according to the signal values.

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

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 6-7

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 Open model `xpcosc`.
- 2 Connect to the target computer. On the **Real-Time** tab, toggle the **Disconnected** indicator to **Connected**.
- 3 Build and download the real-time application to the target computer. On the **Real-Time** tab, click **Run on Target**.
- 4 Open Simulink Real-Time Explorer. On the **Real-Time** tab, click **Prepare > SLRT Explorer**.


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**.

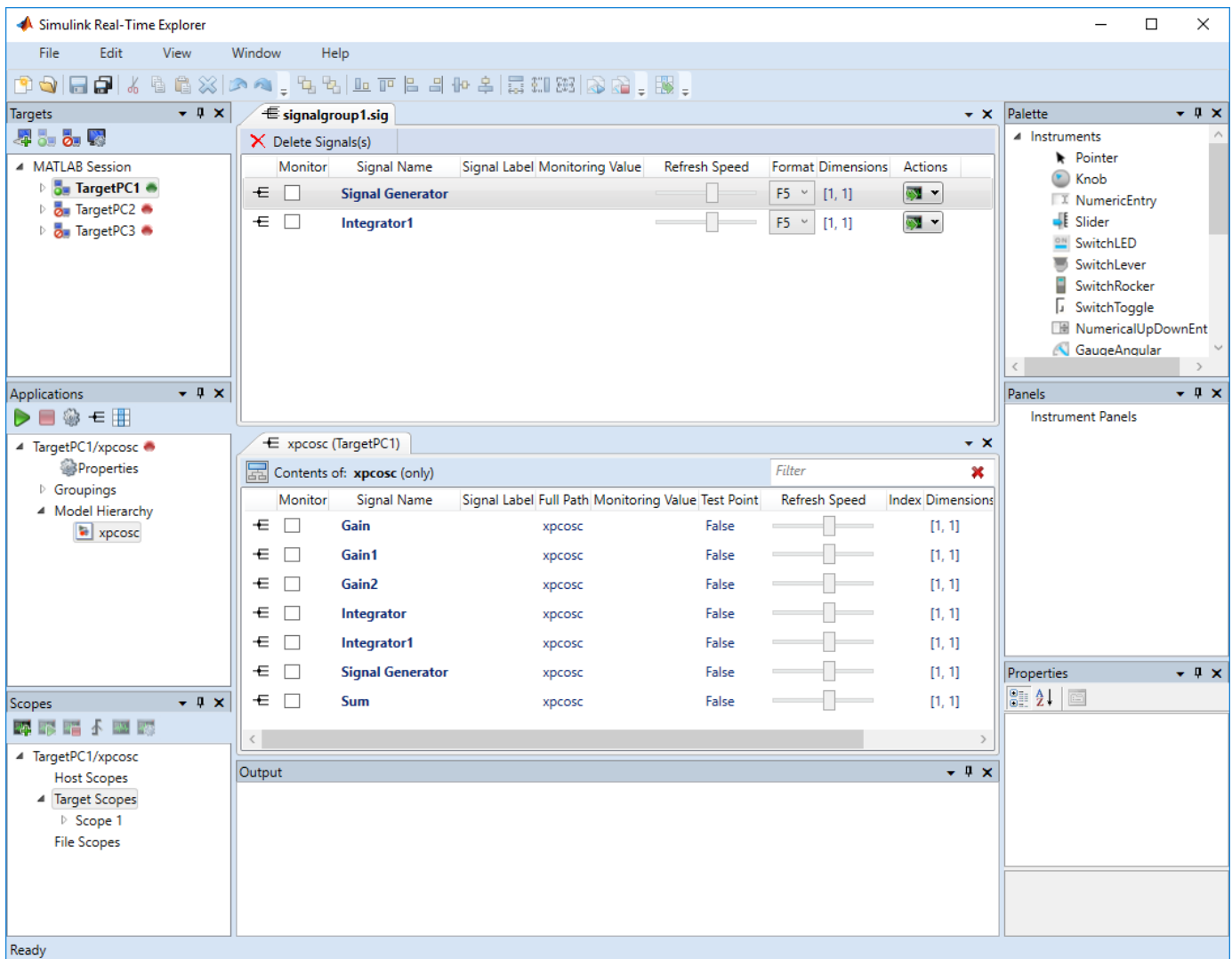
A new signal group appears, along with its Signal Group workspace.

- 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 6-12.



Signals are defined within a particular real-time application. To open a signal group from the **File > Open > Group** menu, you must first select an application.

To remove signals from the signal group, select the signal items in the group list and click **Delete Signals**.

To remove the signal group, navigate to the signal group under **Groupings > Signals**, right-click the signal group, and click **Remove**.

To make both workspaces visible at the same time, drag one workspace tab down until the  icon appears in the middle of the dialog box. Continue to drag 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 6-4
- “Create Target Scopes with Simulink Real-Time Explorer” on page 6-24
- “Create Host Scopes with Simulink Real-Time Explorer” on page 6-52
- “Create File Scopes with Simulink Real-Time Explorer” on page 6-81
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147

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 6-49
- “Create Host Scopes with Simulink Real-Time Explorer” on page 6-52
- “Simulink Real-Time Scope Usage” on page 6-17
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147

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.

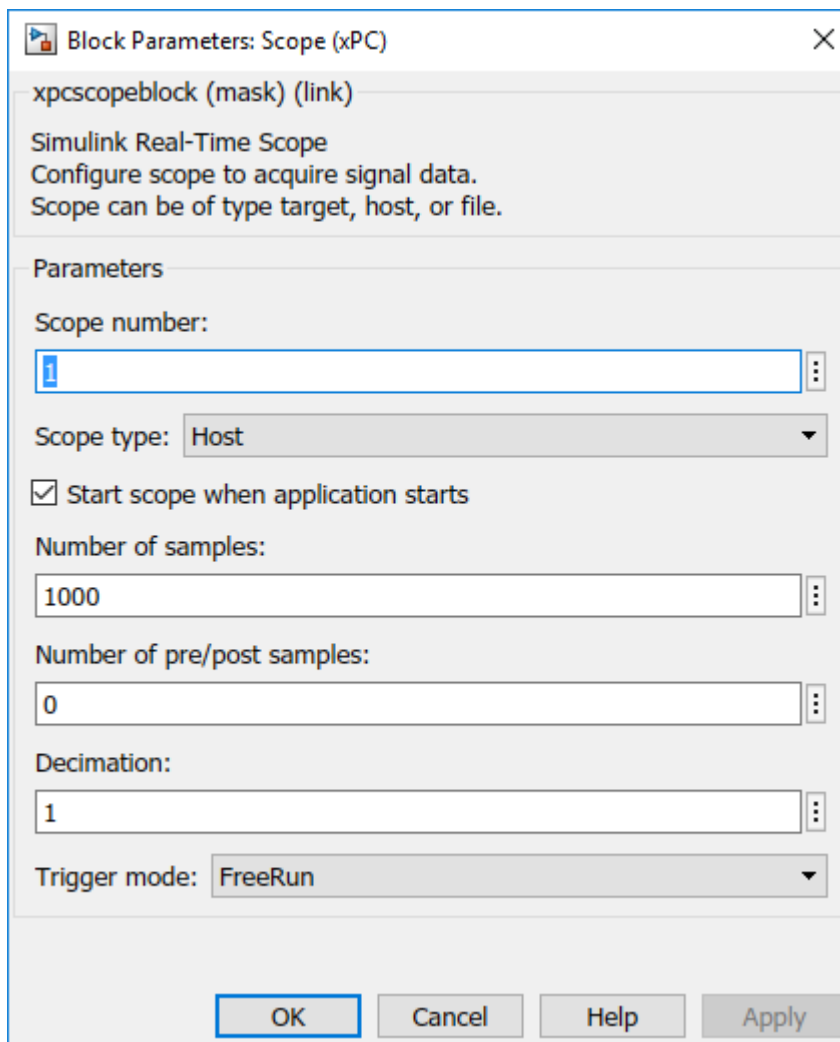
- 6 In the **Number of samples** box, enter the number of values to be acquired in a data package.
- 7 In the **Number of pre/post samples** box, enter the number of samples to save or skip. To save *N* samples before a trigger event, specify the value $-N$. To skip *N* samples after a trigger event, 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 Save the model as `ex_slrt_host_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_host_osc')))`). On the **Simulation** tab, from **Save**, click **Save As**.

See Also

Scope

More About

- “Simulink Real-Time Scope Usage” on page 6-17
- “Host Scope Usage” on page 6-48
- “Create Host Scopes with Simulink Real-Time Explorer” on page 6-52
- “Trigger One Scope with Another Scope” on page 11-15

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 6-49 and “Host Scope Usage” on page 6-48.



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 Open model `ex_slrt_sf_car`. Set property **Stop time** to `inf`. On the **Real-Time** tab, select **Run on Target > Stop Time** and set **Stop Time** to `inf`.
- 2 Connect to the target computer. On the **Real-Time** tab, toggle the **Disconnected** indicator to **Connected**.
- 3 Build and download the real-time application to the target computer. On the **Real-Time** tab, click **Run on Target**.
- 4 Open Simulink Real-Time Explorer. On the **Real-Time** tab, click **Prepare > SLRT Explorer**.

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**.

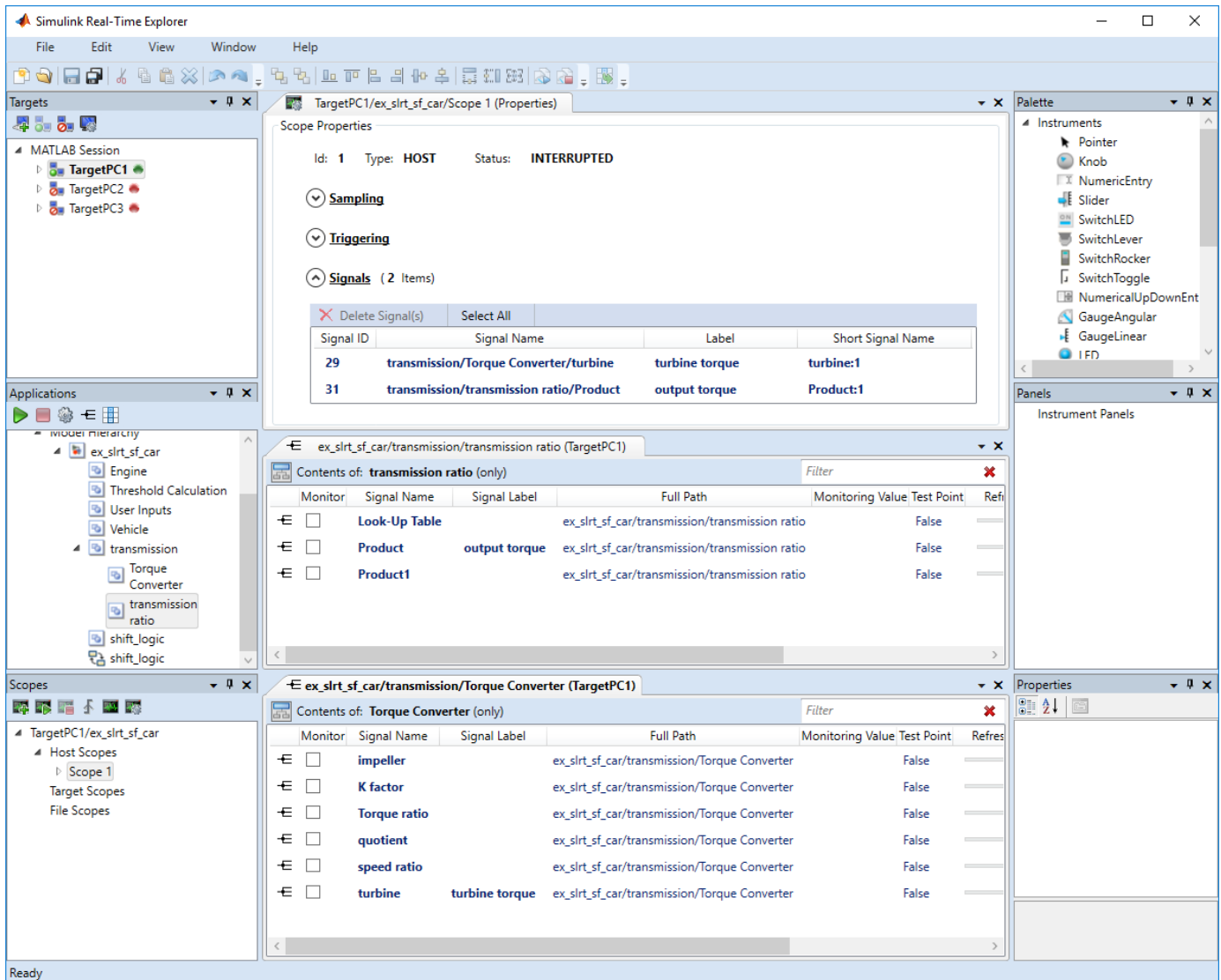
- 4 In the **Applications** pane, expand the real-time application node, and then the node **Model Hierarchy**.
- 5 Double-click the `ex_slrt_sf_car > transmission > Torque Converter` node.

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

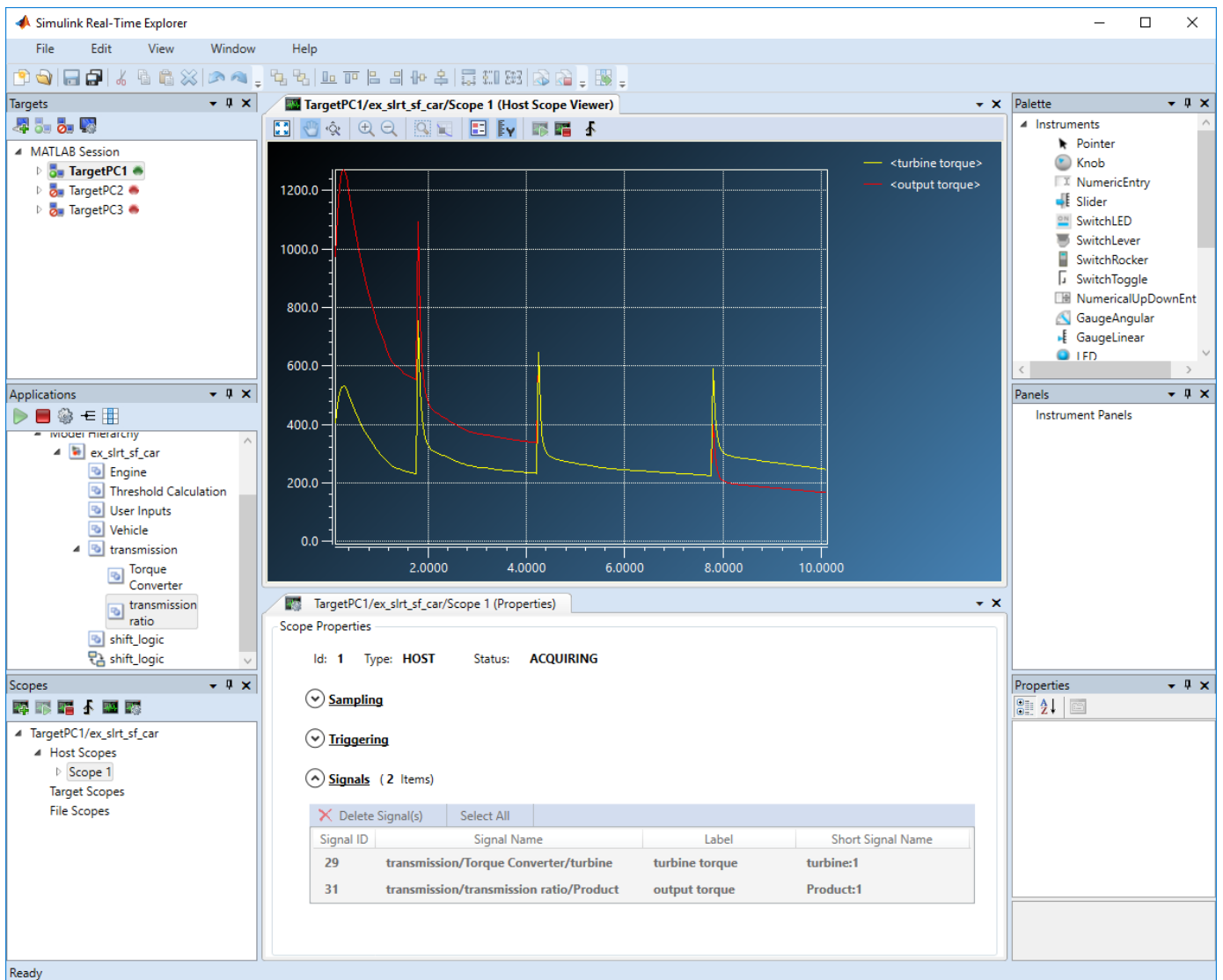
- 1 Select **Scope 1**, and then click the **View Scope** button  on the toolbar.



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 6-48
- “Configure Real-Time Host Scope Blocks” on page 6-49
- “Configure the Host Scope Viewer” on page 6-56
- “Create Signal Groups with Simulink Real-Time Explorer” on page 6-45

- “Configure Scope Sampling with Simulink Real-Time Explorer” on page 6-29
- “Trigger Scopes with Simulink Real-Time Explorer” on page 6-32
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147

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 6-52. 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 6-32

Trace Signals with Simulink External Mode

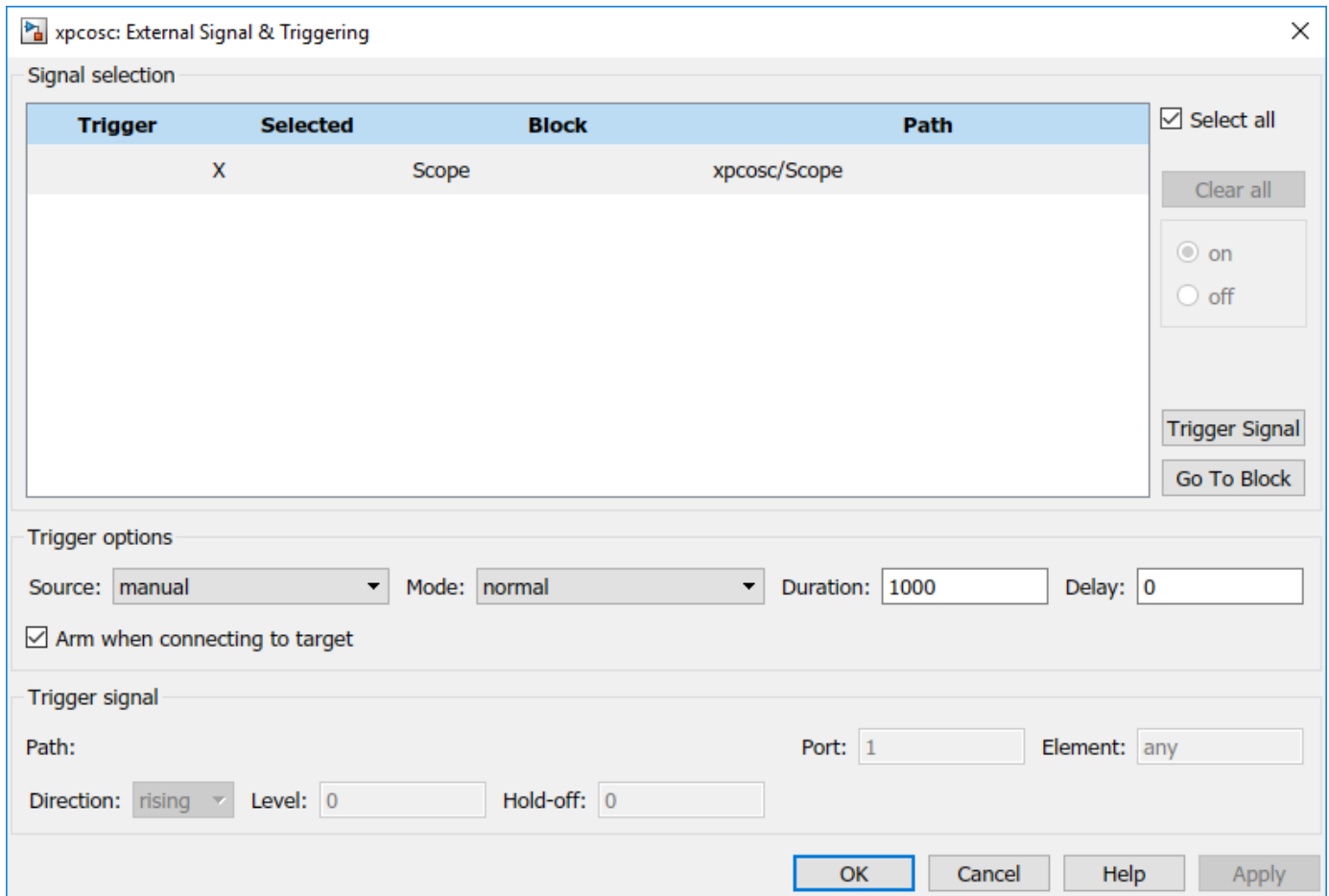
You can use Simulink external mode to establish a communication channel between your Simulink block diagram and your real-time application. The block diagram becomes a user interface to your real-time application. Simulink scopes can display signal data from the real-time application, including from 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.

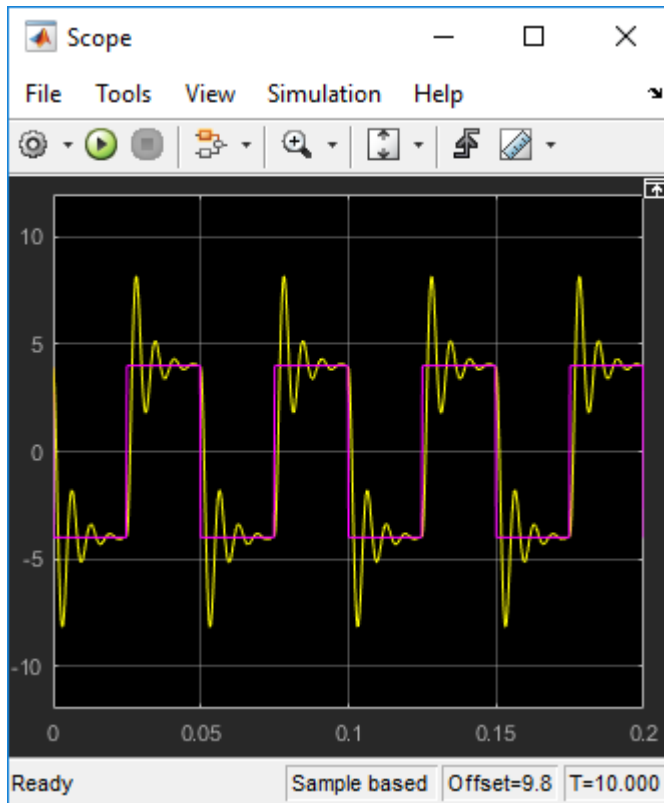
- 1 Open model `xpcosc`.
- 2 Open the external mode control panel. In the Simulink Editor, on the **Real-Time** tab, click **Prepare > Control Panel**.
- 3 In the external mode control panel, click the **Signal & Triggering** button.
- 4 In the External Signal & Triggering dialog box, set the **Source** 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 Save the model as `ex_slrt_ext_osc`. On the **Simulation** tab, from **Save**, click **Save As**.
- 13 If a scope window is not displayed for the Scope block, double-click the Scope block.
- 14 Connect to the target computer. On the **Real-Time** tab, toggle the **Disconnected** indicator to **Connected**.
- 15 Build and download the real-time application to the target computer. On the **Real-Time** tab, click **Run on Target**.

The real-time application begins running on the target computer. The Scope window displays plotted data.



16 To stop the simulation, on the **Real-Time** tab click **Stop**.

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.

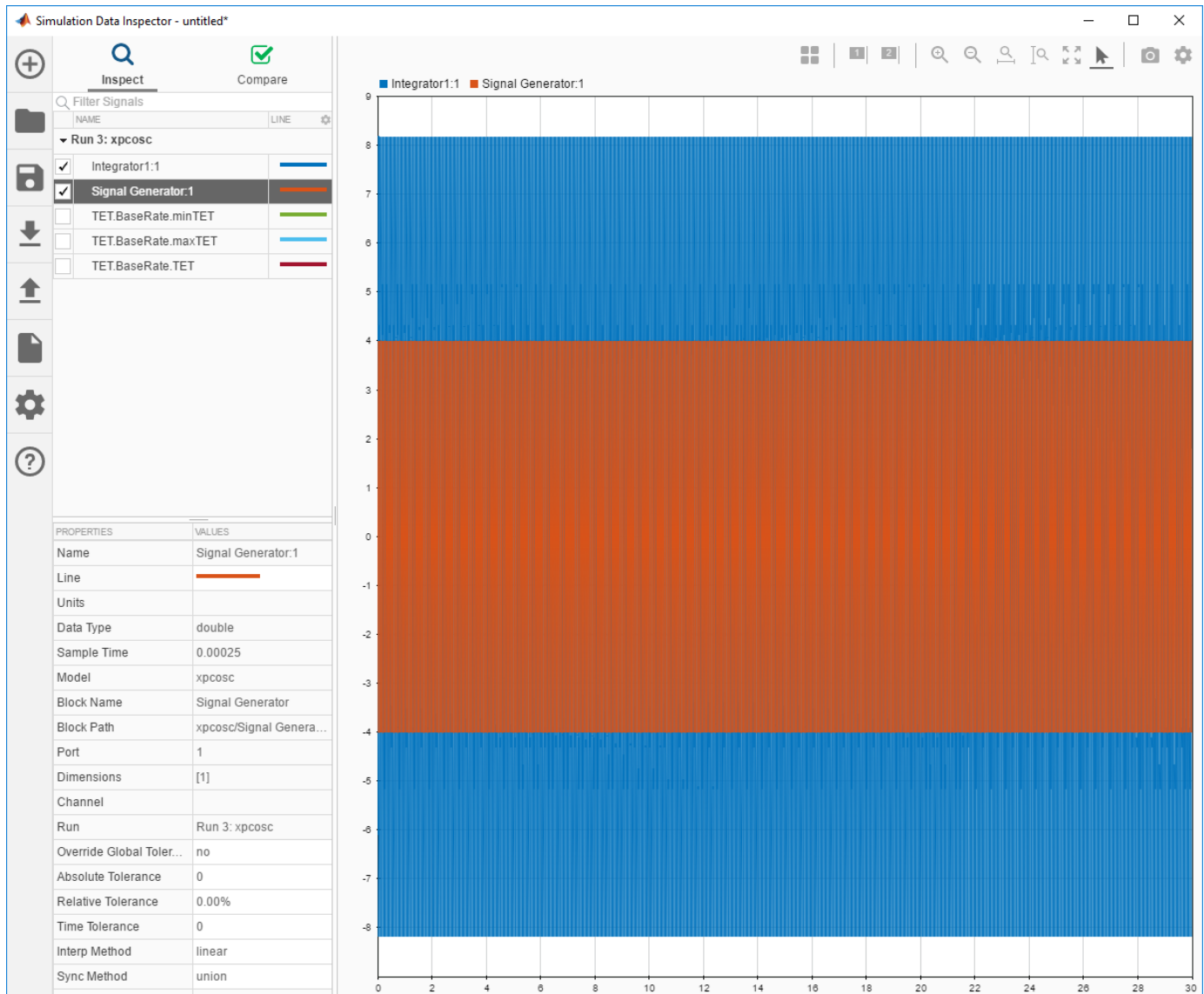
Setup the Simulation Data Inspector

Make sure that you have started the target computer and established communication between the development and target computers.

- 1 Open model `xpcosc`.
- 2 Increase the simulation stop time to, for example, 10 seconds. On the **Real-Time** tab, pull down **Run on Target** and type the value in the **Stop Time** box.
- 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. On the **Real-Time** tab, click **Run on Target**.

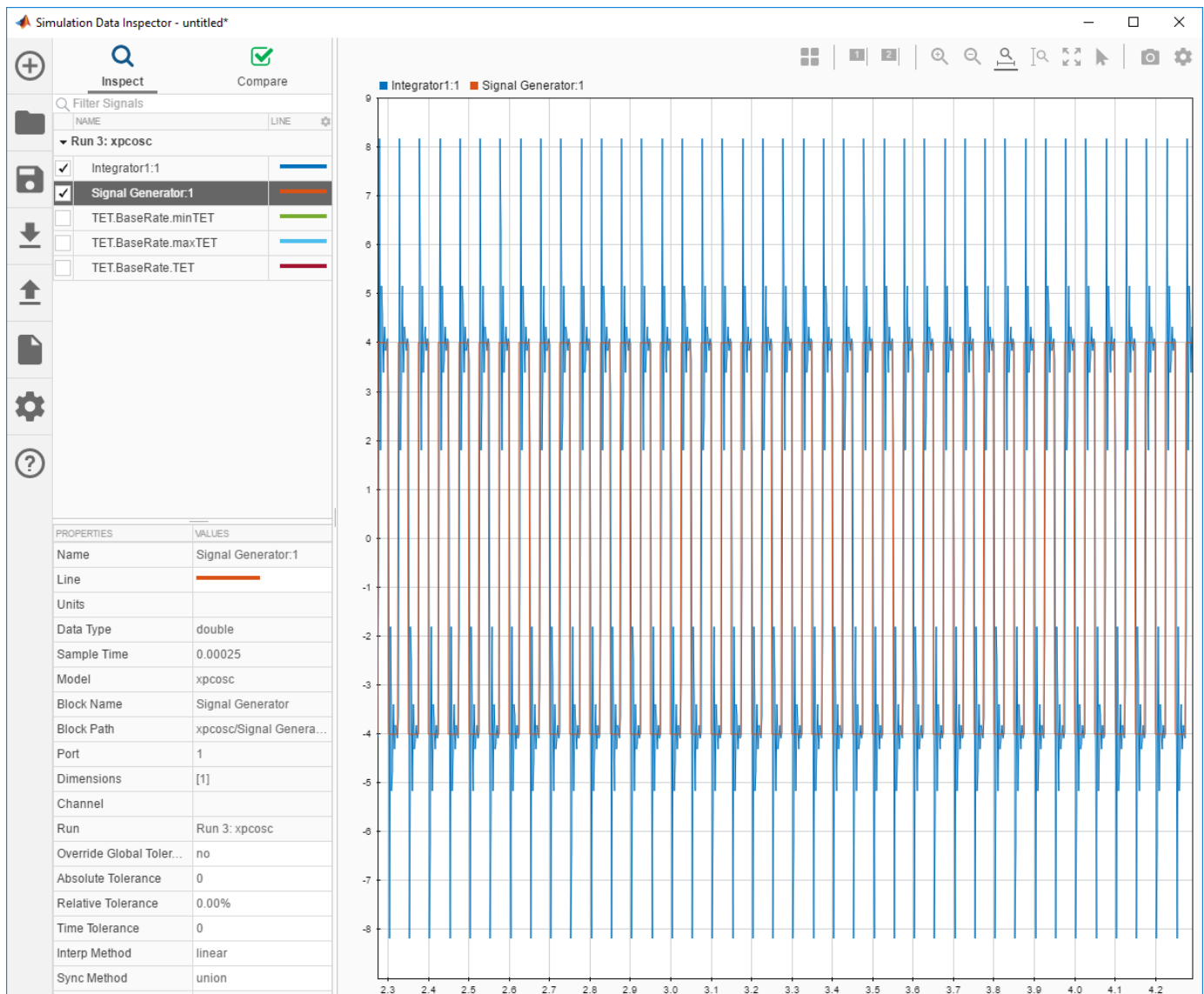
Inspect Signal Data

- 1 Open Simulink Real-Time Explorer. On the **Real-Time** tab, pull down the **Prepare** section and click **SLRT Explorer**.
- 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 the Simulation Data Inspector, select the signals `Integrator1:1` and `SignalGenerator:1`. The Simulation Data Inspector displays plotted signal data.



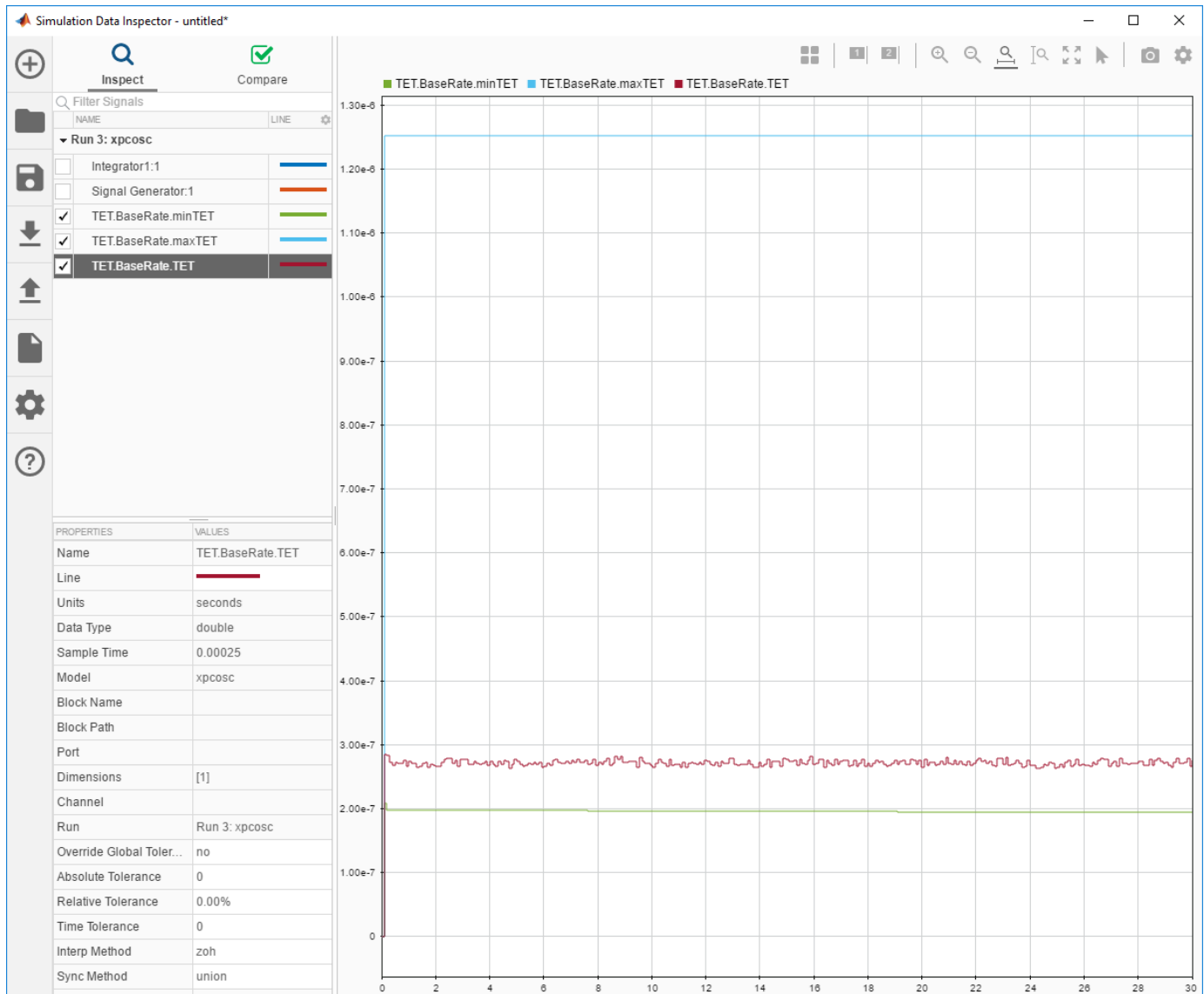
5. Stop the real-time application. On the **Real-Time** tab, click **Stop**.

6. After the simulation, use the Simulation Data Inspector 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 `.mldatx` file, click **Save**.

See Also

`SimulinkRealTime.utils.TETMonitor.open`

More About

- “Trace or Log Data with the Simulation Data Inspector” on page 6-68
- Simulation Data Inspector

Stream Signal Data from Target Computer to Simulation Data Inspector

This example shows how to create a signal list for a Simulink Real-Time model by using the streaming signals API. After you build the real-time application from the model and run the application on the target computer, you can stream signal data to the Simulation Data Inspector from dynamically selected signals.

Create a Signal List Object for Blocks

Open the model, identify the programmatic names of blocks in the model, and create a signal list object. For more information, see `SimulinkRealTime.SignalList`.

```
open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcosc'));
find_system('xpcosc')
mySignals=SimulinkRealTime.SignalList()
```

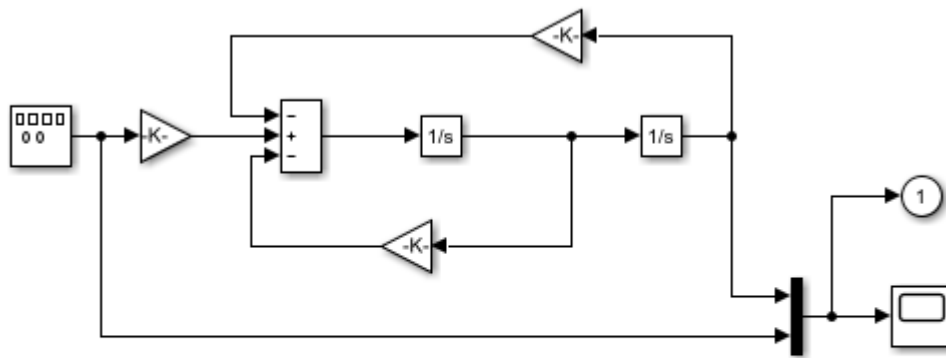
```
ans =
```

```
11x1 cell array
```

```
{'xpcosc'           }
{'xpcosc/Gain'     }
{'xpcosc/Gain1'    }
{'xpcosc/Gain2'    }
{'xpcosc/Integrator' }
{'xpcosc/Integrator1' }
{'xpcosc/Mux'      }
{'xpcosc/Scope'    }
{'xpcosc/SignalGenerator' }
{'xpcosc/Sum'      }
{'xpcosc/Outport'  }
```

```
mySignals =
```

```
SignalList with no properties.
```



Model xpcosc
 Simulink Real-Time example model
 Copyright 1999-2013 The MathWorks, Inc.

Add Signals to Signal List Object

To add signals by name to the signal list object, get parameter information for the signals and use the parameter information in the `add` command. Adding signals by name makes it easier to find signals in the Simulation Data Inspector when viewing the signal data.

```
p = get_param('xpcosc/Integrator', 'PortHandles');
l = get_param(p.Outport, 'Line');
set_param(l, 'Name', 'Integ_out');
add(mySignals, 'Integ_out')
p = get_param('xpcosc/Integrator1', 'PortHandles');
l = get_param(p.Outport, 'Line');
set_param(l, 'Name', 'Integ1_out');
add(mySignals, 'Integ1_out')
```

You also can add signals to the signal list object by block path and port index. To add the `xpcosc` model `Integrator` block input to the signal list by block path and port index, the command is:

```
add(mySignals, 'xpcosc/Integrator', 1);
```

View Signals in Signal List Object

To view signals in the signal list object, use the `view` command.

```
view(mySignals)

Integ_out
Integ1_out
```

Build the Model and Run the Application

Build the model and download the real-time application to the target computer. After building the real-time application, you can close the model.

The model does not need to be open to stream signal data from the real-time application.

```
evalc('rtwbuild(''xpcosc'')');
tg = slrt('TargetPC1');
```

```
load(tg, 'xpcosc');  
bdclose('all');
```

Set the stop time and start the real-time application. To generate signal data, run the application for 20 seconds.

```
tg.StopTime=Inf;  
start(tg);  
pause(20);
```

Stream Signal data to Simulation Data Inspector

To select signals to stream, use the `setStreamingSignals` command. This signal selection with the signal list object is dynamic because the selection occurs after you build and download the real-time application.

After you select signals to stream, display the signals in the Simulation Data Inspector (SDI). For more information, see `setStreamingSignals`.

```
setStreamingSignals(tg, mySignals);  
Simulink.sdi.view;
```

To stop streaming signals, use the command:

```
setStreamingSignals(tg, []);  
stop(tg);
```

See Also

`SimulinkRealTime.utils.TETMonitor.open`

More About

- “Trace or Log Data with the Simulation Data Inspector” on page 6-68
- Simulation Data Inspector

Trace or Log Data with the Simulation Data Inspector

With the Simulation Data Inspector 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.

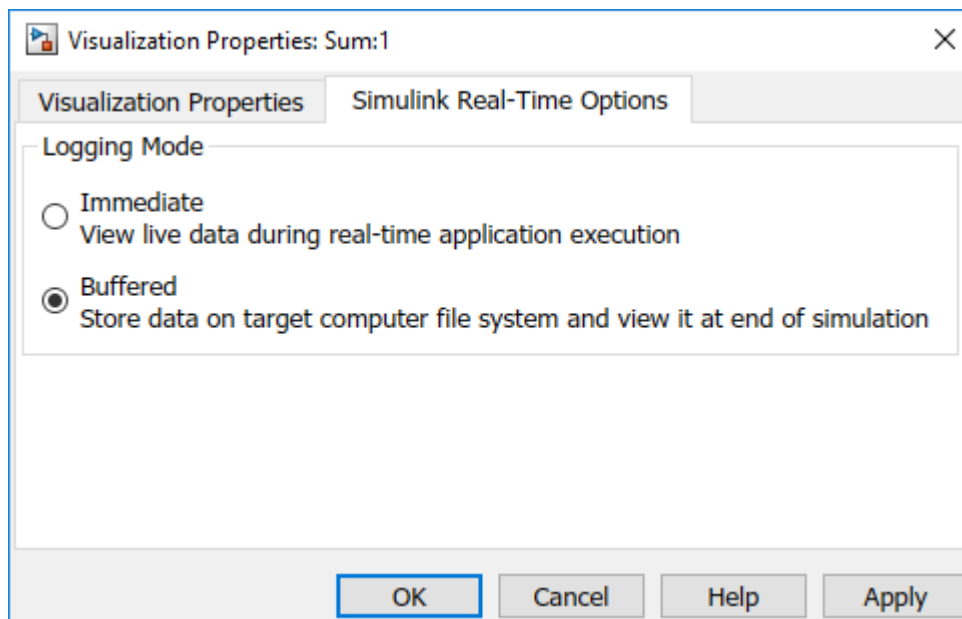
To avoid dropped data points caused by network buffer overruns, you can use buffered logging mode. In buffered mode, the kernel stores data for the buffered signals in a file on the target computer. At the end of execution, it transmits it to the development computer. You can then view the most important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

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 the Simulation Data Inspector

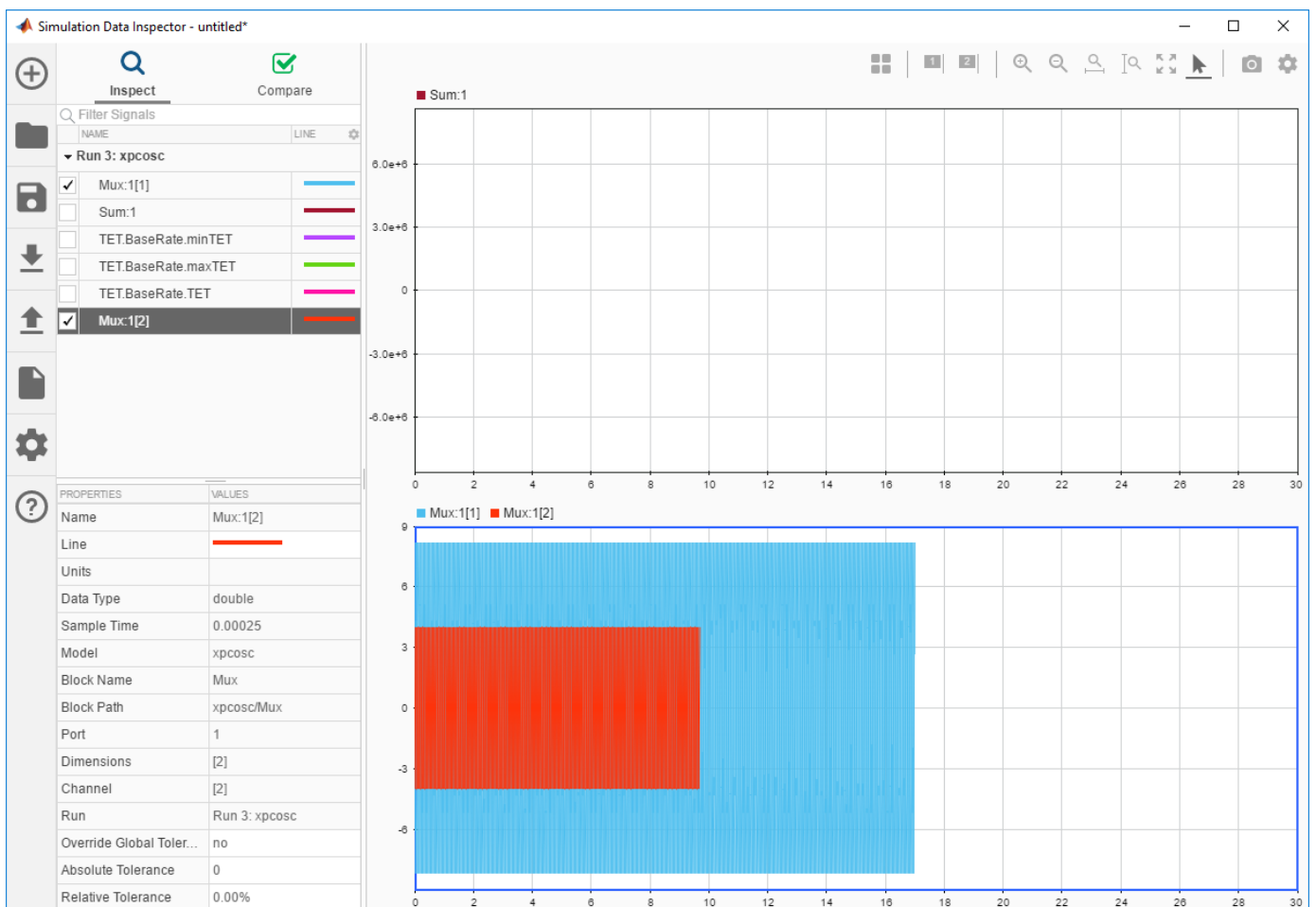
- 1 Open the Simulation Data Inspector (📊).

- 2 Click the **Layout** button (☐).
- 3 Select two horizontal displays.


View Simulation Data

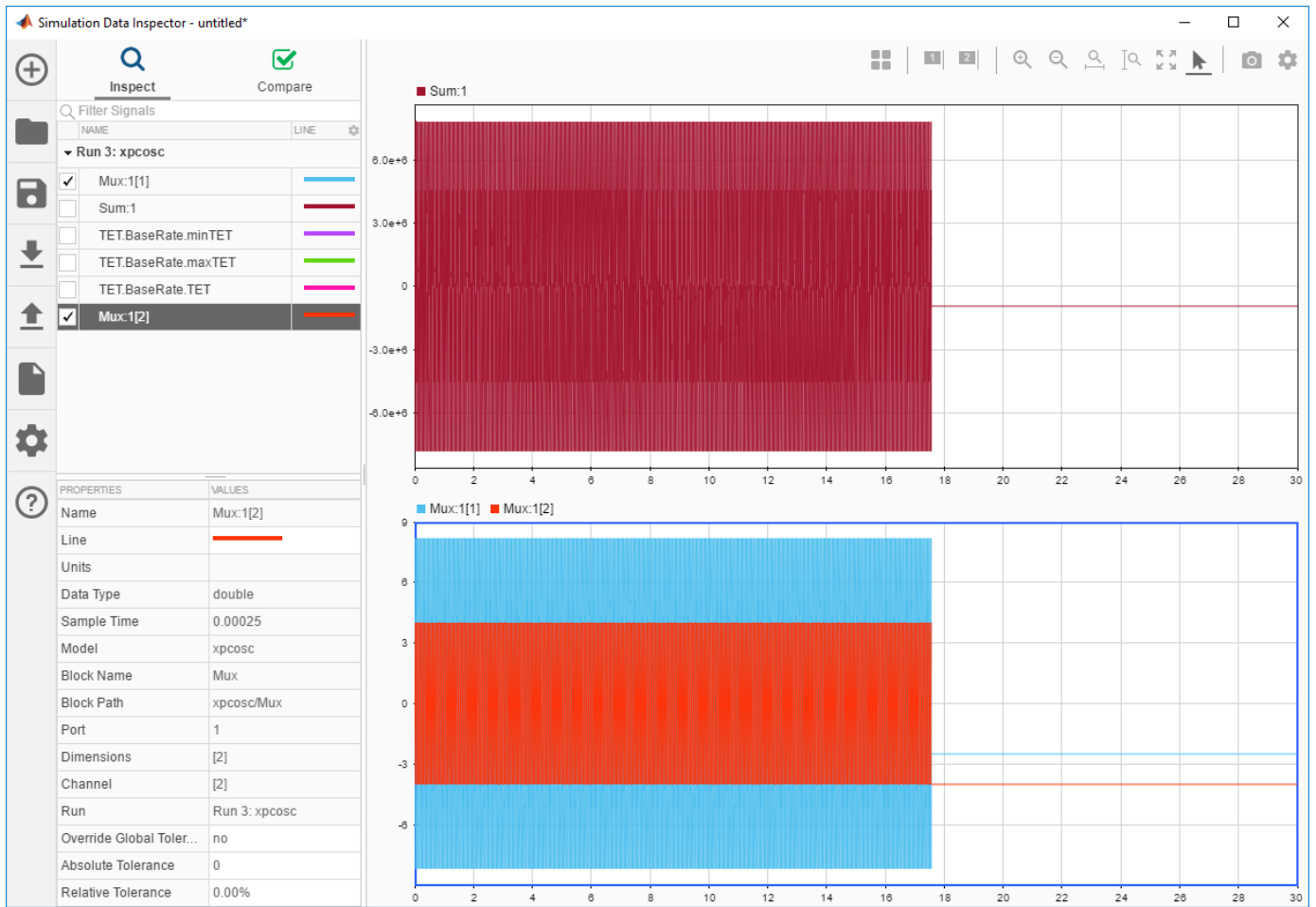
- 1 Build and download xpcosc.
- 2 Start real-time execution.
- 3 When the Simulation Data Inspector button glows , click the top display and select the Sum output signal.

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

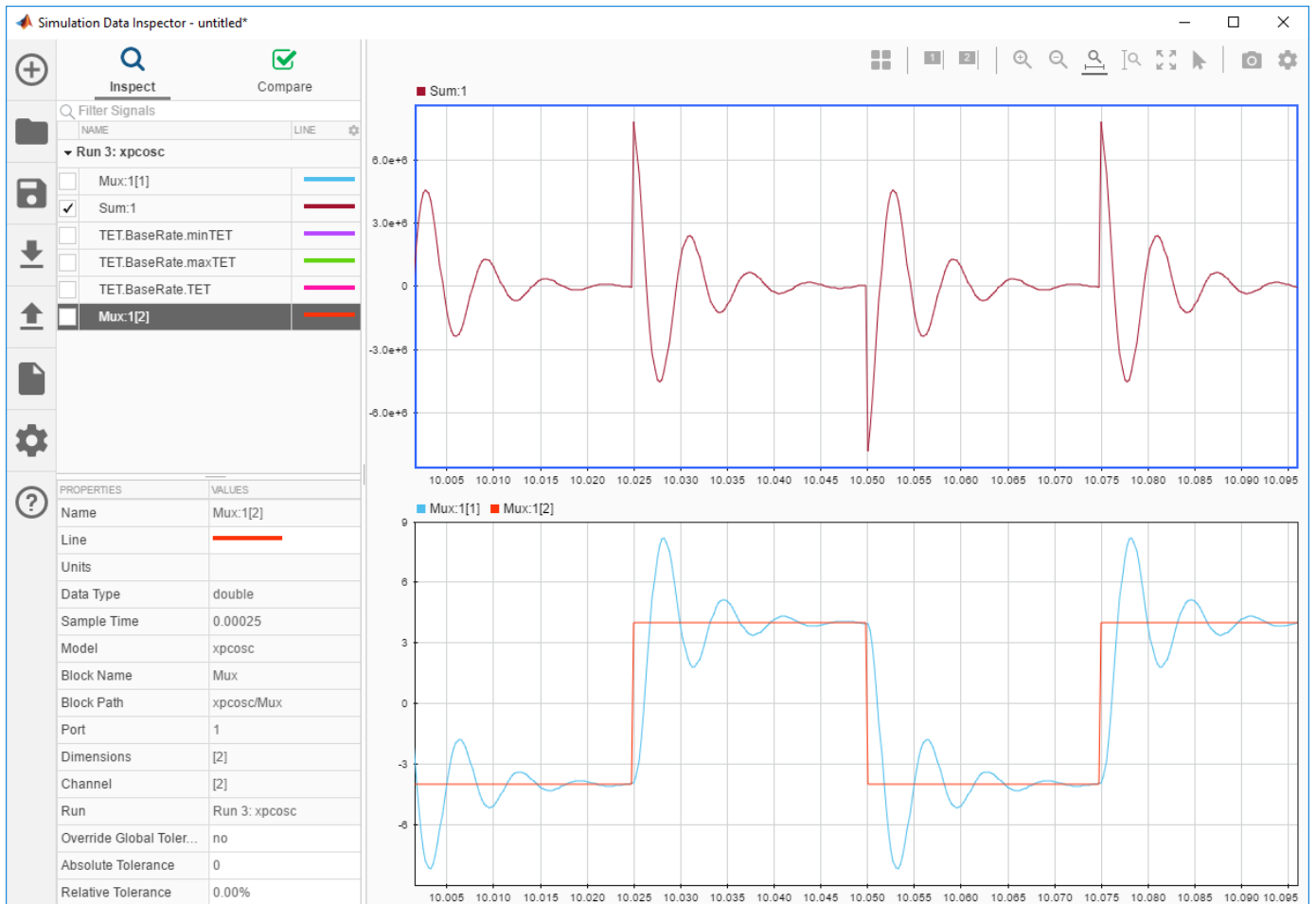


- 4 Stop real-time execution.

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 `.mldatx` file, click **Save**.

See Also

More About

- “Inspect Simulink® Real-Time™ Data with Simulation Data Inspector” on page 6-61
- Simulation Data Inspector

External Mode Usage

- When setting up signal triggering (Source set to signal), explicitly specify the element number of the signal in the **Trigger signal:Element** box. If the signal is a scalar, enter a value of 1. If the signal is a wide signal, enter a value from 1 to 10. When uploading Simulink Real-Time signals to Simulink scopes, do not enter Last or Any in this box.
- The **Direction:Holdoff** value does not affect the Simulink Real-Time signal uploading feature.

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 6-85
- “Log Signal Data with Outport Blocks and Simulink Real-Time Explorer” on page 6-92
- “Log Signal Data with Outport Block and MATLAB Language” on page 6-97
- “Troubleshoot Signals Not Accessible by Name” on page 6-155
- “Simulink Real-Time Scope Usage” on page 6-17
- “Target Scope Usage” on page 6-18
- “Host Scope Usage” on page 6-48
- “File Scope Usage” on page 6-74
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147

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 in the operating system on the target computer can have a maximum of 260 characters. If the file name is longer than eight-dot-three format (eight character file name, period, three character extension), the operating system represents the file name in truncated form (for example, six characters followed by '~1'). MATLAB commands can access the file using the fully qualified file name or the truncated representation of the name. Some block parameters, such as the Scope block `filename` parameter, require 8.3 format for the file name.
- 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

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

More About

- “Configure Real-Time File Scope Blocks” on page 6-76
- “Create File Scopes with Simulink Real-Time Explorer” on page 6-81
- “Log Signal Data into Multiple Files” on page 6-89
- “Simulink Real-Time Scope Usage” on page 6-17
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147
- “Using SimulinkRealTime.fileSystem Objects” on page 12-4

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 in the operating system on the target computer can have a maximum of 260 characters. If the file name is longer than eight-dot-three format (eight character file name, period, three character extension), the operating system represents the file name in truncated form (for example, six characters followed by '~1'). MATLAB commands can access the file using the fully qualified file name or the truncated representation of the name. Some block parameters, such as the Scope block filename parameter, require 8.3 format for the file name.

- 11** From the **Mode** list, select either Lazy or Commit.

With the Commit mode, each file write operation simultaneously updates the FAT entry for the file. The file system maintains the actual file size after each write. With the Lazy mode, the FAT entry is updated only when the file is closed.

If your system stops responding, you lose **WriteSize** bytes of data.

- 12** In the **WriteSize** box, enter the block size, in bytes, of the data chunks. This parameter specifies that a memory buffer of length Number of samples is written to the file in chunks of size **WriteSize**. By default, this parameter is 512 bytes. Using a block size that is the same as the disk sector size improves performance.

If your system stops responding, you lose **WriteSize** bytes of data.

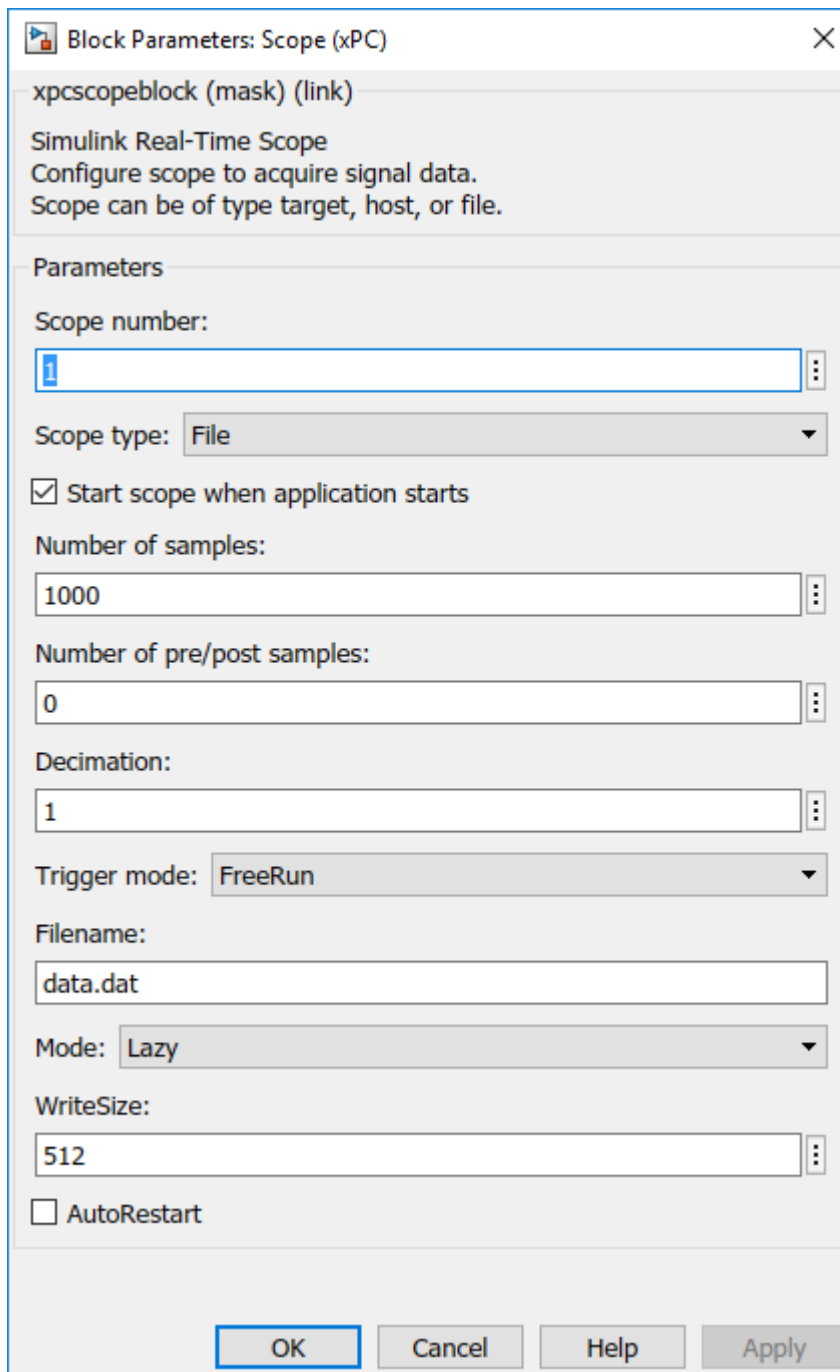
- 13** To have the file scope collect data up to **Number of samples** and then start over again reading new data, select the **AutoRestart** check box.

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.



- 14 Click **OK**.
- 15 Save the model as `ex_slrt_file_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_file_osc')))`). On the **Simulation** tab, from **Save**, click **Save As**.

See Also

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

More About

- “Simulink Real-Time Scope Usage” on page 6-17
- “File Scope Usage” on page 6-74
- “Trigger One Scope with Another Scope” on page 11-15

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 6-76 and “File Scope Usage” on page 6-74.


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

- 1 Open model `xpcosc`. Set property **Stop time** to `inf`. On the **Real-Time** tab, select **Run on Target > Stop Time** and set **Stop Time** to `inf`.
- 2 Connect to the target computer. On the **Real-Time** tab, toggle the **Disconnected** indicator to **Connected**.
- 3 Build and download the real-time application to the target computer. On the **Real-Time** tab, click **Run on Target**.
- 4 Open Simulink Real-Time Explorer. On the **Real-Time** tab, click **Prepare > SLRT Explorer**.


To configure a file scope:

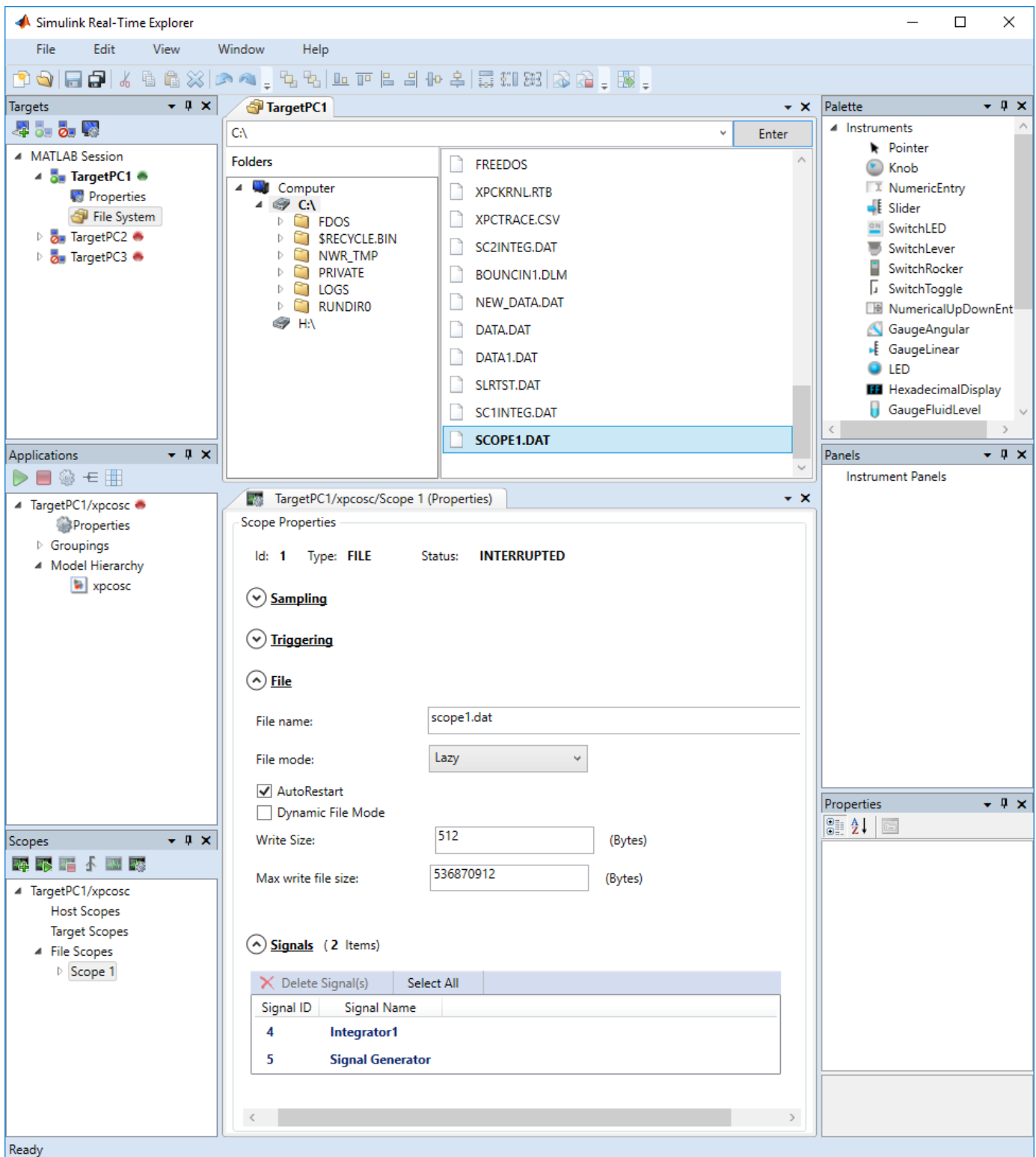
- 1 In the **Scopes** pane, expand the `xpcosc` node.
 - 2 To add a file scope, select **File Scopes**, and then click the **Add Scope** button  on the toolbar.
 - 3 Expand **Scope 1**, and then click the **Properties** button  on the toolbar.
 - 4 In the **Scope Properties** pane, click **Signals**.
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.




- 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 6-89
- “Configure File Scopes with Simulink Real-Time Explorer” on page 6-85
- “Using `SimulinkRealTime.fileSystem` Objects” on page 12-4
- “Create Signal Groups with Simulink Real-Time Explorer” on page 6-45
- “Configure Scope Sampling with Simulink Real-Time Explorer” on page 6-29
- “Trigger Scopes with Simulink Real-Time Explorer” on page 6-32
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147

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 6-81. 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 in the operating system on the target computer can have a maximum of 260 characters. If the file name is longer than eight-dot-three format (eight character file name, period, three character extension), the operating system represents the file name in truncated form (for example, six characters followed by '~1'). MATLAB commands can access the file using the fully qualified file name or the truncated representation of the name. Some block parameters, such as the Scope block `filename` parameter, require 8.3 format for the file name.

If a file with this name exists when you start the file scope, the file scope overwrites the old data with the new data.

- 4 Select **File mode** `Commit`.


The default **File mode** is `Lazy`. When real-time execution stops without an error, both the `Lazy` and `Commit` settings of the **Mode** box have the same result. Both settings cause the model to open a file, write signal data to the file, and close that file at the end of the session. The differences are in when the software updates the FAT entry for the file.

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



`Lazy` mode is faster than `Commit` mode. However, if the target computer enters an error state, the system can stop responding before the file is closed. In `Lazy` mode, the actual file size can be lost, even though the file was written. You can lose an amount of data equivalent to the setting of the **WriteSize** parameter.

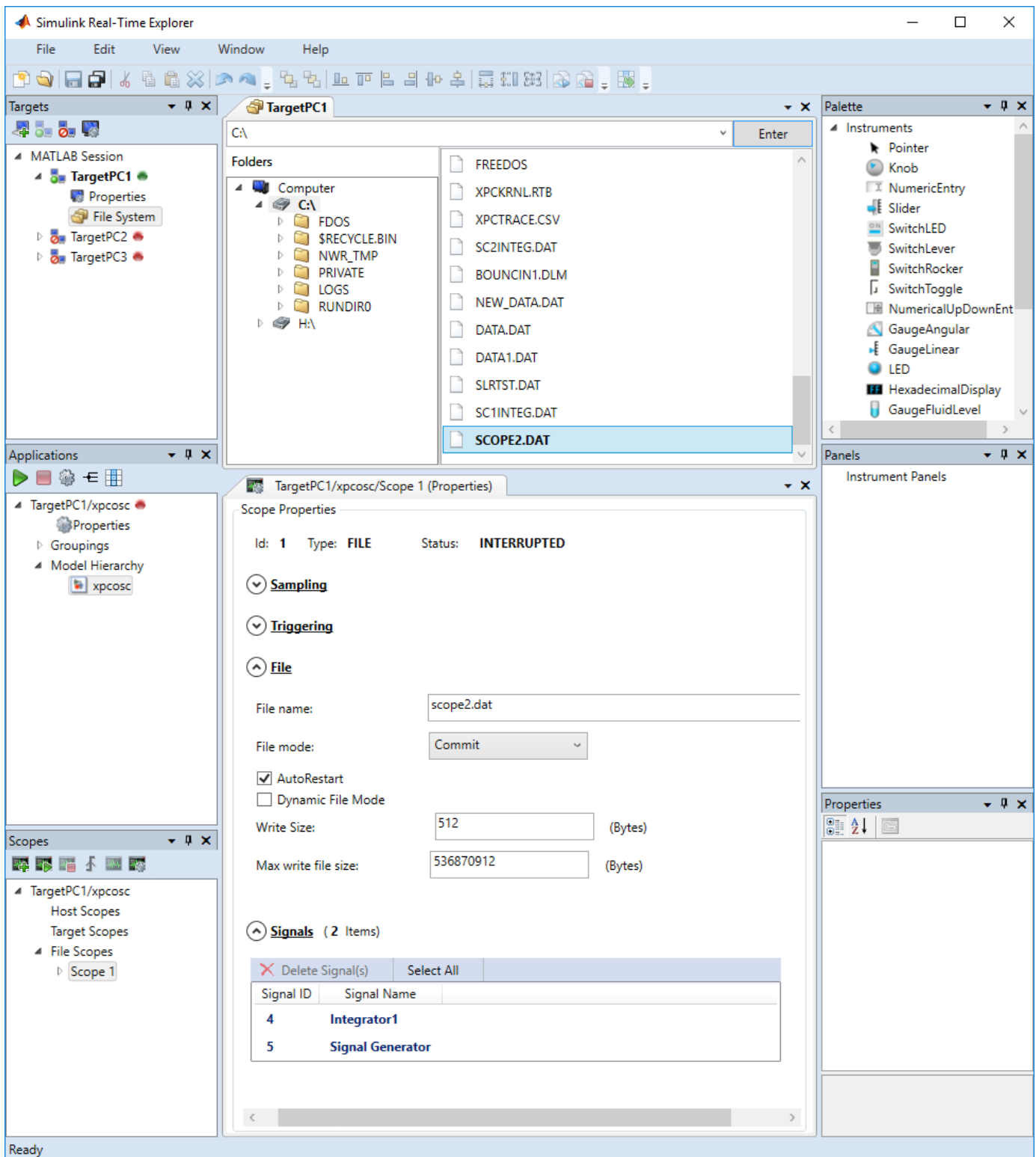
- 5 To have the file scope collect data up to **Number of samples** and then start over again reading new data, select the **AutoRestart** check box.
- 6 Leave the **Dynamic File Mode** check box cleared.
- 7 Leave **Write Size** set to the default value of 512.

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).



- 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.utils.bytes2file` | `SimulinkRealTime.utils.getFileScopeData` | `mkdir`


More About

- “Log Signal Data into Multiple Files” on page 6-89
- “Using `SimulinkRealTime.fileSystem` Objects” on page 12-4
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147

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 6-81.


- 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.

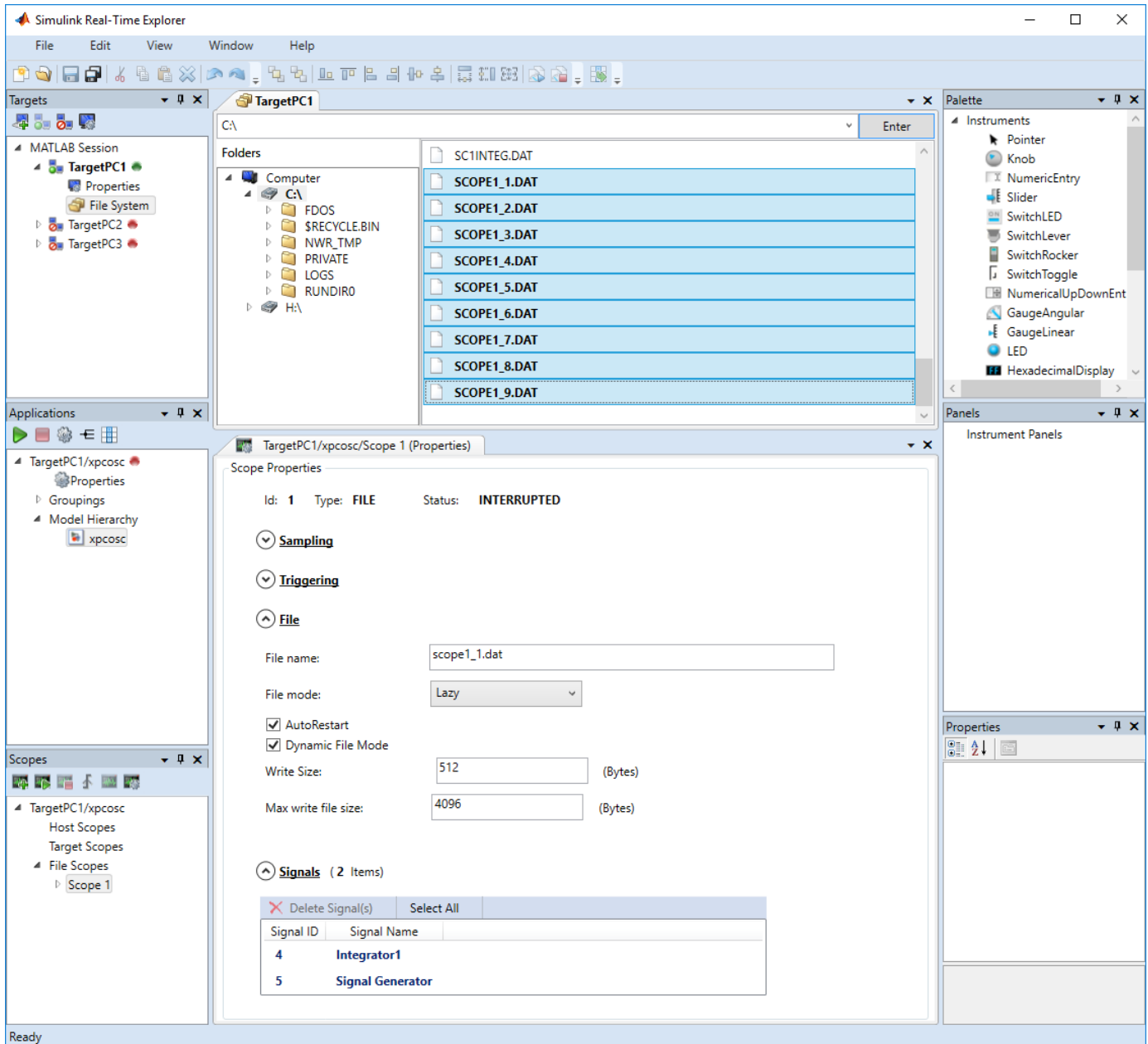
- 8 In the **Max write file size** box, enter a value to limit the size of the signal log files. This value must be a multiple of the **Write Size** value. For example, if the write size is 512, enter 4096 to limit each log file size to 4096 bytes.
- 9 To start execution, click the real-time application 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

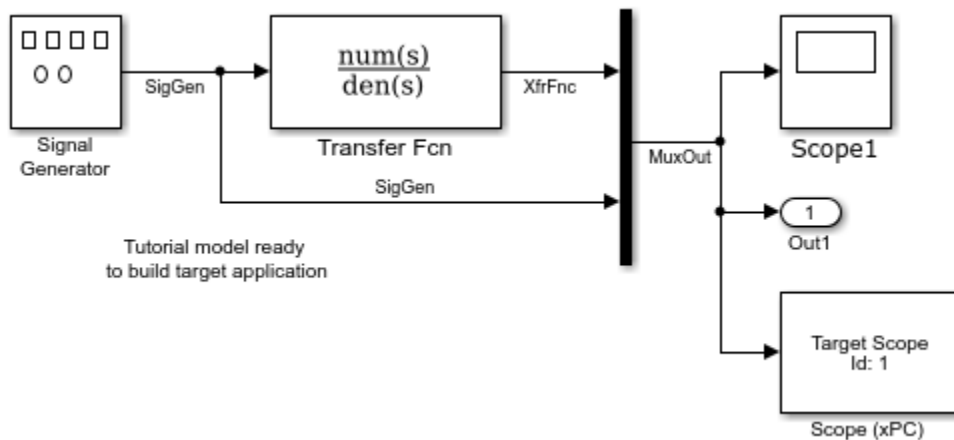
- “File Scope Usage” on page 6-74
- “Using `SimulinkRealTime.fileSystem` Objects” on page 12-4
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147

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 `SimulinkRealTime.target` 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.

Data Logs

Simulink Real-Time stores logged data in four data logs that you can access on the development computer by using `SimulinkRealTime.target` 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 `getlog`.

Note

- The data logging variables `tout`, `xout`, `yout`, and `logout` are available only when you use Simulink to simulate the model in non-real-time.
 - You cannot use the 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 Configuration Parameters. On the **Real-Time** tab, click **Hardware Settings**.
- 2 To allow Simulink to log signals. Select **Data Import/Export > Time** and select **Data Import/Export > Output**. These check boxes are selected by default.
- 3 To plot the task execution time, select **Code Generation > Simulink Real-Time Options > Monitor Task Execution Time**. This check box is selected by default.
- 4 To create a buffer for the signals that you are logging, set **Code Generation > Simulink Real-Time Options > Signal logging buffer size in doubles** to the required value.

The default value of 100000 units is large enough for this model.

- 5 Save the model as `ex_slrt_outport_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_outport_osc')))`). On the **Simulation** tab, from **Save**, click **Save as**.

Log the Data

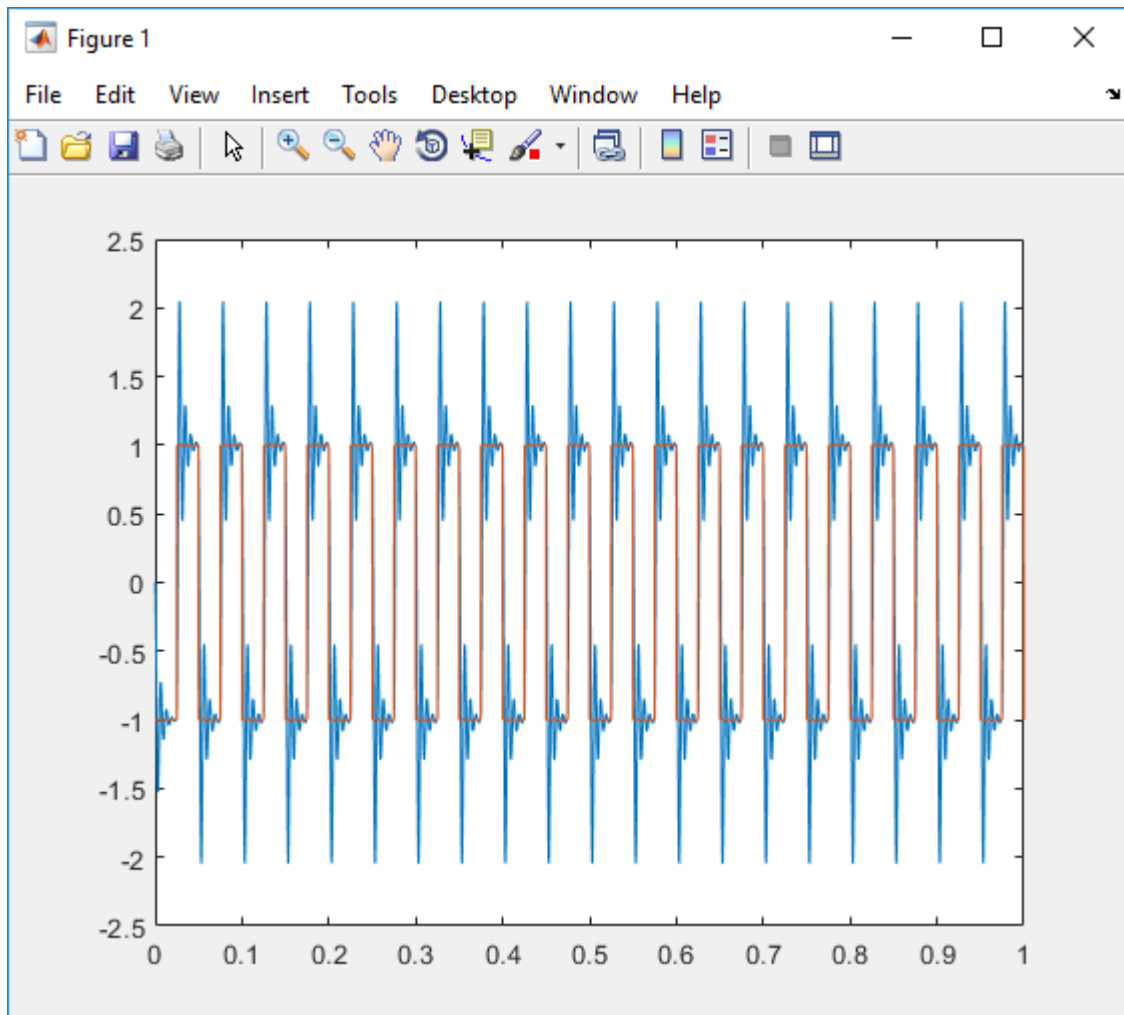
- 1 Connect to the target computer. On the **Real-Time** tab, toggle the **Disconnected** indicator to **Connected**.
- 2 Build and download the real-time application to the target computer. On the **Real-Time** tab, click **Run on Target**.
- 3 Open Simulink Real-Time Explorer. On the **Real-Time** tab, click **Prepare > SLRT Explorer**.
- 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:

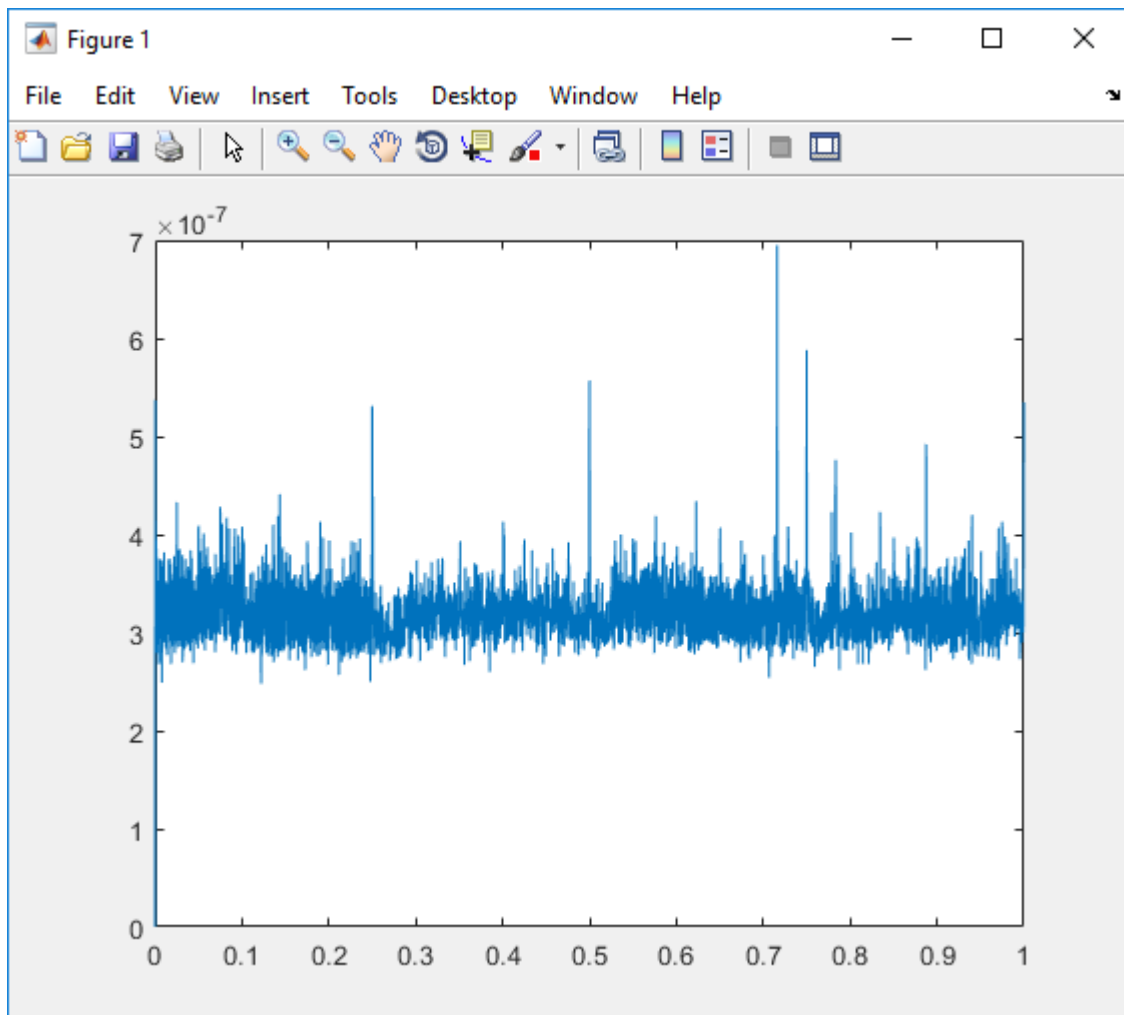
```
tg = slrt;
timelog = tg.TimeLog;
```

```
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 | `getlog`

More About

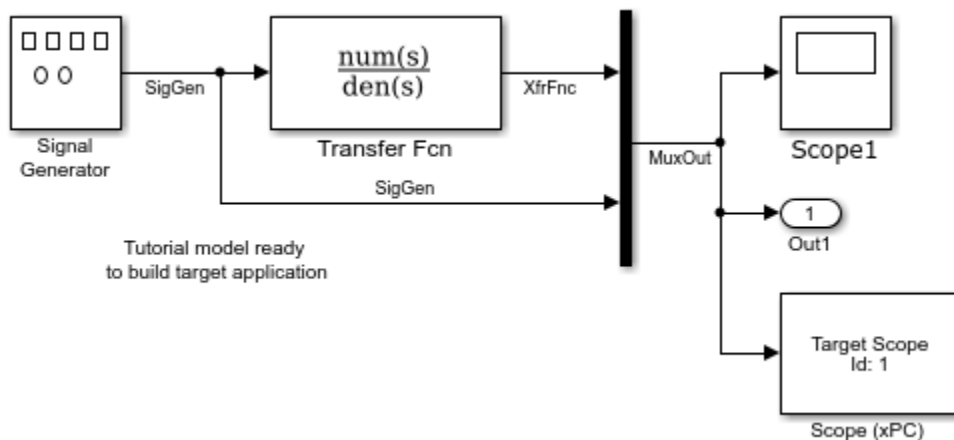
- “Simulate Simulink Model by Using MATLAB Language”
- “Log Signal Data with Outport Block and MATLAB Language” on page 6-97
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147
- “Signal Logging Buffer Size” on page 6-103

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 `SimulinkRealTime.target` 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.

Data Logs with `SimulinkRealTime.target` Properties

Simulink Real-Time stores logged data in four data logs that you can access on the development computer by using `SimulinkRealTime.target` 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.

Note The `tg.TimeLog`, `tg.OutputLog`, `tg.TeTLog`, and `tg.StateLog` properties will be removed in a future release.

Turn on logging for only the data that you are interested in.

Each Output block has an associated column vector in `tg.OutputLog`. You can access the data that corresponds to a particular Output block by specifying the column vector for that block. For example, to access the data that corresponds to Output 2, use `tg.outputlog(:,2)`.

To download part of the logs, use the target object method `getlog`.

Note

- The data logging variables `tout`, `xout`, `yout`, and `logout` are available only when you use Simulink to simulate the model in non-real-time.
 - You cannot use the Simulation Data Inspector to create a data log on the target computer. You can log only signals that are connected to an Output block.
-

Data Logs with the Simulation Data Inspector and Data Profiler

The `tg.TimeLog`, `tg.OutputLog`, `tg.TeTLog`, and `tg.StateLog` properties will be removed in a future release. Consider these replacement approaches.

| If using . . . | Replace with . . . |
|---|---|
| <code>tg.TimeLog</code> , <code>tg.OutputLog</code> , or <code>tg.StateLog</code> | Access the <code>Simulink.sdi.Run</code> object for a run created by logging signals of interest to the Simulation Data Inspector. From the <code>Simulink.sdi.Run</code> object you can get <code>Simulink.sdi.Signal</code> objects that you can use to view data. For example: <pre>run = Simulink.sdi.getRun(runID); % Get signal objects for the signals in the run signal1 = fuelRun.getSignalByIndex(4); signal2 = fuelRun.getSignalByIndex(9);</pre> |
| <code>tg.TETLog</code> | The Code Execution Profiling Report. Setup the profiler and then use: <pre>profiler_object = getProfilerData(tg);</pre> |

For an example, see “Data Logging With Simulation Data Inspector (SDI)” on page 15-190.

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);
tg = slrt('TargetPC1');
load(tg, mdl);

```

- 2 Set the stop time and start execution.

```

tg.stoptime = 1;
start(tg);

```

- 3 Stop execution.

```

stop(tg);

```

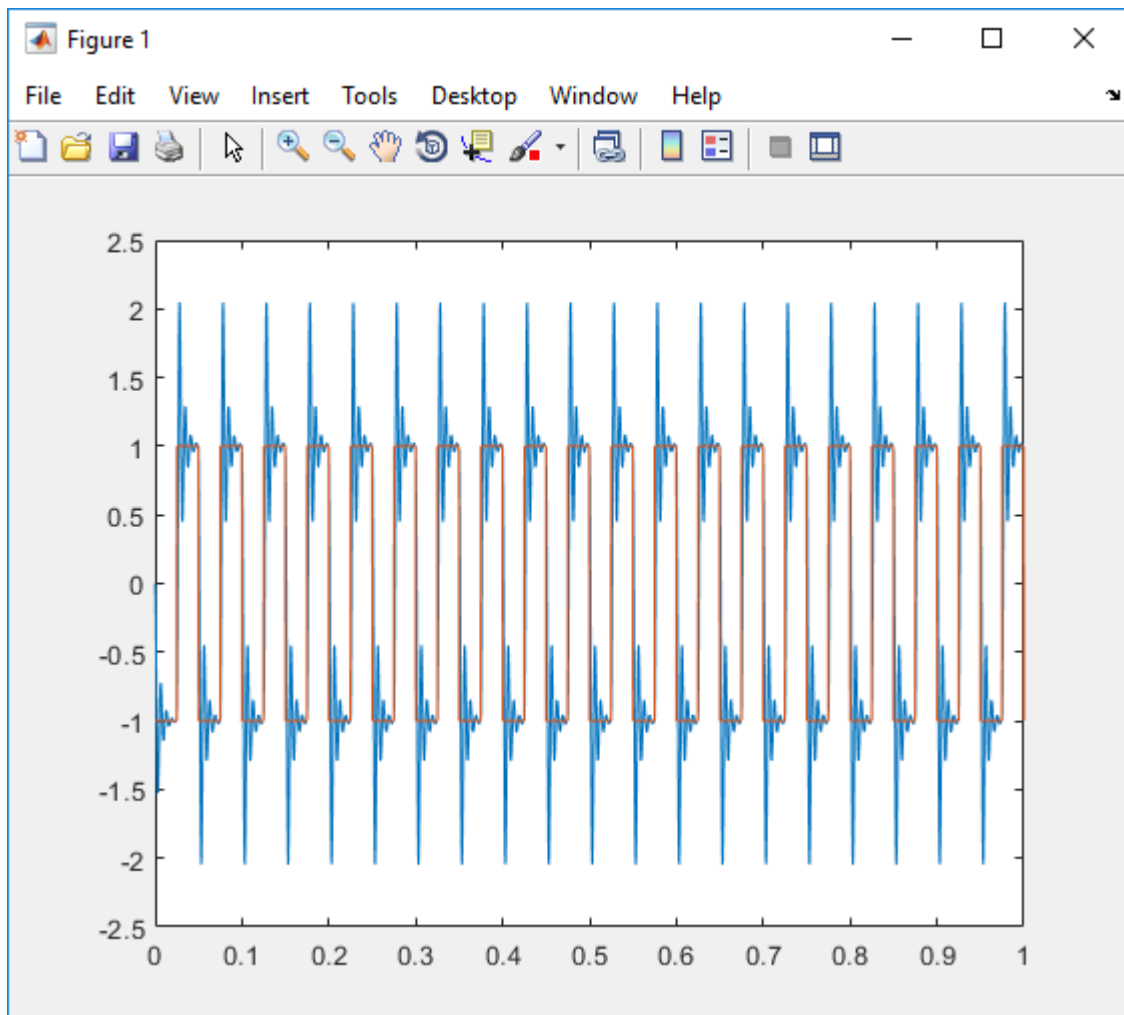
Download and Plot the Data

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

```

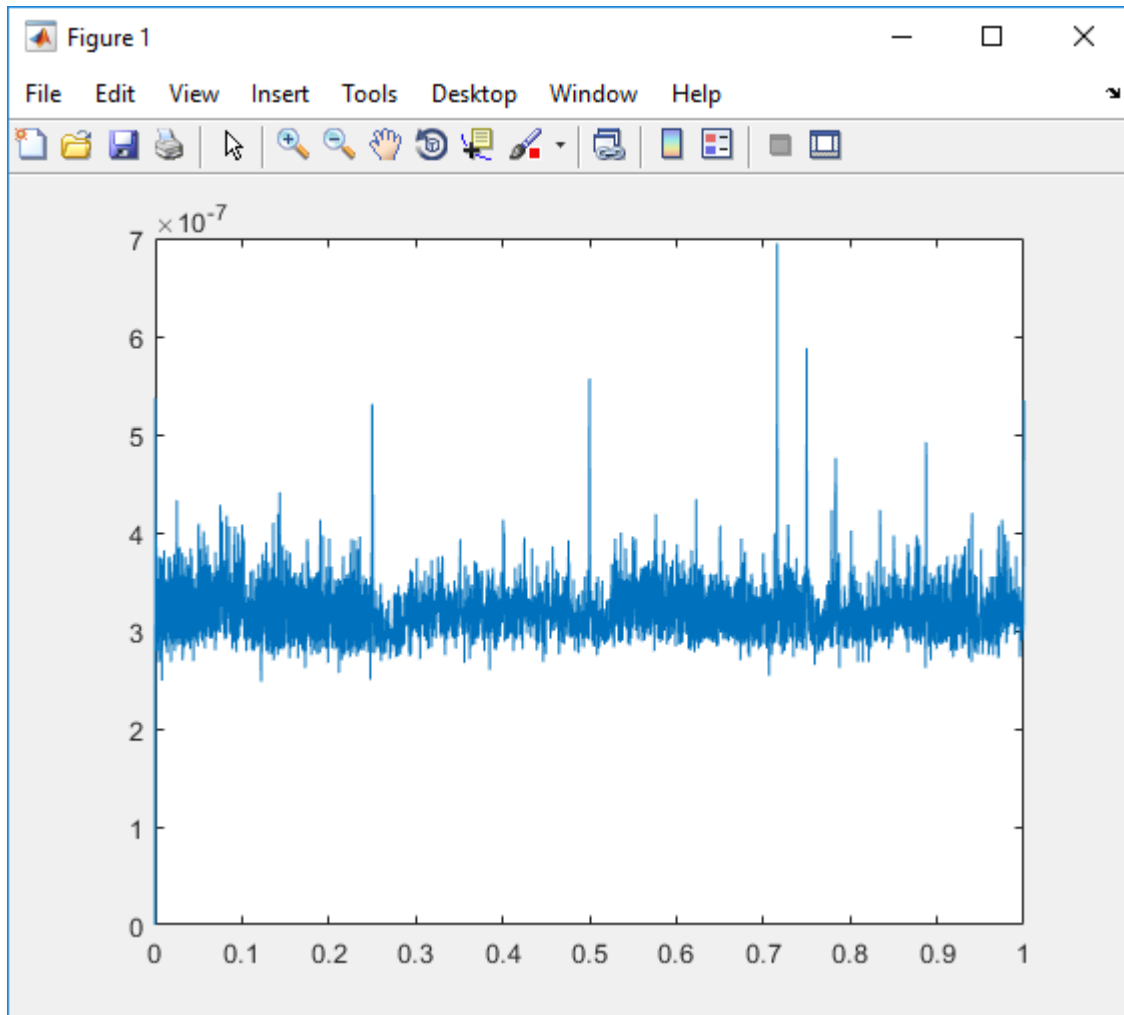
tg = slrt;
timelog = tg.TimeLog;
outputlog = tg.OutputLog;
plot(timelog, outputlog)

```



- 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 | `getlog`

More About

- “Simulate Simulink Model by Using MATLAB Language”
- “Log Signal Data with Outport Blocks and Simulink Real-Time Explorer” on page 6-92
- “Signal Logging Buffer Size” on page 6-103

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:

$$N = \text{Buffer size in doubles} / \text{Logged signals}$$

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 12-4
- “Monitor Signals with MATLAB Language” on page 6-7


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 Open model `xpcosc`. Set property **Stop time** to `inf`. On the **Real-Time** tab, select **Run on Target > Stop Time** and set **Stop Time** to `inf`.
- 2 Connect to the target computer. On the **Real-Time** tab, toggle the **Disconnected** indicator to **Connected**.
- 3 Build and download the real-time application to the target computer. On the **Real-Time** tab, click **Run on Target**.
- 4 Open Simulink Real-Time Explorer. On the **Real-Time** tab, click **Prepare > SLRT Explorer**.
- 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.

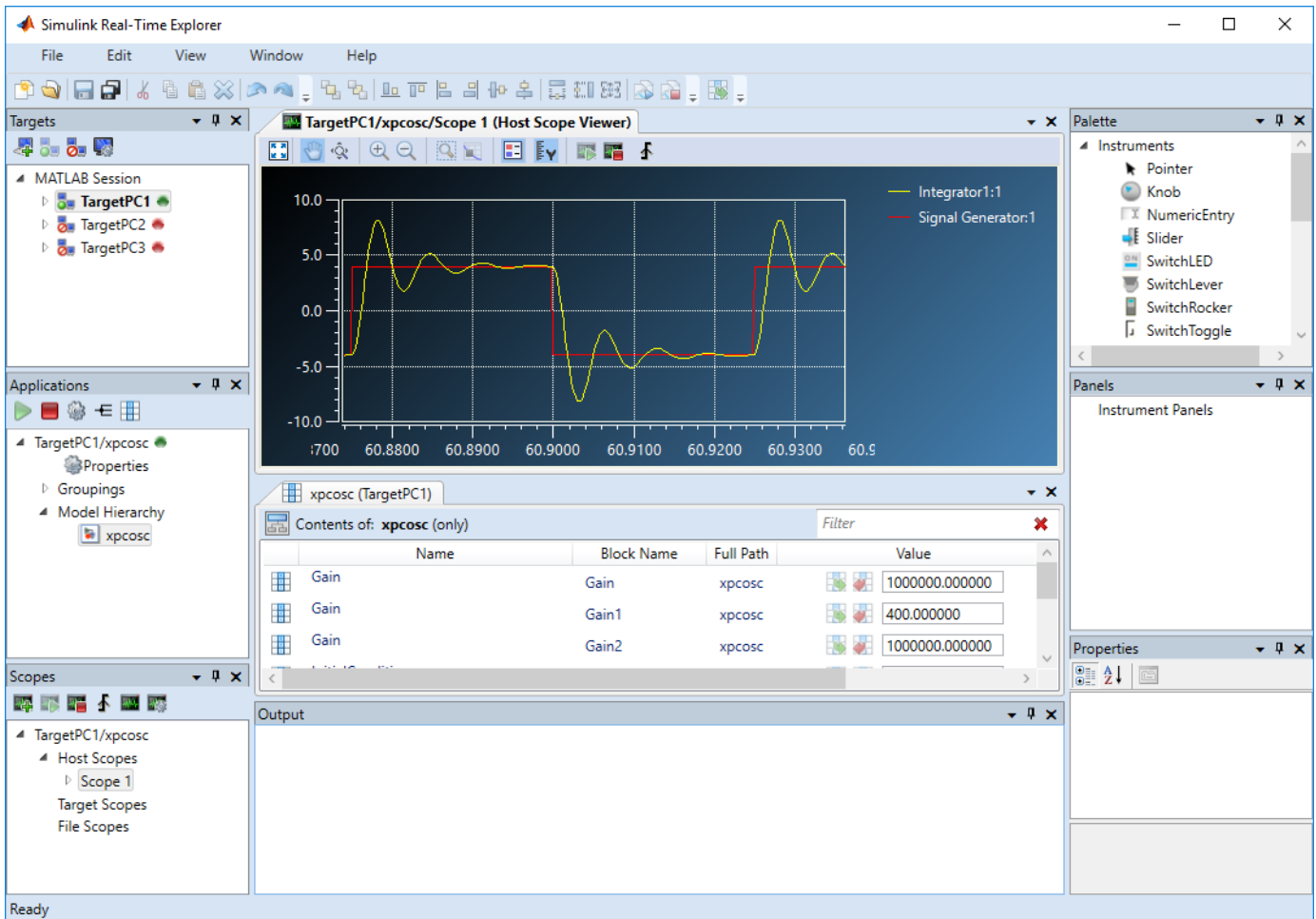
Add signal `Integrator1` to **Scope1** in the same way.

- 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 6-52.


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.

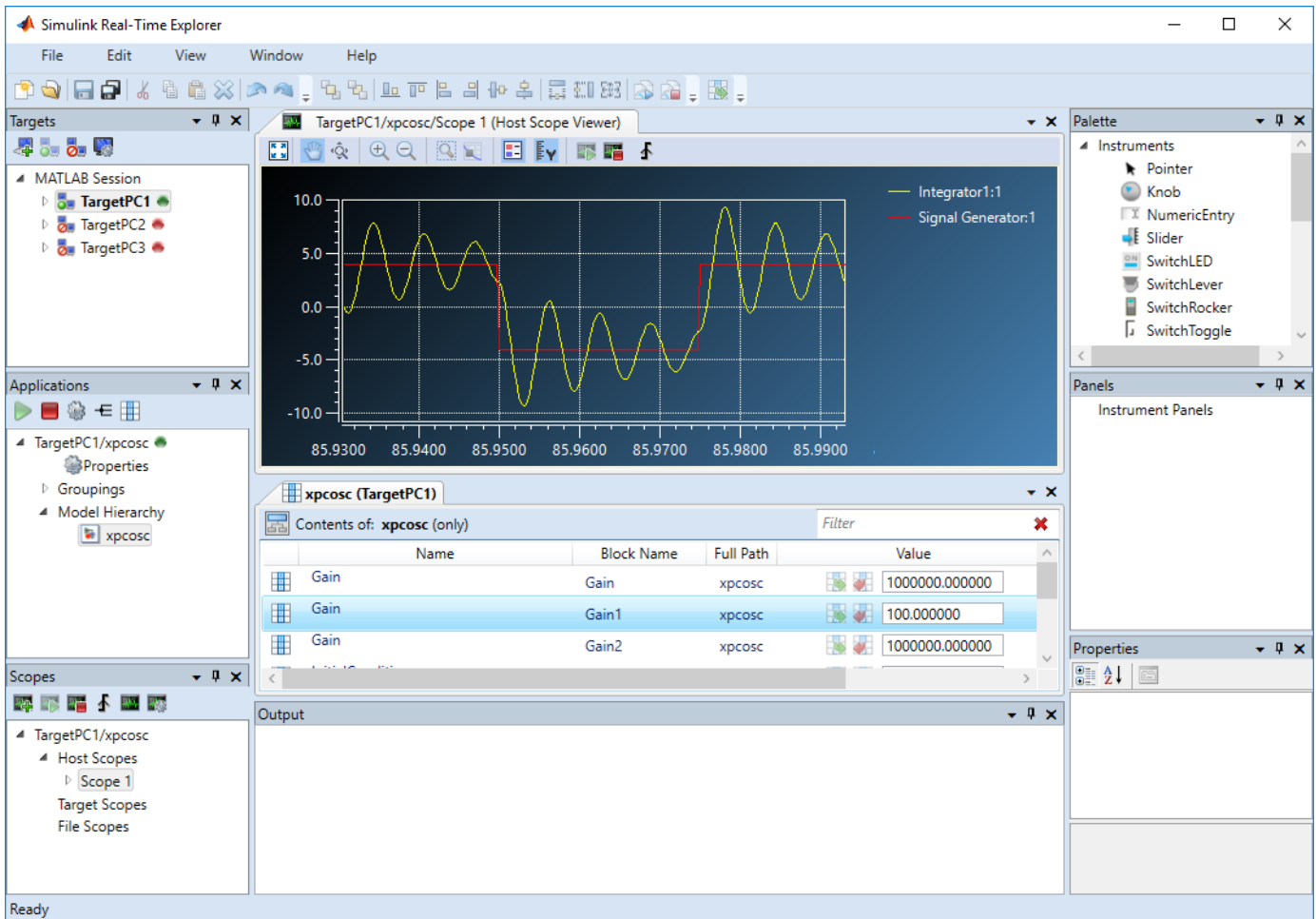
The Parameters workspace opens, showing a table of parameters with properties and actions.



- 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 6-52
- “Configure Real-Time Host Scope Blocks” on page 6-49
- “Create Parameter Groups with Simulink Real-Time Explorer” on page 6-111

- “Display and Filter Hierarchical Signals and Parameters” on page 6-147
- “Troubleshoot Parameters Not Accessible by Name” on page 6-156
- “Troubleshoot Instrument Label Not Present” on page 6-158


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:

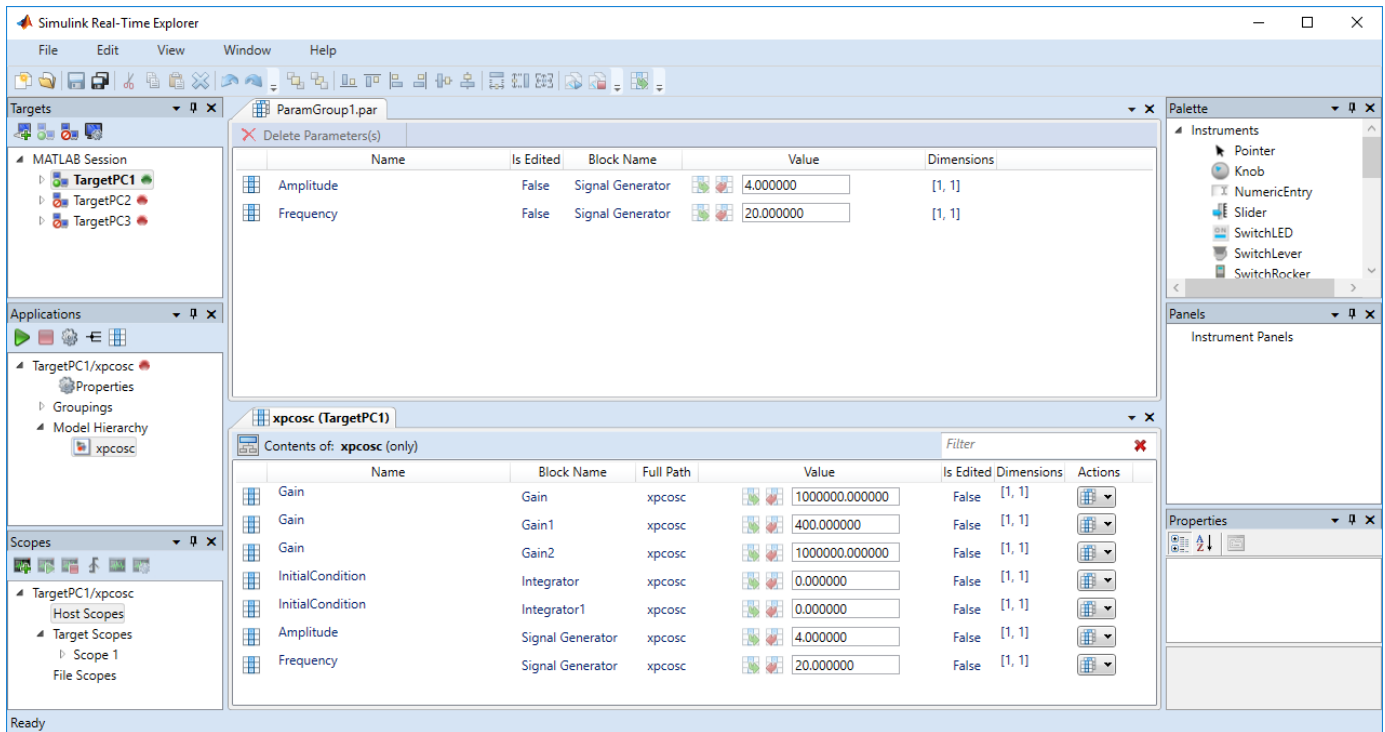
- 1 Open model `xpcosc`.
- 2 Connect to the target computer. On the **Real-Time** tab, toggle the **Disconnected** indicator to **Connected**.
- 3 Build and download the real-time application to the target computer. On the **Real-Time** tab, click **Run on Target**.
- 4 Open Simulink Real-Time Explorer. On the **Real-Time** tab, click **Prepare > SLRT Explorer**.

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 6-107
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147

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 6-156
- “Troubleshoot Instrument Label Not Present” on page 6-158

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 Open model `xpcosc`.
- 2 Connect to the target computer. On the **Real-Time** tab, toggle the **Disconnected** indicator to **Connected**.
- 3 Build and download the real-time application to the target computer. On the **Real-Time** tab, click **Run on Target**.

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)
```

- 4 From the Simulation block diagram, double-click the block labeled **Gain1**
- 5 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.

- 6 To stop the simulation, on the **Real-Time** tab click **Stop**.
- 7 Disconnect to the target computer. On the **Real-Time** tab, toggle the **Connected** indicator to **Disconnected**.

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.

Tuning with Batch Mode and Update All Parameters

By using Batch Mode, you can tune multiple parameters and apply the tuning changes at once, instead of tuning one parameter at a time. This example uses model `xpcosc` (matlab: `open_system(docpath(fullfile(docroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcosc')))`)).

- 1 Open model `xpcosc`.
- 2 In the Simulink Editor, on the **Real-Time** tab, click **Run on Target**.

- 3 On the **Real-Time** tab, click **Prepare > Batch Mode**. The editor remains in batch mode until you click **Batch Mode** again.

To set parameter values, you can either set values by clicking on each block or by using the Model Data Editor.

- 4 On the **Real-Time** tab, click **Prepare > Signal Table**.
- 5 In the Model Data Editor, click the **Parameters** tab. Modify parameters values in the Model Data Editor.
- 6 On the **Real-Time** tab, **Prepare > Update All Parameters**.
- 7 To stop the simulation before it ends, on the **Real-Time** tab, click **Stop**.

See Also

More About

- “Troubleshoot Parameters Not Accessible by Name” on page 6-156
- “Troubleshoot Instrument Label Not Present” on page 6-158

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.

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 6-117).

- 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 6-117).

- 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 6-107
- “Tune Parameters with MATLAB Language” on page 6-113
- “Tune Parameters with Simulink External Mode” on page 6-115

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 Generation > Optimization** 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 either:

- Press **Ctrl+D**.
- On the **Real-Time** tab, click **Prepare > Signal Table**. On the **Parameters** tab, edit the parameters and click **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

`getparam` | `setparam`

More About

- [“Tune Inlined Parameters with Simulink Real-Time Explorer”](#) on page 6-122
- [“Default parameter behavior”](#) (Simulink Coder)
- [“Specify Source for Data in Model Workspace”](#) (Simulink)
- [“Troubleshoot Parameters Not Accessible by Name”](#) on page 6-156
- [“Troubleshoot Instrument Label Not Present”](#) on page 6-158
- [“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)
- [“Preserve Variables in Generated Code”](#) (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.

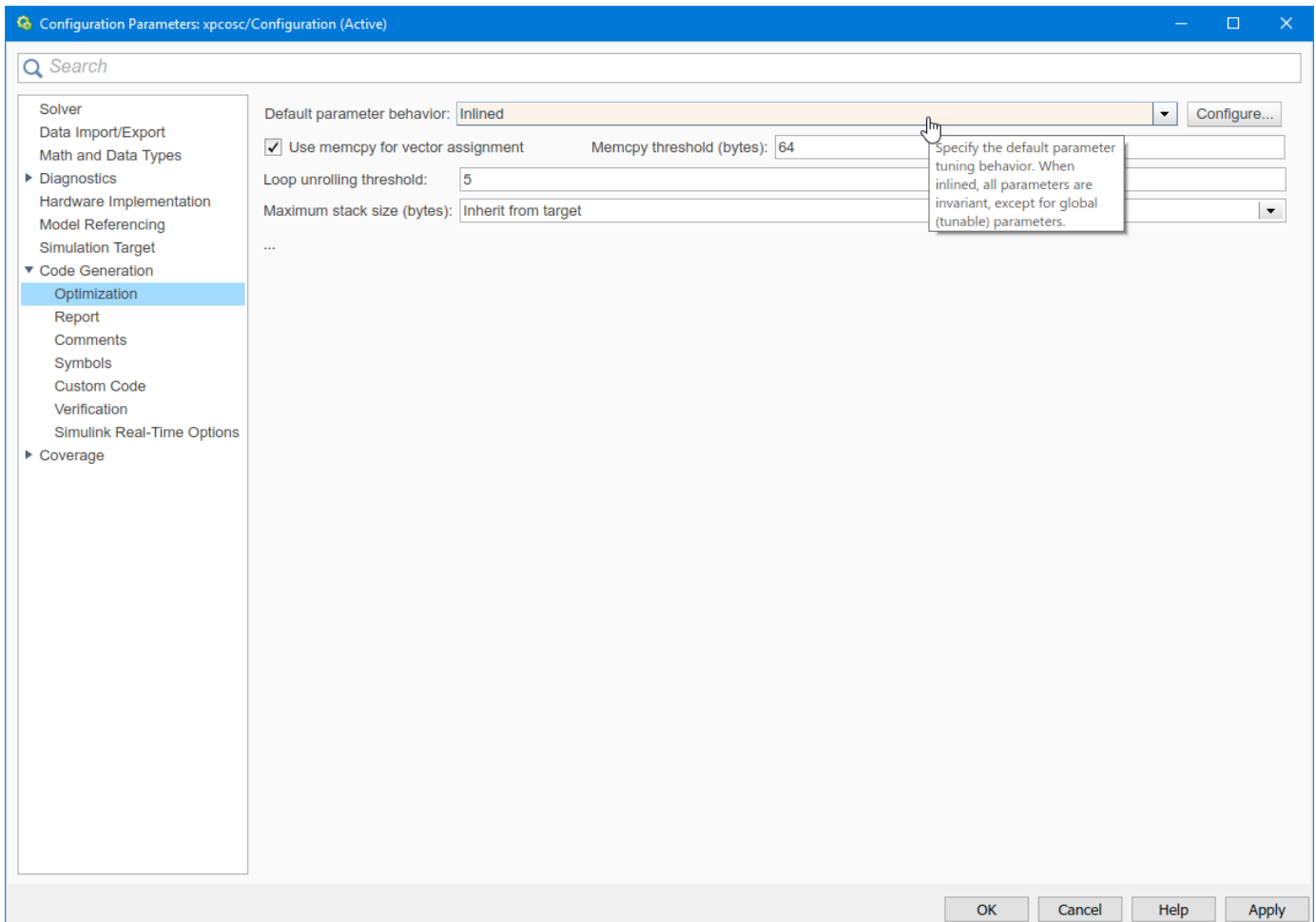
Note Simulink Real-Time does not support parameters of multiword data types.

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 Open model `xpcosc`.
- 2 In the Simulink Editor, select the input to the Scope block and mark it for data logging with the Simulation Data Inspector.
- 3 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`.
 - a Double-click the Signal Generator block, and then enter `A` for the **Amplitude** parameter. Click **OK**.
 - b Assign a constant to variable `A`. In the Command Window, type:
$$A = 4$$

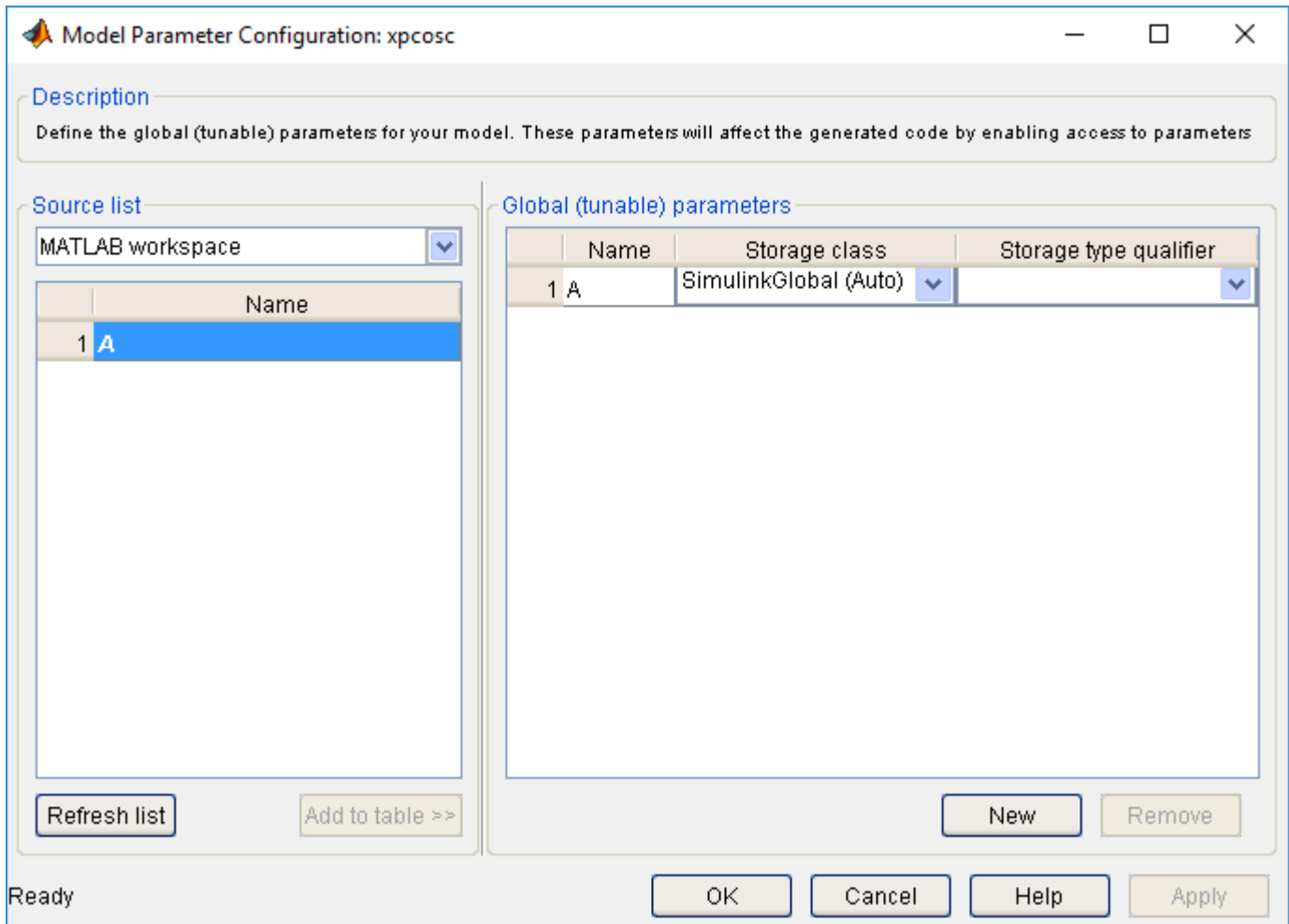
The value is displayed in the MATLAB workspace.
- 4 Open Configuration Parameters. On the **Real-Time** tab, click **Hardware Settings**.
- 5 Select **Code Generation > Optimization > Default parameter behavior > Inlined**.



6 Click **Configure**.

The Model Parameter Configuration dialog box opens. The MATLAB workspace contains the constant you assigned to **A**.

7 Select the line that contains your constant. Click **Add to table**.





Add the remaining global parameters that you want to tune.

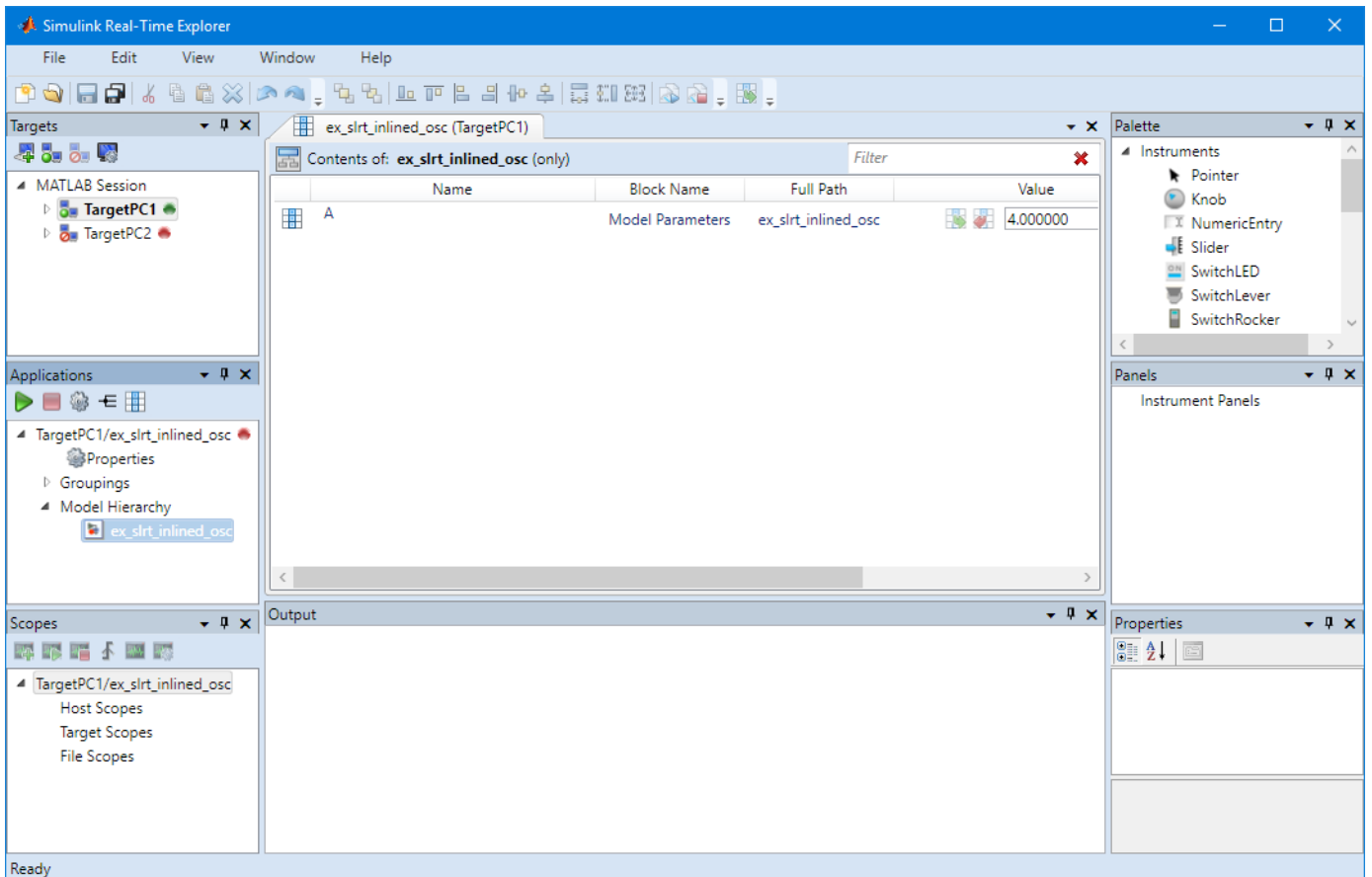
- 8 Click **Apply**, and then click **OK**.
- 9 In the Configuration Parameters dialog box, click **Apply**, and then **OK**.
- 10 Save the model as `ex_slrt_inlined_osc`. On the **Simulation** tab, from **Save**, click **Save As**. For example, save it as `ex_slrt_inlined_osc`. For example model, see `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_inlined_osc')))`.
- 11 Build and download the model to your target computer. On the **Real-Time** tab, click **Run on Target**.

Initial Value

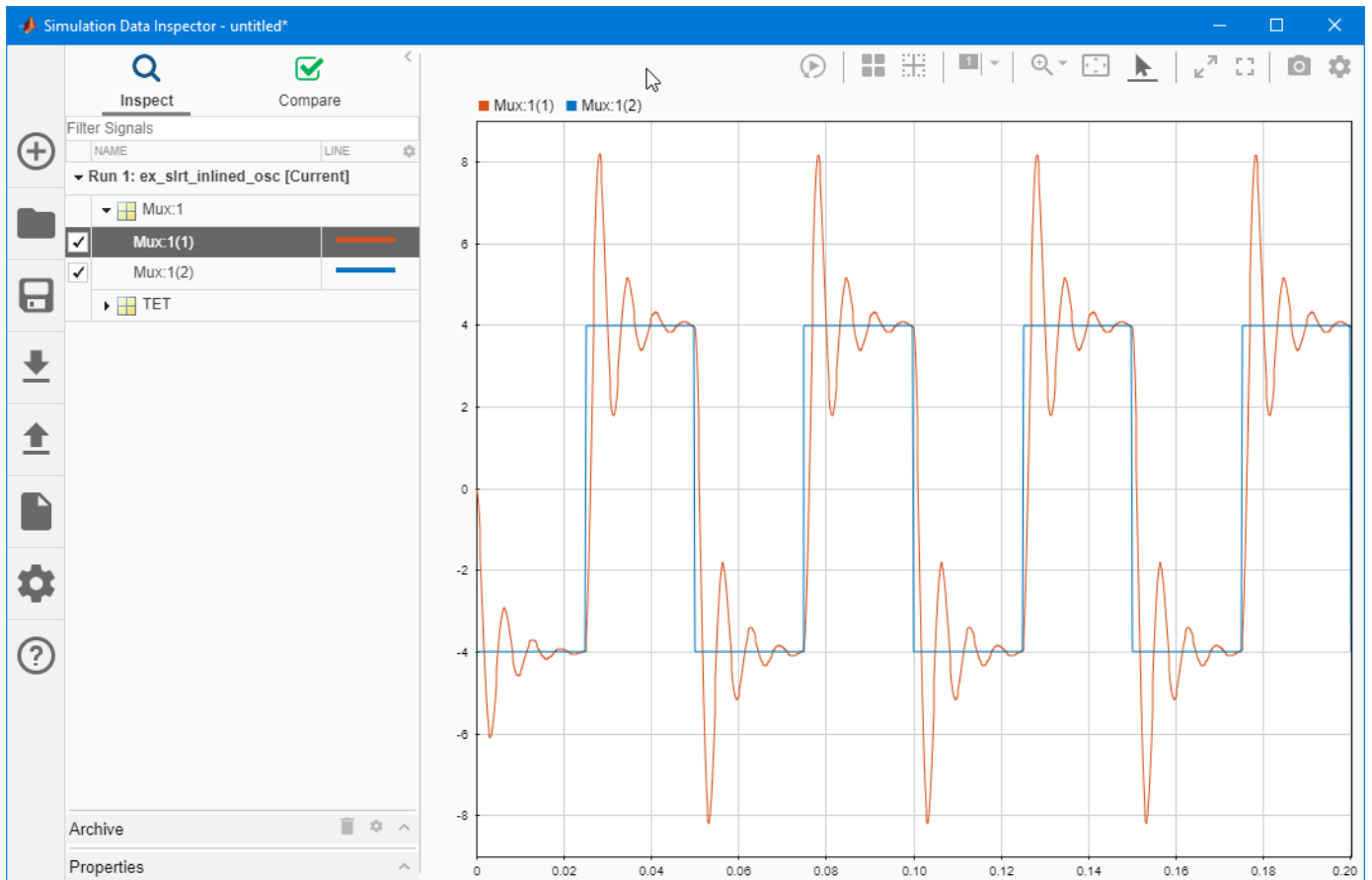
This procedure assumes that you have completed the steps in “Configure Model to Tune Inlined Parameters” on page 6-122.

- 1 Open Simulink Real-Time Explorer. On the **Real-Time** tab, click **Prepare > SLRT Explorer**.
- 2 Select the real-time application in the **Applications** pane (for example, `ex_slrt_inlined_osc`).

- 3 To start execution, click the real-time application, and then click the **Start** button  on the toolbar.
- 4 In the **Applications** pane, expand both the real-time application node and the **Model Hierarchy** node.
- 5 Select the model node, and then click the **View Parameters** button  on the toolbar. The Parameters workspace opens, showing a table of parameters with properties and actions.




- 6 Open the Simulation Data Inspector and view the signals you marked for signal logging. On the **Real-Time** tab, click **Data Inspector**.





Updated Value

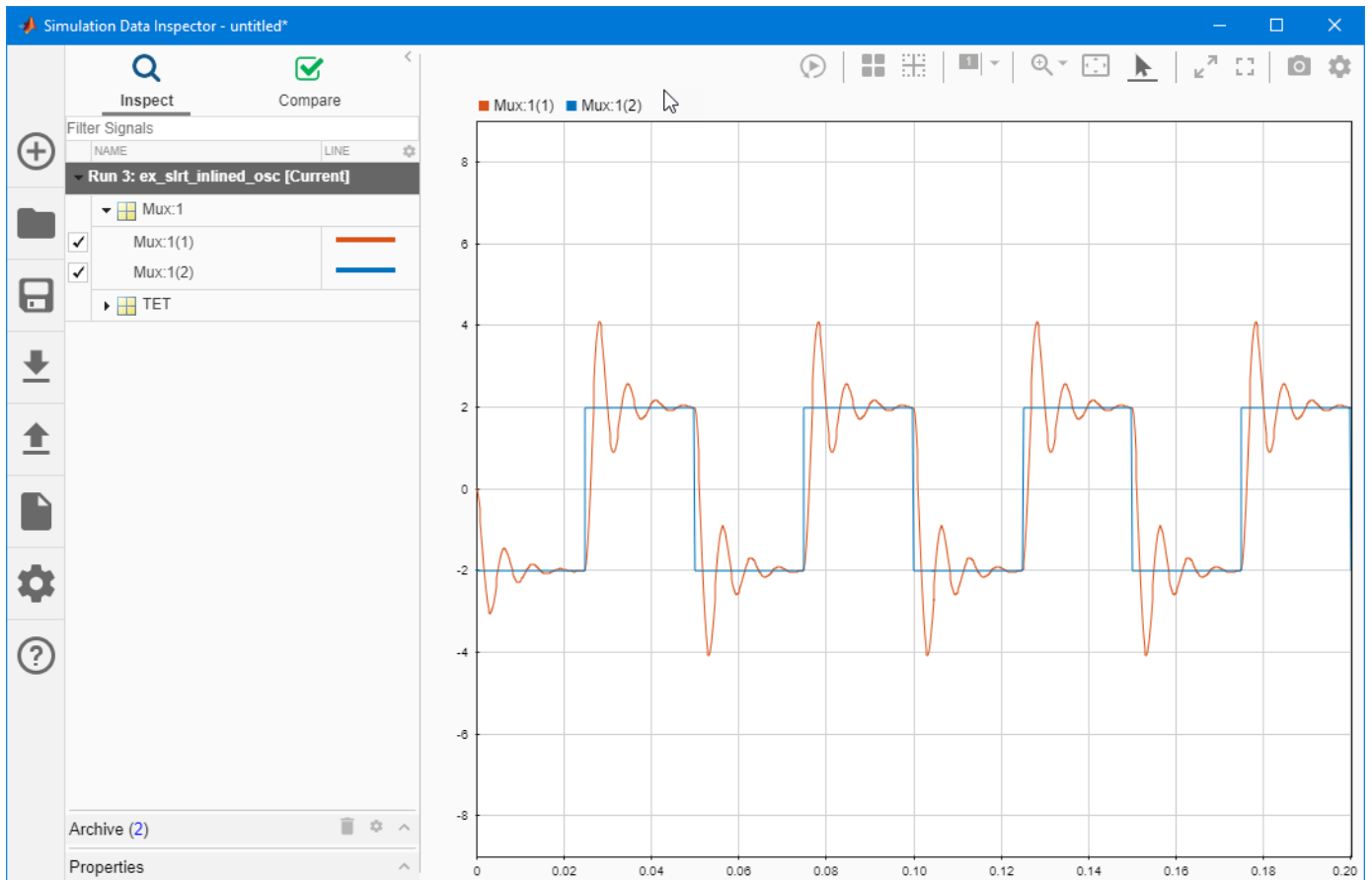
This procedure assumes that you have completed the steps in “Initial Value” on page 6-124.

- 1 Change the value of the MATLAB variable A to 2. In Simulink Real-Time Explorer, 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 , and then click the **Start** button  on the toolbar.

The Simulation Data Inspector looks like this figure.



- 3 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 6-128
- “Display and Filter Hierarchical Signals and Parameters” on page 6-147
- “Troubleshoot Parameters Not Accessible by Name” on page 6-156
- “Troubleshoot Instrument Label Not Present” on page 6-158

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 6-156
- “Troubleshoot Instrument Label Not Present” on page 6-158

Tune Parameter Structures with Simulink Real-Time Explorer

In this section...

“Create Parameter Structure” on page 6-129

“Replace Block Parameters with Parameter Structure Fields” on page 6-130

“Tune Parameters in a Parameter Structure” on page 6-130


“Save and Load Parameter Structure” on page 6-132

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.

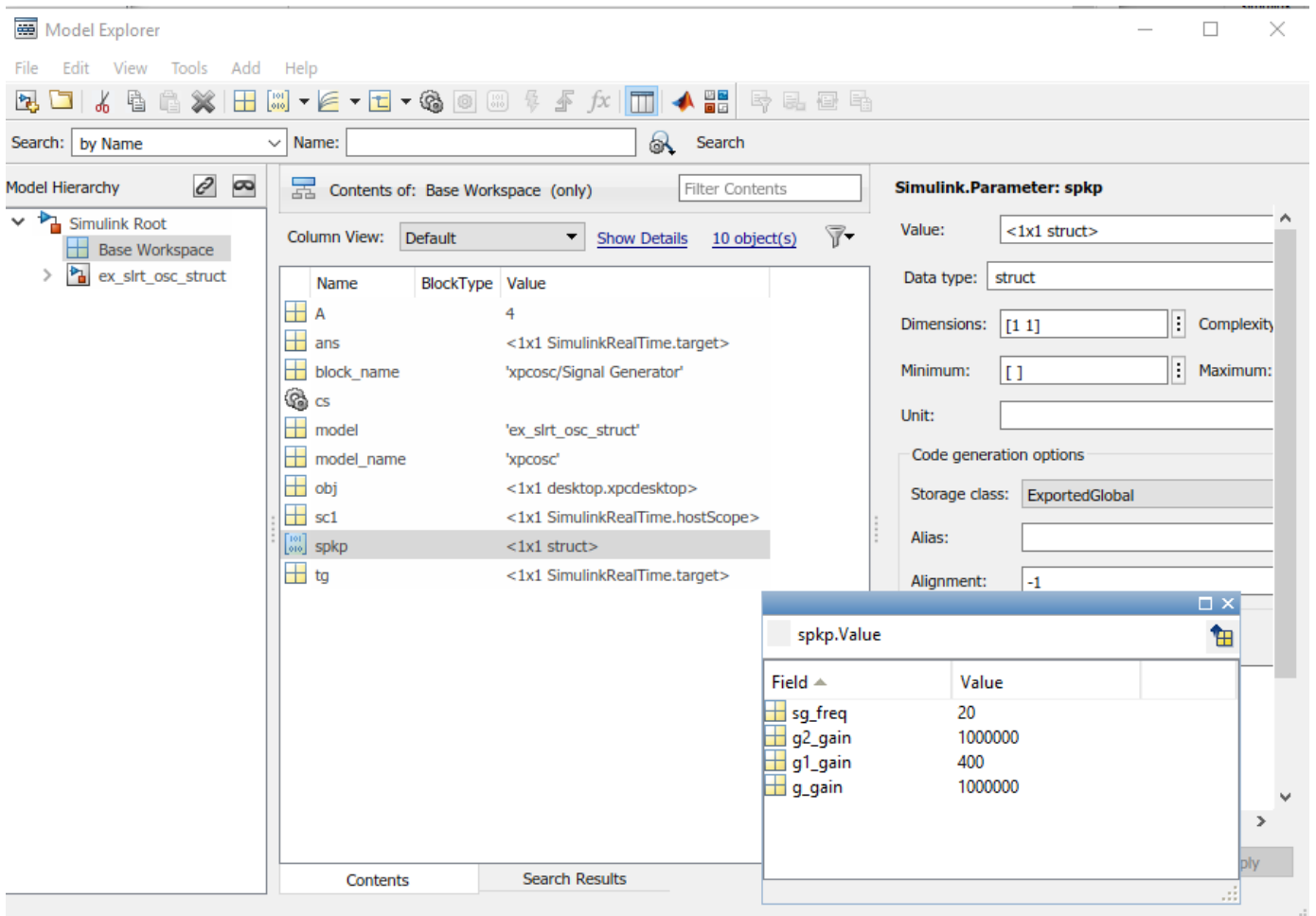
| Block | Parameter | Structure Field Expression | Initial Value |
|------------------|-------------|----------------------------|---------------|
| Signal Generator | Freq | <code>spkp.sg_freq</code> | 20 |
| Gain | Gain | <code>spkp.g_gain</code> | 1000^2 |
| Gain1 | Gain | <code>spkp.g1_gain</code> | $2*0.2*1000$ |
| Gain2 | Gain | <code>spkp.g2_gain</code> | 1000^2 |

Create Parameter Structure

- 1 Open model `xpcosc`, and save a copy of the model to a working folder.
- 2 Open the Base Workspace in the Model Explorer. On the **Modeling** tab, click **Base Workspace**.
- 3 Click the **Add Simulink Parameter** button .
- 4 In the **Name** column, type the name `spkp`.
- 5 In the **Storage class** field, select `ExportedGlobal`.
- 6 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 .







- 7 Click **Apply**.
- 8 Save the model as `ex_slrt_osc_struct`. On the **Simulation** tab, from **Save**, click **Save As**.

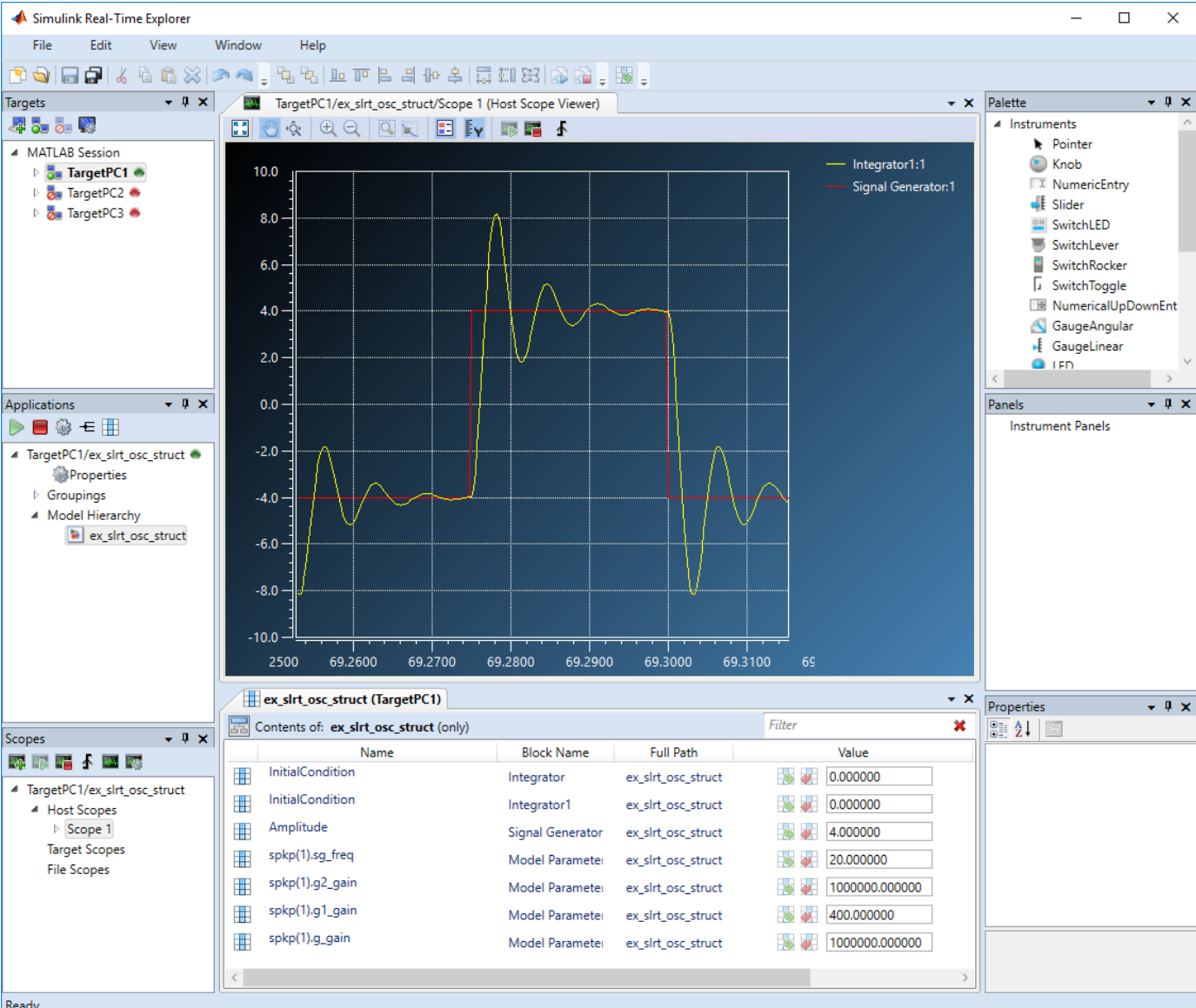
Replace Block Parameters with Parameter Structure Fields

- 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 Simulink Real-Time Explorer. In the **Real-Time** tab, click **Prepare** > **SLRT Explorer**.
- 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 (.

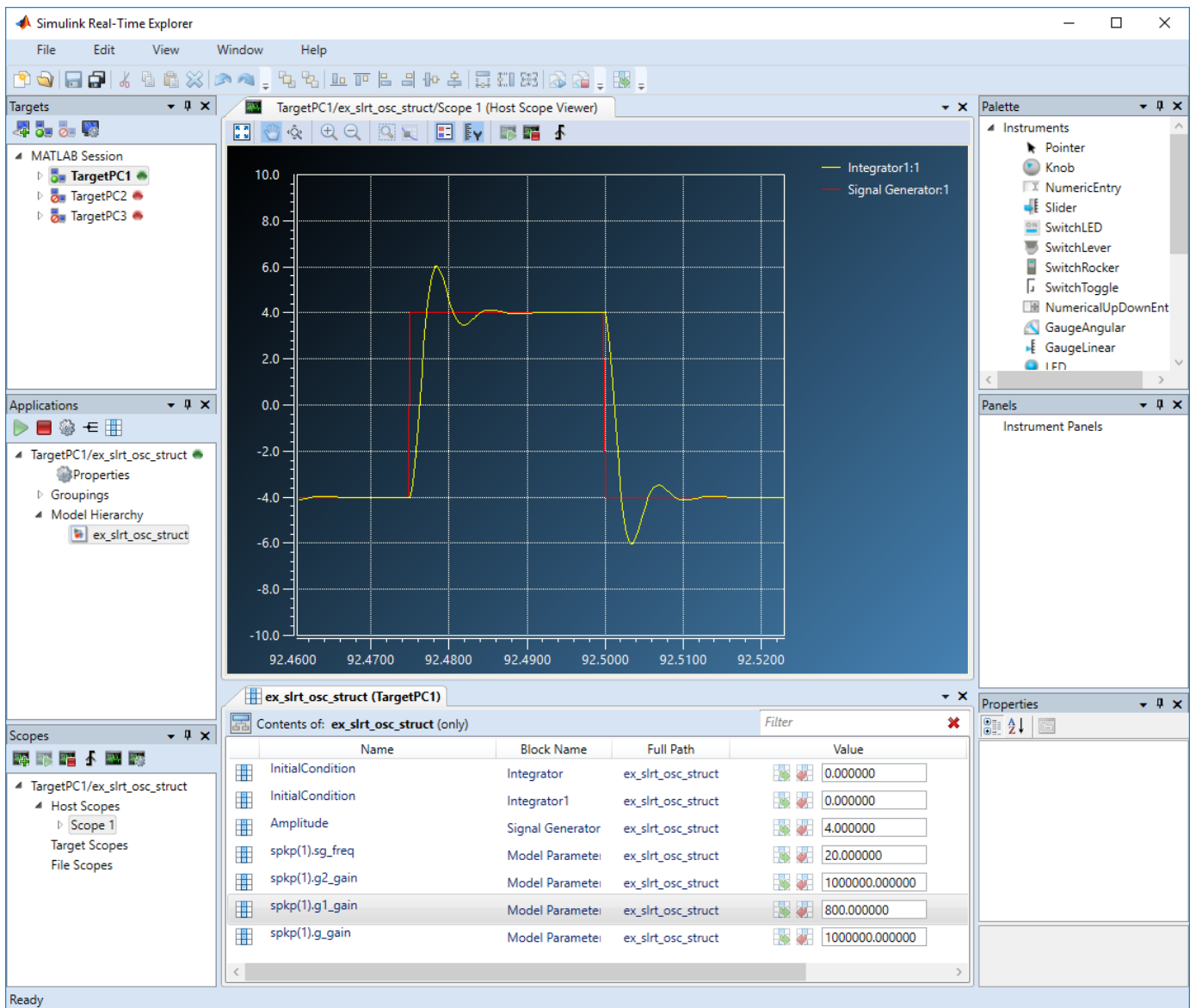


The screenshot shows the Simulink Real-Time Explorer interface. The main window displays a scope plot titled "TargetPC1/ex_slrt_osc_struct/Scope 1 (Host Scope Viewer)". The plot shows two signals: "Integrator:1" (yellow line) and "Signal Generator:1" (red line). The x-axis represents time from 2500 to 65, and the y-axis represents signal amplitude from -10.0 to 10.0. The Signal Generator signal is a step function that transitions from -4.0 to 4.0 at approximately t=69.275. The Integrator signal shows a transient response to this step change, peaking at approximately 8.0 before settling back to 4.0.

Below the scope plot, the "Values" table for the "ex_slrt_osc_struct" model is displayed. The table lists various parameters and their current values.

| Name | Block Name | Full Path | Value |
|------------------|------------------|--------------------|----------------|
| InitialCondition | Integrator | ex_slrt_osc_struct | 0.000000 |
| InitialCondition | Integrator1 | ex_slrt_osc_struct | 0.000000 |
| Amplitude | Signal Generator | ex_slrt_osc_struct | 4.000000 |
| spkp(1).sg_freq | Model Parameter | ex_slrt_osc_struct | 20.000000 |
| spkp(1).g2_gain | Model Parameter | ex_slrt_osc_struct | 1000000.000000 |
| spkp(1).g1_gain | Model Parameter | ex_slrt_osc_struct | 400.000000 |
| spkp(1).g_gain | Model Parameter | ex_slrt_osc_struct | 1000000.000000 |

- 8 In the **Values** text box for `spkp(1).g1_gain`, change the value to 800, click outside of 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 6-147
- “Model Callbacks” (Simulink)

Tune Parameter Structures with MATLAB Language

In this section...

“Create Parameter Structure” on page 6-134

“Replace Block Parameters with Parameter Structure Fields” on page 6-135

“Tune Parameters in a Parameter Structure” on page 6-135

“Save and Load Parameter Structure” on page 6-136

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 | <code>spkp.sg_freq</code> | 20 |
| Gain | Gain | <code>spkp.g_gain</code> | 1000^2 |
| Gain1 | Gain | <code>spkp.g1_gain</code> | $2*0.2*1000$ |
| Gain2 | Gain | <code>spkp.g2_gain</code> | 1000^2 |

Create Parameter Structure

- 1 Open model `xpcosc` and save a copy to 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');
tg = slrt('TargetPC1');
load(tg, 'xpcosc');
```

- 2 Set stop time to `inf`.

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

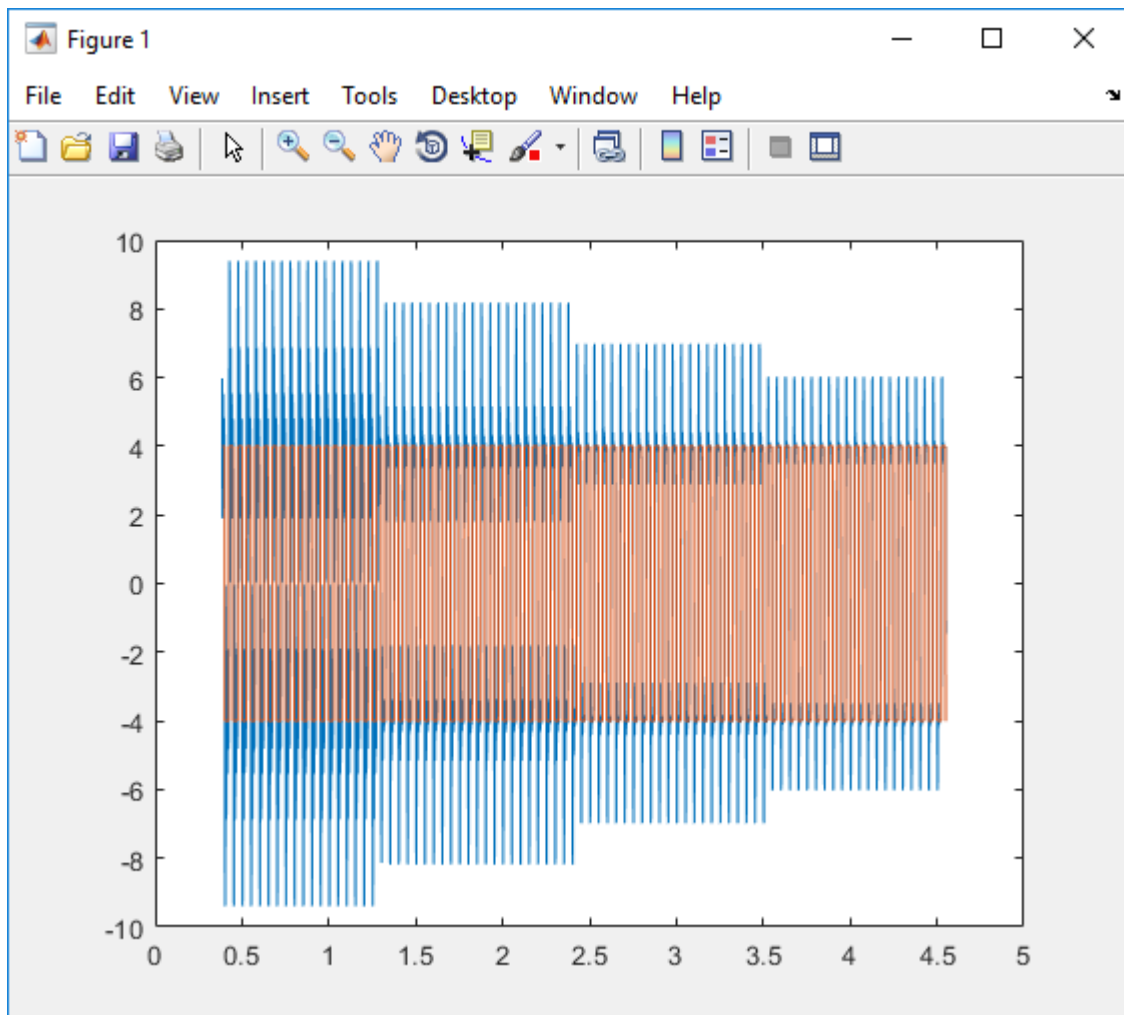
- 3 Sweep the Gain value of the Gain1 block from 200 to 800.

```
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 6-137

“Map Inport to Use Square Wave” on page 6-137

“Update Inport to Use Sawtooth Wave” on page 6-139

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:

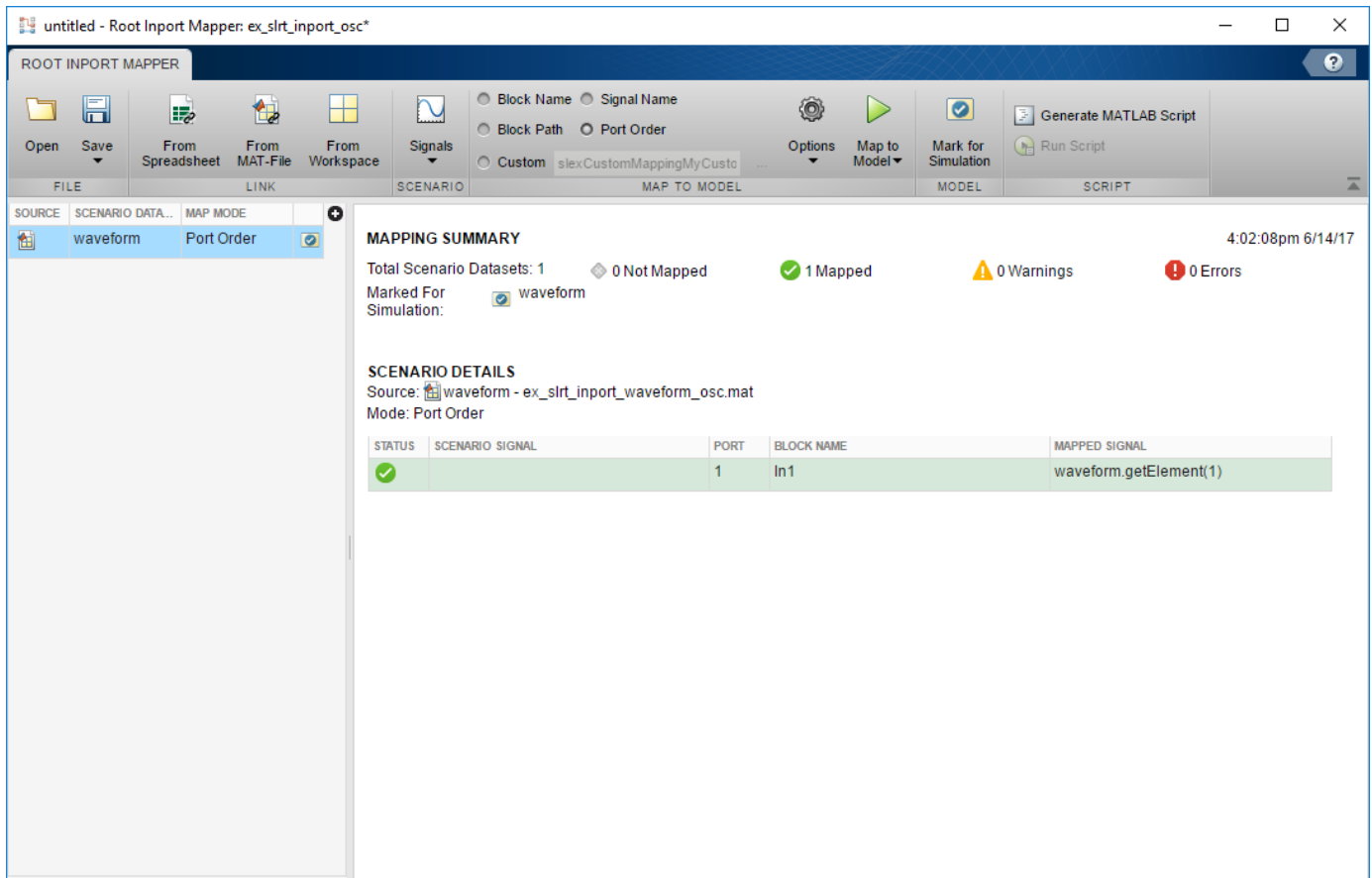
- `ex_slrt_inport_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_inport_osc')))`) — Damped oscillator that takes its input data from input port `In1` and sends its muxed output to output port `Out1`.
- `ex_slrt_inport_square.mat` (`load(docpath(fullfile(docroot, 'toolbox', '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 model `ex_slrt_inport_osc` and save a copy 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**.



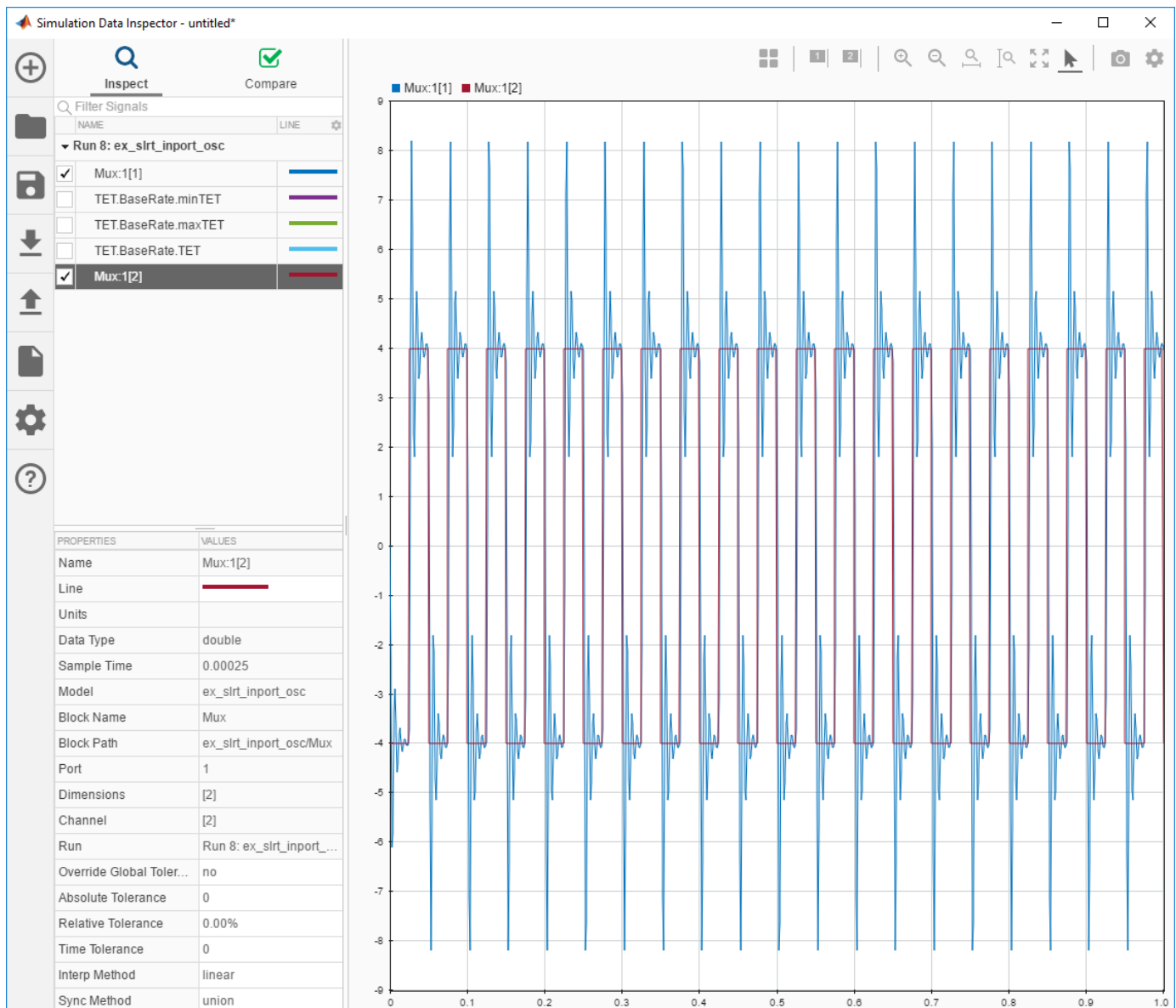
10 Click **Save**.

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 the 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 the 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');
```

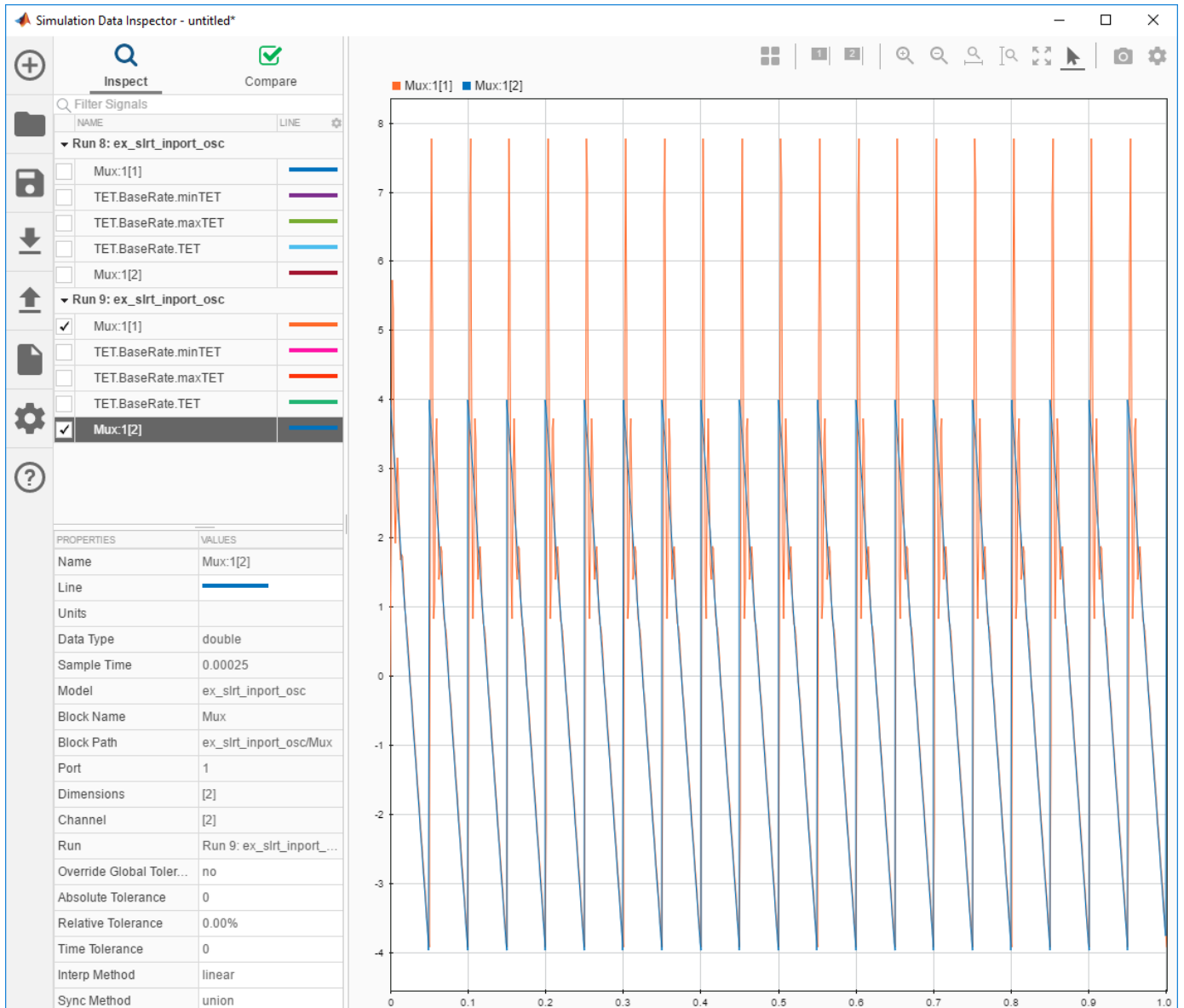
- 3 Update the application object.

```
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 the Simulation Data Inspector.



See Also

More About

- “Define and Update Inport Data with MATLAB Language” on page 6-142
- “Load Data to Root-Level Input Ports” (Simulink)
- “Inport Data Mapping Limitations” on page 6-146
- “Inspect Simulink® Real-Time™ Data with Simulation Data Inspector” on page 6-61

Define and Update Inport Data with MATLAB Language

In this section...

“File Dependencies” on page 6-142

“Map Inport to Use Square Wave” on page 6-142

“Update Inport to Use Sawtooth Wave” on page 6-143

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.

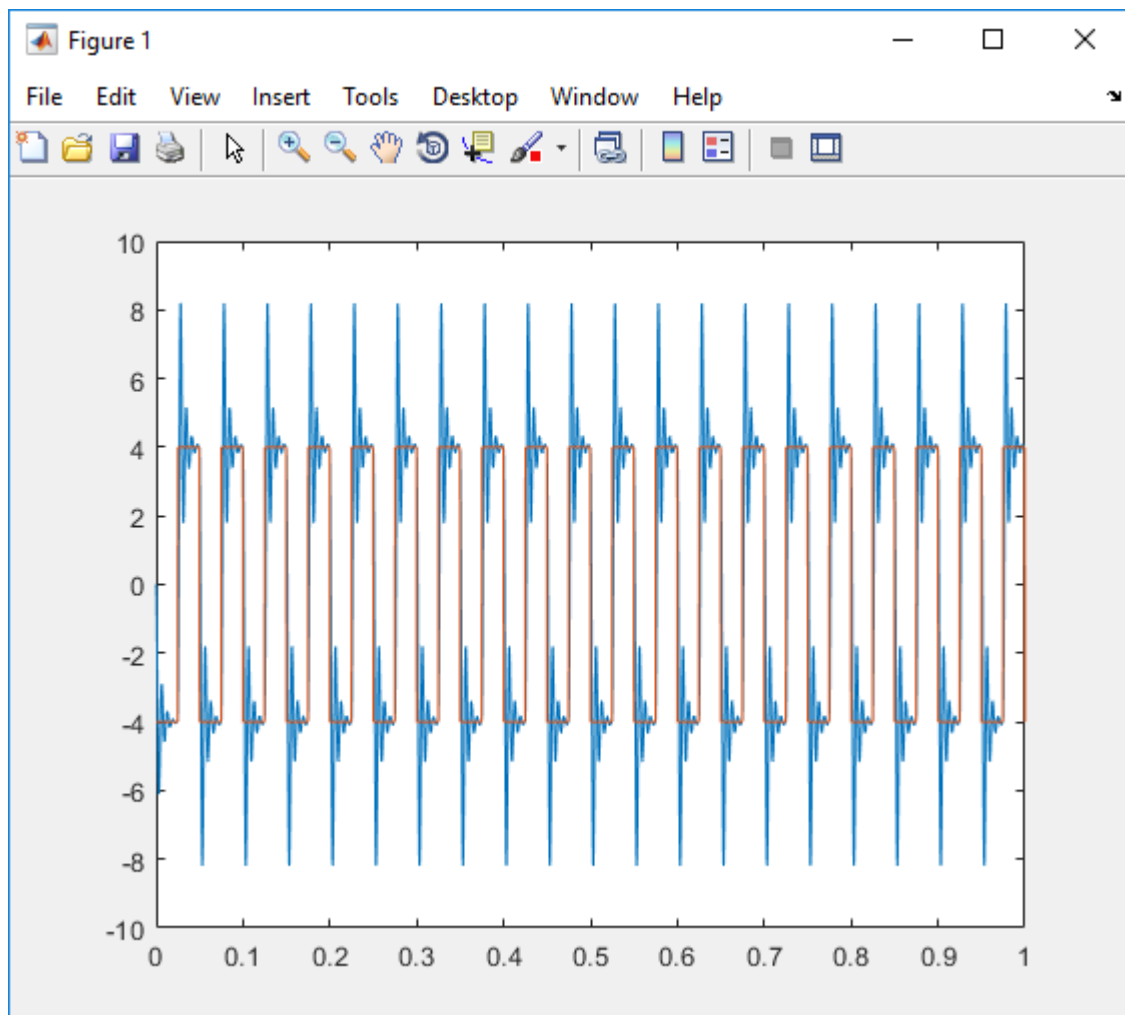
```
set_param(model, 'LoadExternalInput', 'on');
```

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

```
rtwbuild(model);
tg = slrt('TargetPC1');
load(tg,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.

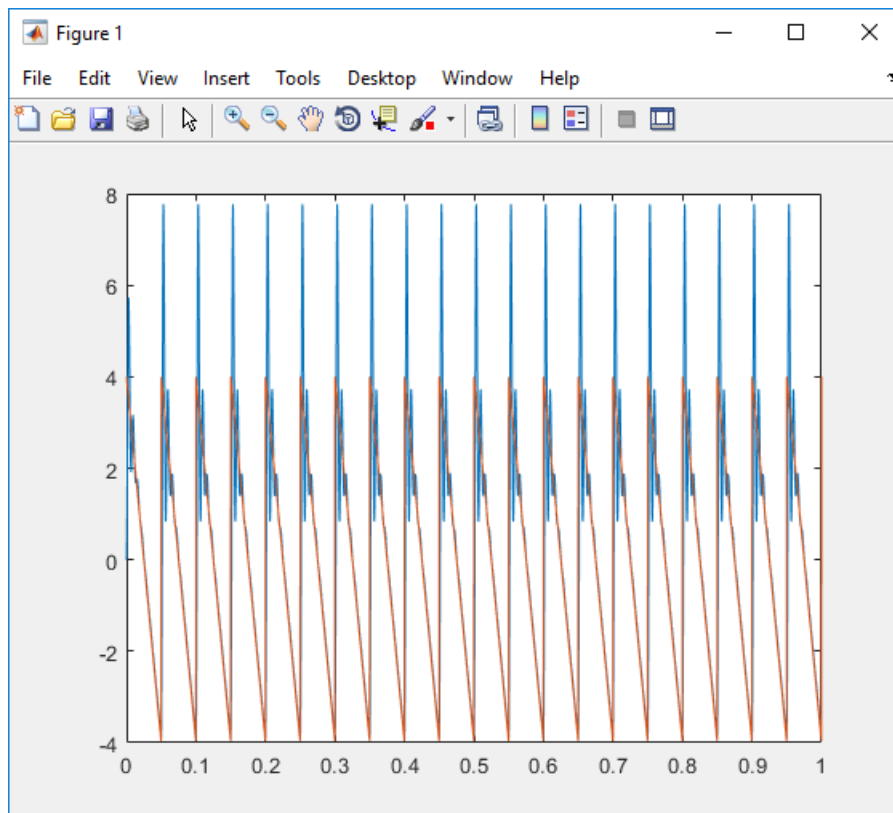
```
updateRootLevelInportData(app_object);
```

- 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 6-137
- “Load Data to Root-Level Input Ports” (Simulink)
- “Inport Data Mapping Limitations” on page 6-146

- “Inspect Simulink® Real-Time™ Data with Simulation Data Inspector” on page 6-61

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 6-137
- “Load Data to Root-Level Input Ports” (Simulink)

Display and Filter Hierarchical Signals and Parameters

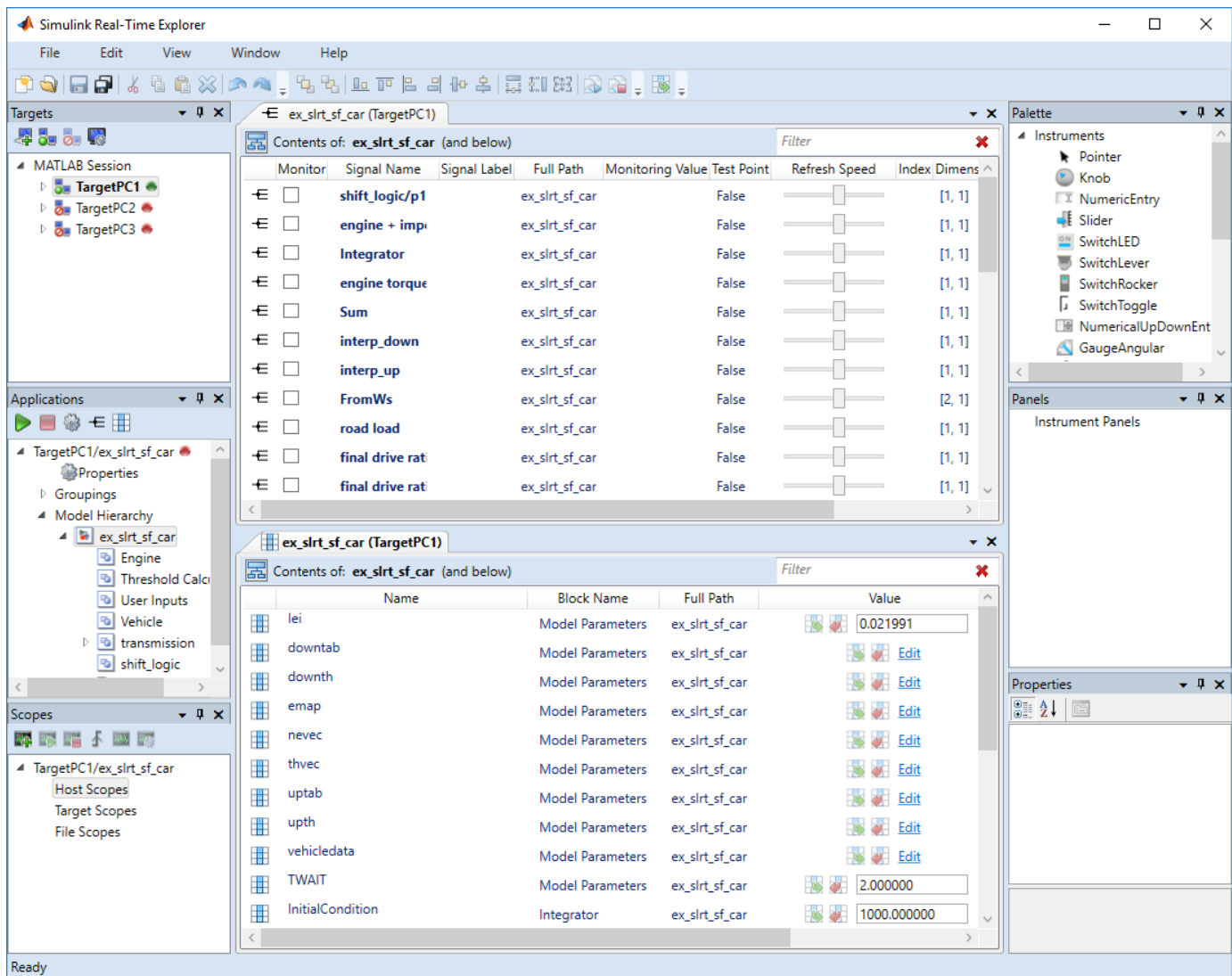
| In this section... |
|--------------------------------------|
| “Hierarchical Display” on page 6-147 |
| “Filtered Display” on page 6-148 |
| “Grouped Display” on page 6-149 |

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.



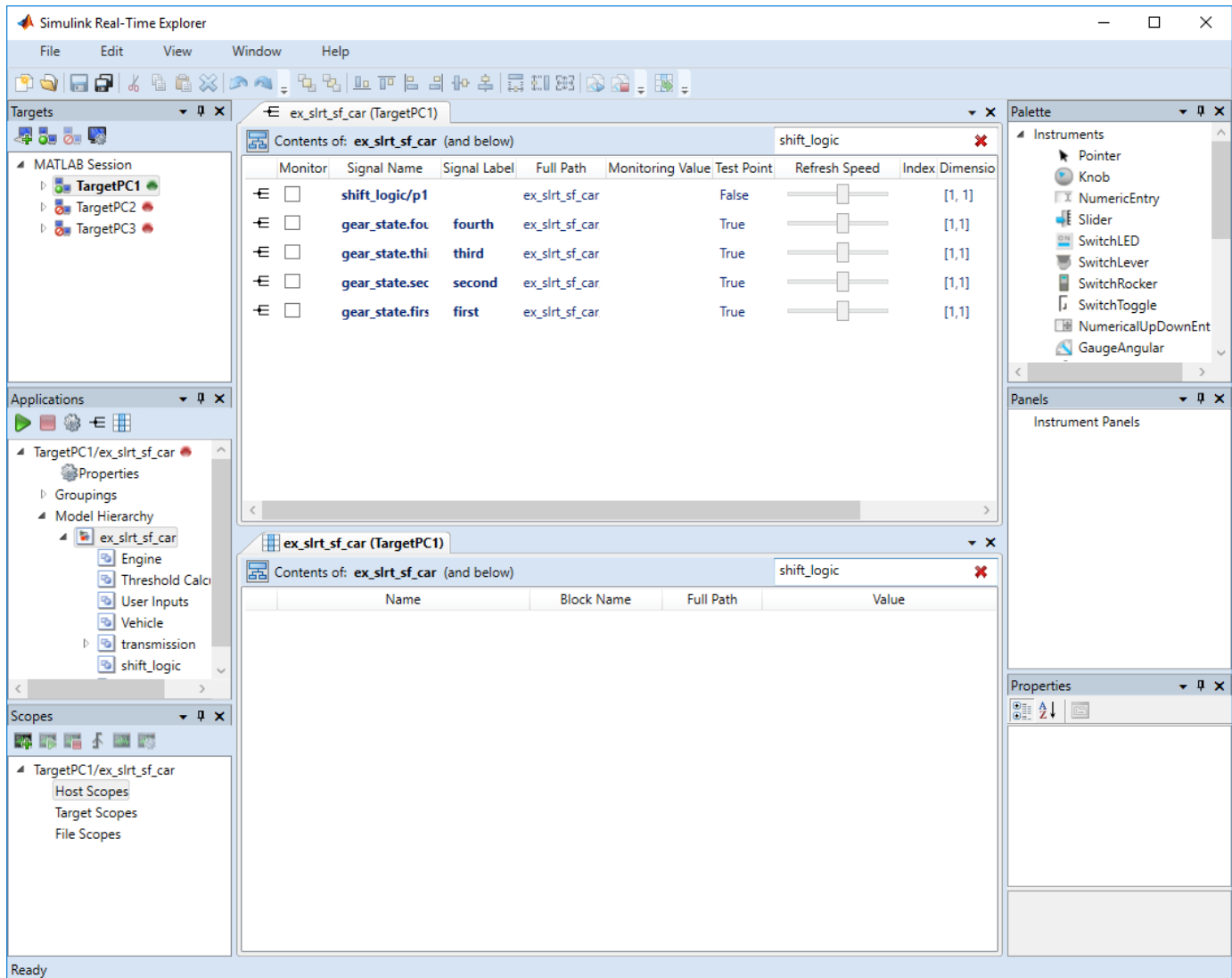
Filtered Display

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.



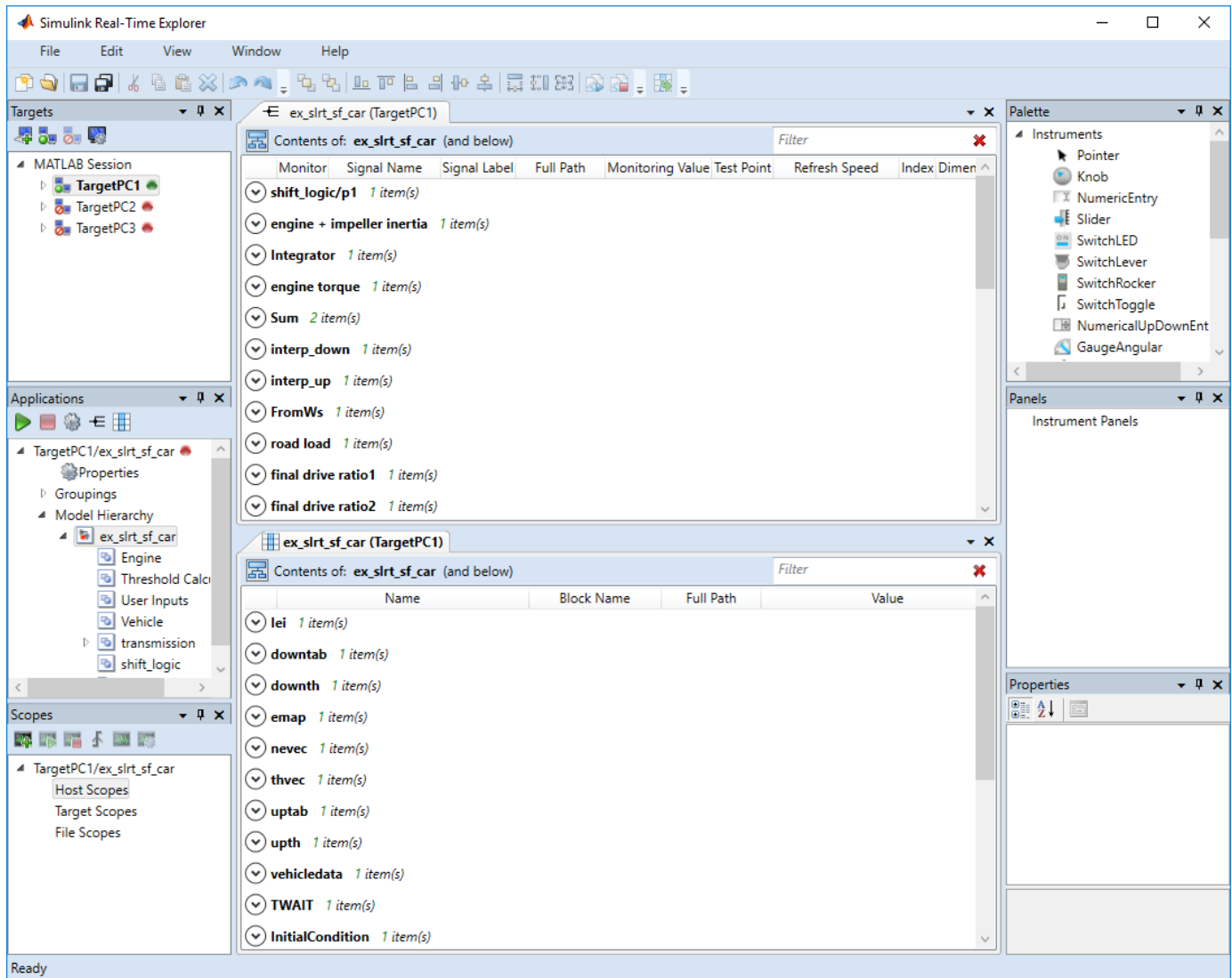
Grouped Display

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**.



Display and Filter Hierarchical Signals and Parameters (tech preview)

In Simulink Real-Time Target Computer Explorer (tech preview), the **Signal** tab and the **Parameters** tab display the signals and parameters only at the level of the hierarchy that you selected. You can display signals and parameters for the current hierarchy node and below. You can filter the display to show only the items in which you are interested.

Use the steps in this example in MATLAB and Simulink to build and deploy the real-time application `sf_car_slrt`. In the MATLAB Command Window type:

```
open_system(fullfile(matlabroot,'toolbox','rtw','targets','xpc','xpcdemos','sf_car_slrt'));
rtwbuild('sf_car_slrt');
tg = slrt('TargetPC1');
```

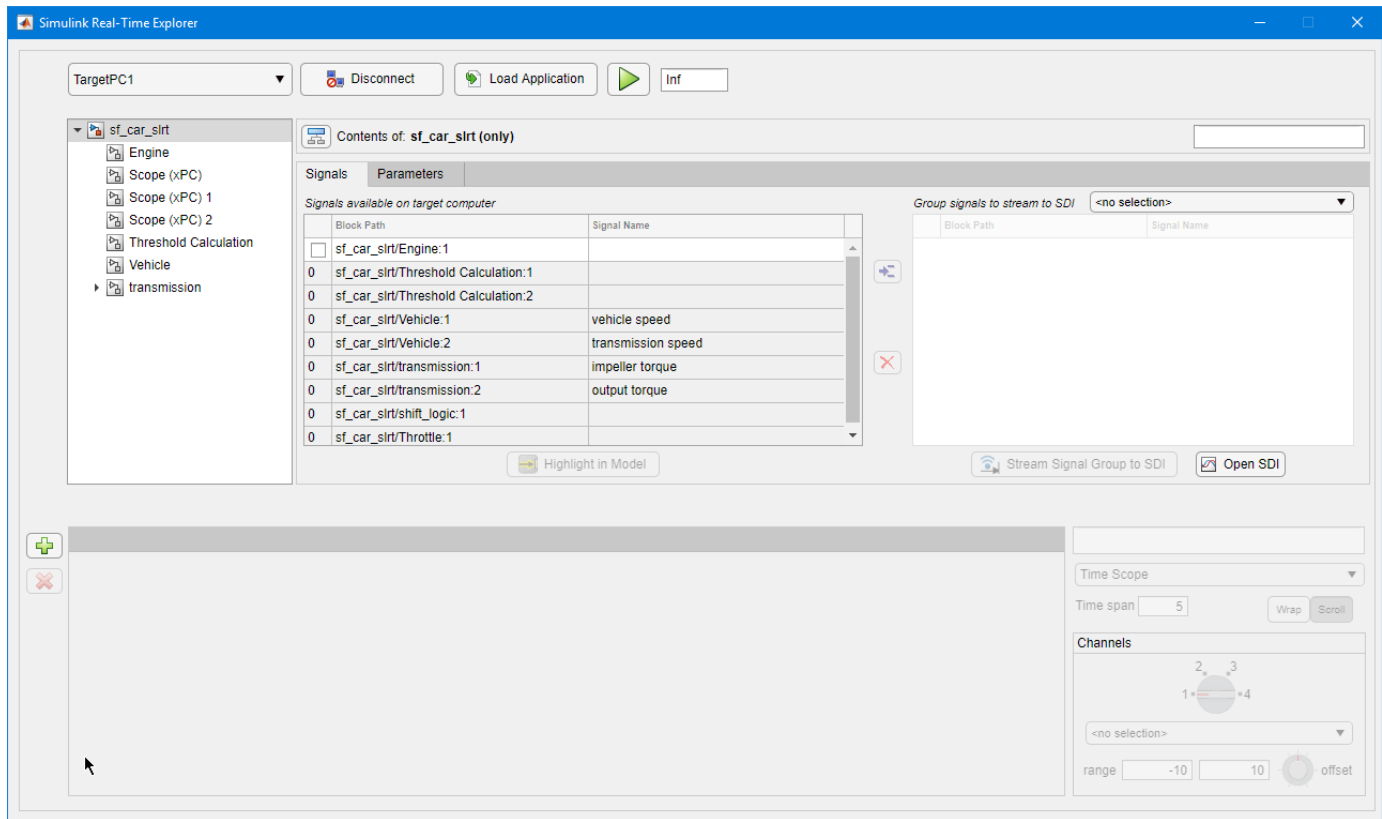
To explore the real-time application, open the Simulink Real-Time Target Computer Explorer tech preview, connect to the target computer, and load the real-time application.

- 1 Open the Target Computer Explorer (tech preview). In the Command Window, type:
`SimulinkRealTime.prototype.Explorer`
- 2 To connect to the real-time application, click **Connect**.
- 3 Click **Load Application**. Browse to and select the `sf_car_slrt.mldatx` file.
- 4 In the hierarchical display, click the `sf_car_slrt` node.

Hierarchical Display

To show signals and parameters from the current hierarchical level and below it, navigate to the hierarchical level in which you are interested.

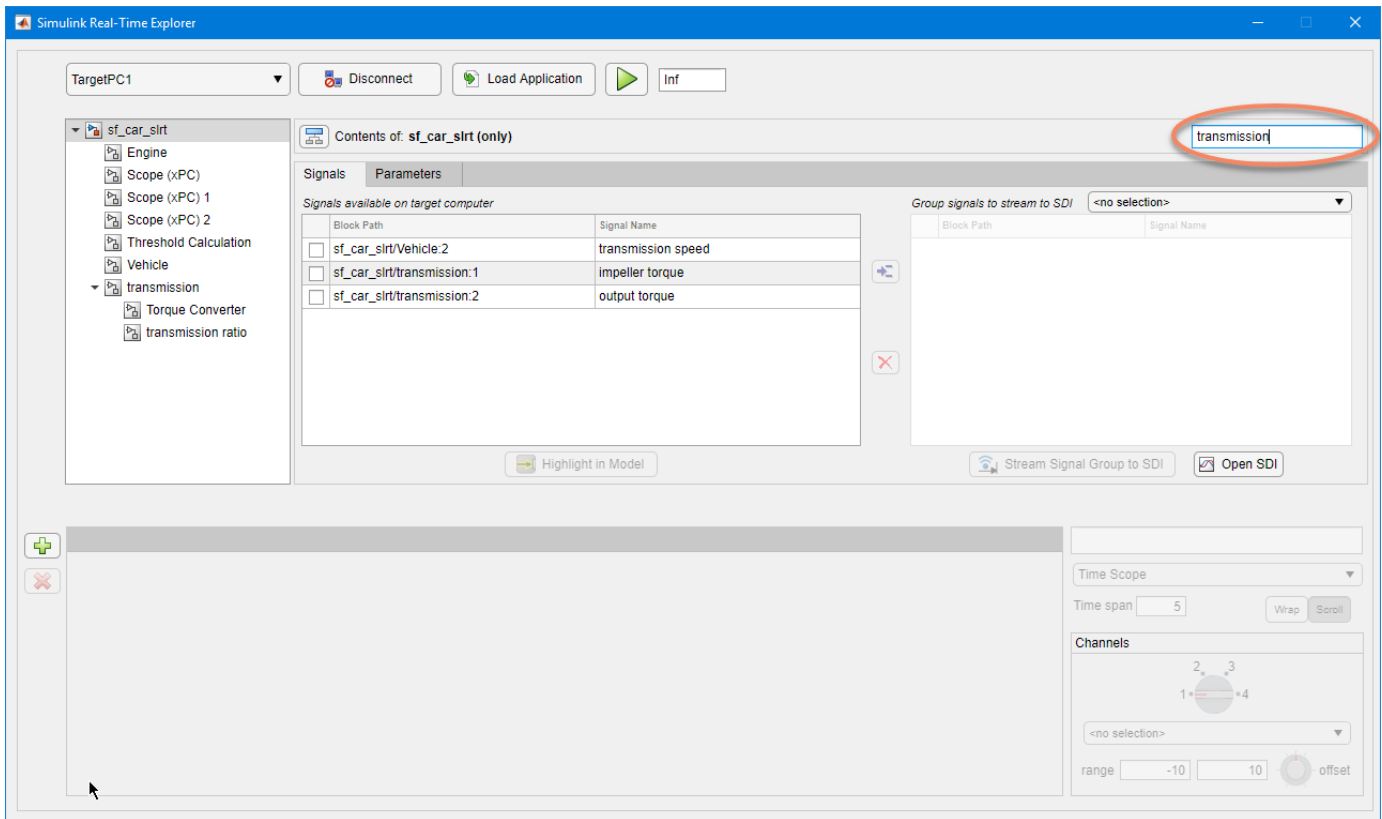
The contents of the top level of real-time application `sf_car_slrt` appear in the figure.





Filtered Display

To restrict the display to signals or parameters with a particular characteristic, use the filter text box. This box appears in the figure, circled in red. If you display only one level, the filter applies only to that level.

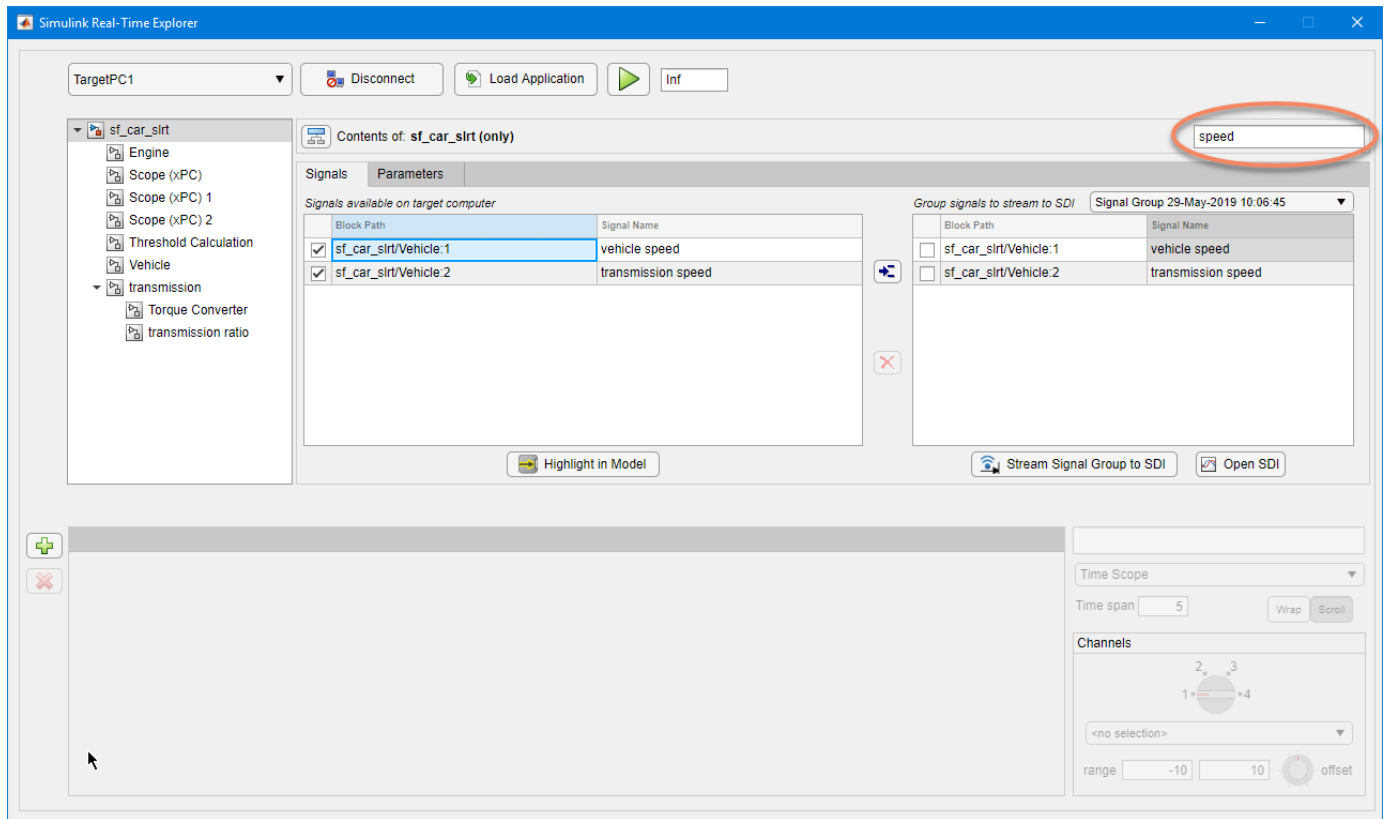
This example shows how to restrict the display of signals and parameters to the **transmission** subsystem by using the filter text **transmission**. The filter applies to the **Block Path** and the **Signal Name**.



Grouped Display

To group signals to stream to the Simulation Data Inspector, select the signal and click **Add selected signals to signal group** . To remove signals from the signal group, click **Remove selected signals from signal group** .

This example filters the hierarchy by using filter text speed and selects the vehicle speed and transmission speed signals to stream to the Simulation Data Inspector.



See Also

`SimulinkRealTime.prototype.Explorer`

More About

- Simulink Real-Time Target Computer Explorer tech preview

Troubleshoot Signals Not Accessible by Name

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

What This Issue Means

You cannot monitor, trace, or log by name these types of signals in the real-time application:

- 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 Simulink optimizes 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 the signal, add a unit Gain block (a Gain block with gain 1) before the model 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 Workaround

Check the signal types for the issues described in “What This Issue Means” on page 6-155.

See Also

Gain

More About

- “Nonvirtual and Virtual Blocks” (Simulink)
- “Types of Composite Signals” (Simulink)
- “Signal storage reuse” (Simulink Coder)
- “Block reduction” (Simulink)
- “Troubleshoot Parameters Not Accessible by Name” on page 6-156
- “Internationalization Issues” on page 6-160

Troubleshoot Parameters Not Accessible by Name

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

What This Issue 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.
- During execution, you cannot tune parameters 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 Workaround

Check the parameters for the issues described in “What This Issue Means” on page 6-156.

See Also

More About

- “Troubleshoot Signals Not Accessible by Name” on page 6-155
- “Internationalization Issues” on page 6-160

Troubleshoot Instance-Specific Parameters Not Saved

The `saveparamset` function does not save instance-specific parameters and parameters with custom storage classes to a MAT file for loading with the `loadparamset` function. When I use the `saveparamset` function on a model that contains only instance-specific parameters, I get an error message.

```
Error using SimulinkRealTime.target/saveparamset
TargetPC1: Error writing file
```

What This Issue Means

The `saveparamset` function saves parameters that appear in the `rtP` structure of the model. Instance-specific parameters and parameters with custom storage classes are global variables that are not by default represented in the `rtP` structure.

Try This Workaround

You can use the `saveparamset` function to save parameter sets from models that include instance-specific parameters or parameters with custom storage classes. But, these parameters do not appear in the saved parameter set.

See Also

More About

- “Save and Reload Parameters with MATLAB Language” on page 6-117

Troubleshoot Instrument Label Not Present

I get an instrument label error when working with instrumentation panels in Simulink Real-Time Explorer.

```
'The given key was not present in the dictionary'
```

What This Issue Means

This error indicates that one or more labels added to instrument panels in Simulink Real-Time Explorer could have a corrupted name.

This label issue causes errors when opening and closing the instrument panel.

Try This Workaround

Resolve the errors for corrupted labels by renaming instruments. Simulink Real-Time Explorer names labels in the form `Label<num>`, where `<num>` is an integer.

The numbering starts at "1" and increments by one each time a new label is added.

Occasionally, a label may receive a name with a very large value for `<num>` (for example, `Label2147483648`).

Use Simulink Real-Time Explorer to find these labels with large values for `<num>` and rename them to avoid errors.

See Also

Instrument Properties

More About

- “Instrumentation for Real-Time Applications”

Troubleshoot Internationalization Issues

I want to identify signal and parameter names through the target computer keyboard when the names contain Unicode characters from a locale other than English.

What This Issue Means

Simulink Real-Time supports the internationalization because the products that it works with support internationalization. These products include 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. When you use host scopes to observe signals, the non-ASCII signal names are displayed with the expected characters.

The Simulink Real-Time kernel does not support international (non-ASCII) characters. It generates messages in English by 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 Workaround

For more information, see “Internationalization Issues” on page 6-160.

See Also

More About

- “Internationalization Issues” on page 6-160

External Websites

- <https://www.speedgoat.com/support>

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 6-155
- “Troubleshoot Parameters Not Accessible by Name” on page 6-156
- “Troubleshoot Instrument Label Not Present” on page 6-158

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 `-axpcCPUlockPoll` 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 ($T_s = 10\text{-}50\ \mu\text{s}$).
- DSP systems — Sample-based DSP system models of average model size and I/O complexity that are executed at high sample rates ($F_s = 20\text{-}100\ \text{kHz}$). 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 10-9.

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 25-5

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 8-5
“Creating a Real-Time Application Model” on page 8-6
“Marking Block Parameters” on page 8-6
“Marking Block Signals” on page 8-7

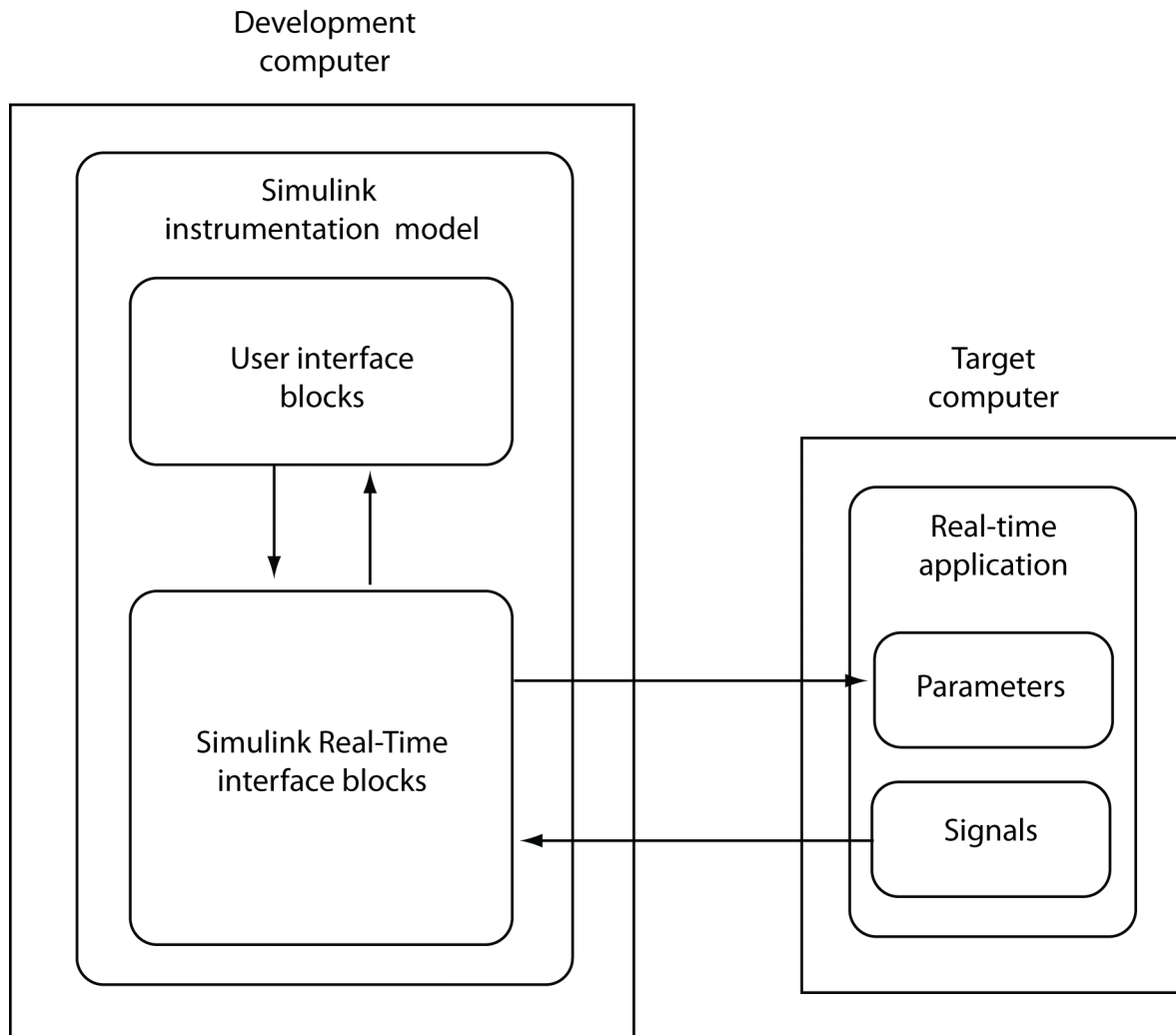
Simulink User Interface Model

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 8-6.
- 3 Tag the signals in Simulink model that you want to be connected to a display device. See “Marking Block Signals” on page 8-7.
- 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.

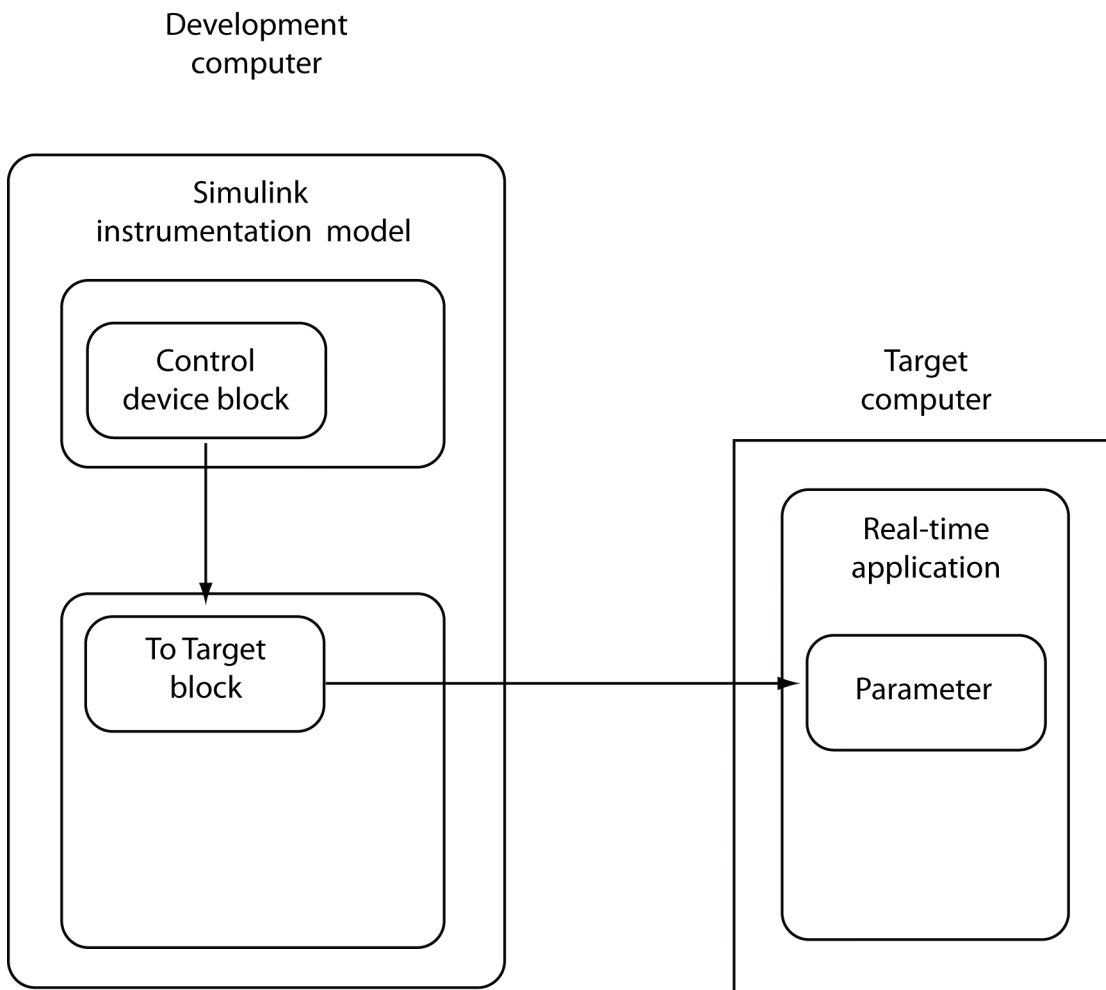
- 5 To the user interface template model, add Simulink interface blocks from add-on products (Simulink 3D Animation, Altia Design).
 - You can connect Altia blocks to the Simulink Real-Time To PC Target interface blocks. Connect the To Target blocks on the left to control devices.
 - You can connect 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.



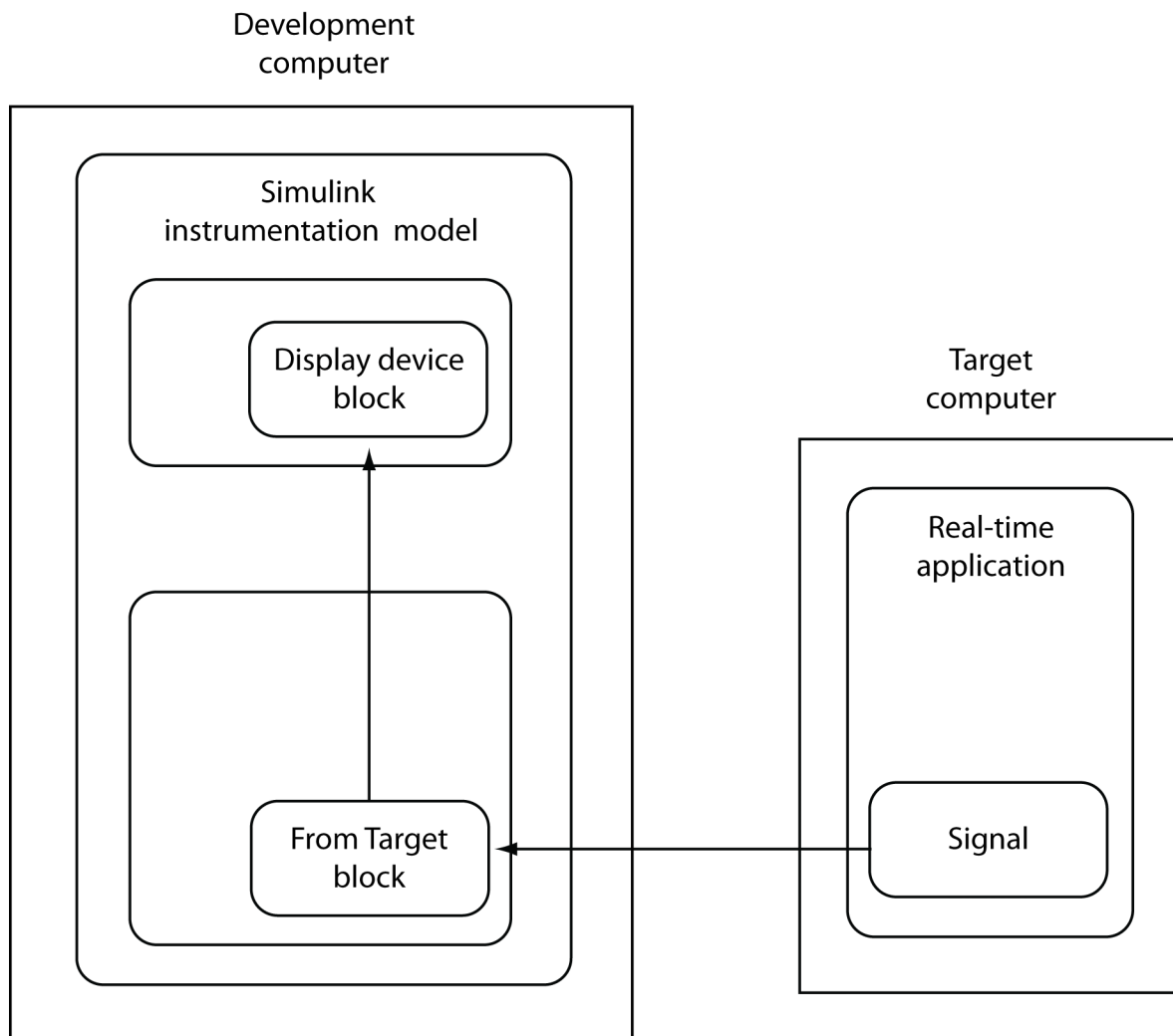
This block is implemented as a MATLAB S-function. The block only changes a parameter on the real-time application when the input value differs from the value that existed at the last time step. This block uses the parameter downloading feature of the Simulink Real-Time command-line interface.

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 8-6 and “Marking Block Signals” on page 8-7 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`.

- 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_1 label_2 label_3 label_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;
```

- On the **Simulation** tab, from **Save**, 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 8-7 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 8-6.

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.

- 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 On the **Simulation** tab, from **Save**, 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 8-6. 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.

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 9-4.

- 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

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 9-4.

- 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 10-9
- “Limits on Sample Time” on page 10-10
- “CPU Overload Options” on page 10-11
- “Execution Profiling for Real-Time Applications” on page 10-15
- “Reduce Build Time for Simulink Real-Time Referenced Models” on page 10-20
- “Sample Time and Throughput in Real-Time Applications” on page 10-22

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

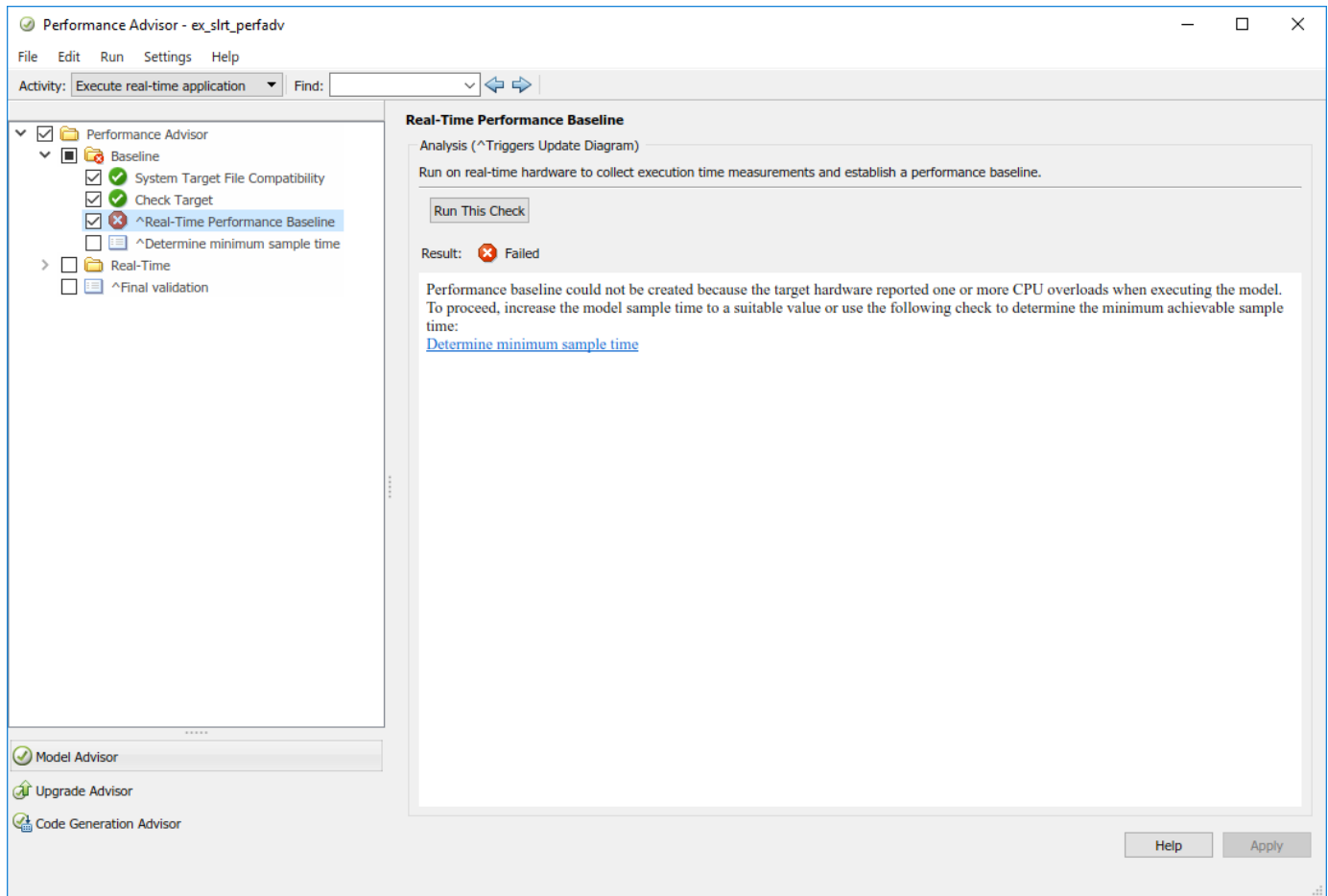
Generate Baseline

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

1. Open model `ex_slrt_perfadv`.
2. Open Performance Advisor. On the **Debug** tab, click **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**.

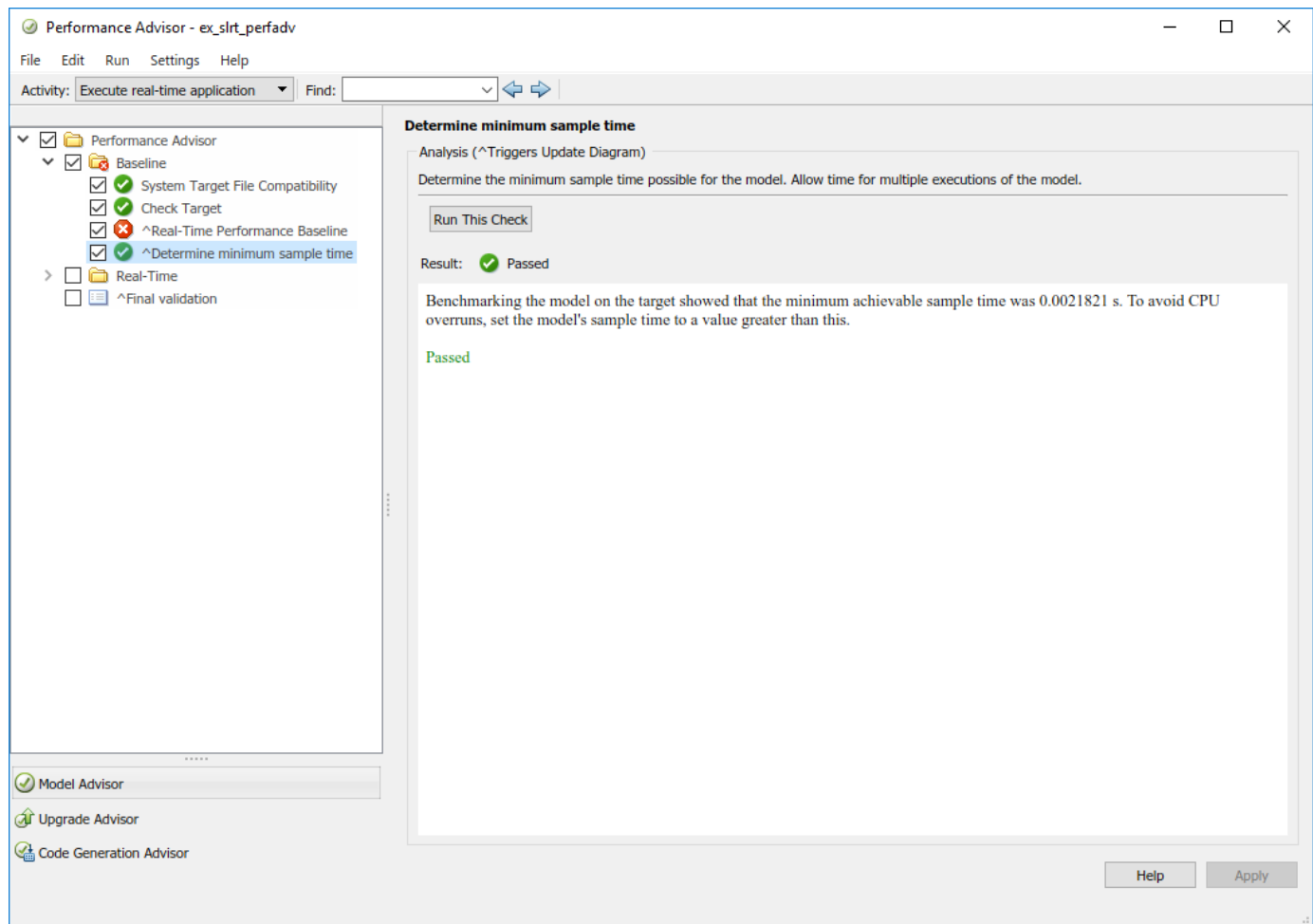
Determining the minimum sample time can be a lengthy process for a large model with a long execution time.

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**.



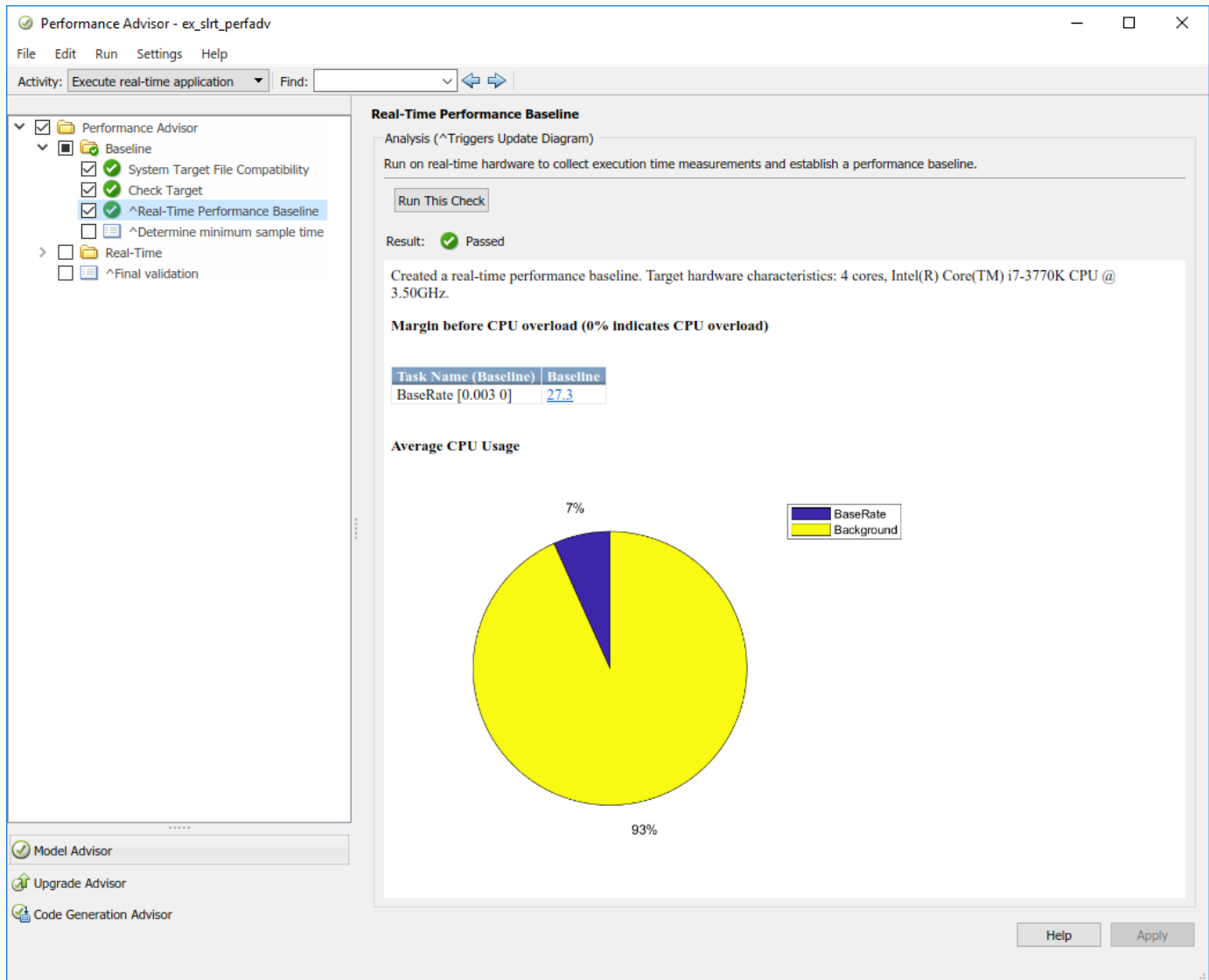
To avoid the overloads that random variations can cause, set T_s to a value above the lower limit. For example, set it to 0.003 s.

7. In the Command Window, type:

$T_s = 0.003$

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**.



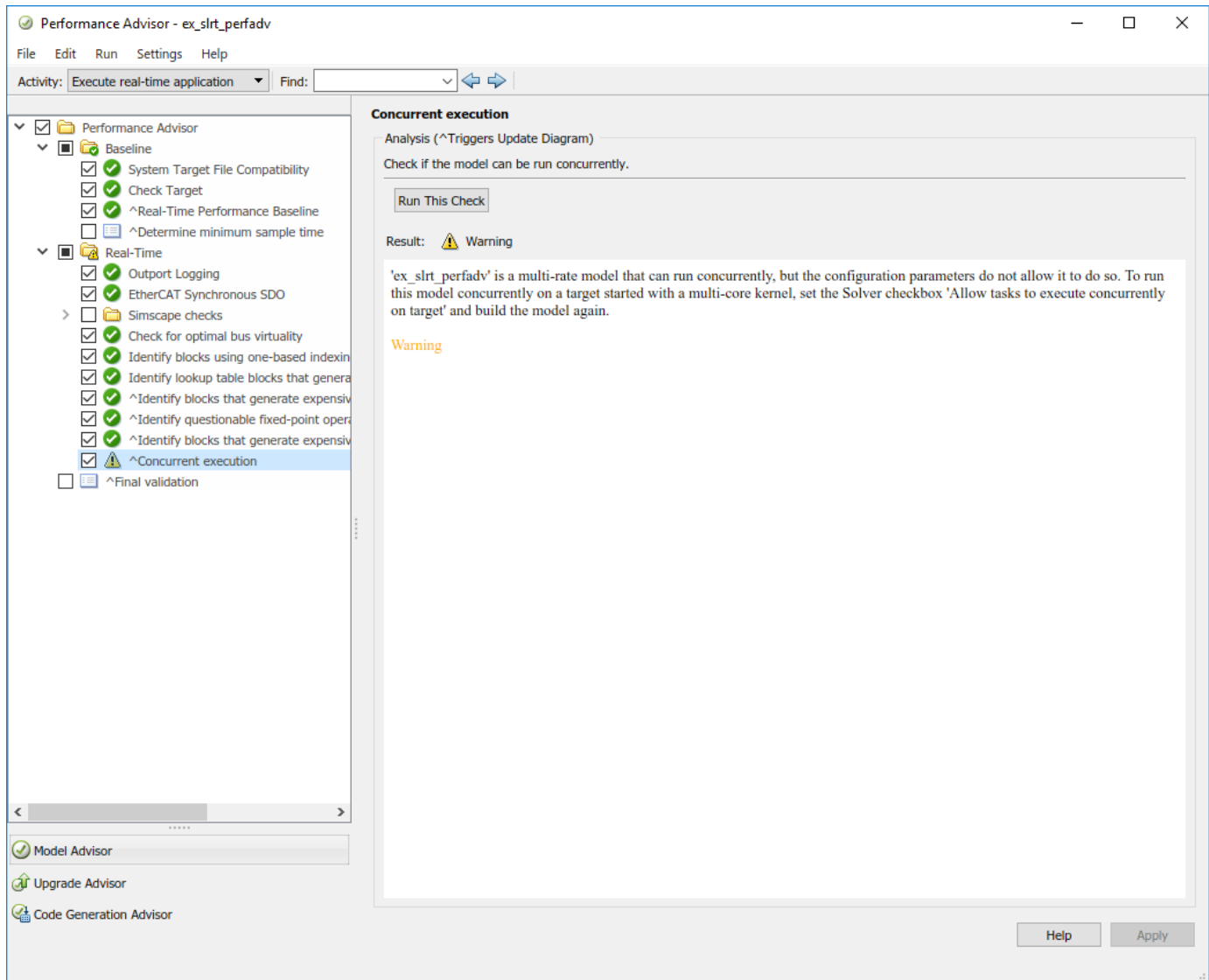
Perform Real-Time Checks

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.

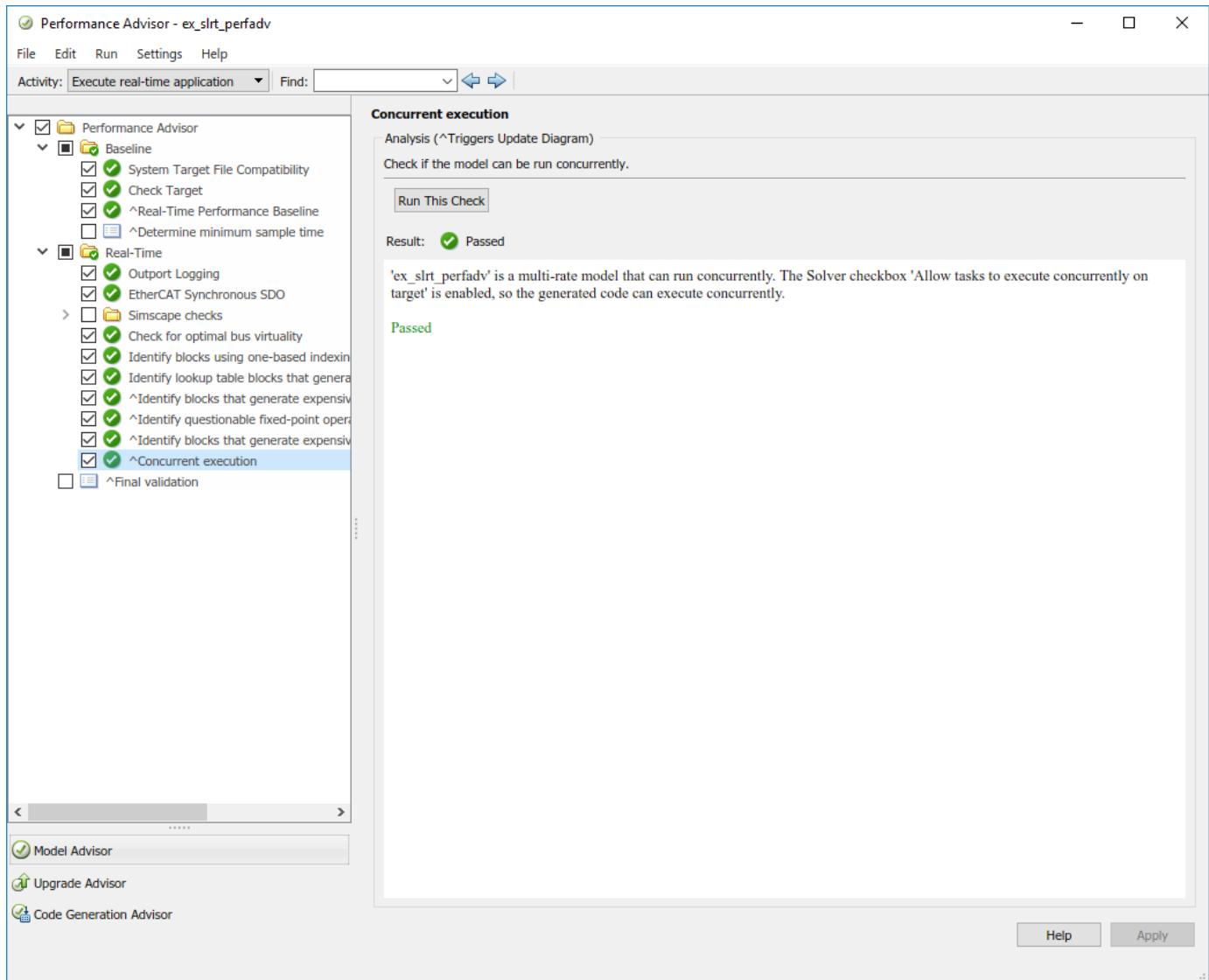
If you have a license for Simscape™ or its related products, such as Simscape Driveline™ and Simscape Electrical™, clear those checks. `ex_slrt_perfadv` contains no Simscape blocks.

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 T_s to a value above the lower limit. For example, set it to 0.001 s.

7. In the Command Window, type:

```
 $T_s = 0.001$ 
```

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**.

Performance Advisor - ex_slrt_perfadv

File Edit Run Settings Help

Activity: Execute real-time application Find: []

- Performance Advisor
 - Baseline
 - System Target File Compatibility
 - Check Target
 - Real-Time Performance Baseline
 - Determine minimum sample time
 - Real-Time
 - Output Logging
 - EtherCAT Synchronous SDO
 - Simscape checks
 - Check for optimal bus virtuality
 - Identify blocks using one-based indexing
 - Identify lookup table blocks that generate expensive operations
 - Identify blocks that generate expensive operations
 - Identify questionable fixed-point operations
 - Identify blocks that generate expensive operations
 - Concurrent execution
 - Final validation

Final validation

Analysis (^Triggers Update Diagram)

Compare performance against baseline collected in 'Real-Time Performance Baseline'.

Run This Check

Result: ✔ Passed

Margin before CPU overload (0% indicates CPU overload)

| Task Name (Baseline) | Margin | Task Name (Final) | Margin |
|----------------------|--------|---------------------|--------|
| BaseRate [0.003 0] | 27.3 | Model1_R1 [0.001 0] | 94.5 |
| | | Model2_R1 [0.001 0] | 93.5 |
| | | Model1_R3 [0.003 0] | 81.2 |
| | | Model1_R2 [0.002 0] | 86.4 |
| | | Model2_R3 [0.003 0] | 74.4 |
| | | Model1_R4 [0.004 0] | 98.2 |
| | | Model2_R4 [0.004 0] | 84.5 |

Average CPU Usage

Final validation checks completed.

Help Apply

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'.

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 Open Model Explorer. In the Simulink Editor, on the **Modeling** tab, select **Model Workspace**.
- 6 In the **Model Hierarchy** pane, expand the model node and select the Configurations node.
- 7 In the **Contents** pane, right-click the new configuration and select **Open** to open the Configuration Parameters dialog box.
- 8 In the Configuration Parameters dialog box, select **Solver**.
- 9 Check **Allow tasks to execute concurrently on target**.
- 10 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 25-5
- “Execution Modes” on page 7-2

CPU Overload Options

In this section...

“Option Behavior” on page 10-11
 “Violation of xPCMaxOverloads” on page 10-12
 “Violation of xPCMaxOverloadLen” on page 10-13
 “Violation of xPCStartupFlag” on page 10-13

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`.

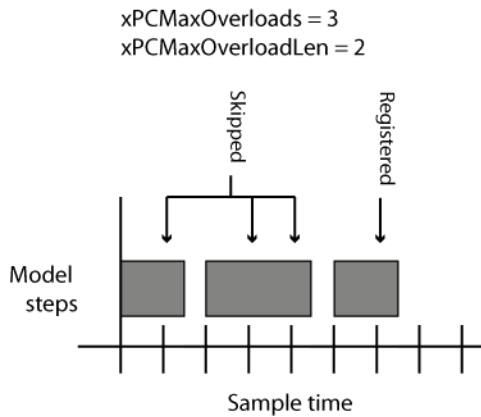
```
set_param('xpcosc','TLCOptions','-axPCMaxOverloads=30  
-axPCOverLoadLen=2 -axPCStartupFlag=5')
```

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

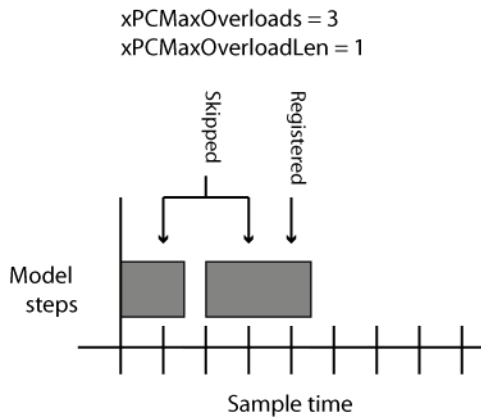
Violation of `xPCMaxOverloads`

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.



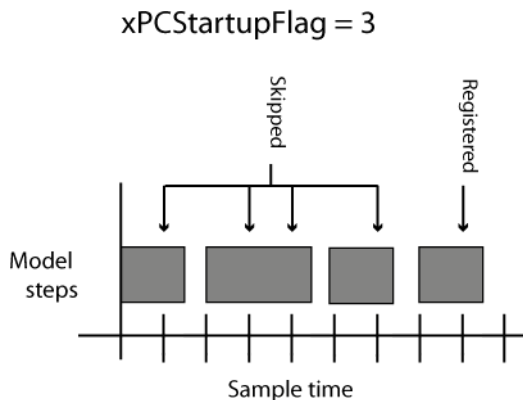
Violation of xPCMaxOverloadLen

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. For **Measure function execution times**, select **Coarse (reference models and subsystems only)**. The **Measure task execution time** check box is checked and locked. Or, in the MATLAB command window, type:

```
set_param('dxpcmds6t','CodeProfilingInstrumentation','Coarse');
```

4. Click **OK**. Save model `dxpcmds6t` in a local folder.

Generate Real-Time Application Execution Profile

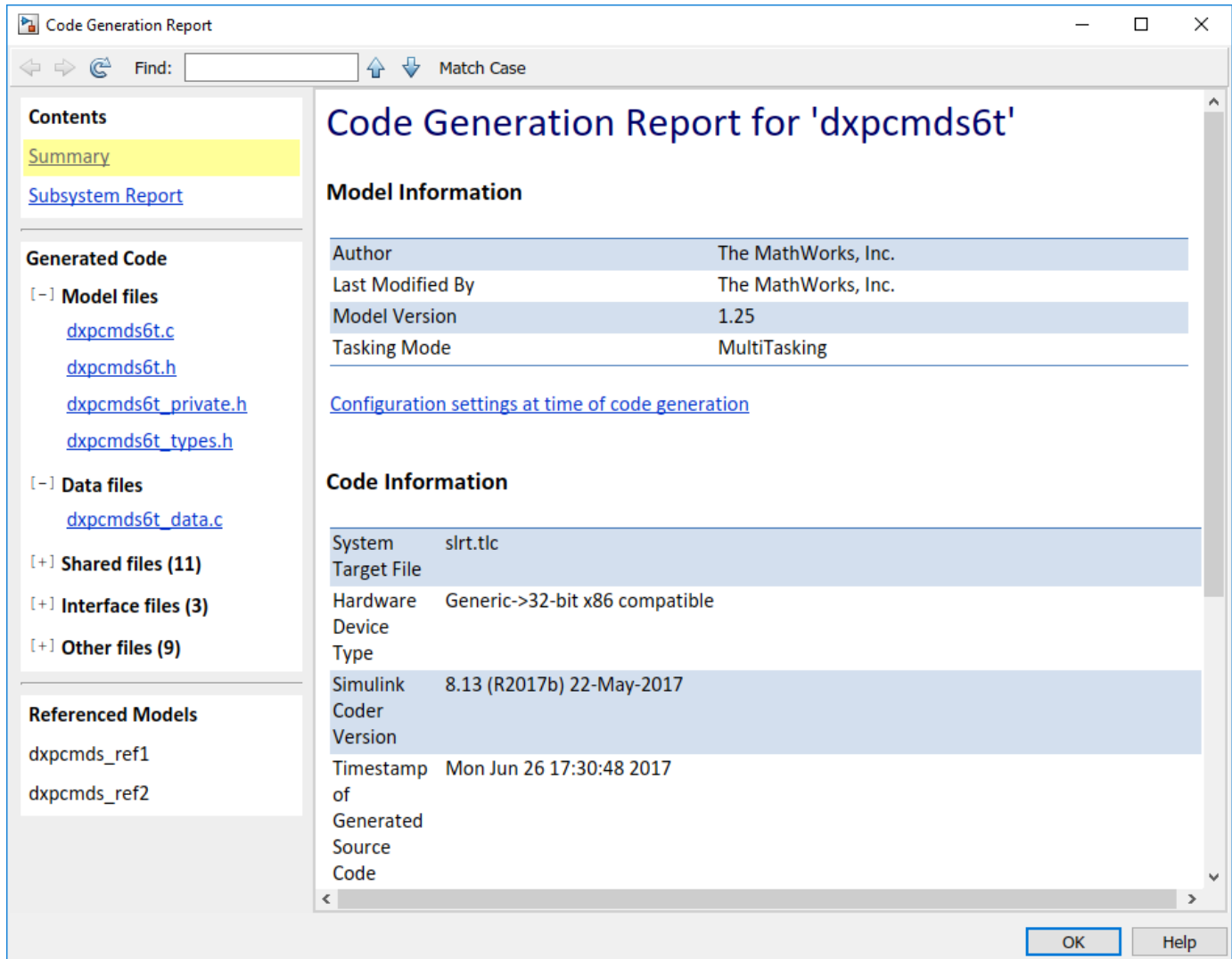
In this section, generate profile data for model `dxpcmds6t` on a multicore target computer.

This procedure assumes that you have configured the target computer to take advantage of multiple cores. It also assumes that you previously configured the model for task and function execution profiling.

1. Open, build, and download the model.

```
mdl = 'dxpcmds6t';
open_system(mdl);
rtwbuild(mdl);
tg = slrt('TargetPC1');
load(tg,mdl);
```

When you include profiling, the Code Generation Report is generated by default. It contains links to the generated C code and include files. By clicking these links, you can examine the generated code and interpret the Code Execution Profile Report.



2. Start the profiler and then execute the real-time application.

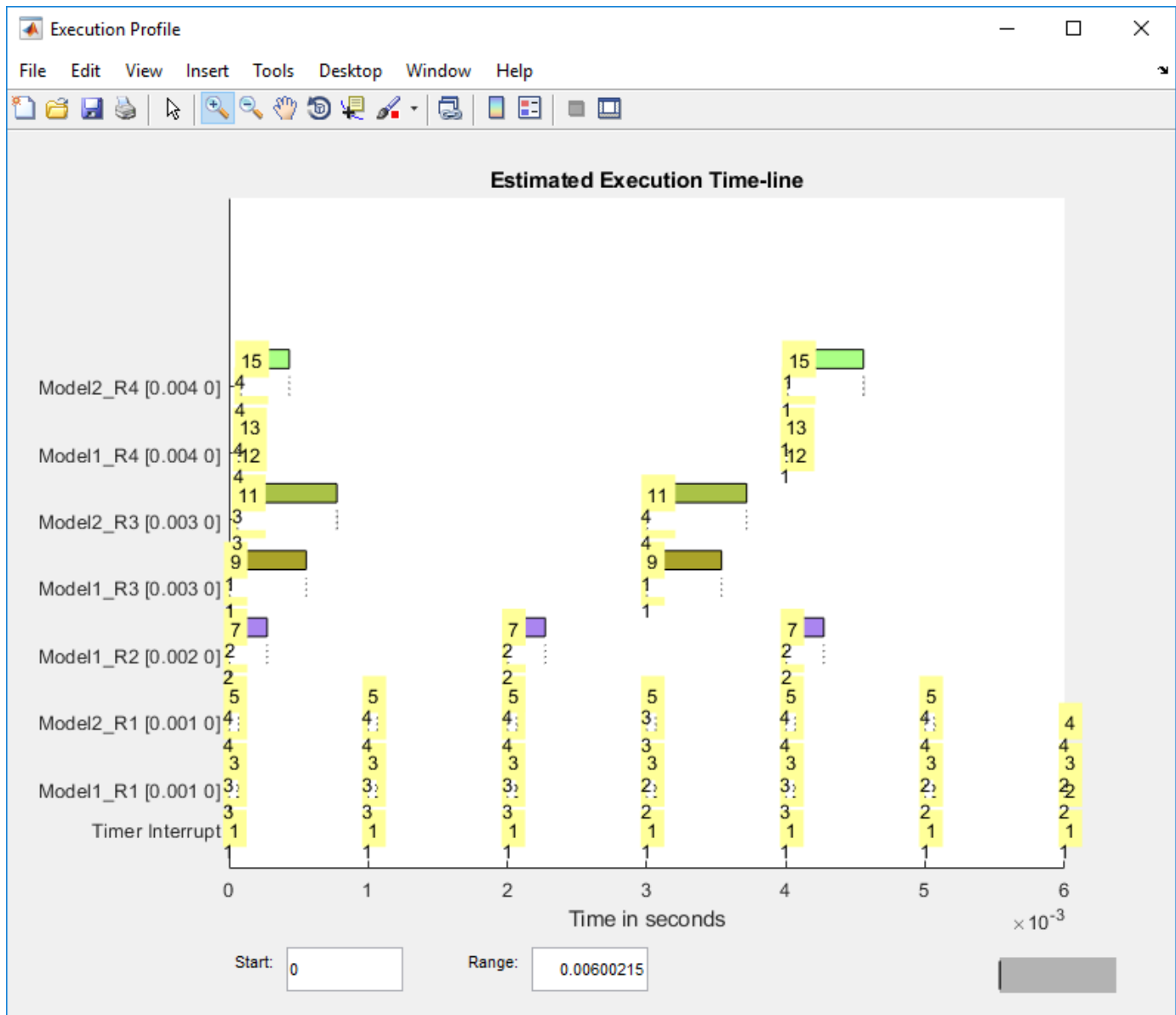
```
startProfiler(tg);
start(tg);
pause(1)
stopProfiler(tg);
stop(tg);
```

3. Display the profiler data.

```
profiler_data = getProfilerData(tg)
plot(profiler_data)
```

The Execution Profile plot shows the allocation of execution cycles across the four processors, indicated by the colored horizontal bars. The model sections are listed in the Code Execution Profiling Report. The cores are indicated by the numbers underneath the bars.

For more information about the time line in the execution profile plot, see [timeline](#).



The Code Execution Profiling Report displays model execution profile results for each task.

- To display the profile data for a section of the model, in the **Section** column, click the **Membrane** button next to the task.
- To display the TET data for the section in Simulation Data Inspector, click the **Plot time series data** button.
- To view the section in Simulink Editor, click the link next to the **Expand Tree** button.
- To view the lines of generated code corresponding to the section, click the **Expand Tree** button and then click the **View Source** button.

Code Execution Profiling Report
— □ ×

Code Execution Profiling Report for dxpcmds6t

The code execution profiling report provides metrics based on data collected from real-time simulation. Execution times are calculated from data recorded by instrumentation probes added to the generated code. See [Code Execution Profiling](#) for more information.

1. Summary

| | |
|------------------------------------|---|
| Total time | 1619431194 |
| Unit of time | ns |
| Command | report(, 'Units', 'Seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f'); |
| Timer frequency (ticks per second) | 1e+09 |
| Profiling data created | 26-Jun-2017 17:31:55 |

2. Profiled Sections of Code

| Section | Maximum Turnaround Time in ns | Average Turnaround Time in ns | Maximum Execution Time in ns | Average Execution Time in ns | Calls | |
|---|-------------------------------|-------------------------------|------------------------------|------------------------------|-------|--|
| Timer Interrupt | 4170 | 1496 | 4170 | 1496 | 2002 | |
| [+] Model1 R1 [0.001 0] | 59032 | 54435 | 59032 | 54435 | 2001 | |
| [+] Model2 R1 [0.001 0] | 65251 | 63273 | 65251 | 63273 | 2001 | |
| [+] Model1 R3 [0.003 0] | 555712 | 537251 | 555712 | 537251 | 667 | |
| [+] Model1 R2 [0.002 0] | 269707 | 268149 | 269707 | 268149 | 1001 | |
| [−] Model2 R3 [0.003 0] | 727574 | 713323 | 727574 | 713323 | 667 | |
| Model2 | 726716 | 712610 | 726716 | 712610 | 667 | |
| [+] Model1 R4 [0.004 0] | 12146 | 8165 | 12146 | 8165 | 501 | |
| [+] Model2 R4 [0.004 0] | 571241 | 547431 | 571241 | 547431 | 501 | |

3. Definitions

Execution Time: Time between start and end of code section, which excludes preemption time.

Turnaround Time: Time between start and end of code section, which includes preemption time.

OK Help

4. To investigate further improvements, see Improve Performance of Multirate Model.

See Also
timeline

Reduce Build Time for Simulink Real-Time Referenced Models

In a parallel computing environment, you can increase the speed of code generation and compilation for models containing large model reference hierarchies. Achieve the speed by building referenced models in parallel whenever conditions allow. For example, if you have Parallel Computing Toolbox software, code generation and compilation for each referenced model can be distributed across the cores of a multicore host computer. If you also have MATLAB Parallel Server™ software, you can distribute code generation and compilation for each referenced model across remote workers in your MATLAB Parallel Server configuration.

The Simulink Real-Time software allows you to build referenced models in parallel on a compute cluster. In this way, you can more quickly build and download real-time applications to the target computer.

The following procedure assumes that you have a functioning Simulink Real-Time installation on your development computer.

- 1 Identify a set of worker computers, which can be separate cores on your development computer or computers in a remote cluster running under Windows.
- 2 If you intend to use separate cores on the development computer, install Parallel Computing Toolbox on the development computer.
- 3 If you intend to use computers in a remote cluster:
 - a Install the following on each cluster computer:
 - MATLAB
 - Parallel Computing Toolbox
 - MATLAB Parallel Server
 - Simulink Real-Time
 - Build compiler

Install the same compiler and compiler version at the same location as on the development computer.
 - b Start and configure the remote cluster according to the instructions at www.mathworks.se/support/product/DM/installation/ver_current.
- 4 Run MATLAB on the development computer.
- 5 In MATLAB, call the `parpool` function to open a parallel pool on the cluster.
- 6 To configure the compiler for the remote workers as a group, call the `pctRunOnAll` function. For example:

```
pctRunOnAll('slrtsetCC(''VisualC'', '''')')
pctRunOnAll('mex -setup')
```

In this configuration, the development computer and the remote workers have installed a supported version of Microsoft Visual Studio. See Supported and Compatible Compilers - All Products.

- 7 From the top model of the model reference hierarchy, open the Configuration Parameters dialog box. Go to the **Model Referencing** pane and select the “Enable parallel model reference builds” (Simulink) option. This selection enables the parameter “MATLAB worker initialization for builds” (Simulink). For more information, see “Reduce Build Time for Referenced Models by Using Parallel Builds” (Simulink Coder).

8 Build and download your model.

See Also

`parpool` | `pctRunOnAll`

More About

- “Reduce Build Time for Referenced Models by Using Parallel Builds” (Simulink Coder)

Sample Time and Throughput in Real-Time Applications

After you design and debug the functionality of your model in Simulink, test and debug it as a real-time application. While testing your real-time application, you can encounter performance issues.

Real-Time Performance Factors

Real-time performance consists of *sample time* and *throughput*.

Sample time refers to the time during which the real-time application reads data into blocks and processes it. Physical systems have an inherent sample time (the Nyquist sample time) that is based on physical constraints. For example, when you use the brakes in a truck, the inertia of the truck limits how fast the road speed can change. A significant change requires about a second. Therefore, the speedometer does not need to sample the road speed more often than every tenth of a second.

If the data changes significantly between samples taken at the inherent sample time, sample times longer than that rate can miss those changes. If the data includes undesirable noise, sample times shorter than the inherent sample time can capture that noise.

Throughput refers to how much data the real-time application can process without a CPU overload in a given sample time. Throughput is limited by the resources that are available from the target computer. Sample times that are too short can overload the target computer CPU and stop execution.

For more information, see:

- “Sample Times in Systems” (Simulink)

Resources

The target computer system resources that affect a real-time application include:

- CPU cycles available on multicore systems
- Target computer RAM access speed
- RAM available for RAM disk
- Backplane I/O channel bandwidth and latency
- Disk storage bandwidth and latency

A multicore target computer can improve throughput and sample time. A multicore computer contains multiple CPUs, or cores, that share the processing load. In a four-core target computer, for example, the following tasks can happen simultaneously on different cores:

- Execute a referenced model
- Acquire data through an I/O channel
- Log results to a RAM disk
- Communicate with the development computer

The strategy that you use to improve throughput depends on your application system requirements.

| Application System Requirement | Hardware Capabilities | Modeling Style | Available Tools |
|--|--|---|--|
| Heavy sensor and effector I/O | Fast I/O channels | | Simulink Real-Time profiler |
| Heavy real-time computation | <ul style="list-style-type: none"> • Additional multicore processors • Faster multicore processors • Faster RAM speed | Polling mode | <ul style="list-style-type: none"> • Simulink Performance Advisor • Minimum sample time function |
| Reference models with different inherent rates | Multicore processors | <ul style="list-style-type: none"> • Rate transition blocks • Trade off deterministic data transfer for data transfer speed • Concurrent execution options • Reference model task mapping | Simulink Performance Advisor |
| Real-time applications connected by network | <ul style="list-style-type: none"> • Multiple target computers • Fast network switches | Multiple real-time applications that use network blocks for communication | <ul style="list-style-type: none"> • Simulink Real-Time profiler • Network analyzer |
| Data logging | <ul style="list-style-type: none"> • Large fast hard drive • Large RAM disk | <ul style="list-style-type: none"> • Selective marking of signals for capture • File scopes | Simulation Data Inspector in buffered mode |
| Low-level mechanical and electronic control | FPGA | | HDL Coder HDL Workflow Advisor |

Improving Performance with Concurrency

Whether you can use concurrency to improve real-time performance depends on the model. For example, a model that has heavy data traffic between referenced models is limited by data propagation and not by data processing. For more information, see:

- “Multicore Programming with Simulink” (Simulink)
- “Limitations with Multicore Programming in Simulink” (Simulink)

To use concurrency, first convert the blocks at the root level of your model into MATLAB System blocks or into models that are referenced with Model blocks. Do not use Subsystem blocks.

Simulink provides concurrency settings in the **Solver** pane, under **Additional options**:

- **Allow tasks to execute concurrently on target** — 'on' (default) or 'off'. When this parameter is 'on' (the default), the kernel allocates tasks to the next available CPU core. For most models, use the default value.

When **Allow tasks to execute concurrently on target** is 'off', the parameter **Treat each discrete rate as a separate task** is available. When **Treat each discrete rate as a separate task** is 'off', the real-time application executes in single-tasking mode. In single-tasking mode, the application does not take advantage of a multicore target computer.

In a future release, single-tasking mode will not work for multirate Simulink Real-Time models.

- **Enable explicit model partitioning for concurrent behavior** — 'on' or 'off' (default). This parameter is available only if **Allow tasks to execute concurrently on target** is 'on' and you click **Configure tasks**.

This scenario shows how to use the inherent sample time of a model and concurrency to improve the sample time and throughput of a model. As frequently happens during prototyping, the original version is a single-rate model. Using Simulink Performance Advisor and the profiler, this scenario iterates through these tasks:

- Eliminating CPU overloads while executing in the required sample time range
- Converting the single-rate model to multirate by using the design specification
- Improving multirate performance by using concurrency with implicit partitioning
- Refactoring a multirate model to reduce the CPU requirements of individual referenced models
- Improving multirate performance by using concurrency with explicit partitioning

At each stage, you view the allocation of single-rate and multirate models among the cores of a multicore target computer by using the Simulink Real-Time profiler functions.

Prerequisites

This scenario assumes that you can:

- 1 Open Simulink Real-Time Explorer.
- 2 Start the target computer.
- 3 Connect Explorer to the target computer.
- 4 Build and download a real-time application to the target computer.
- 5 Execute a real-time application on the target computer.

For more information, see Related Topics.

Single-Rate Model

You implemented the basic functionality as a single-rate model. To expedite tuning the sample time, you used variable T_s to define the base sample time for the constant blocks in the ref1 and ref2 referenced models.

You debugged the model at a sample time of $T_s = 1.0e-3$ s. To meet its real-time performance requirement, this model must achieve a base sample time in the range $1.0e-4 \leq T_s \leq 3.0e-4$ while running on a four-core target computer.

Test Against Requirement

To test the model, set its base sample time to the top of the required range, $3.0e-4$ s.

- 1 To open this model, open these files in sequence:

- a `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_single_rate_ref1')))`
- b `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_single_rate_ref2')))`
- c `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_single_rate')))`

To view the sample time legend, right-click in Simulink Editor and click **Sample Time Display > All**. For a single-rate model, the top-level sample time legend color applies to all referenced models.

- 2 Set $T_s = 3.0e-4$.
- 3 Build, download, and execute the real-time application.

The real-time application overloads the CPU. The target computer does not have enough CPU cycles to completely execute the model at the basic sample time.

Determine Minimum Sample Time

Because the CPU overloaded, you cannot take a baseline until you have determined the minimum sample time.

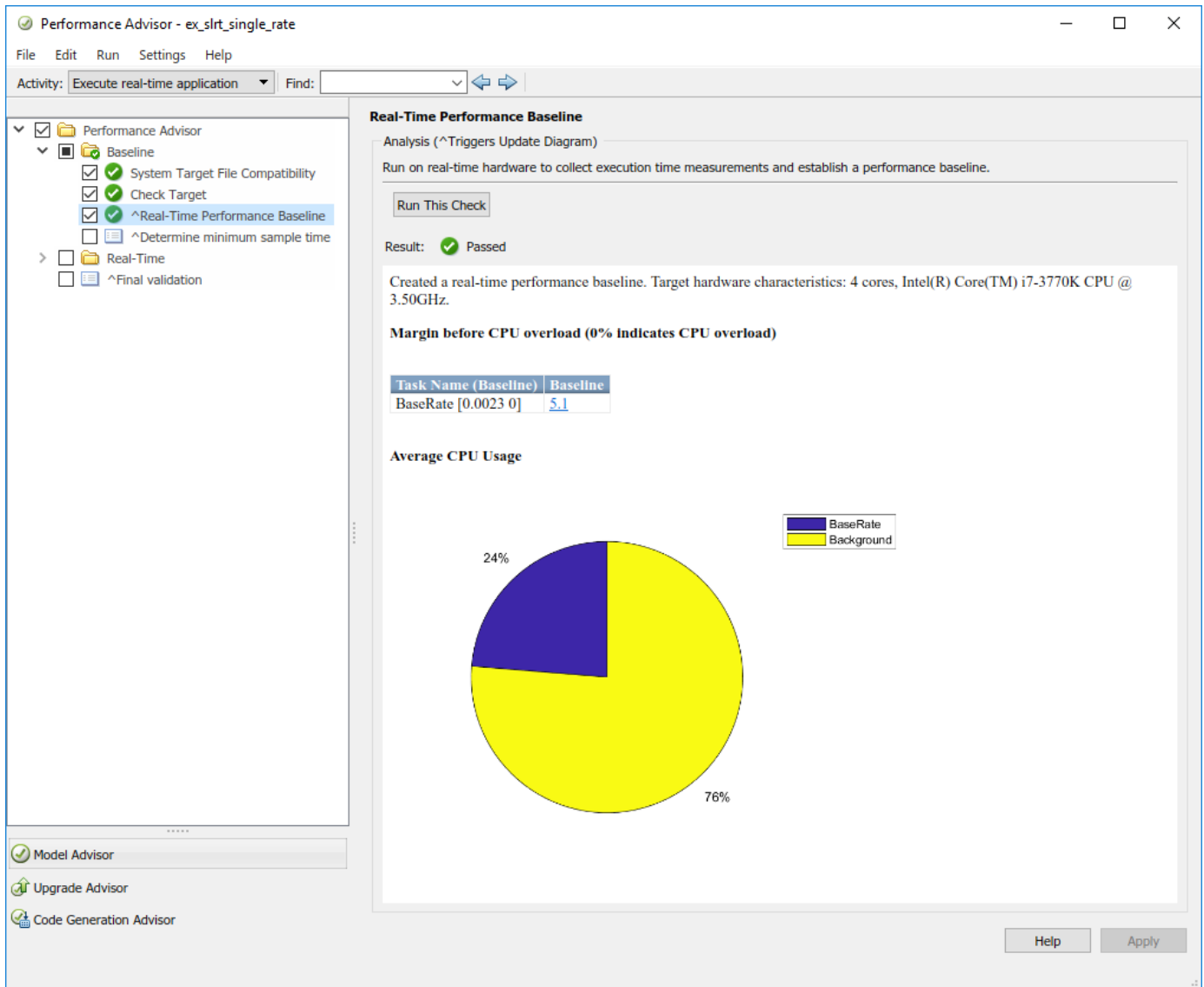
- 1 Open the Performance Advisor. On the **Debug** tab, click **Performance Advisor**.
- 2 Select the **Execute real-time execution** activity.
- 3 Select and run the baseline checks other than **Real-Time Performance Baseline**, including **Determine minimum sample time**.
- 4 Evaluate the smallest sample time this model can attain, about $2.2e-3$.
- 5 To avoid CPU overloads caused by random variations, set T_s to a value about 5% higher than that sample time, or $T_s = 2.3e-3$.

Determine Baseline

Using Performance Advisor, establish a baseline and evaluate whether improvement is possible for this version of the model.

- 1 In Performance Advisor, run **Real-Time Performance Baseline**.

The run succeeds and produces a pie chart.



This chart shows two usage allocations, **BaseRate** and **Background**. The **BaseRate** allocation shows the execution of the single-rate real-time application as one task. The **Background** allocation shows the execution of the kernel tasks, such as accessing the target computer disk for data logging or communicating between the development and target computers.

This example uses a four-core target computer, but the real-time application only uses a quarter of the available CPU cycles. **BaseRate** has a low margin before CPU overload, about 5%. To improve performance, the real-time application must use more of the available CPU resources.

- As a best practice, run all of the **Real-Time** checks except **Simscape checks**.

The **Real-Time** checks pass. This version of the model cannot be improved further.

Evaluate Task Allocation

Evaluate the allocation of tasks across the four cores.

- 1 In the Command Window, run the profiler:

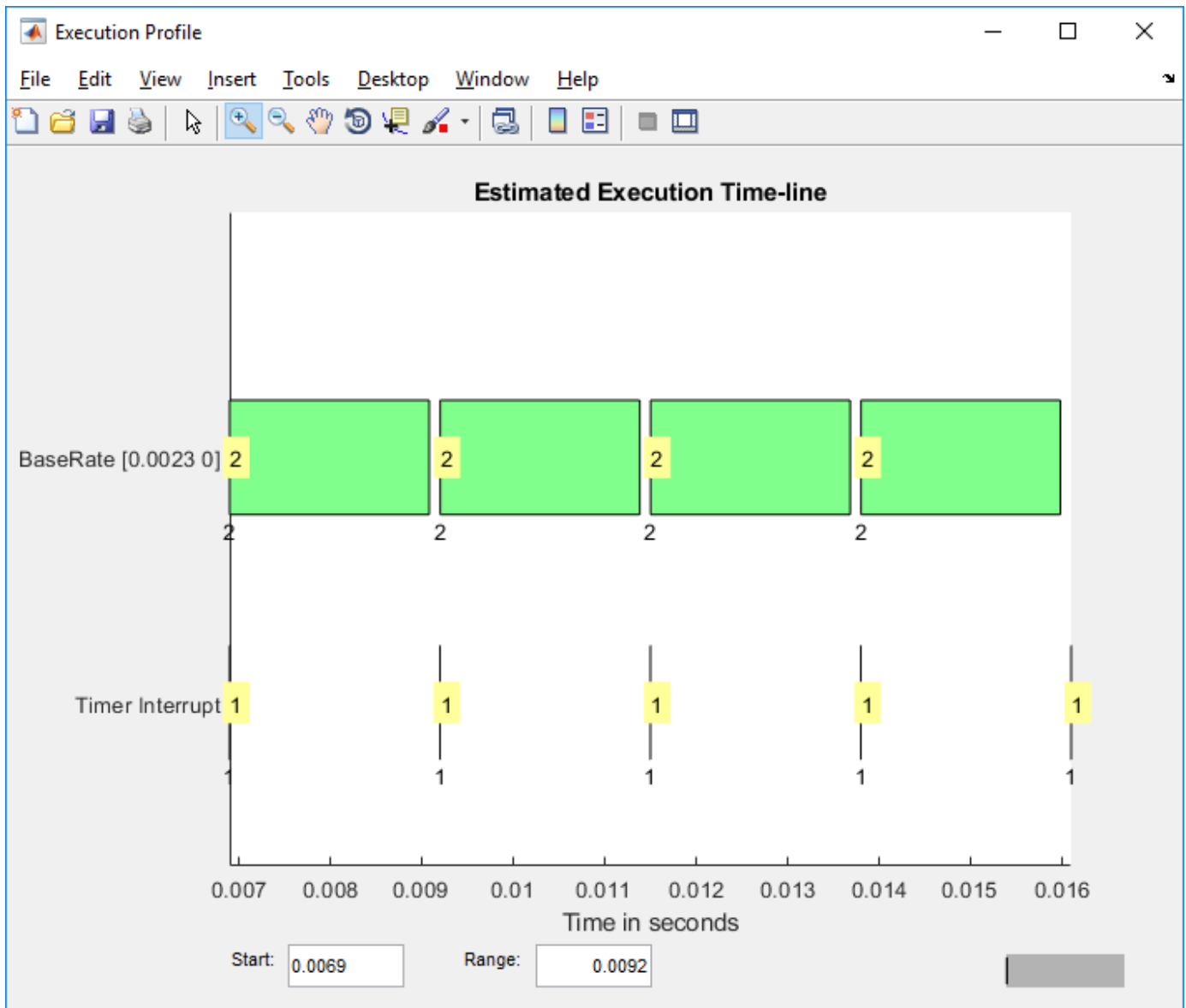
```
tg = slrt;
startProfiler(tg);
start(tg);
stop(tg);
```

The stop function also calls the stopProfiler function.

- 2 Retrieve the profiler data and display the results:

```
profiler_data = getProfilerData(tg);
plot(profiler_data);
```

To skip initialization, start the display at $3 \cdot T_s$. To show a representative example of concurrency, use a range of $4 \cdot T_s$.



In the profiler display, the highlighted numbers within each task bar give the task number. Task number 2 shows how much of the available time is being used by the BaseRate task. Task number 1 is the timer interrupt, part of the background tasks.

The labels under the task bars give the CPU core on which each task runs. Because this model is a single-rate model, the referenced model tasks run one after the other on core 2 at the same rate after each timer interrupt.

The execution bar at one timer interrupt almost fills the time until the next timer interrupt. If the execution bar at one interrupt overlaps with the execution bar at the next, the target computer CPU overloads and stops execution.

Multirate Model: Concurrency On, Implicit Partitioning

At this stage of the optimization process, the current value of $T_s = 2.3e-3$, which is outside the required range of $1.0e-4 \leq T_s \leq 3.0e-4$.

To improve the sample time of the real-time application, start with the inherent rates of the model. After converting the single-rate model to a multirate model, you can turn on concurrency with implicit partitioning.

Convert to Multirate Model

From the design specification, determine which parts of the model can run at lower rates and which cannot.

- 1 Specify rates for parts of the model.

As a best practice, specify rates that are multiples of a single base rate. In this model, the valid rates are multiples of T_s : T_s , $2*T_s$, $3*T_s$, and $4*T_s$.

- 2 In the original model, Ref1/Out4 connects directly to Ref2/In1. Because Ref1/Out4 and Ref2/In1 have different rates, add a Rate Transition block to Ref1.

- 3 Configure the Rate Transition block:

- Set the **Ensure data integrity during data transfer** parameter.
- Clear the **Ensure deterministic data transfer (maximum delay)** parameter.

- 4 To open this model, open these files in sequence:

- a `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_multirate_ref1')))`
- b `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_multirate_ref2')))`
- c `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_multirate')))`

For this model, the sample time legend colors for the top level also apply to the Ref1 referenced model. A separate set of sample time legend colors appears for the Ref2 referenced model.

Configure Implicit Partitioning

To configure implicit partitioning, turn on task-level concurrency and take the defaults.

- 1 Open the Configuration Parameters for the top-level model. On the **Real-Time** tab, click **Hardware Settings**.

- 2 Select **Solver > Allow tasks to execute concurrently on target.**
- 3 On the **Simulation** tab, click **Prepare > Update Model.**

Test Against Requirement

To test the model, set its base sample time to the top of the required range, $3.0e-4$ s.

- 1 Set $T_s = 3.0e-4$.
- 2 Build, download, and execute the real-time application.

The real-time application overloads the CPU. The target computer does not have enough CPU cycles to completely execute the model at the basic sample time.

Determine Minimum Sample Time

Because the CPU overloaded, you cannot take a baseline until you have determined the minimum sample time.

- 1 Open the Performance Advisor. On the **Debug** tab, click **Performance Advisor.**
- 2 Select the **Execute real-time execution** activity.
- 3 Select and run the baseline checks other than **Real-Time Performance Baseline**, including **Determine minimum sample time.**
- 4 Evaluate the smallest sample time this model can attain, about $4.2e-4$.
- 5 To avoid CPU overloads caused by random variations, set T_s to a value about 5% higher than that sample time, or $T_s = 4.4e-4$.

Determine Baseline

Using Performance Advisor, establish a baseline and evaluate whether improvement is possible for this version of the model.

- 1 To take a baseline for optimization, run **Real-Time Performance Baseline.**

The run succeeds and produces a pie chart.

Performance Advisor - ex_slrt_multirate

File Edit Run Settings Help

Activity: Execute real-time application Find: []

Real-Time Performance Baseline

Analysis (^Triggers Update Diagram)

Run on real-time hardware to collect execution time measurements and establish a performance baseline.

Run This Check

Result: ✔ Passed

Created a real-time performance baseline. Target hardware characteristics: 4 cores, Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz.

Margin before CPU overload (0% indicates CPU overload)

| Task Name (Baseline) | Baseline |
|----------------------|----------|
| BaseRate [0.00044 0] | 73.5 |
| SubRate1 [0.00088 0] | 69.2 |
| SubRate3 [0.00176 0] | 67.4 |
| SubRate2 [0.00132 0] | 4.89 |

Average CPU Usage

Legend:

- BaseRate
- SubRate1
- SubRate3
- SubRate2
- Background

Model Advisor
Upgrade Advisor
Code Generation Advisor

Help Apply

The CPU core usage has improved, but the real-time application only uses half of the available CPU cycles. Also, `SubRate2` has a low margin before CPU overload, about 5%. The real-time application needs better load balancing to improve the base sample time and to make its execution more likely to succeed.

- 2 As a best practice, run all of the **Real-Time** checks except **Simscape checks**.

The **Real-Time** checks pass. This version of the model cannot be improved further.

Evaluate Task Allocation

Evaluate the allocation of tasks across the four cores.

- 1 In the Command Window, run the profiler:

```
tg = slrt;
startProfiler(tg);
```

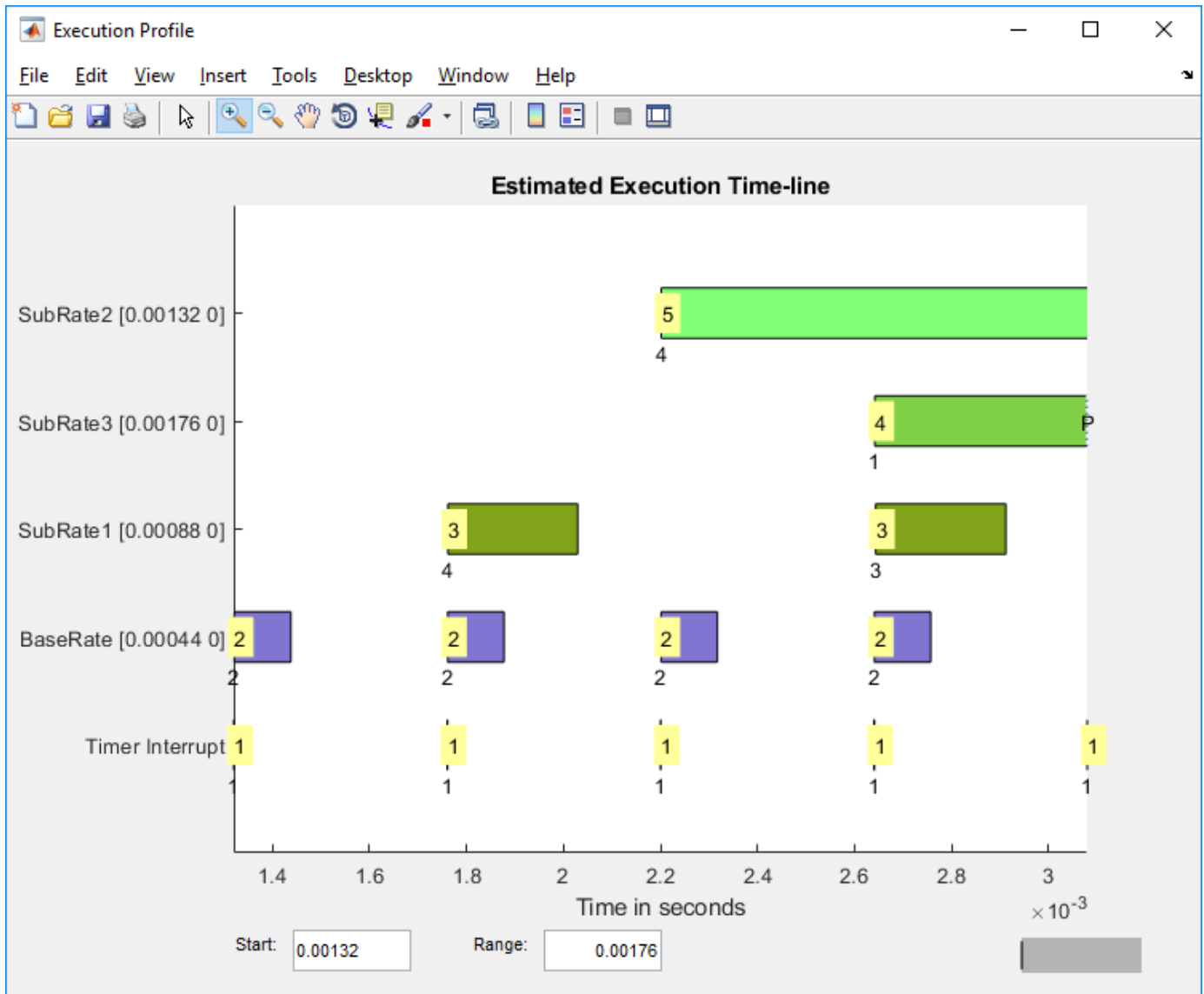


```
start(tg);
stop(tg);
```

- Retrieve the profiler data and display the results:

```
profiler_data = getProfilerData(tg);
plot(profiler_data);
```

To skip initialization, start the display at $3 \cdot T_s$. To show a representative example of concurrency, use a range of $4 \cdot T_s$.



The execution bars for SubRate2, the task with the largest CPU requirement, almost overlap. Concurrency is in full operation as of time tick 1.32×10^{-3} .

Refactored Multirate Model: Concurrency On, Explicit Partitioning

At this stage of the optimization process, the current value of $T_s = 4.4 \times 10^{-4}$, which is still outside the required range of $1.0 \times 10^{-4} \leq T_s \leq 3.0 \times 10^{-4}$.

You can improve the performance of your real-time application by explicitly balancing the load of the different processing nodes in the multicore target computer. This process involves iteratively refactoring the model, moving tasks between different processing nodes, and testing the result. For more information, see “Concepts in Multicore Programming” (Simulink).

Before refactoring a model, note which tasks of a system depend on the output of other tasks. The data dependency between tasks determines their execution order within a time step. Two or more partitions containing data dependencies in a cycle creates a data dependency loop, also known as an algebraic loop. To detect these loops, in the **Diagnostics** pane, set the **Algebraic loop** parameter to error. Simulink identifies algebraic loops during execution, displays an error message, and highlights the portion of the block diagram that comprises the loop. Remove these loops from your model. For more information, see “Algebraic loop” (Simulink).

Refactor Model

In this scenario, the multirate model consists of two referenced models, each containing many signals to process during each sample time. With implicit partitioning, each referenced model task is assigned to a core. To improve interleaving among CPU cores, divide each referenced model in half. Each half contains half the number of signals in the original referenced model. This configuration produces the same number of referenced models as cores with each referenced model having smaller CPU requirements than the original.

- 1 Split the Ref1 referenced models into two referenced models, Ref1A and Ref1B. Each block has half the number of signals as Ref1.
- 2 Split the Ref2 referenced models into two referenced models, Ref2A and Ref2B. Each block has half the number of signals as Ref2.
- 3 To open this model, open these files in sequence:
 - a `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_multirate_partition_ref1A')))`
 - b `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_multirate_partition_ref1B')))`
 - c `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_multirate_partition_ref2A')))`
 - d `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_multirate_partition_ref2B')))`
 - e `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_multirate_partition')))`

Configure Explicit Partitioning

To configure explicit partitioning, turn on task-level concurrency and explicitly configure the tasks for each referenced model. Explicit partitioning can increase the CPU interleaving of real-time tasks.

- 1 Open the Configuration Parameters for the top-level model. On the **Real-Time** tab, click **Hardware Settings**.
- 2 Select **Solver > Allow tasks to execute concurrently on target**.
- 3 Click **Configure Tasks**, and then select **Enable explicit model partitioning for concurrent behavior**.
- 4 Under **Concurrent Execution > Tasks and Mapping**, open **CPU > Periodic**.

- 5 Create periodic tasks for each rate in each referenced model. Name the tasks Model1_R1, Model1_R2, and so on.

You use a periodic trigger to represent periodic interrupt sources, such as a timer. The periodicity of the trigger is either the base rate of the task or the period of the trigger. See “Concepts in Multicore Programming” (Simulink).

- 6 Assign each periodic task to the corresponding rate in each referenced model.
 - Model1, Model3 — Four tasks of rates T_s , $2*T_s$, $3*T_s$, and $4*T_s$.
 - Model2, Model4 — Three tasks of rates T_s , $3*T_s$, and $4*T_s$

At the end of this process, the Concurrent Execution window looks like the figure.

The screenshot shows the 'Concurrent Execution: ex_slrt_multirate_partition (Active)' window. The left pane shows a tree view of the execution environment, including 'Concurrent Execution', 'Data Transfer', 'Tasks and Mapping', 'CPU', and 'Periodic' tasks. The right pane, titled 'Map blocks to tasks', contains a table with columns: Name, TriggerType, Period, and Autogenerated. The table lists tasks for four blocks: Model1, Model2, Model3, and Model4.

| Name | TriggerType | Period | Autogenerated |
|--------------------|-------------|---------|---------------|
| Block: Model1 | | | |
| Periodic:Model1_R1 | Periodic | T_s | No |
| Periodic:Model1_R2 | Periodic | $2*T_s$ | No |
| Periodic:Model1_R3 | Periodic | $3*T_s$ | No |
| Periodic:Model1_R4 | Periodic | $4*T_s$ | No |
| Block: Model2 | | | |
| Periodic:Model2_R1 | Periodic | T_s | No |
| Periodic:Model2_R3 | Periodic | $3*T_s$ | No |
| Periodic:Model2_R4 | Periodic | $4*T_s$ | No |
| Block: Model3 | | | |
| Periodic:Model3_R1 | Periodic | T_s | No |
| Periodic:Model3_R2 | Periodic | $2*T_s$ | No |
| Periodic:Model3_R3 | Periodic | $3*T_s$ | No |
| Periodic:Model3_R4 | Periodic | $4*T_s$ | No |
| Block: Model4 | | | |
| Periodic:Model4_R1 | Periodic | T_s | No |
| Periodic:Model4_R3 | Periodic | $3*T_s$ | No |
| Periodic:Model4_R4 | Periodic | $4*T_s$ | No |

- 7 On the **Simulation** tab, click **Prepare > Update Model**.

For this model, the sample time legend colors for the top level also apply to the Ref1A and Ref1B referenced models. A separate set of sample time legend colors appears for the Ref2A and Ref2B referenced models.

Test Against Requirement

To test the model, set its base sample time to the top of the required range, $3.0e-4$ s.

- 1 Set $T_s = 3.0e-4$.
- 2 Build, download, and execute the real-time application.

The real-time application runs. Your model has met the basic sample-time requirement.

Determine Minimum Sample Time

To evaluate where this version of the model falls in the sample-time range and how much margin it has:

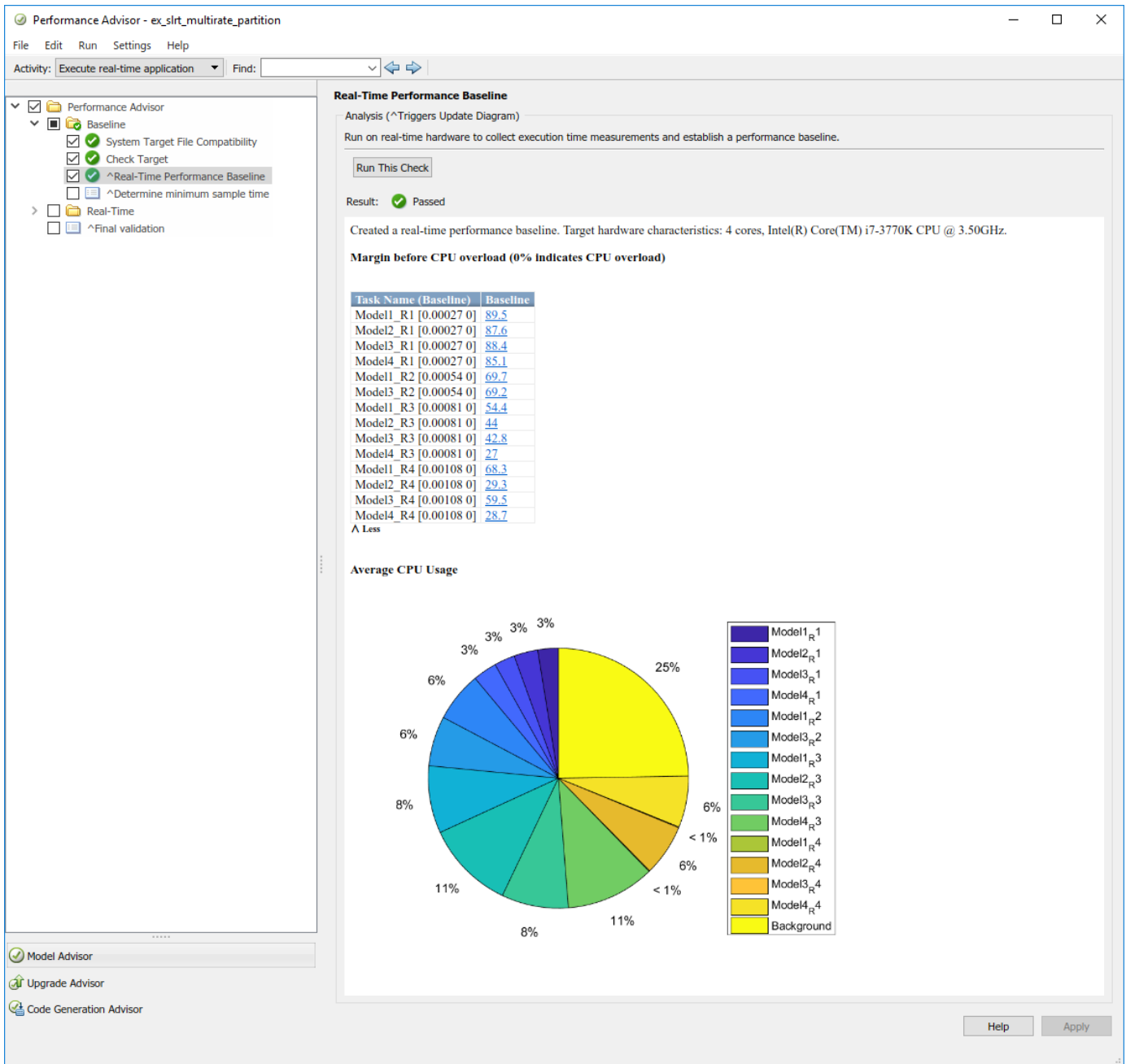
- 1 Open the Performance Advisor. On the **Debug** tab, click **Performance Advisor**.
- 2 Select the **Execute real-time execution** activity.
- 3 Select and run the baseline checks other than **Real-Time Performance Baseline**, including **Determine minimum sample time**. You cannot take a baseline until you have determined the minimum sample time.
- 4 Evaluate the smallest sample time this model can attain, about $2.6e-4$.
- 5 To avoid CPU overloads caused by random variations, set T_s to a value about 5% higher than that sample time, or $T_s = 2.7e-4$.

Determine Baseline

Using Performance Advisor, establish a baseline and evaluate whether improvement is possible for this version of the model.

- 1 To take a baseline, run **Real-Time Performance Baseline**.

The run succeeds and produces output like the figure.



At the lowest achievable sample time, this real-time application uses three-quarters of the available CPU cycles. The smallest margin before CPU overload is about 27%, which is an improvement over the 5% margin in the previous version.

- As a best practice, run all of the **Real-Time** checks except **Simscape checks**.

The **Real-Time** checks pass. This version of the model cannot be improved further.

Evaluate Task Allocation

Evaluate the allocation of tasks across the four cores.

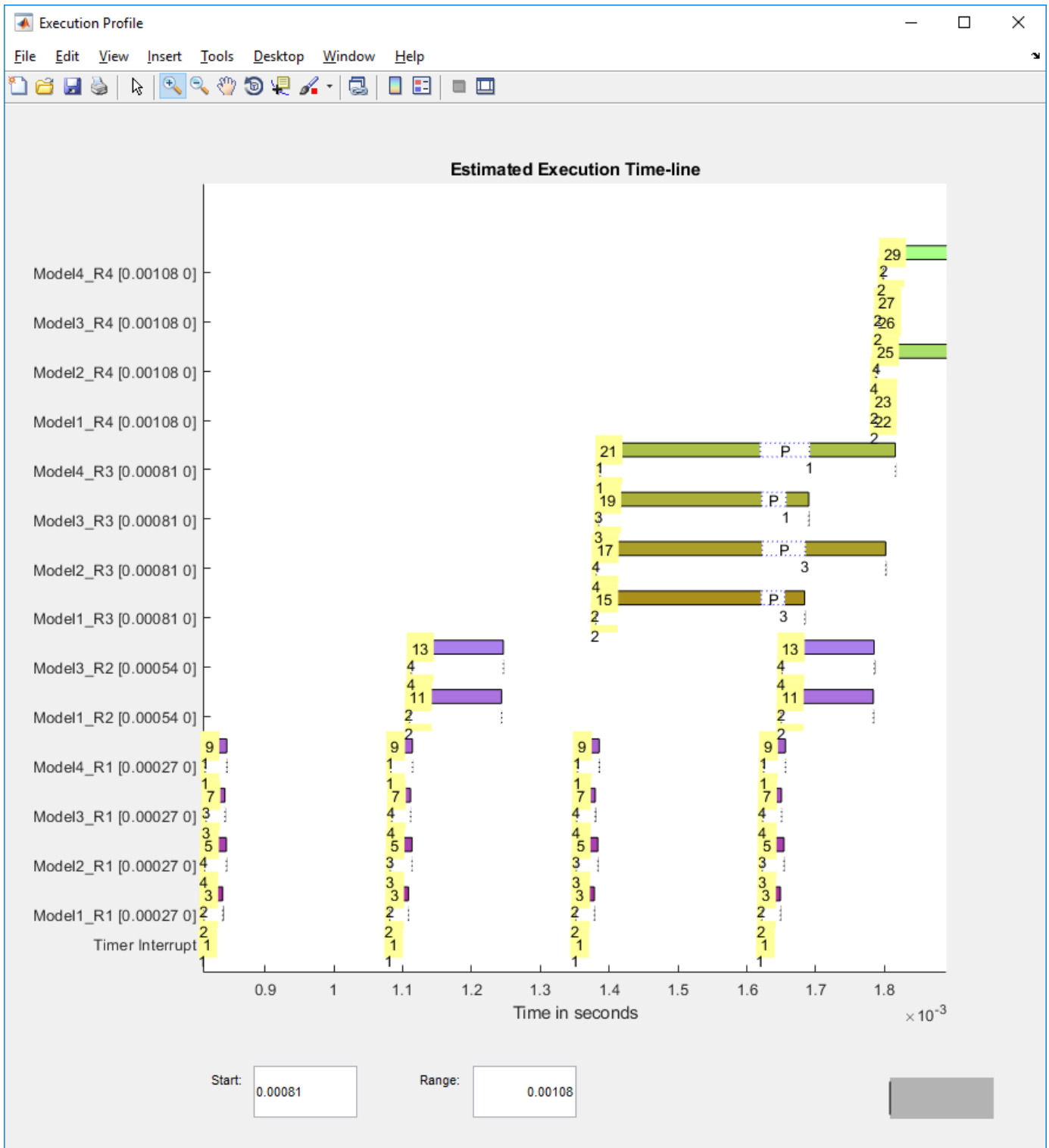
- 1 In the Command Window, run the profiler:

```
tg = slrt;  
startProfiler(tg);  
start(tg);  
stop(tg);
```

- 2 Retrieve the profiler data and display the results:

```
profiler_data = getProfilerData(tg);  
plot(profiler_data);
```

To skip initialization, start the display at $3 \cdot T_s$. To show a representative example of concurrency, use a range of $4 \cdot T_s$.



The Model*_R3 tasks start running on all four processors, but Model*_R1 tasks preempt them. The overhead of preemption limits the performance improvement that you can achieve by using concurrency alone.

Additional Optimizations

In the model scenario, the change that produced the greatest improvement was going from single-rate to multirate execution with the default task mapping (over 5X improvement). The other optimization produced less improvement (1.5X), but was required to reach the required sample time of $1.0e-4 \leq Ts \leq 3.0e-4$.

| Optimization | Achievable Sample Time T_s |
|--|------------------------------|
| Single-rate | 2.3e-3 |
| Multirate, implicit task mapping | 4.4e-4 |
| Partitioned multirate, explicit task mapping | 2.7e-4 |

To gain more improvement, consider the following optimizations.

Isolated Rate Transitions

If a multirate model contains many rate-transition blocks covering a few overlapping rates, consider extracting each similar rate transition into a new referenced model. You can then set the **Enable explicit model partitioning for concurrent behavior** parameter and create an explicit periodic task mapping for the new referenced models. If a referenced model does not contain a block with the base-rate sample time, add a separate base-rate task to the mapping table for that model.

For this model, factoring out rate transitions provides only a small improvement. To open `ex_slrt_multirate_refactor`, open these files in sequence:

- `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_multirate_refactor_ref1A')))`
- `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_multirate_refactor_ref2A')))`
- `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_multirate_refactor_ref3A')))`
- `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_multirate_refactor_ref1B')))`
- `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_multirate_refactor_ref2B')))`
- `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_multirate_refactor_ref3B')))`
- `open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_multirate_refactor')))`

Explicit Partitioning of Single-Rate Model

If the model is a single-rate model with a high computational requirement for each referenced model without data dependencies between them, consider setting the **Enable explicit model partitioning for concurrent behavior** parameter. You can then create an explicit periodic task mapping for each of the referenced models.

The improvement that you can achieve by explicitly mapping the tasks of a single-rate model is limited by the number of cores. For example, if you have four cores and the tasks run at a single rate, the most you can achieve is a 4X improvement in CPU usage.

Function Execution Optimization

To find additional optimizations, consider running the Simulink Real-Time profiler with function execution time logging enabled (see “Profiling and Optimization”). The profiler provides detailed, low-level information about the CPU tasks. You can then identify bottleneck blocks and replace or improve them.

FPGA Coprocessing

In cases where you cannot meet your system requirements by other optimization methods, consider embedding the crucial algorithms in an FPGA by using HDL Coder HDL Workflow Advisor.

See Also

`EnableProfiler` | `ProfilerData` | `RateTransition` | `SimulinkRealTime`. - `utils.minimumSampleTime` | `getProfilerData` | `resetProfiler` | `startProfiler` | `stopProfiler`

More About

- “Simulink Real-Time Performance Advisor Checks”
- “Get Started with Simulink Real-Time”
- “Improve Simulation Performance Using Performance Advisor” (Simulink)
- “Multicore Programming with Simulink” (Simulink)
- “Sample Times in Systems” (Simulink)
- “Limitations with Multicore Programming in Simulink” (Simulink)
- “Concepts in Multicore Programming” (Simulink)
- “Improve Performance of Multirate Model” on page 10-2
- “Profiling and Optimization”
- “FPGA Subsystem Configuration”
- “Algebraic loop” (Simulink)

Execution with MATLAB Scripts

Real-Time Applications and Scopes in the MATLAB Interface

- “Real-Time Application Objects” on page 11-2
- “Real-Time Scope Objects” on page 11-5
- “Acquire Signal Data with File Scopes” on page 11-9
- “Acquire Signal Data into Dynamically Named Files” on page 11-10
- “Scope Trigger Configuration” on page 11-12
- “Pretriggering and Posttriggering of Scopes” on page 11-13
- “Trigger One Scope with Another Scope” on page 11-15
- “Minimize Data Gaps with Two Scopes” on page 11-22

Real-Time Application Objects

The Simulink Real-Time software uses a `SimulinkRealTime.target` object to represent the target kernel and your real-time application. Use real-time application object functions to run and control real-time applications on the target computer with scope objects to collect signal data.

An understanding of the real-time application object properties and functions helps you to control and test your real-time application on the target computer.

A real-time application object on the development computer represents the interface to a real-time application and the kernel on the target computer. You use real-time application objects to run and control the real-time application.

When you change a real-time application object property on the development computer, information is exchanged with the target computer and the real-time application.

To create a real-time application object:

- 1 Build a real-time application. The Simulink Real-Time software creates a real-time application object during the build process.
- 2 Use the real-time application object function `SimulinkRealTime.target`. In the MATLAB Command Window, type:

```
tg = SimulinkRealTime.target
```

A `SimulinkRealTime.target` object has properties and functions specific to that object. The real-time application object functions allow you to control a real-time application on the target computer from the development computer. You enter real-time application object functions in the MATLAB window on the development computer, or you can use MATLAB code scripts. To access the help for these functions from the command line, use the syntax:

```
doc SimulinkRealTime.target/function_name
```

If you want to control the real-time application from the target computer, use target computer commands (see “Control Real-Time Application at Target Computer Command Line” on page 9-2).

Create Real-Time Application Objects

To create a real-time application object:

- 1 Build a real-time application. The Simulink Real-Time software creates a real-time application object during the build process.
- 2 To create a specific real-time application object, or to create multiple real-time application objects in your system, use the real-time application object function `SimulinkRealTime.target` with arguments. For example, to create a real-time application object for target `TargetPC1`, in the MATLAB Command Window, type:

```
tg = SimulinkRealTime.target('TargetPC1')
```

The resulting real-time application object is `tg`.

Using this function clarifies which application object is associated with a particular target computer.

- 3 To check a connection between development and target computers, use the target function `ping`. For example, type:

```
ping(tg)
```

- 4 To create a real-time application object for the default target computer, use the creation function `SimulinkRealTime.target` without arguments. For example, in the MATLAB Command Window, type:

```
tg = SimulinkRealTime.target
```

The resulting real-time application object is `tg`.

Note If you use `SimulinkRealTime.target` without arguments to create a real-time application object, use Simulink Real-Time Explorer to configure your target computer. Doing so clarifies which real-time application object is associated with a particular target computer.

Display Application Object Properties

To monitor a real-time application, list the real-time application object properties. The properties include the execution time and the average task execution time.

After you build a real-time application and real-time application object from a Simulink model, you can list the real-time application object properties. This procedure uses the default real-time application object name `tg` as an example.

- 1 In the MATLAB window, type:

```
tg = slrt;
```

The current real-time application properties are uploaded to the development computer. MATLAB displays a list of the real-time application object properties with the updated values.

The real-time application object properties for `TimeLog`, `StateLog`, `OutputLog`, and `TETLog` are not yet updated.

- 2 Type:

```
start(tg)
```

The `Status` property changes from `stopped` to `running`. The log properties change to `Acquiring`.

For a list of real-time application object properties with a description, see `SimulinkRealTime.target`.

Set Real-Time Application Object Property Values

You can change a real-time application object property by using the Simulink Real-Time dot notation on the development computer. (For limitations on target property changes to sample times, see “Target Computer Configuration and Control Methods”.)

With the Simulink Real-Time software, you can use object property syntax to change the real-time application object properties.

```
target_object.property_name = new_property_value
```

For example, to change the stop time for the real-time application running on target `tg`, in the MATLAB window, type:

```
tg = slrt;  
tg.StopTime = 1000
```

When you change a real-time application object property, the new property value is downloaded to the target computer. The Simulink Real-Time kernel then receives the information and changes the behavior of the real-time application.

To get a list of the writable properties, type `target_object`. The build process assigns the default name of the real-time application object to `tg`.

Get Real-Time Application Object Property Values

You can list a property value in the MATLAB window or assign that value to a MATLAB variable. With the Simulink Real-Time software, you can use object property syntax.

```
target_object.property_name
```

For example, to access the stop time for the real-time application running on target `tg`, in the MATLAB window, type:

```
tg = slrt;  
endrun = tg.StopTime
```

To get a list of readable properties, type `target_object`. Without assignment to a variable, the property values are listed in the MATLAB window.

Signals are not real-time application object properties. To get the value of the Integrator1 signal from the model `xpcosc`, in the MATLAB window, type:

```
outputvalue = getsignal(tg, 0)
```

0 is the signal index.

Note Function names are case-sensitive. Type the entire name. Property names are not case-sensitive. You do not need to type the entire name, as long as the characters that you do type are unique for the property.

Use Real-Time Application Object Functions

To run a real-time application object function, use the `function_name(target_object, argument_list)` syntax.

Unlike properties, for which partial but unambiguous names are permitted, you must enter function names in full, in lowercase. For example, to add a target scope with a scope index of 1, in the MATLAB window, type:

```
tg = slrt;  
addscope(tg, 'target', 1)
```


Real-Time Scope Objects

The Simulink Real-Time software uses scope objects to represent scopes on the target computer. Use scope object functions to view and collect signal data.

The Simulink Real-Time software uses scopes and scope objects as an alternative to using Simulink scopes and external mode. A scope can exist as part of a Simulink model system or outside a model system.

- A scope that is part of a Simulink model system is a Scope block. You add a Simulink Real-Time Scope block to the model, build a real-time application from that model, and download that application to the target computer.
- A scope that is outside a model is not a Scope block. For example, if you create a scope with the `addscope` function, that scope is not defined within the model. After the model has been downloaded and initialized, you add this scope to the model.

This difference affects when and how the scope executes to acquire data.

Scope blocks inherit sample times. A Scope block in the root model or a normal subsystem executes at the sample time of its input signals. A Scope block in a conditionally executed (triggered/enabled) subsystem executes whenever the containing subsystem executes. In the latter case, the scope can acquire samples at irregular intervals.

Note If you display multiple scopes on the target computer and these scopes use different sample times, the time scale for all scopes is set by the scope with the fastest sampling time.

A scope that is not part of a model executes at the base sample time of the model. For signals with a sample time longer than the base sample time, it acquires repeated identical samples. For example, assume that the model base sample time is `0.001` and that you dynamically add to the scope a signal whose sample time is `0.005`. The scope acquires five identical samples for this signal at each signal sample time.

Understanding the structure of scope objects helps you to use the MATLAB command-line interface to view and collect signal data. A scope object on the development computer represents a scope on the target computer. You use scope objects to observe the signals from your real-time application during a real-time run or analyze the data after the run is finished.

To create a scope object:

- Add a Simulink Real-Time Scope block to your Simulink model. To determine the scope type, set the **Scope type** parameter. To create a scope on the target computer, build and download the model. Use the real-time application object function `getscope` to create a scope object on the development computer.
- Build and download a model. Use the real-time application object function `addscope` to create a scope on the development computer. To determine the scope type, pass one of the following values as input parameter: `target`, `host`, or `file`.

Upon creation, the Simulink Real-Time software assigns the required scope object class for the scope type: `SimulinkRealTime.targetScope`, `SimulinkRealTime.hostScope`, or `SimulinkRealTime.fileScope`.

A scope object has properties and functions specific to its scope type, as well as properties and functions in common with the other scopes. The scope object functions allow you to control scopes on your target computer.

To control the real-time application from a target computer keyboard, use target computer commands (see “Control Real-Time Application at Target Computer Command Line” on page 9-2).

Display Scope Object Properties for One Scope

To list the properties of a single scope object, `sc1`, in the MATLAB window, type:

```
tg = slrt;  
sc1 = getscope(tg,1)
```

MATLAB creates the scope object `sc1` from a previously created scope.

The current scope properties are uploaded to the development computer. MATLAB displays a list of the scope object properties with the updated values. Because `sc1` is a vector with a single element, you could also type `sc1(1)` or `sc1([1])`.

Note Only scopes of type `host` store data in the properties `scope_object.Time` and `scope_object.Data`.

For a list of real-time application object properties with a description, see the target function `SimulinkRealTime.target`.

Display Scope Object Properties for Multiple Scopes

To list the properties of the current scope objects associated with the real-time application object `tg`, in the MATLAB window, type:

```
tg = slrt;  
getscope(tg)
```

The `getscope` function supports vector arguments. For example, to list the first and third scopes, type:

```
getscope(tg,[1,3])
```

To assign a list of current scopes to a variable, type:

```
allscopes = getscope(tg)
```

For a list of real-time application object properties, see the target function `SimulinkRealTime.target`.

Set Scope Property Values

With the Simulink Real-Time software, you can use object property syntax to set a single scope object property.

```
scope_object.property_name = new_property_value
```

For example, to change the trigger mode for scope 1, in the MATLAB window, type:

```
tg = slrt;
sc1 = getscope(tg, 1);
sc1.triggermode = 'signal'
```

You cannot use dot notation to set vector object properties. To assign a property value to a vector of scopes, use the `set` function. For example, assume that you have two scopes, 1 and 2. First assign a vector containing these scopes to the variable `sc12`:

```
sc12 = getscope(tg, [1,2]);
```

To set the `NumSamples` property of these scopes to 300, type:

```
set(sc12, 'NumSamples', 300);
```

To get a list of the writable properties, type `scope_object`.

Note

- You cannot set a property of a vector of scopes to a vector of property values. For example, you cannot set property `NumSamples` of vector `sc12` to `[100,200]`.
 - Function names are case-sensitive. Type the entire name. Property names are not case-sensitive. You do not need to type the entire name, as long as the characters that you do type are unique for the property.
-

Get Scope Property Values

You can list a property value in the MATLAB window or assign that value to a MATLAB variable. With the Simulink Real-Time software, you can use object property syntax to get scope property values.

```
scope_object_vector(index_vector).property_name
```

For example, to get the number of samples from scope 1, in the MATLAB window, type:

```
tg = slrt;
sc1 = getscope(tg, 1);
sc1.NumSamples
```

To get the values of vector object properties set using the `set` function, you can use the corresponding `get` function. For example, assume that you have two scopes, 1 and 2, with a `NumSamples` property of 300.

First assign a vector containing these scopes to the variable `sc12`.

```
sc12 = getscope(tg, [1,2]);
```

To get the value of `NumSamples` for these scopes, type:

```
get(sc12, 'NumSamples')
```

You get a result like:

```
ans =  
    [300]  
    [300]
```

Although you cannot use dot notation to set the values of vector object properties, you can use it to get those values:

```
sc12.NumSamples
```

You get a result like:

```
ans =  
    300
```

```
ans =  
    300
```

To get a list of readable properties, type `scope_object`. The property values are listed in the MATLAB window.

Note Function names are case-sensitive. Type the entire name. Property names are not case-sensitive. You do not need to type the entire name, as long as the characters that you do type are unique for the property.

Use Scope Object Functions

Use the function syntax to run a scope object functions:

```
function_name(scope_object, argument_list)
```

Unlike properties, for which partial but unambiguous names are permitted, enter function names in full, in lowercase. For example, to add signals to the first scope in a vector containing the current scopes, in the MATLAB window, type:

```
allscopes = getscope(tg)  
addsignal(allscopes(1), [0,1])
```

Acquire Signal Data with File Scopes

You can acquire signal data into a file on the target computer. To do so, you can include a real-time file scope in your Simulink Real-Time model. Alternatively, after you build the real-time application and download it to the target computer, you can add a file scope to that application.

For example, to add a file scope named `sc` to the real-time application, and to add signal 4 to that scope:

- 1 In the MATLAB window, type:

```
tg = slrt;  
sc = addscope(tg, 'file')
```

The Simulink Real-Time software creates a file scope for the real-time application.

- 2 To add signal 4, type:

```
addsignal(sc, 4)
```

- 3 **Caution** Before starting the scope, copy previously acquired data to the development computer. When the file scope starts, the software overwrites previously acquired data in files of the specified name or name pattern. A partially overwritten file or a file that is opened but left unwritten loses its original contents.
-

To start the scope, type:

```
start(sc)
```

- 4 To start the real-time application, type:

```
start(tg)
```

The Simulink Real-Time software adds signal 4 to the file scope. When you start the scope and the real-time application, the scope saves the signal data for signal 4 to a file, by default named `C:\data.dat`.

- For more information on file scopes, see “Configure Real-Time File Scope Blocks” on page 6-76.
- To retrieve the file programmatically from the target computer for analysis, see “Using SimulinkRealTime.fileSystem Objects” on page 12-4.
- To acquire signal data into multiple files, see “Acquire Signal Data into Dynamically Named Files” on page 11-10.

Acquire Signal Data into Dynamically Named Files

You can acquire signal data into multiple, dynamically named files on the target computer. For example, you can acquire data into multiple files to examine one file while the scope continues to acquire data into other files.

To acquire data into multiple files, you can include a real-time file scope in your Simulink Real-Time model. Alternatively, after you build a real-time application and download it to the target computer, you can add a file scope to that application. You can then configure that scope to log signal data to multiple files.

For example, configure a file scope named `sc` to the real-time application. The file scope has these characteristics:

- Logs signal data into up to nine files whose sizes do not exceed 4096 bytes.
- Creates files whose names contain the character vector `file_.dat`.
- Contains signal 4.

1 In the MATLAB window, type:

```
tg = slrt;  
tg.StopTime = Inf;
```

This parameter value directs the real-time application to run indefinitely.

2 To add a file scope, type:

```
sc = addscope(tg, 'file');
```

3 To enable the file scope to create multiple log files, type:

```
sc.DynamicFileName = 'on';
```

Enable this setting to enable logging to multiple files.

4 To enable file scopes to collect data up to the number of samples, and then start over again, type:

```
sc.AutoRestart = 'on';
```

Use this setting for the creation of multiple log files.

5 To limit each log file size to 4096, type:

```
sc.MaxWriteFileSize = 4096;
```

You must use this property. Set `MaxWriteFileSize` to a multiple of the `WriteSize` property.

6 To enable the file scope to create multiple log files with the same name pattern, type:

```
sc.FileName = 'file_<%.>.dat';
```

This sequence directs the software to create up to nine log files, `file_1.dat` to `file_9.dat` on the target computer file system.

7 To add signal 4 to the file scope, type:

```
addsignal(sc, 4);
```

8 **Caution** Before starting the scope, copy previously acquired data to the development computer. When the file scope starts, the software overwrites previously acquired data in files of the

specified name or name pattern. A partially overwritten file or a file that is opened but left unwritten loses its original contents.

To start the scope, type

```
start(sc)
```

- 9 To start the real-time application, type

```
start(tg)
```

The software creates a log file named `file_1.dat` and writes data to that file. When the size of `file_1.dat` reaches 4096 bytes (value of `MaxWriteFileSize`), the software closes the file and creates `file_2.dat`. When its size reaches 4096 bytes, the software closes it and creates `file_3.dat`, and so on.

The software repeats this sequence until it fills the last log file, `file_9.dat`. If the real-time application continues to run and collect data after `file_9.dat`, the software reopens `file_1.dat` and overwrites the existing contents. It cycles through the other log files sequentially.

- For more information on file scopes, see “Configure Real-Time File Scope Blocks” on page 6-76.
- To retrieve the file programmatically from the target computer for analysis, see “Using SimulinkRealTime.fileSystem Objects” on page 12-4.
- To acquire signal data into a single file, see “Acquire Signal Data with File Scopes” on page 11-9.

Scope Trigger Configuration

You can configure Simulink Real-Time scopes to acquire data right away, or define triggers for scopes so that the Simulink Real-Time scopes wait until they are triggered to acquire data. You can configure Simulink Real-Time scopes to start acquiring data when a predefined trigger condition is met. The exact condition depends on the trigger mode that you specify.

- **Freerun** — Acquires data when the scope is started (default).
- **Software** — Acquires data in response to a user request, such as a call to one of the Scope functions (`trigger (fileScope)`, `trigger (hostScope)`, and `trigger (targetScope)`) or a call to a C or .NET API function (`xPCScSoftwareTrigger`, `xPCScope.Trigger`).
- **Signal** — Acquires data when a particular signal has crossed a preset level.
- **Scope** — Acquires data when another (triggering) scope starts.

You can use several additional properties to refine when a scope starts to acquire data. For example, if you want the scope to be triggered when another signal crosses a certain value, use **Signal** trigger mode. Specify:

- The signal to trigger the scope.
- The trigger level that the signal must cross to trigger the scope to start acquiring data.
- Whether the scope is triggered on a rising signal, falling signal, or either one.

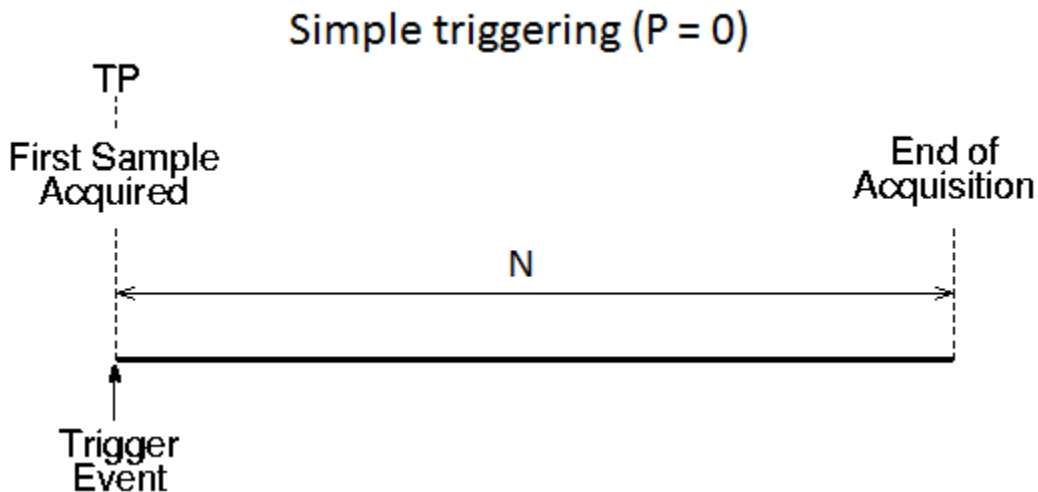
The *trigger point* is the sample at which the scope trigger condition is satisfied. For signal triggering, the trigger point is the sample at which the trigger signal passes through the trigger level. At the trigger point, the scope acquires the first sample. By default, scopes start acquiring data from the trigger point onwards. You can modify this behavior using pretriggering and posttriggering with the `NumPrePostSamples` scope property. See “Pretriggering and Posttriggering of Scopes” on page 11-13.

Pretriggering and Posttriggering of Scopes

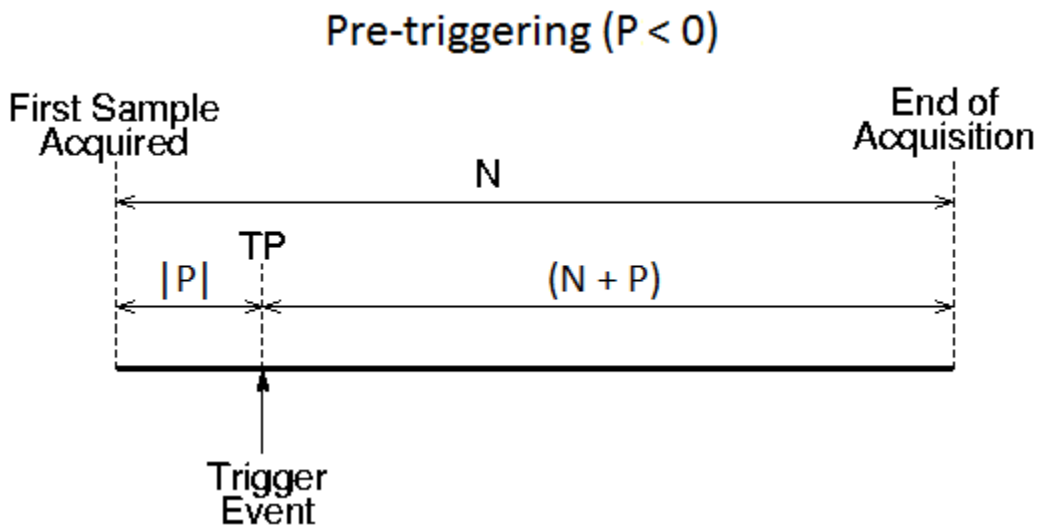
By default, the scope starts acquiring data at the same time as the trigger event (the trigger point). Sometimes, to observe the values that led to the trigger, you start acquiring data a given number of samples before the trigger event (pretriggering). Other times, to observe the system settling after the trigger, you delay acquiring data a given number of samples after the trigger event (posttriggering).

Use the `NumPrePostSamples` scope property to specify pretriggering and posttriggering. A negative value indicates pretriggering and a positive value indicates posttriggering. For example, suppose that P is the value of `NumPrePostSamples` for Scope 1 and TP is the trigger point, the sample where the trigger event occurs.

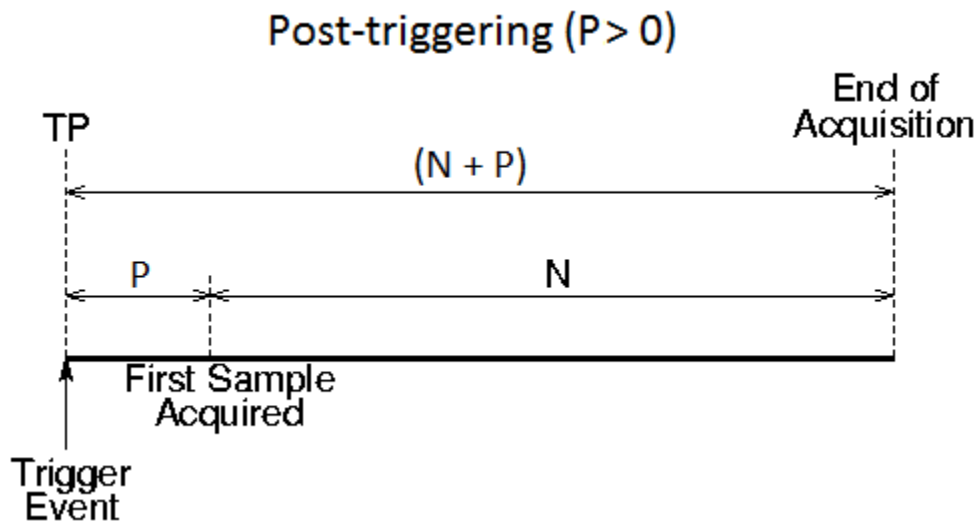
- $P = 0$ — Scope 1 starts acquiring data immediately at trigger point TP .



- $P < 0$ — Scope 1 starts acquiring data $|P|$ samples before trigger point TP .



- $P > 0$ — Scope 1 starts acquiring data P samples after trigger point TP .



Trigger One Scope with Another Scope

When you have started two scopes that you want to keep synchronized, you can trigger one scope with another to acquire data. Set up the first scope with the trigger of your choice, and then trigger the second scope from the first.

In this setup, Scope 1 triggers Scope 2.

- 1 Two scope objects are configured as a vector with the command:

```
tg = slrt;  
sc = addscope(tg, 'host', [1 2]);
```

- 2 For Scope 1, set these values:

```
sc(1).ScopeId = 1  
sc(1).NumSamples = N1  
sc(1).NumPrePostSamples = P1
```

- 3 For Scope 2, set these values:

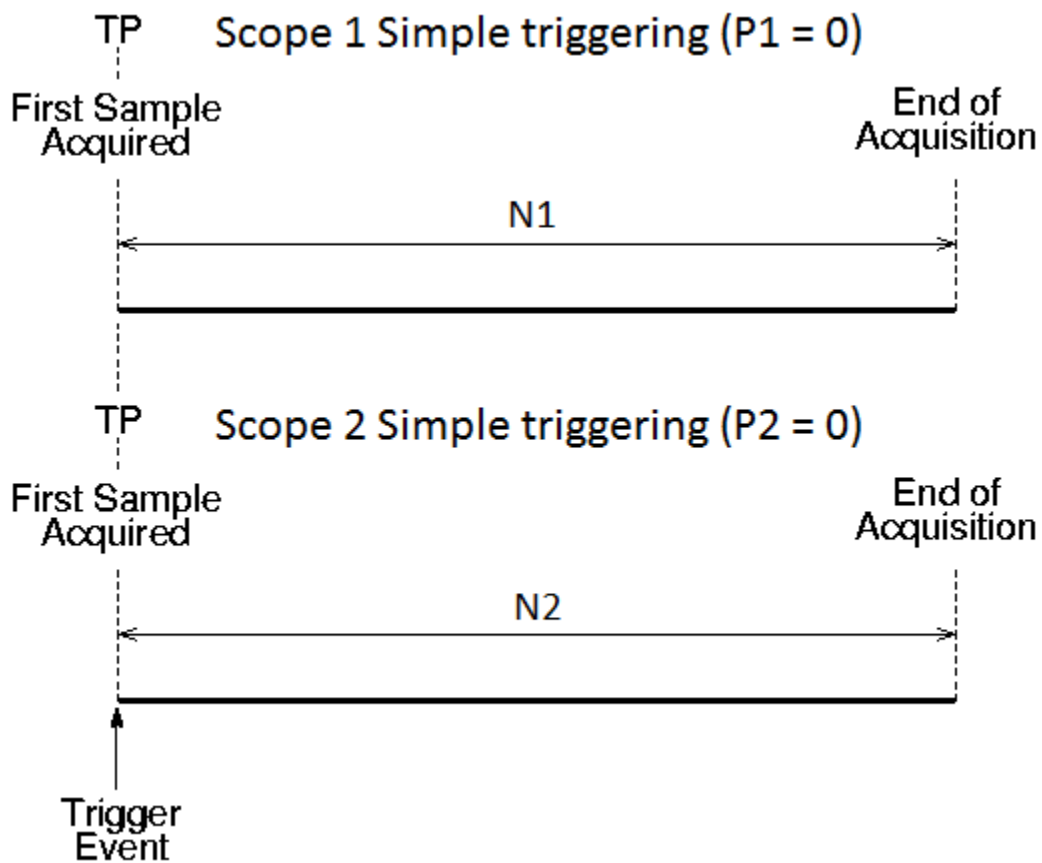
```
sc(2).ScopeId = 2  
sc(2).NumSamples = N2  
sc(2).TriggerMode = 'Scope'  
sc(2).TriggerScope = 1  
sc(2).NumPrePostSamples = P2
```

Because Scope 1 triggers Scope 2, the trigger point TP is the same for both scopes. However, Scopes 1 and 2 can acquire different samples.

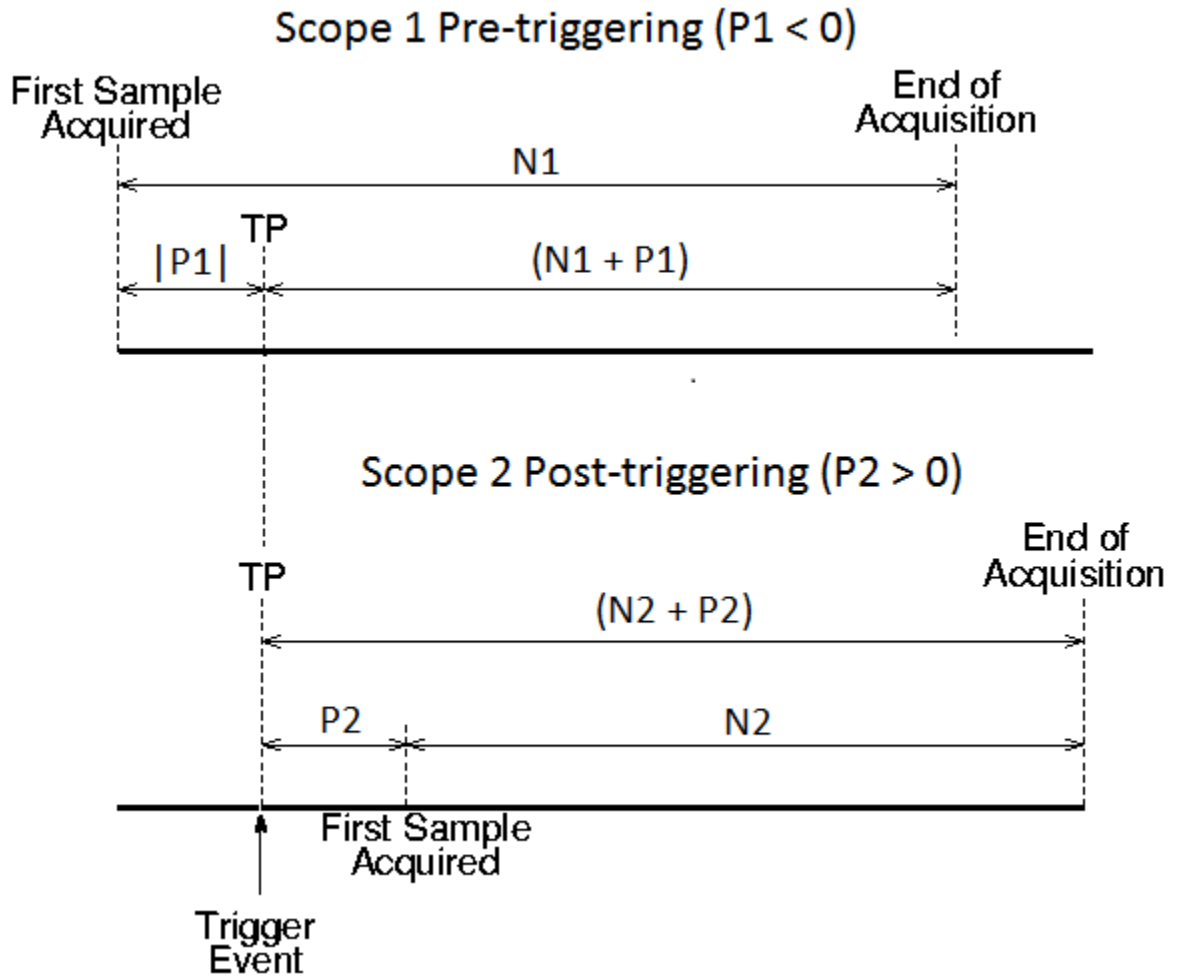
Scope-Triggered Data Acquisition

Some representative relationships between data acquisitions by Scope 1 and Scope 2 are shown in the figures. P1 and P2 are the values of NumPrePostSamples for Scopes 1 and 2. TP is the trigger point, the sample where a trigger event occurs, for both Scopes 1 and 2. Scope 2 begins acquiring data as described.

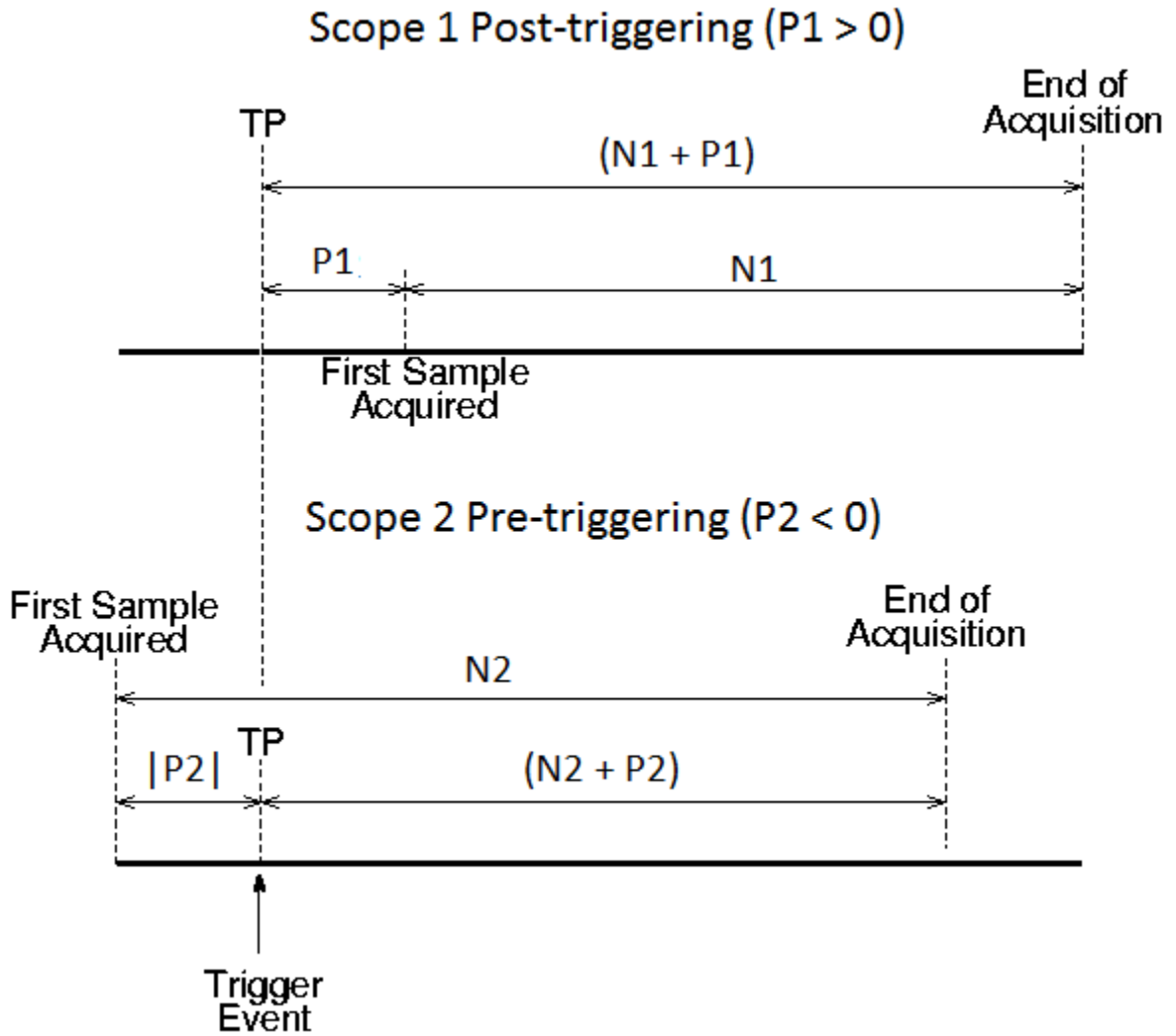
- P1 = 0 and P2 = 0 — Scope 1 and Scope 2 start acquiring data immediately at trigger point TP.



- $P1 < 0$ and $P2 > 0$ — Scope 1 starts acquiring data $|P1|$ samples before trigger point TP. Scope 2 starts acquiring data $P2$ samples after trigger point TP.



- $P1 > 0$ and $P2 < 0$ — Scope 1 starts acquiring data $P1$ samples after trigger point TP . Scope 2 starts acquiring data $|P2|$ samples before trigger point TP .

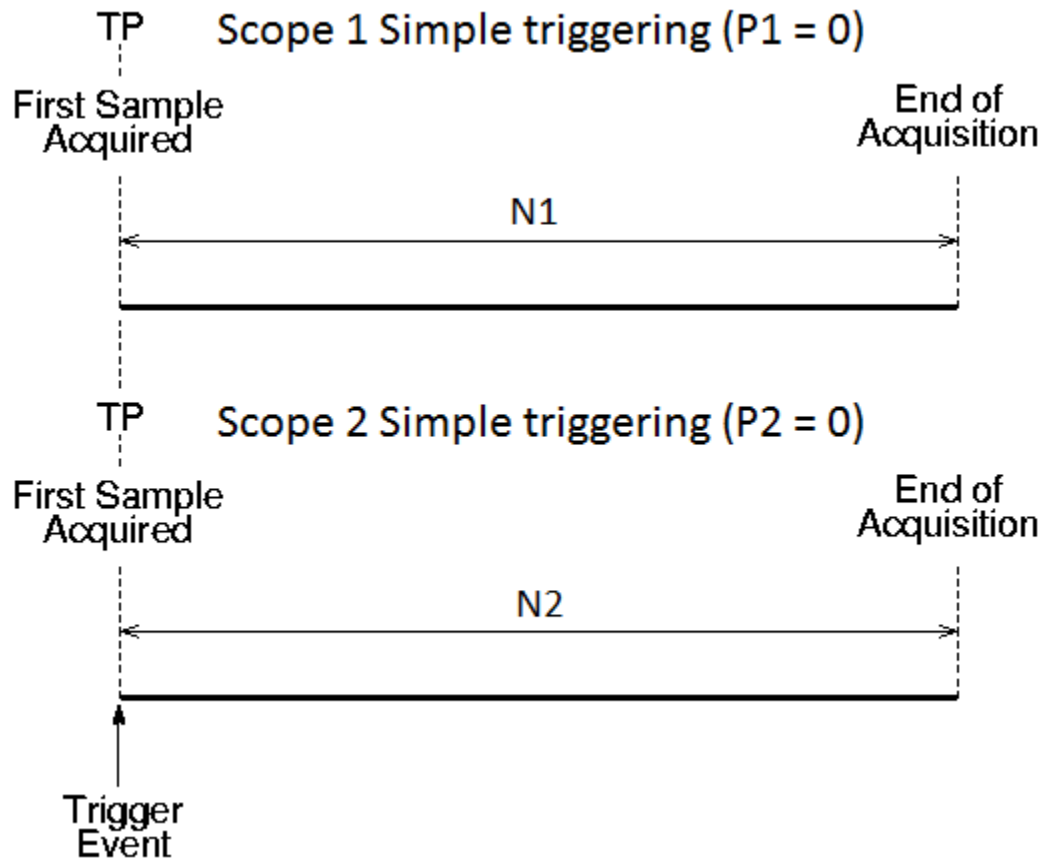


Trigger Sample Setting

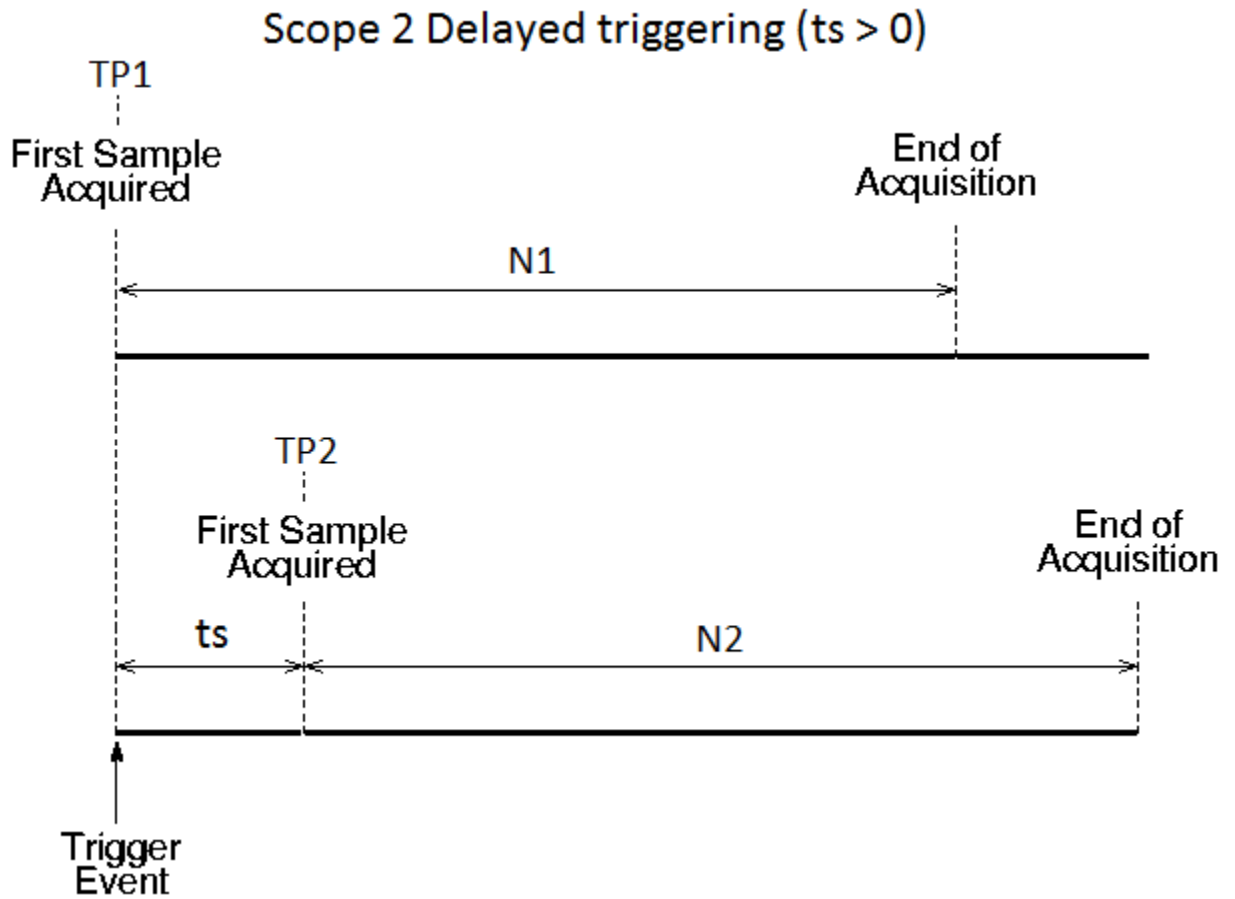
For additional flexibility in scope triggering, you can use the Scope 2 trigger sample setting.

`sc(2).TriggerSample = range 0 to (N + P1 - 1)`

- `sc(2).TriggerSample = 0` (default) — Scope 2 triggers when Scope 1 triggers. Trigger point TP is the same sample for both scopes.



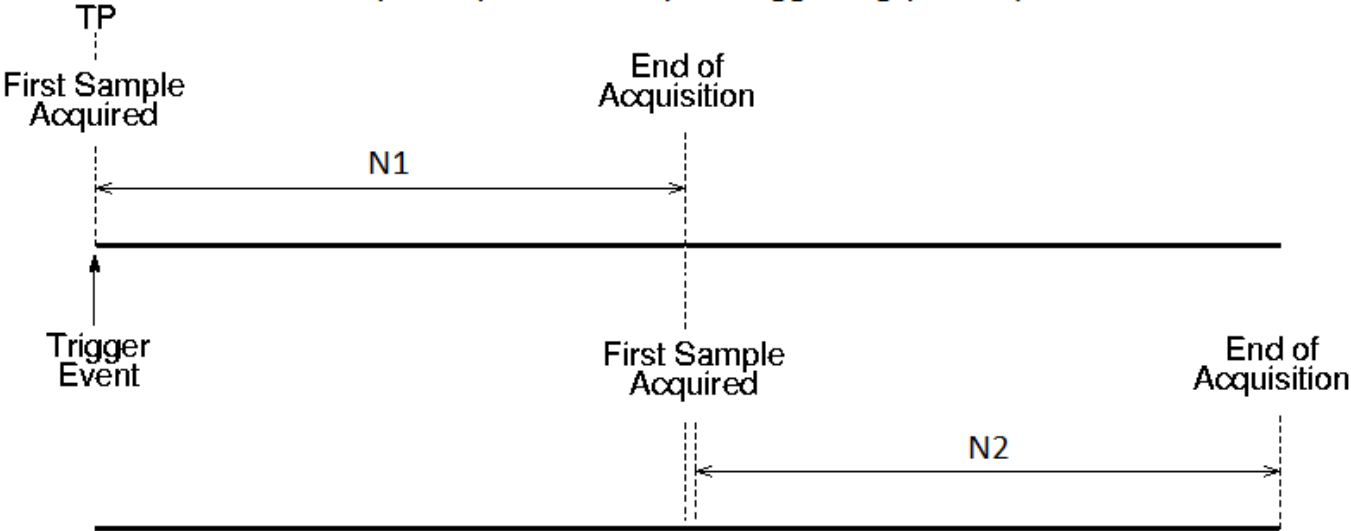
- `sc(2).TriggerSample = ts > 0` — Scope 2 triggers `ts` samples after Scope 1 is triggered. Trigger point TP2 for Scope 2 is `ts` samples after TP1 for Scope 1.



Setting `sc(2).TriggerSample` to a value `ts` larger than $(N + P - 1)$ does not cause an error. It implies that Scope 2 cannot be triggered, because Scope 1 cannot acquire more than $(N + P - 1)$ samples after TP.

- `sc(2).TriggerSample = -1` (special case) — Causes Scope 2 to start acquiring data from the sample after Scope 1 stops acquiring.

Scope 2 Special delayed triggering ($t_s = -1$)

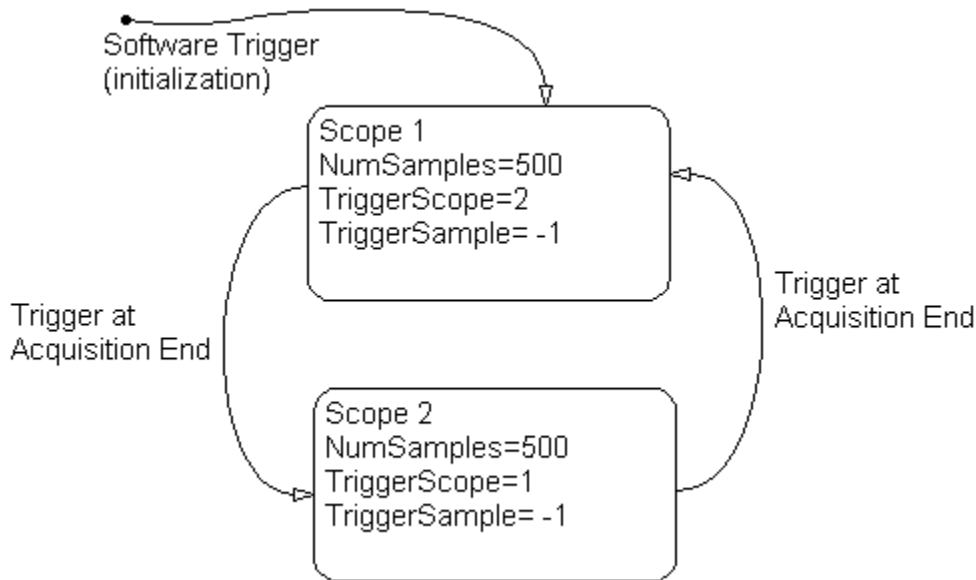


Minimize Data Gaps with Two Scopes

With two scopes, you can minimize data overlap or gaps. The first scope acquires data up to sample N , then stops. The second scope begins to acquire data at sample $N+1$.

In this example, the `TriggerMode` property of Scope 1 is set to 'Scope', but it is explicitly triggered with the MATLAB function `trigger(sc1)`.

You can use the `trigger` function to force real-time scopes to trigger, regardless of trigger mode setting and regardless of whether the triggering criteria were met.



To minimize gaps by acquiring data with two scopes:

- 1 Build and download the Simulink model `xpcosc` to the target computer.
- 2 In the MATLAB Command Window, assign `tg` to the target computer and set the `StopTime` property to 10.


```
tg = slrt;
tg.StopTime = 10;
```
- 3 Add a vector of two host scopes to the real-time application. Use the vector index to switch from one scope to the other.

```
sc = addscope(tg, 'host', [1 2]);
```

- 4 Add signals 4 and 5 to both scopes.

```
addsignal(sc, [4 5]);
```

- 5 Set the `NumSamples` property for both scopes to 500 and the `TriggerSample` property for both scopes to -1. With this property setting, each scope triggers the next scope *at the end* of its 500 sample acquisition.

```
sc(1).NumSamples = 500;
sc(1).TriggerSample = -1;
```

```
sc(2).NumSamples = 500;
sc(2).TriggerSample = -1;
```

- 6** Set the `TriggerMode` property for scopes 1 and 2 to 'Scope'. Set the `TriggerScope` property such that each scope triggers the other.

```
sc(1).TriggerMode = 'Scope';
sc(1).TriggerScope = 2;
```

```
sc(2).TriggerMode = 'Scope';
sc(2).TriggerScope = 1;
```

- 7** Set up storage for time, `t`, and signal, data acquisition.

```
t = [];
data = zeros(0, 2);
```

- 8** Start both scopes and the model.

```
start(sc);
start(tg);
```

Both scopes receive the same signals, 4 and 5.

- 9** To start acquiring data, explicitly trigger scope 1.

```
scNum = 1;
trigger(sc(scNum));
```

Setting `scNum` to 1 indicates that Scope 1 acquires data first.

- 10** Start acquiring data using the two scopes to double buffer the data.

```
while (1)

    % Busy wait until this scope has finished acquiring 500 samples
    % or the model stops (scope is interrupted).
    while ~(strcmp(sc(scNum).Status, 'Finished') || ...
            strcmp(sc(scNum).Status, 'Interrupted'))
        end

    % Stop buffering data when the model stops.
    % Pause to be certain that the status property has been updated.

    pause(0.1)

    if strcmp(tg.Status, 'stopped')
        break
    end

    % Save the data.
    t( end + 1 : end + 500) = sc(scNum).Time;
    data(end + 1 : end + 500, :) = sc(scNum).Data;

    % Restart this scope.
    start(sc(scNum));

    % Switch to the next scope.
    if(scNum == 1) scNum = 2;
    else scNum = 1;
```

```
end
```

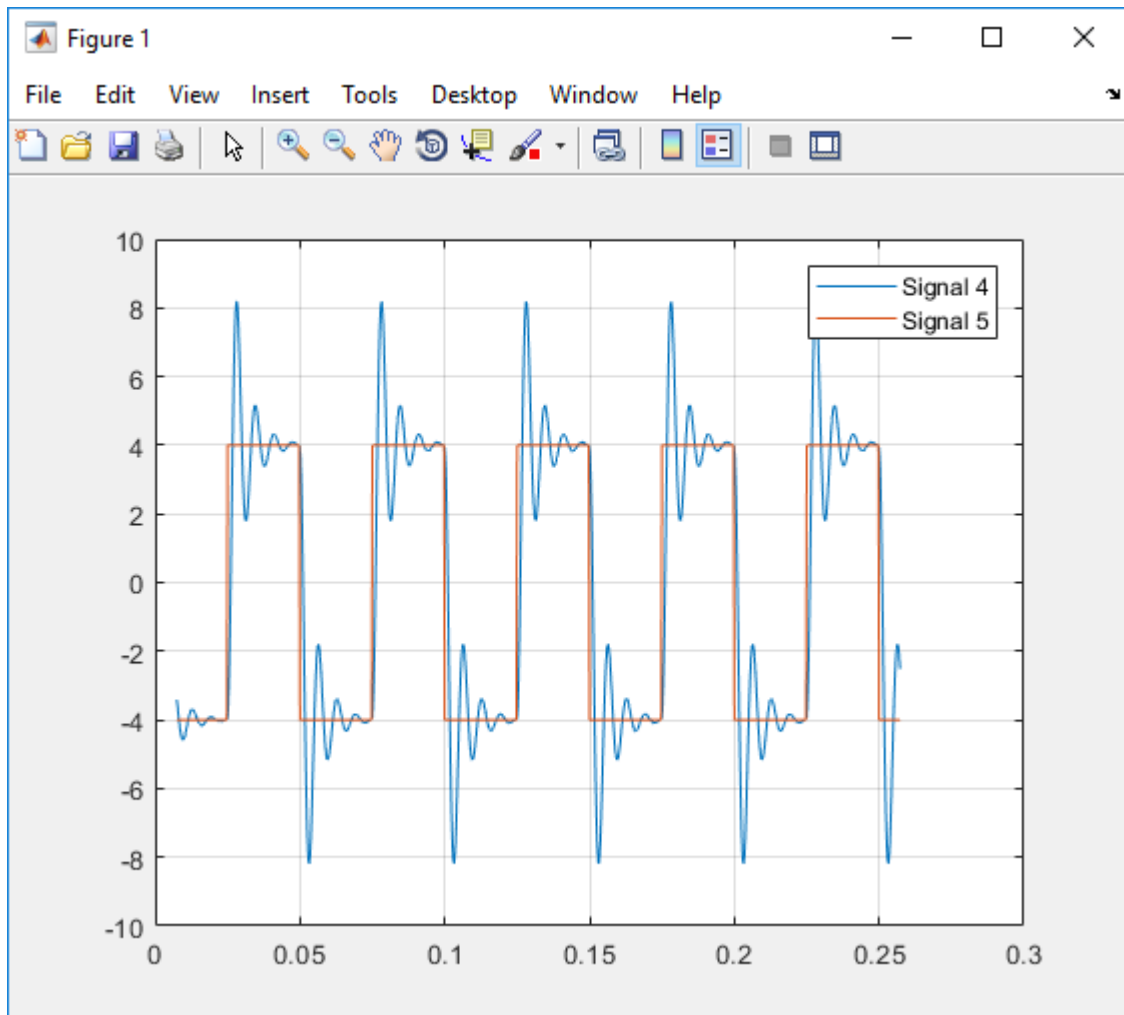
```
end
```

- 11 When done, remove the scopes.

```
% Remove the scopes we added.
remscope(tg,[1 2]);
```

- 12 Plot the data.

```
plot(t,data);
grid on;
legend('Signal 4','Signal 5');
```



Following is a complete code listing for the preceding double-buffering data acquisition procedure. After you download the model (xpcosc) to the target computer, you can copy and paste this code into a MATLAB file and run it. Communication between the development and target computers must be fast enough to transmit the entire set of samples before the next acquisition cycle starts. In a similar way, you can use more than two scopes to implement a triple- or quadruple-buffering scheme.

```
% Assumes model xpcosc program text has been
% built and loaded on the target computer.
```

```

% Attach to the target computer and set StopTime to 10 sec.
tg = slrt;
tg.StopTime = 10;

% Add two host scopes.
sc = addscope(tg,'host', [1 2]);

% [4 5] are the signals of interest. Add to both scopes.
addsignal(sc,[4 5]);

% Each scope triggers the next scope at end of a 500 sample acquisition.
sc(1).NumSamples = 500;
sc(1).TriggerSample = -1;

sc(2).NumSamples = 500;
sc(2).TriggerSample = -1;

sc(1).TriggerMode = 'Scope';
sc(1).TriggerScope = 2;

sc(2).TriggerMode = 'Scope';
sc(2).TriggerScope = 1;

% Initialize time and data log.
t = [];
data = zeros(0, 2);

% Start the scopes and the model.
start(sc);
start(tg);

% To start the capture, explicitly trigger scope 1.
scNum = 1;
trigger(sc(scNum));

% Use the two scopes as a double buffer to log the data.
while (1)

    % Busy wait until this scope has finished acquiring 500 samples
    % or the model stops (scope is interrupted).
    while ~(strcmp(sc(scNum).Status, 'Finished') || ...
            strcmp(sc(scNum).Status, 'Interrupted'))
        end

    % Stop buffering data when the model stops.
    % Pause to be certain that the status property has been updated.

    pause(0.1)

    if strcmp(tg.Status, 'stopped')
        break
    end

    % Save the data.
    t( end + 1 : end + 500) = sc(scNum).Time;
    data(end + 1 : end + 500, :) = sc(scNum).Data;

    % Restart this scope.
    start(sc(scNum));

    % Switch to the next scope.
    if(scNum == 1) scNum = 2;
    else scNum = 1;
    end

end

% Remove the scopes we added.
remscope(tg,[1 2]);

% Plot the data.

```

```
plot(t,data);  
grid on;  
legend('Signal 4','Signal 5');
```

Logging Signal Data with File System Objects

- “File System Basics” on page 12-2
- “Using SimulinkRealTime.fileSystem Objects” on page 12-4

File System Basics

Simulink Real-Time file scopes create files on the target computer. To work with these files from the development computer, see `SimulinkRealTime.fileSystem`. The `SimulinkRealTime.fileSystem` object allows you to perform file system-like operations on the target computer file system.

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

You cannot direct the scope to write the data to a file on the Simulink Real-Time development computer. When the software has written the signal data file to the target computer, you can access the contents of the file from the development computer.

The software can write data files to:

- **Hard drive** — The target computer hard drive supports a serial ATA (SATA) drive. The Simulink Real-Time software supports file systems of type FAT-32 only.

Check that the hard drive is not cable-selected and that the target computer can detect it. The maximum file size is limited by the FAT-32 file system type.

A Simulink Real-Time file scope can access the target computer hard drive, provided it is formatted with the FAT-32 file system. Simulink Real-Time ignores other file systems.

Note In a future release, the `SecondaryIDE` target setting will be read-only and set to 'off'.

- **ERAM drive** — If the target computer has more than 4 GB of RAM, the kernel automatically formats the excess memory as an extended RAM (ERAM) drive. The kernel assigns the ERAM drive the drive letter 'H:'. Use the ERAM drive when you need faster file I/O than you can achieve with other drive types.

The limitations for hard drives also apply to the ERAM drive.

- **USB drive** — To write data files to a USB drive, you must set the **USB Support** property in Simulink Real-Time.
- **3.5-inch disk drive** — Writing data files to a 3.5-inch disk drive is considerably slower than writing to a hard drive.

There are the following limitations:

- 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 in the operating system on the target computer can have a maximum of 260 characters. If the file name is longer than eight-dot-three format (eight character file name, period, three character extension), the operating system represents the file name in truncated form (for example, six characters followed by '~1'). MATLAB commands can access the file using the fully qualified file name or the truncated representation of the name. Some block parameters, such as the Scope block `filename` parameter, require 8.3 format for the file name.

- Do not write data to the `private` folder on your target computer. It is reserved for Simulink Real-Time internal use.

You can access signal data files, or other target computer system files, in one of the following ways:

- If running the target computer standalone, you can access a file by restarting the target computer under an operating system such as DOS. You can access the file through the operating system utilities.
- If running the target computer linked to a development computer, you can access the target computer file system from the development computer using a `SimulinkRealTime.fileSystem` function.

You can perform file transfer operations using the functions `SimulinkRealTime.copyFileToHost` and `SimulinkRealTime.copyFileToTarget`.

You can perform file system-like tasks using functions such as `fopen` and `fread` on the signal data file. File system functions work like the corresponding MATLAB file I/O functions.

The `SimulinkRealTime.fileSystem` class also includes file system utilities that allow you to collect target computer file system information for the disk and disk buffers.

This topic focuses primarily on using the `SimulinkRealTime.fileSystem` functions to work with target computer data files that you generate from a real-time Scope of type `file`.

For an example of how to perform data logging with the Simulation Data Inspector, see “Data Logging With Simulation Data Inspector (SDI)” on page 15-190.

Using SimulinkRealTime.fileSystem Objects

In this section...

“Copying Files from the Target Computer to the Development Computer” on page 12-5

“Copying Files from the Development Computer to the Target Computer” on page 12-5

“Accessing File Systems on a Specific Target Computer” on page 12-6

“Reading the Contents of a File on the Target Computer” on page 12-6

“Removing a File from the Target Computer” on page 12-8

“Getting a List of Open Files on the Target Computer” on page 12-8

“Getting Information About a File on the Target Computer” on page 12-9

“Getting Information About a Disk on the Target Computer” on page 12-9

The `fileSystem` object enables you to work with the target computer file system from the development computer. You enter target object functions in the MATLAB window on the development computer or use scripts. The `fileSystem` object has functions that allow you to use

- `cd` to change folders
- `dir` to list the contents of the current folder
- `mkdir` to make a folder
- `pwd` to get the current working folder path
- `rmdir` to remove a folder
- `diskinfo` to get information about the specified disk
- `fclose` to close a file (similar to MATLAB `fclose`)
- `fileinfo` to get information about a particular file
- `filetable` to get information about files in the file system
- `fopen` to open a file (similar to MATLAB `fopen`)
- `fread` to read a file (similar to MATLAB `fread`)
- `fwrite` to write a file (similar to MATLAB `fwrite`)
- `getfilesize` to get the size of a file in bytes
- `removefile` to remove a file from the target computer

Note: The `SimulinkRealTime.fileSystem` object will be removed in a future release. See the release note for file system commands to use instead. These commands use the `SimulinkRealTime.openFTP` function and the functions for the MATLAB `ftp` object.

Useful global functions:

- `SimulinkRealTime.copyFileToHost` to retrieve a file from the target computer to the development computer
- `SimulinkRealTime.copyFileToTarget` to place a file from the development computer on the target computer
- `SimulinkRealTime.utils.getFileScopeData`, to interpret the raw data from the `fread` function

These procedures assume that the target computer has a signal data file created by a Simulink Real-Time file scope. This file has the path name `C:\data.dat`.

There are the following limitations:

- 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 in the operating system on the target computer can have a maximum of 260 characters. If the file name is longer than eight-dot-three format (eight character file name, period, three character extension), the operating system represents the file name in truncated form (for example, six characters followed by '~1'). MATLAB commands can access the file using the fully qualified file name or the truncated representation of the name. Some block parameters, such as the Scope block `filename` parameter, require 8.3 format for the file name.
- Do not write data to the `private` folder on your target computer. It is reserved for Simulink Real-Time internal use.

Copying Files from the Target Computer to the Development Computer

You can copy a data file from the target computer to the development computer using a `SimulinkRealTime` package function on the development computer.

For example, to retrieve a file named `data.dat` from the target computer `C:\` drive (default):

- 1 If you have not already done so, in the MATLAB window, type the following to assign the default `SimulinkRealTime.target` object to a variable.

```
tg = slrt;
```

- 2 Type

```
SimulinkRealTime.copyFileToHost(tg, 'data.dat')
```

This command retrieves the file and saves that file to the variable `data`. This content is in the Simulink Real-Time file format.

Copying Files from the Development Computer to the Target Computer

You can copy a file from the development computer to the target computer using a `SimulinkRealTime` package function on the development computer.

For example, to copy a file named `data2.dat` from the development computer to the target computer `C:\` drive (default),

- 1 If you have not already done so, in the MATLAB window, type the following to assign the default `SimulinkRealTime.target` object to a variable.

```
tg = slrt;
```

- 2 Type the following to save that file to the variable `data`.

```
SimulinkRealTime.copyFileToTarget(tg, 'data2.dat')
```

Accessing File Systems on a Specific Target Computer

You can access specific target computer files from the development computer for the `SimulinkRealTime.fileSystem` object.

Use the `SimulinkRealTime.fileSystem` creator function. If your system has multiple targets, you can access specific target computer files from the development computer for the `SimulinkRealTime.fileSystem` object.

For example, to list the name of the current folder of target computer 'TargetPC1':

- 1 In the MATLAB window, type a command like the following to assign the `SimulinkRealTime.fileSystem` object for the default computer to a variable.

```
fsys = SimulinkRealTime.fileSystem;
```

- 2 Type

```
dir(fsys)
```

Alternatively, you can use the `SimulinkRealTime.target` constructor to construct a target object for a specific computer, then use that target object as an argument to `SimulinkRealTime.fileSystem`.

- 1 In the MATLAB window, type a command like the following to assign the `SimulinkRealTime.target` object for target computer 'TargetPC1' to a variable.

```
tg1 = SimulinkRealTime.target('TargetPC1');
```

- 2 To assign the `SimulinkRealTime.fileSystem` object to a variable, type:

```
fsys = SimulinkRealTime.fileSystem(tg1);
```

- 3 Type

```
dir(fsys)
```

Reading the Contents of a File on the Target Computer

You can read the contents of a data file from the target computer by using `SimulinkRealTime.fileSystem` functions on the development computer. Use this procedure as an alternative to the method described in “Configure File Scopes with MATLAB Language” on page 6-104.

To run a `SimulinkRealTime.fileSystem` object function, use the `function_name(fs_object, argument_list)` syntax. For example, to retrieve the contents of a file named `data.dat` from the target computer C:\ drive (default):

- 1 If you have not already done so, in the MATLAB window, type the following to assign the `SimulinkRealTime.fileSystem` object to a variable.

```
fsys = SimulinkRealTime.fileSystem;
```

- 2 Type

```
h = fopen(fsys, 'data.dat');
```

This command opens the file `data.dat` for reading and assigns the file identifier to `h`.

3 Type

```
data2 = fread(fsys,h);
```

This command reads the file `data.dat` and stores the contents of the file to `data2`. This content is in the Simulink Real-Time file format.

4 Type

```
fclose(fsys, h)
```

This command closes the file `data.dat`.

Before you can view or plot the contents of this file, you must convert the contents. See “Converting Simulink Real-Time File Format Content to Double Precision Data” on page 12-7.

Converting Simulink Real-Time File Format Content to Double Precision Data

The Simulink Real-Time software provides the function `SimulinkRealTime.utils.getFileScopeData` to convert Simulink Real-Time file format content (in bytes) to double precision data representing the signals and timestamps. The `SimulinkRealTime.utils.getFileScopeData` function takes in data from a file in Simulink Real-Time format. The data must be a vector of bytes (`uint8`). To convert the data to `uint8`, use a command like the following:

```
data2 = uint8(data2');
```

This section assumes that you have a variable, `data2`, that contains data in the Simulink Real-Time file format (see “Reading the Contents of a File on the Target Computer” on page 12-6).

- 1 In the MATLAB window, change folder to the folder that contains the Simulink Real-Time format file.
- 2 Type

```
new_data2 = SimulinkRealTime.utils.getFileScopeData(data2);
```

`SimulinkRealTime.utils.getFileScopeData` converts the format of `data2` from the Simulink Real-Time file format to an array of bytes. It also creates a structure for that file in `new_data2`, of which one of the elements is an array of doubles, `data`. The `data` member is also appended with a timestamp vector. The data is returned as doubles, which represent the real-world values of the original Simulink signals at the specified times during target execution.

You can view or examine the signal data. You can also plot the data with `plot(new_data2.data)`.

If you use Simulink Real-Time in standalone mode, you can extract the data from the data file:

- First determine the file header size. To obtain the file header size, ignore the first 8 bytes of the file. The next 4 bytes store the header size as an unsigned integer.
- After the header size number of bytes, the file stores the signals sequentially as doubles. For example, assume that the scope has three signals, `x`, `y`, and `z`. Assume that `x[0]` is the value of `x` at sample 0, `x[1]` is the value at sample 1, and so forth. Also assume `t[0]`, `t[1]` are the simulation time values at samples 0, 1, and so forth. The file saves the data using the following pattern:

```
x[0] y[0] z[0] t[0] x[1] y[1] z[1] t[1] x[2] y[2] z[2] t[2]...
x[N] y[N] z[N] t[N]
```

N is the number of samples acquired. The file saves x, y, z, and t as doubles at 8 bytes each.

Removing a File from the Target Computer

You can remove a file from the target computer by using Simulink Real-Time functions on the development computer for the `SimulinkRealTime.fileSystem` object. If you have not already done so, close this file first with `fclose`.

To run a `SimulinkRealTime.fileSystem` object function, use the `function_name(fs_object, argument_list)` syntax. For example, to remove a file named `data2.dat` from the target computer C:\ drive (default),

- 1 If you have not already done so, in the MATLAB window, type the following to assign the `SimulinkRealTime.fileSystem` object to a variable.

```
fsys = SimulinkRealTime.fileSystem;
```

- 2 Type the following to remove the specified file from the target computer.

```
removefile(fsys, 'data2.dat')
```

Getting a List of Open Files on the Target Computer

You can get a list of open files on the target computer file system by using `SimulinkRealTime.fileSystem` object functions on the development computer. The target computer file system limits the number of open files you can have to eight. Use this list to identify files that you can close.

To run a `SimulinkRealTime.fileSystem` object function, use the `function_name(fs_object, argument_list)` syntax. For example, to get a list of open files for the file system object `fsys`,

- 1 If you have not already done so, in the MATLAB window, type the following to assign the `SimulinkRealTime.fileSystem` object to a variable.

```
fsys = SimulinkRealTime.fileSystem;
```

- 2 Type

```
filetable(fsys)
```

If the file system has open files, a list like the following is displayed:

```
ans =
Index   Handle  Flags   FilePos  Name
-----
      0 00060000 R_      8512  C:\DATA.DAT
      1 00080001 R_         0  C:\DATA1.DAT
      2 000A0002 R_      8512  C:\DATA2.DAT
      3 000C0003 R_      8512  C:\DATA3.DAT
      4 001E0001 R_         0  C:\DATA4.DA
```

- 3 The table returns the open file handles in hexadecimal. To convert a handle to one that other `SimulinkRealTime.fileSystem` functions, such as `fclose`, can use, use the `hex2dec` function. For example,

```
h1 = hex2dec('001E0001')
```

```
h1 =
1966081
```

- 4 To close that file, use the `SimulinkRealTime.fileSystem fclose` function. For example,

```
fclose(fsys, h1)
```

Getting Information About a File on the Target Computer

You can display information for a file on the target computer file system by using `SimulinkRealTime.fileSystem` object functions on the development computer.

To run a `SimulinkRealTime.fileSystem` object function, use the `function_name(fs_object, argument_list)` syntax. For example, to display the information for the file identifier `fid1`,

- 1 If you have not already done so, in the MATLAB window, type the following to assign the `SimulinkRealTime.fileSystem` object to a variable.

```
fsys = SimulinkRealTime.fileSystem;
```

- 2 Type

```
fid1 = fopen(fsys, 'data.dat');
```

This command opens the file `data.dat` for reading and assigns the file identifier to `fid1`.

- 3 Type

```
fileinfo(fsys, fid1)
```

This returns disk information like the following for the C:\ drive file system.

```
ans =
      FilePos: 0
  AllocatedSize: 12288
   ClusterChains: 1
VolumeSerialNumber: 1.0450e+009
      FullName: 'C:\DATA.DAT'
```

Getting Information About a Disk on the Target Computer

You can display information for a disk on the target computer file system by using `SimulinkRealTime.fileSystem` object functions on the development computer.

To run a `SimulinkRealTime.fileSystem` object function, use the `function_name(fs_object, argument_list)` syntax. For example, to display the disk information for the C:\ drive,

- 1 If you have not already done so, in the MATLAB window, type the following to assign the `SimulinkRealTime.fileSystem` object to a variable.

```
fsys = SimulinkRealTime.fileSystem;
```

- 2 Type

```
diskinfo(fsys, 'C:\');
```

This returns disk information like the following for the C:\ drive file system.

```
ans =  
  
  struct with fields:  
  
    DriveLetter: 'C'  
    Label: 'FREEDOS'  
    Reserved: ' '  
    SerialNumber: -857442364  
FirstPhysicalSector: 63  
    FATType: 32  
    FATCount: 2  
    MaxDirEntries: 0  
    BytesPerSector: 512  
    SectorsPerCluster: 64  
    TotalClusters: 1831212  
    BadClusters: 0  
    FreeClusters: 1827614  
    Files: 938  
    FileChains: 942  
    FreeChains: 1  
    LargestFreeChain: 1827614  
    DriveType: DRIVE_FIXED
```


Deploy the MATLAB Application as a Standalone Executable

- “MATLAB Runtime Setup” on page 13-2
- “Deploy MATLAB Application to Control Real-Time Application” on page 13-3

MATLAB Runtime Setup

As part of developing and testing your model, you can write MATLAB applications for:

- Parameter extreme value testing
- Regression testing
- Putting the model into a consistent state for testing another part of the system

To run an application on a Windows computer that does not have MATLAB installed, use MATLAB Compiler to deploy the application as a standalone executable.

To deploy a MATLAB application, first set up MATLAB Runtime:

- 1 Open MATLAB.
- 2 To find the `MCRInstaller` program, type:

```
mcrinstaller
```

The Command Window displays output similar to this output: `C:\Program Files\MATLAB\R2017b\toolbox\compiler\deploy... \win64\MCRInstaller.exe`

- 3 Copy and paste the full path to the `MCRInstaller` program into the Windows Run text box.
- 4 Press **Enter**, and then follow the prompts.

Place the MATLAB Runtime in the default location, for example:

```
C:\Program Files\MATLAB\MATLAB Runtime
```

From the installation dialog box, copy and paste the location where the installer places the MATLAB Runtime binary files:

```
C:\Program Files\MATLAB\MATLAB Runtime\v93
```

- 5 Close MATLAB and open a Windows command prompt window.
- 6 Type the following command:

```
set PATH=C:\Program Files\MATLAB\MATLAB Runtime\v93\bin\win64;%PATH%
```

This command sets the MATLAB Runtime path only for the command prompt window within which you typed it. To make this environment variable setting visible throughout the operating system, see the Windows documentation.

See Also

More About

- “Deploy MATLAB Application to Control Real-Time Application” on page 13-3

Deploy MATLAB Application to Control Real-Time Application

Required Products: Simulink, Simulink Real-Time, MATLAB Compiler, and MATLAB Compiler SDK™

This example shows how to deploy a test script as a standalone executable by using MATLAB Compiler. The test script performs a frequency-response test of the `xpcosc` model. Using this information, in the design phase, you can modify the internal parameters of the model to meet your frequency requirements. In the production phase, you can bin manufactured parts based on frequency response.


The test script is `slrt_freq_sweep_test.m` (`open(fullfile(matlabroot, 'help', 'toolbox', 'xpc', 'examples', 'slrt_freq_sweep_test.m'))`).

Prerequisites

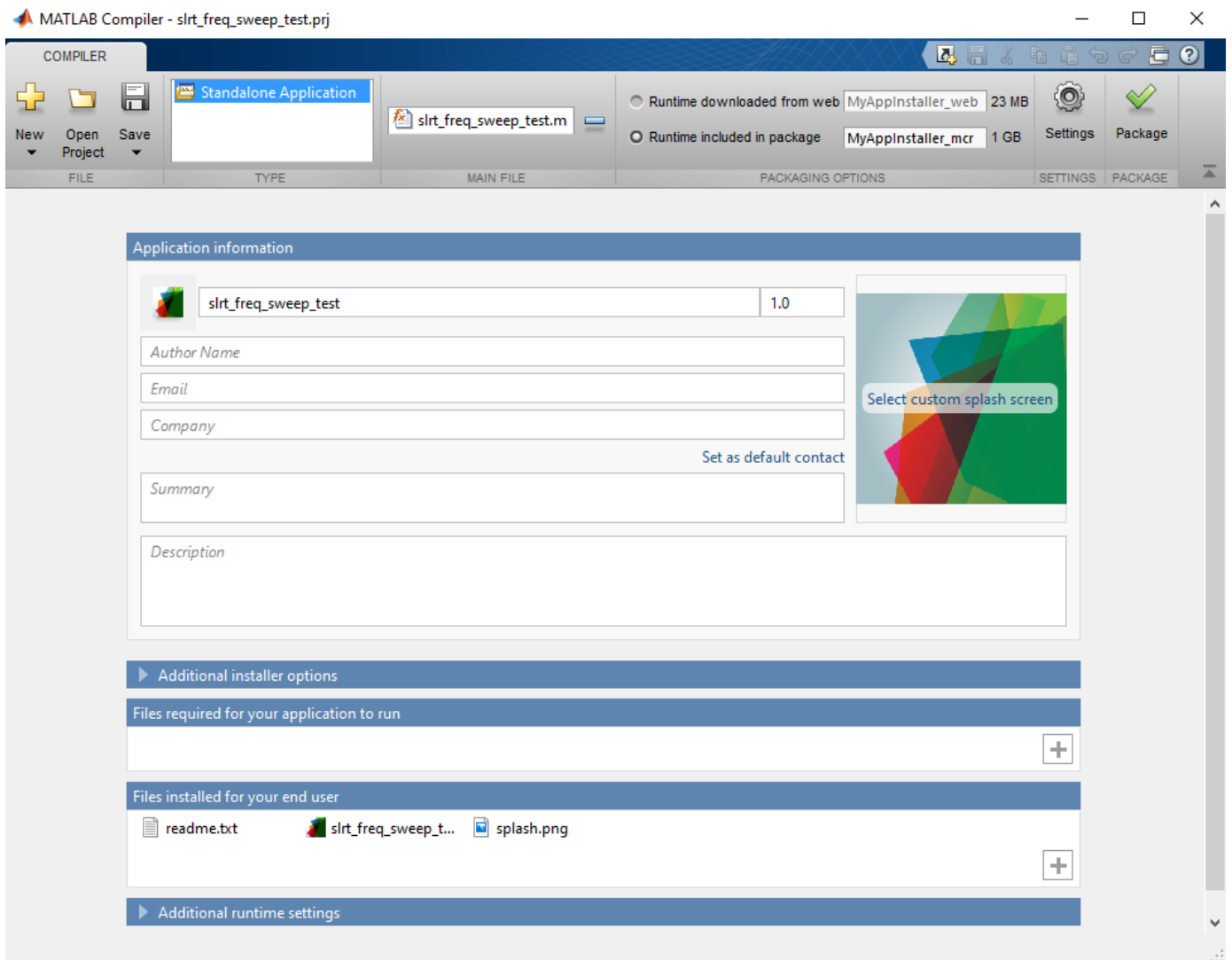
This procedure assumes that you have:

- 1 Completed the steps in “MATLAB Runtime Setup” on page 13-2.
- 2 Opened MATLAB from the Windows command prompt window within which you performed MATLAB run-time setup.
- 3 Configured TCP/IP communication between the development and target computers, recorded the required settings in the test script `slrt_freq_sweep_test.m`, and saved the script in a working folder.
- 4 Built the `xpcosc` real-time application.

Package the MATLAB Application

- 1 Open **Apps > Application Compiler**.
- 2 Enter the name of the application as `slrt_freq_sweep_test`. Add summary information as required.
- 3 To save the project, click **Save**. Save the project under a name such as `slrt_freq_sweep_test.prj`.
- 4 Click the **Add main file** button , and then navigate to the file `slrt_freq_sweep_test.m`.
- 5 Under **PACKAGING OPTIONS**, select the **Runtime included in package** check box.

13 Deploy the MATLAB Application as a Standalone Executable



- 6 Click the **Package** button .

The compiler generates the application and opens the `slrt_freq_sweep_test` folder in Windows Explorer.

- 7 To save the project, click **Save**.

Run the MATLAB Application

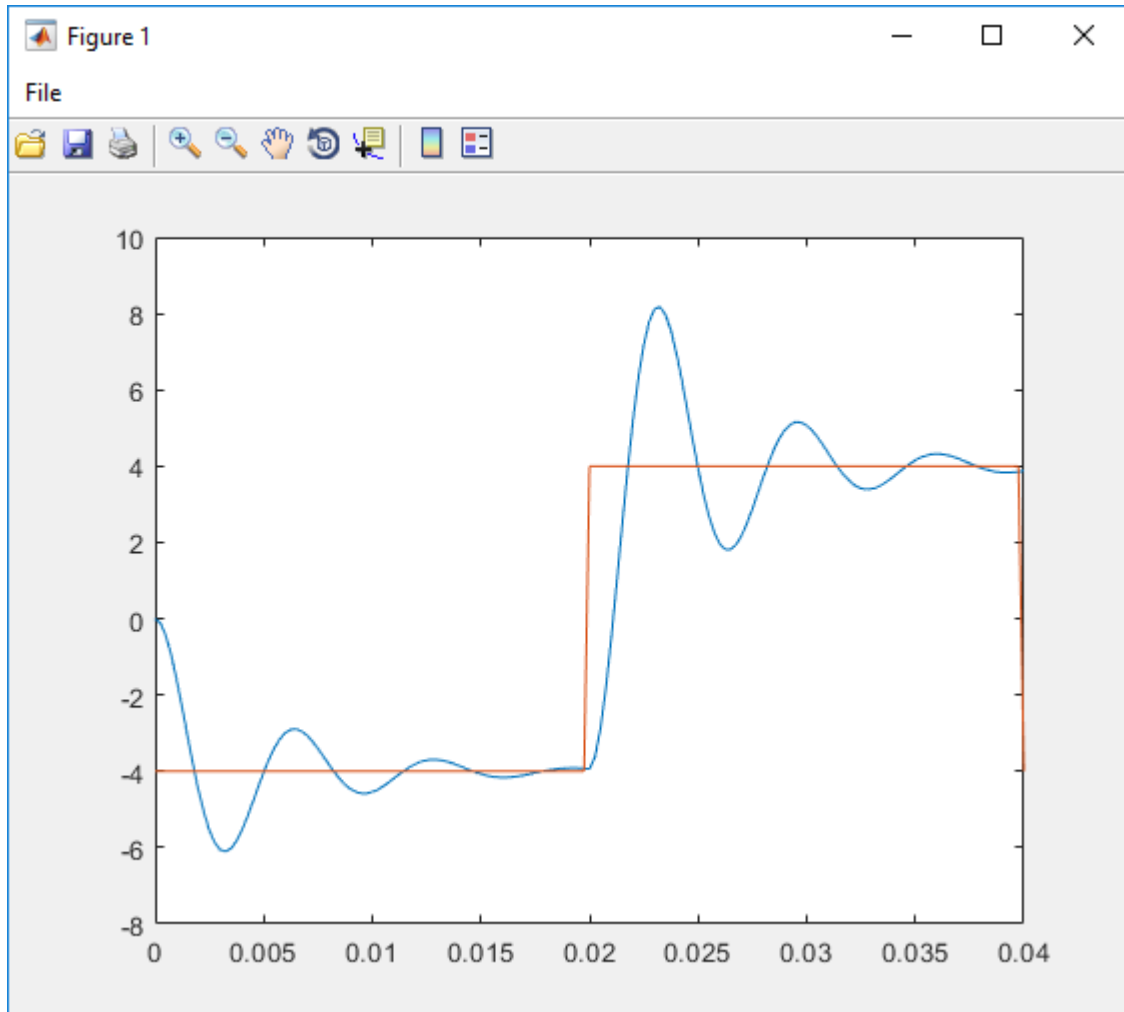
- 1 In Windows Explorer, navigate to `slrt_freq_sweep_test\for_redistribution_files_only`.
- 2 Copy the real-time application file (`xpcosc.mldatx`) into `slrt_freq_sweep_test\for_redistribution_files_only`.

The application assumes that the model file is in the folder where you run the application.

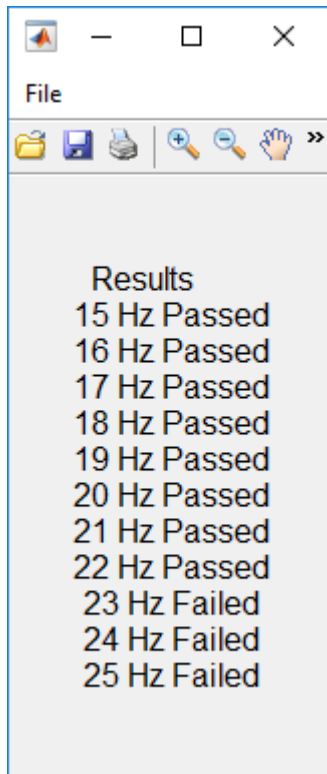
- 3 If you are connected to the target computer within MATLAB, close the connection. Use the `close(tg)` command.

- 4 To run the application, click `slrt_freq_sweep_test.exe`.

The application runs and displays a plot for each frequency.



After the run is complete, the application displays a text box containing the test results.



See Also

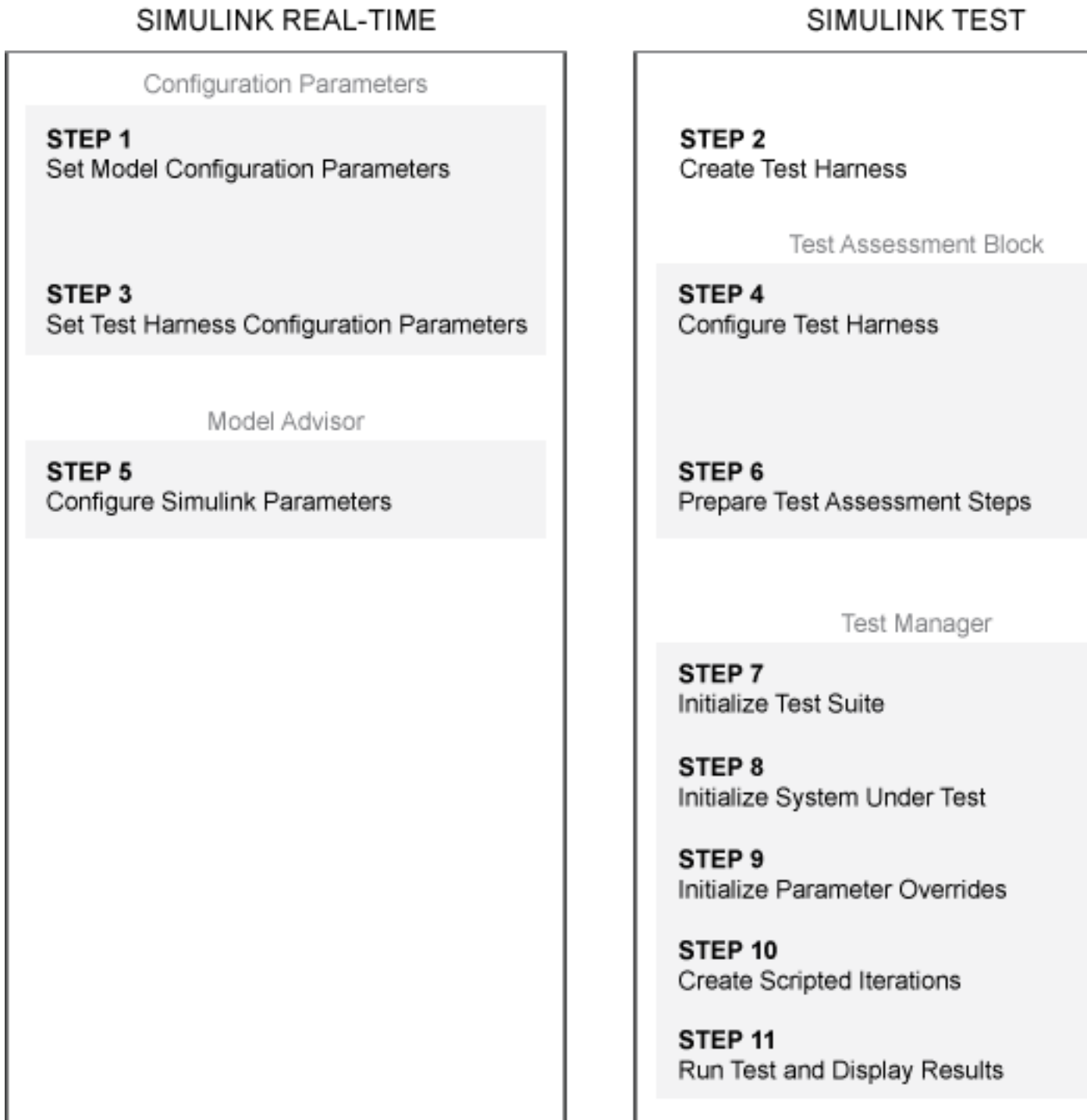
More About

- "Write Deployable MATLAB Code" (MATLAB Compiler)

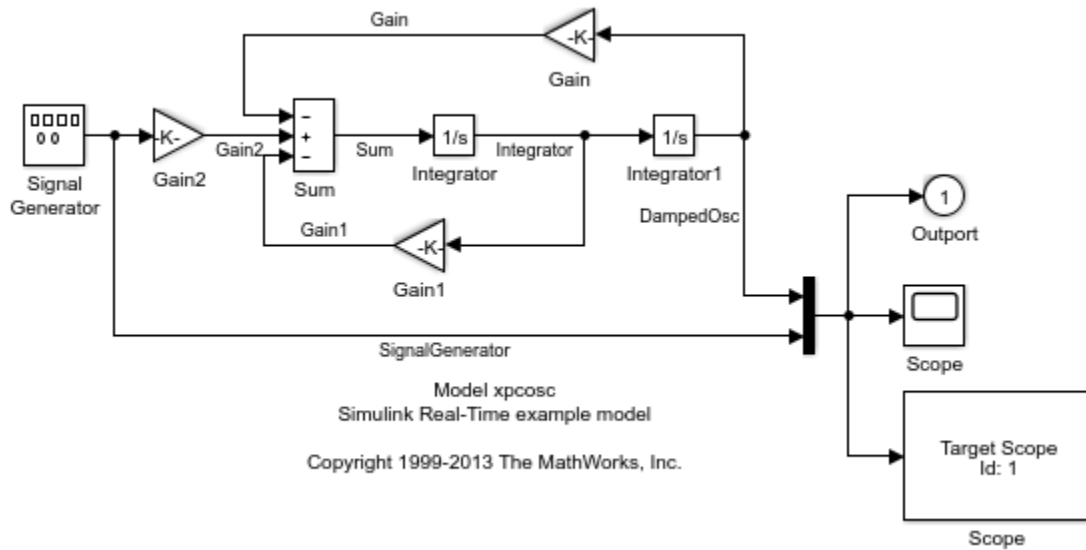
Automated Test with Simulink Test

Test Real-Time Application

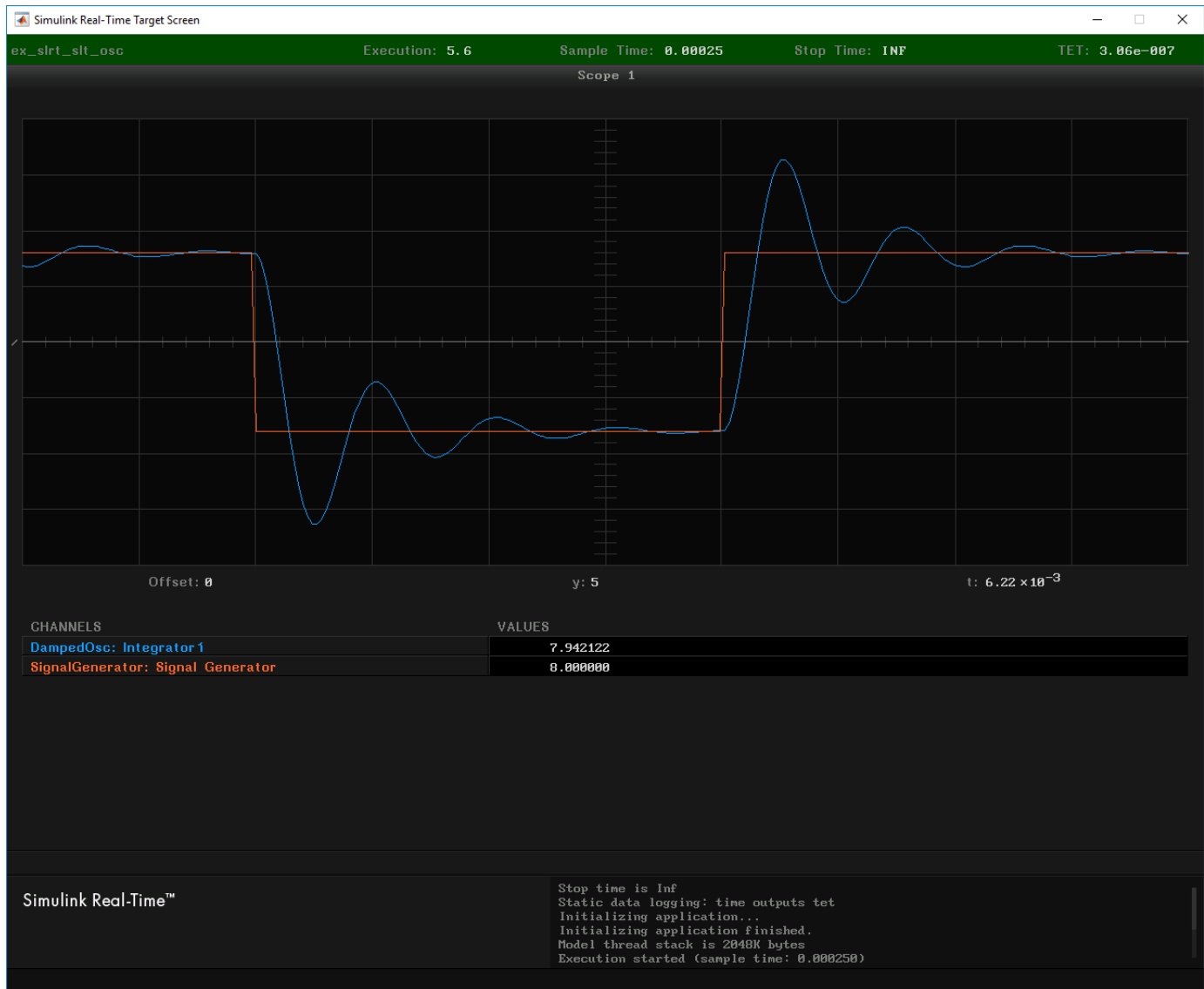
This example shows how to perform a frequency-response test of the model `ex_slrt_slrt_osc` (`open_system(docpath(fullfile(docroot, 'toolbox', 'xpc', 'examples', 'ex_slrt_slrt_osc')))`)).



Using this information, in the design phase, you can modify the internal parameters of the model to meet your frequency requirements. In the production phase, you can bin manufactured parts based on frequency response.



The figure shows representative output from a real-time application running on a target computer. At low frequencies, the output of the Integrator1 block settles to the same value as the output of the Signal Generator block. At high frequencies, the output of the Integrator1 block is still ringing at the end of each pulse.



The test determines the highest frequency at which the output values of the Integrator and Signal Generator blocks are within a specified criterion of each other. The test uses the model itself as a signal source and uses a test harness to compare the outputs of the Integrator and Signal Generator blocks.

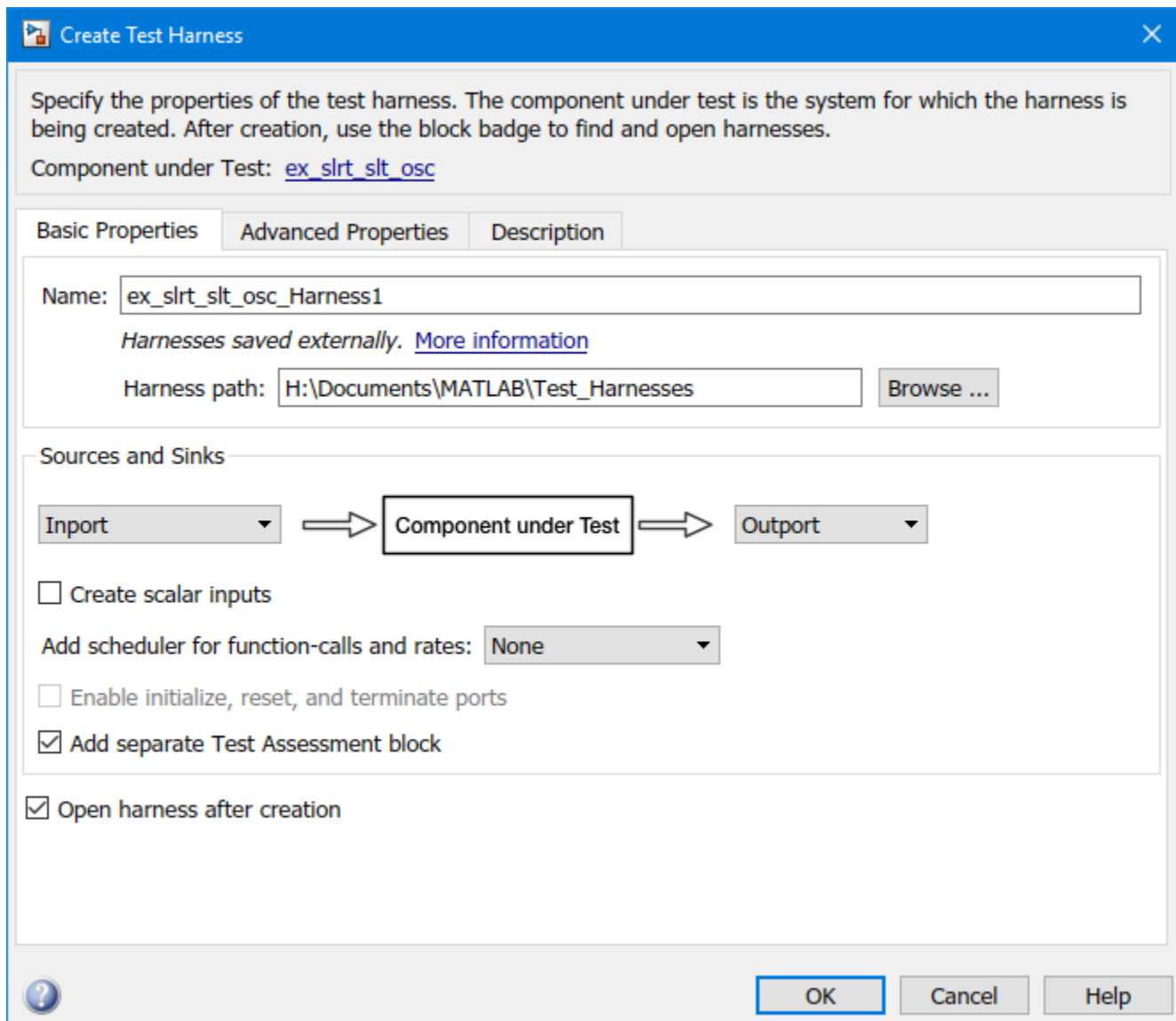
Step 1. Set Model Configuration Parameters

- 1 Open model `ex_slrt_slrt_osc` in a writable folder.
- 2 Open the Configuration Parameters. On the **Real-Time** tab, click **Hardware Settings**.
- 3 Select **Model Referencing** > **Total number of instances allowed per top model** > **One**.
- 4 Select **Data Import/Export** > **Format** > **Structure with time**.
- 5 Select **Data Import/Export** > **Format** > **Time**.
- 6 Select **Data Import/Export** > **Format** > **Output**.
- 7 De-select **Data Import/Export** > **Format** > **States**.

- 8 De-select **Data Import/Export > Format > Final states**.
- 9 De-select **Data Import/Export > Format > Signal logging**.
- 10 De-select **Data Import/Export > Format > Data stores**.
- 11 De-select **Data Import/Export > Format > Log Dataset data to file**.

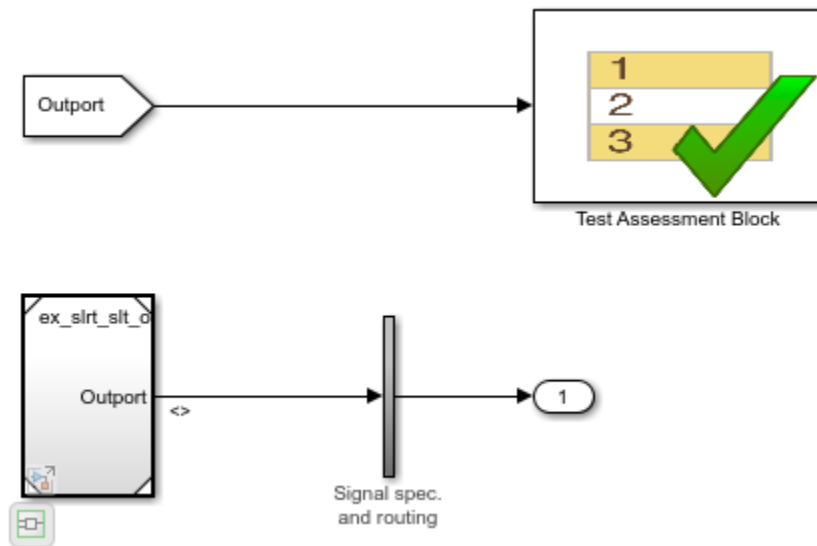
Step 2. Create Test Harness

- 1 On the **Apps** tab, click **Simulink Test**.
- 2 On the **Test** tab, click **Add Test Harness for Model**. The software creates a test harness with the default name `ex_slrt_slrt_osc_Harness1`.
- 3 In the **Basic Properties** pane, select the **Save Test Harnesses Externally** check box.
- 4 For the **Input to Component under Test**, select **None**.
- 5 For the **Output from Component under Test**, select **Output**.
- 6 Select the **Add separate assessment block** check box.
- 7 Select the **Open harness after creation** check box.
- 8 Take the defaults in the remaining panes.



8. Click **OK**.

The test harness looks like this figure.



The example model `ex_slrt_slit_osc` stores the test harness within the model. To access the test harness from the example model:

- 1 In Simulink Editor, on the **Test** tab, click **Manage Test Harnesses**.
- 2 Click `ex_slrt_slit_osc_Harness1`.
- 3 To return to the example model, select it in the perspectives view in the lower right corner of the test harness.

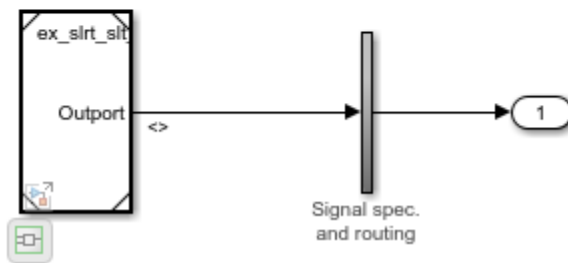
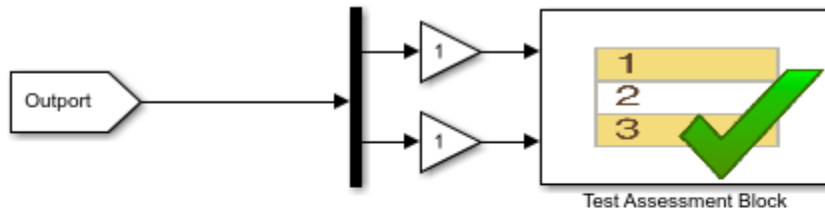
Step 3. Set Test Harness Configuration Parameters

- 1 Open test harness `ex_slrt_slit_osc_Harness1`.
- 2 Open the Configuration Parameters. On the **Real-Time** tab, click **Hardware Settings**.
- 3 Select **Model Referencing** > **Total number of instances allowed per top model** > **One**.
- 4 Select **Data Import/Export** > **Format** > **Structure with time**.
- 5 Select **Data Import/Export** > **Format** > **Time**.
- 6 Select **Data Import/Export** > **Format** > **Output**.
- 7 De-select **Data Import/Export** > **Format** > **States**.
- 8 De-select **Data Import/Export** > **Format** > **Final states**.
- 9 De-select **Data Import/Export** > **Format** > **Signal logging**.
- 10 De-select **Data Import/Export** > **Format** > **Data stores**.
- 11 De-select **Data Import/Export** > **Format** > **Log Dataset data to file**.

Step 4. Configure Test Harness

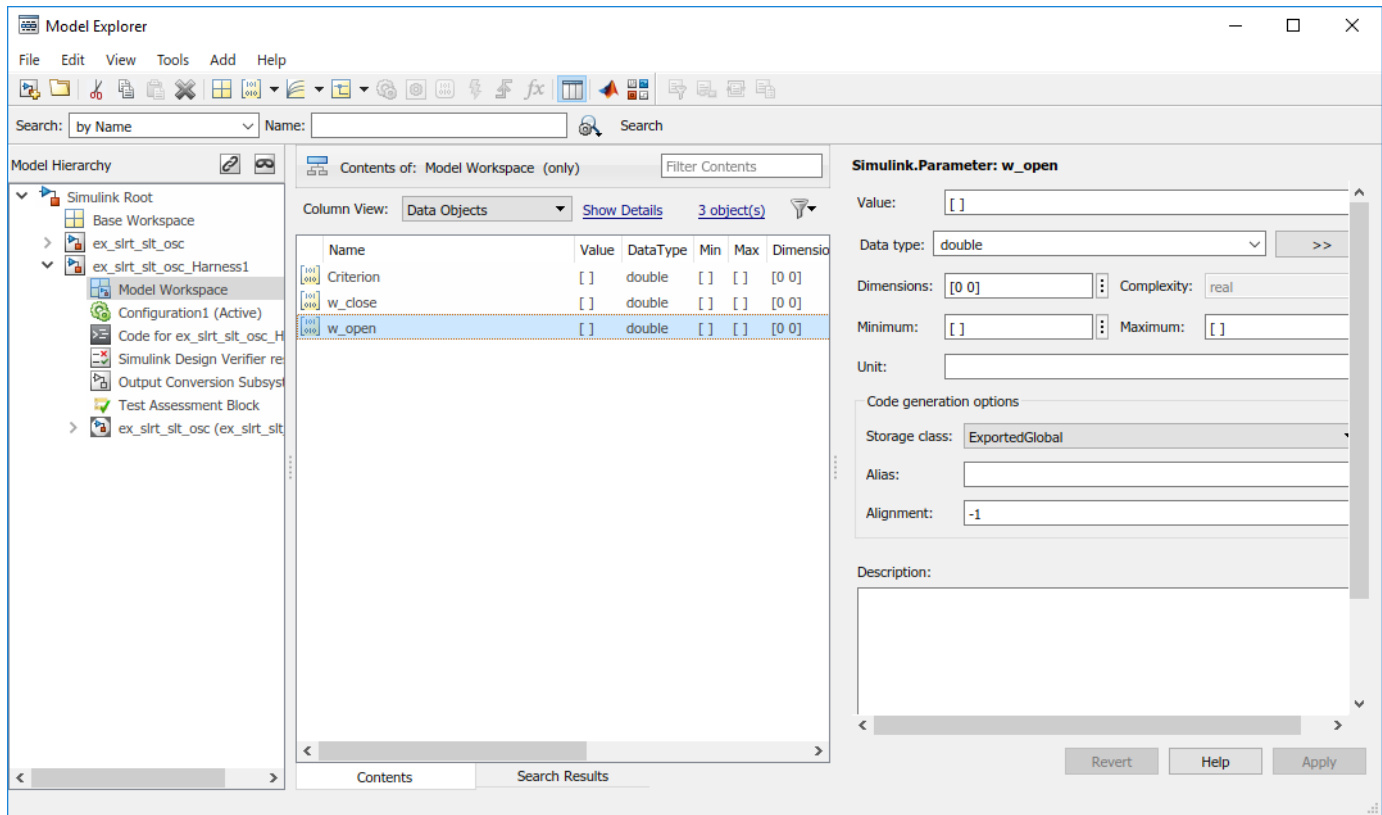
- 1 Open the Test Assessment block.
- 2 To simplify the test assessment configuration, in the **Input** symbol list, replace input `Output` with inputs `Int1` and `SigGen`.
- 3 In `ex_slrt_slit_osc_Harness1`, connect a Demux block to `ex_slrt_slit_osc/Output`.
- 4 In the Demux block dialog box, set **Number of outputs** to 2.

- 5 To make the Demux outputs visible to the Test Assessment block, connect unitary Gain blocks to each of the Demux block outputs.
- 6 Connect the top Demux block output to Test Assessment/Int1 and the bottom output to Test Assessment/SigGen.



Step 5. Configure Simulink Parameters

- 1 Open the Model Explorer. On the **Modeling** tab, pull down the **Design** section and click **Model Explorer**.
- 2 Click node **ex_slrt_slit_osc_Harness1 > Model Workspace**.
- 3 In the toolbar, click the **Add Simulink Parameter** button.
- 4 Add the following data object:
 - Name — Criterion
 - 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_slit_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.

The screenshot shows the Test Sequence Editor interface. The main window displays a test sequence with the following structure:

| Step | Transition | Next Step | Description |
|--|------------|-----------|---|
| CheckSetting | | | |
| Hi when (SigGen > 0) | | | Selects half of waveform |
| HiCheck when ((et >= w_open) && (et <= w_close)) verify((abs(Int1) >= abs(SigGen) * (1.0 - Criterion)) && ... (abs(Int1) <= abs(SigGen) * (1.0 + Criterion))); | | | Selects time window Tests acceptance criterion |
| DefaultStep_1 | | | Required for 'when decomposition' |
| Lo when (SigGen < 0) | | | |
| LoCheck when ((et >= w_open) && (et <= w_close)) verify((abs(Int1) >= abs(SigGen) * (1.0 - Criterion)) && ... (abs(Int1) <= abs(SigGen) * (1.0 + Criterion))); | | | |
| DefaultStep_2 | | | |
| DefaultStep | | | |

The left sidebar shows the Symbols section with Input variables: Int1 and SigGen. The Parameter section lists Criterion, w_close, and w_open. The Data Store Memory section is empty.

Step 7. Initialize Test Suite

- 1 Click on the ex_slrt_slrt_osc subsystem.
- 2 On the **Apps** tab, click **Simulink Test**.
- 3 On the **Test** tab, click **Test Manager**.

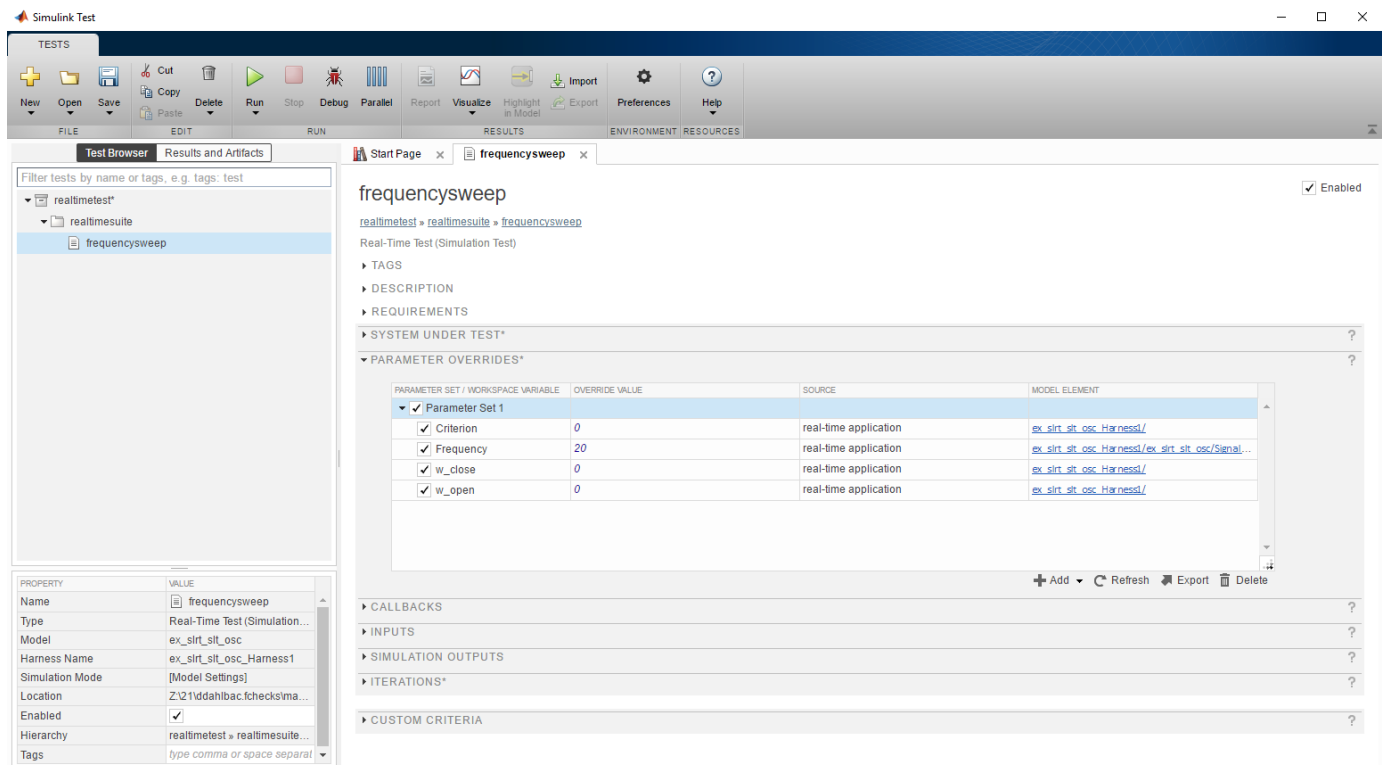
- 4 Select **New > Test File**.
- 5 Name the test file `realtimetest`.
- 6 Right-click the test file and select **New > Real-Time Test**.
- 7 In the new real-time test dialog box, enter `Simulation` in the **Test Type** field.
- 8 Click **Create**.
- 9 Rename the new test suite to `realtimesuite`.
- 10 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_slrt_osc`.
- 5 Set **Target Computer** to `TargetPC1`.
- 6 In tab **Test Harness**, set **Harness** to `ex_slrt_slrt_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_slrt_osc_Harness1` and `ex_slrt_slrt_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

To configure and control iterated runs of the test harness, a number of constants and variables provide input.

Test harness constants include:

- `cStartFreq = 15.0` Start frequency of parameter sweep.
- `cStopFreq = 25.0` End frequency of parameter sweep.
- `cFreqIncr = 1.0` Frequency increment.
- `cWOpen = 0.90` Start of time window for evaluating criterion.
- `cWClose = 0.99` End of time window for evaluating criterion.
- `cCriterion = 0.025` Maximum normalized amplitude difference between Signal Generator and Integrator1 within the time window.

Test harness variables include:

- `vfreq` Frequency at each iteration.
- `vw_open` Window opens once in each half-period.
- `vw_close` Window closes once in each half-period.

- 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;

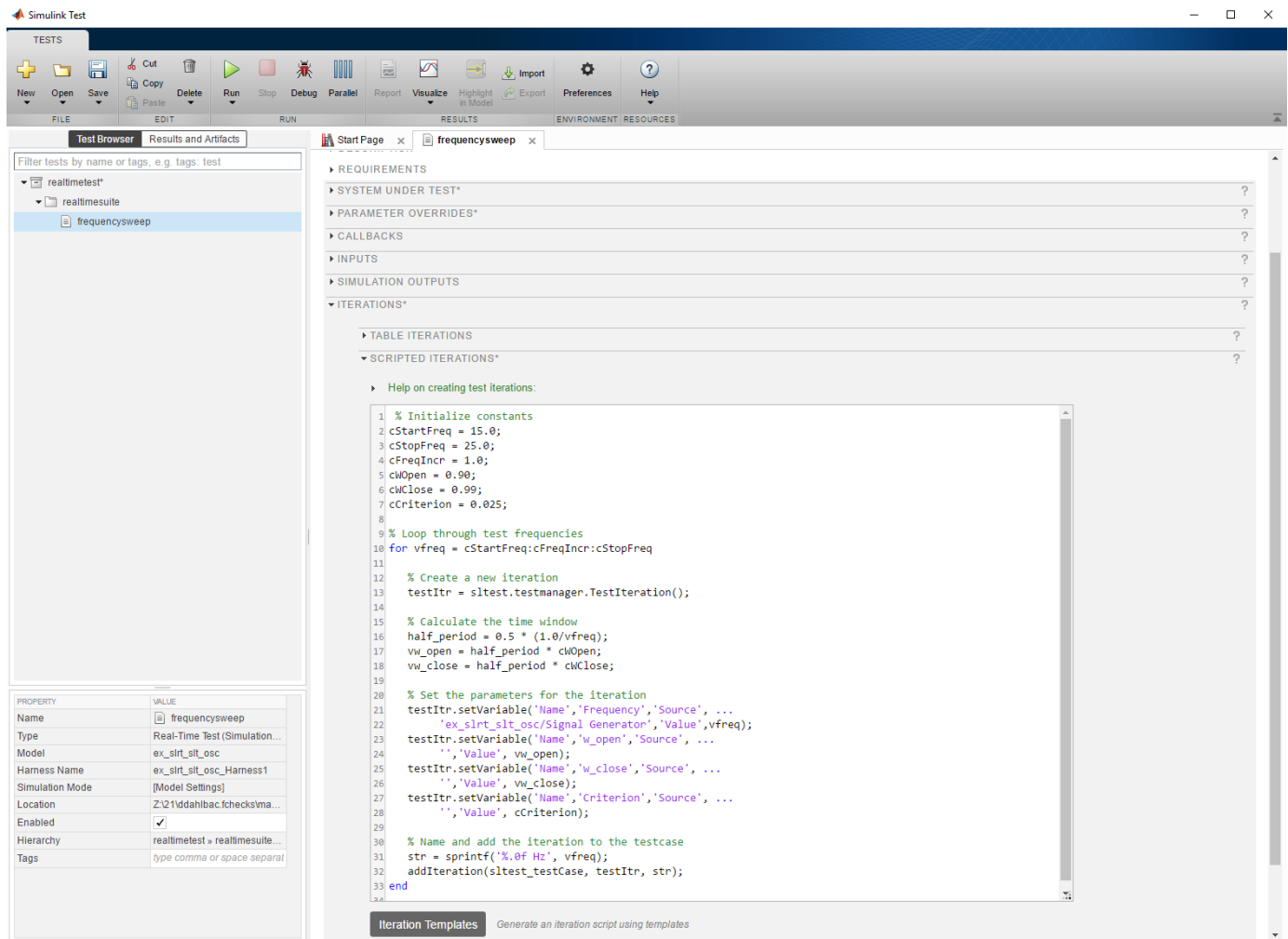
% Loop through test frequencies
for vfreq = cStartFreq:cFreqIncr:cStopFreq

    % 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_slc_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

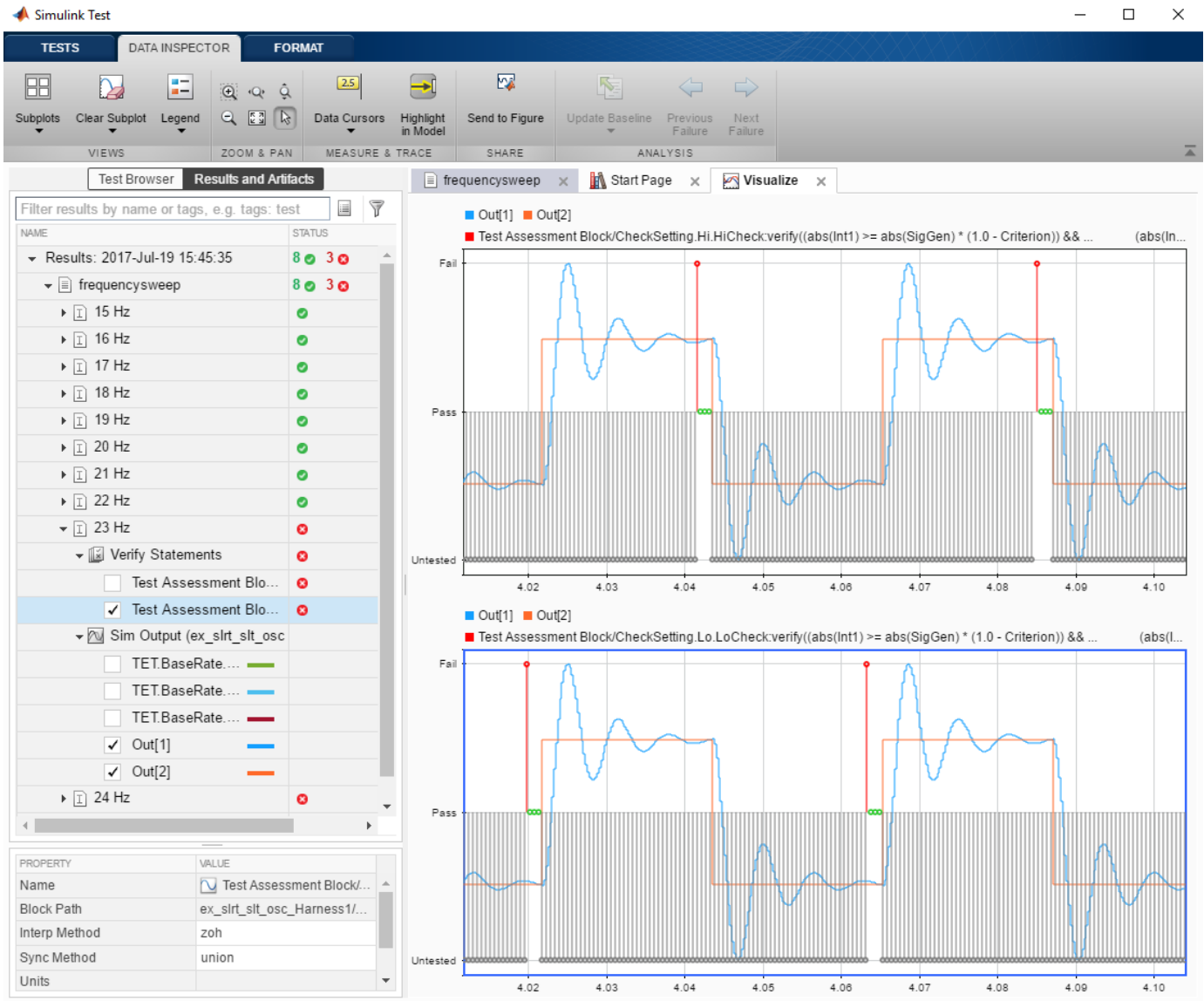
- 1 Build and download ex_slrt_slrt_osc to the target computer.
- 2 In Test Manager, click the **Run** button.
- 3 To view test results, in the left column, click **Results and Artifacts**. In this case, the test failed at iteration **23 Hz**.
- 4 To view the failing results, open nodes **23 Hz > Verify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

Step 12. Display Results

- 1 In the Simulation Data Inspector pane, select the **Layout** button.
- 2 Select two horizontal displays.
- 3 In the Simulation Data Inspector top 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.

- 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_slrt_osc')))`

- “Test Models in Real Time” (Simulink Test)
- “Reuse Desktop Test Cases for Real-Time Testing” (Simulink Test)

Troubleshooting

Simulink Real-Time Examples

Parameter Tuning and Data Logging

This example shows how to do real-time parameter tuning and data logging with Simulink® Real-Time™. After the script builds and downloads the oscillator model, `xpcosc`, to the target computer, it makes multiple runs with the gain 'Gain1/Gain' changed (tuned) before each run. The gain is swept from 0.1 to 0.7 in steps of 0.05. The parameter index of 'Gain1/Gain' is retrieved from the Simulink Real-Time target object using the function `GETPARAMID`.

The data logging capabilities of Simulink Real-Time are used to capture signals of interest during each run. The logged signals are uploaded to the development computer and plotted. Finally, a 3-D plot of the oscillator output vs. time vs. gain is displayed.

Check Connection Between Development and Target Computers

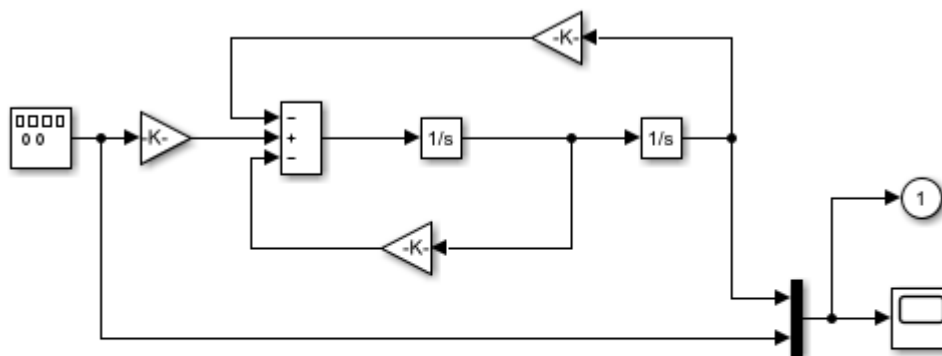
Use `slrtpingtarget` to test the connection between the development and target computers.

```
if ~strcmp(slrtpingtarget, 'success')
    error(message('xPCTarget:examples:Connection'));
end
```

Open, Build, and Download Model to the Target Computer

Open the oscillator model, `xpcosc`. Under the model's configuration parameter Simulink Real-Time option settings, the system target file has been set to `slrt.tlc`. Building the model will create an executable image, `xpcosc.mldatx`, that can be run on a computer booted with the Simulink Real-Time kernel.

```
open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcosc'));
```



Model `xpcosc`
Simulink Real-Time example model

Copyright 1999-2013 The MathWorks, Inc.

Build the model and download the image, `xpcosc.mldatx`, to the target computer.

- Configure for a non-Verbose build.
- Build and download application.

```
set_param('xpcosc', 'RTWVerbose', 'off');
rtwbuild('xpcosc');
```

```
tg = slrt('TargetPC1');
load(tg, 'xpcosc');
```

```
### Starting Simulink Real-Time build procedure for model: xpcosc
Warning: This model contains blocks that do not handle sample time
changes at runtime. To avoid incorrect results, only change
the sample time in the original model, then rebuild the model.
### Successful completion of build procedure for model: xpcosc
### Created MLDATX ..\xpcosc.mldatx
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
```

Run Model, Sweep 'Gain' Parameter, Plot Logged Data

This code accomplishes a number of tasks.

Task 1: Create Target Object

Create the MATLAB® variable, `tg`, containing the Simulink Real-Time target object. This object allows you to communicate with and control the target computer.

- Create a Simulink Real-Time target object.
- Set sample time to 250us.
- Set stop time to 0.2s.

Task 2: Run the Model and Plot Results

Run the model, sweeping through and changing the gain (damping parameter) before each run. Plot the results as you go.

- Get index of parameter 'Gain1/Gain'
- Does the plot figure exist?
- If no, create figure
- If yes, make it the current figure

Task 3: Loop over damping factor z

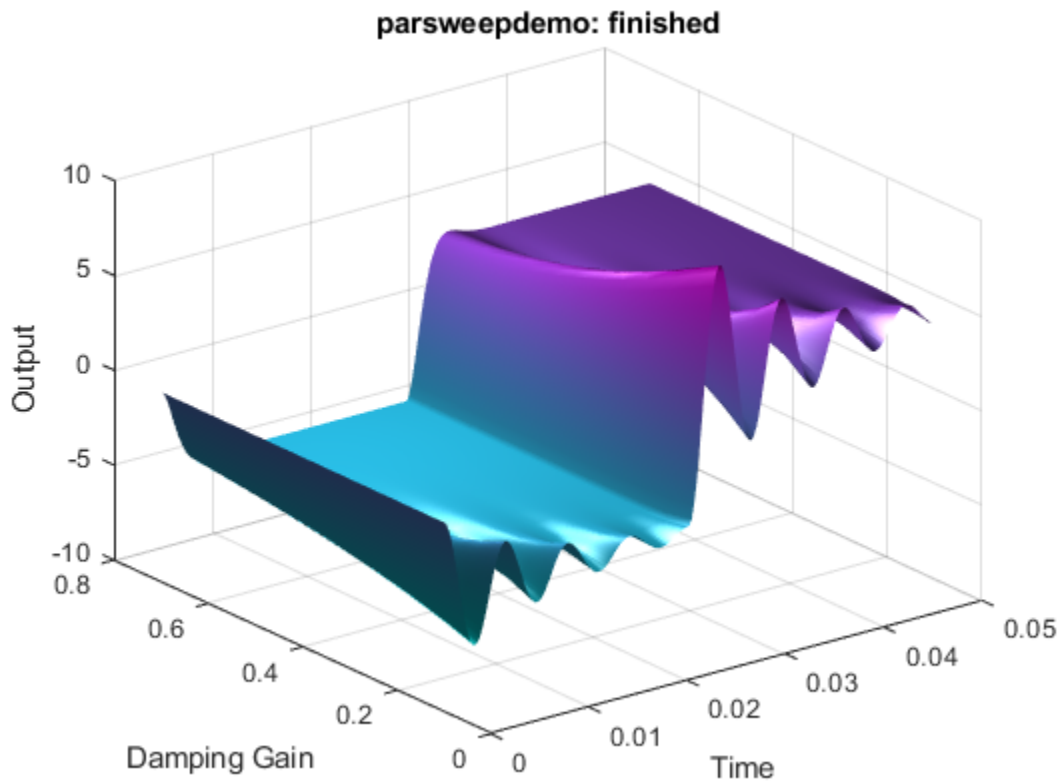
- Set damping factor (Gain1/Gain)
- Start model execution
- Upload output and store in a matrix
- Upload time vector
- Plot data for current run

Task 4: Create 3-D Plot (Oscillator Output vs. Time vs. Gain)

- Create a plot of oscillator output vs. time vs. gain.
- Create 3-D plot

```
tg = slrt; % create target object
tg.SampleTime = 0.000250;
tg.StopTime = 0.2;
tPar = getparamid(tg, 'Gain1', 'Gain'); % run the model
```

```
figh = findobj('Name', 'parsweepdemo');
if isempty(figh)
    figh = figure;
    set(figh, 'Name', 'parsweepdemo', 'NumberTitle', 'off');
else
    figure(figh);
end
y = []; flag = 0; % loop over damping factor
for z = 0.1 : 0.05 : 0.7
    if isempty(find(get(0, 'Children') == figh, 1)), flag = 1; break; end
    setparam(tg,tPar,2 * 1000 * z);
    start(tg);
    pause(2*tg.StopTime);
    outp = tg.OutputLog;
    y = [y, outp(:, 1)];
    t = tg.TimeLog;
    plot(t, outp(:, 1));
    set(gca, 'XLim', [t(1), t(end)], 'YLim', [-10, 10]);
    title(['parsweepdemo: Damping Gain = ', num2str(z)]);
    xlabel('Time'); ylabel('Output');
    drawnow;
end
if ~flag % create 3-D plot
    delete(gca);
    surf(t(1 : 200), 0.1 : 0.05 : 0.7, y(1 : 200, :));
    colormap cool
    shading interp
    h = light;
    set(h, 'Position', [0.0125, 0.6, 10], 'Style', 'local');
    lighting gouraud
    title('parsweepdemo: finished');
    xlabel('Time'); ylabel('Damping Gain'); zlabel('Output');
end
```



Close Model

When done, close the model.

```
close_system('xpcosc',0);
```

Signal Tracing With a Host Scope in Freerun Mode

This example shows how to do freerun signal tracing using an Simulink® Real-Time™ host scope. After the script builds and downloads the oscillator model, `xpcosc`, to the target computer, it adds a scope of type 'host' to the real-time application and the signals 'Integrator1' and 'Signal Generator' to the scope. The application is started and the host scope is used for data acquisition and display. Note:

- The model sample time is 250 usec.
- The scope is set to acquire 200 samples with a decimation factor of 4.
- This corresponds to a display length of $250e-6 * 200 * 4 = 0.2$ seconds.

The scope is started in Freerun mode, and its status is monitored until it reaches the 'Finished' state. Next, the scope data is uploaded to the development computer and plotted. This process repeats 25 times. After every fifth run, the damping gain 'Gain1/Gain' is set to a new random value.

Check Connection Between Development and Target Computers

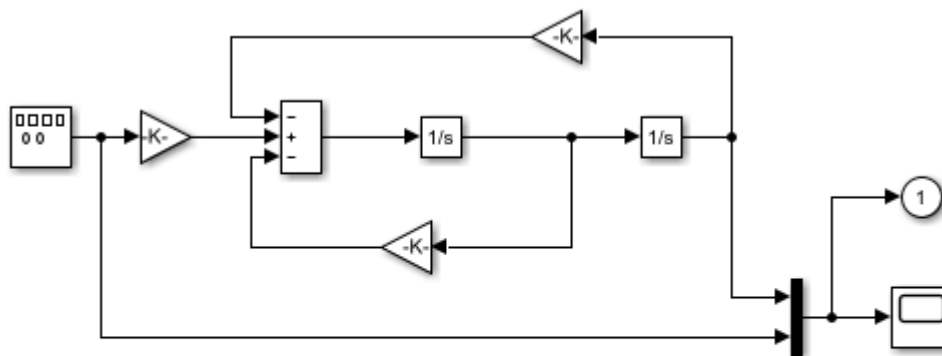
Use `slrtpingtarget` to test the connection between the development and target computers.

```
if ~strcmp(slrtpingtarget, 'success')
    error(message('xPCTarget:examples:Connection'));
end
```

Open, Build, and Download Model to the Target Computer

Open the oscillator model, `xpcosc`. Under the model's configuration parameter Simulink Real-Time option settings, the system target file has been set to `slrt.tlc`. Hence, building the model will create an executable image, `xpcosc.mldatx`, that can be run on a computer booted with the Simulink Real-Time kernel.

```
open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcosc'));
```



Model `xpcosc`
Simulink Real-Time example model
Copyright 1999-2013 The MathWorks, Inc.

Build the model and download the image, `xpcosc.mldatx`, to the target computer.

- Configure for a non-Verbose build.

- Build and download application.

```
set_param('xpcosc', 'RTWVerbose', 'off');
rtwbuild('xpcosc');
tg = slrt('TargetPC1');
load(tg, 'xpcosc');
```

```
### Starting Simulink Real-Time build procedure for model: xpcosc
Warning: This model contains blocks that do not handle sample time
changes at runtime. To avoid incorrect results, only change
the sample time in the original model, then rebuild the model.
### Successful completion of build procedure for model: xpcosc
### Created MLDATX ..\xpcosc.mldatx
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
```

Run Model, Randomize 'Gain' Parameter, Plot Host Scope Data

This code accomplishes a number of tasks.

Task 1: Create Target Object

Create the MATLAB® variable, `tg`, containing the Simulink Real-Time target object. This object allows you to communicate with and control the target computer.

- Create a Simulink Real-Time target object
- Set sample time to 250us
- Set stop time to a high value (10000s)
- Start model execution

Task 2: Create, configure, and plot to the host scope during each run.

- Get index of parameter 'Gain1/Gain'
- Get index of signal 'Integrator1'
- Get index of signal 'Signal Generator'
- Define (add) a host scope object
- Add signals to signal list of scope object
- Set number of samples
- Set decimation factor
- Set trigger mode

Task 3: Check for plot figure.

Does the plot figure exist?

- If no, create figure
- If yes, make it the current figure

Task 4: Loop to acquire 25 data packages from the scope object.

- Change parameter Gain1/Gain every fifth acquisition loop to a random value between 0 and 2000.

- Start scope object
- Wait until scope-object has 'finished' acquiring the data.

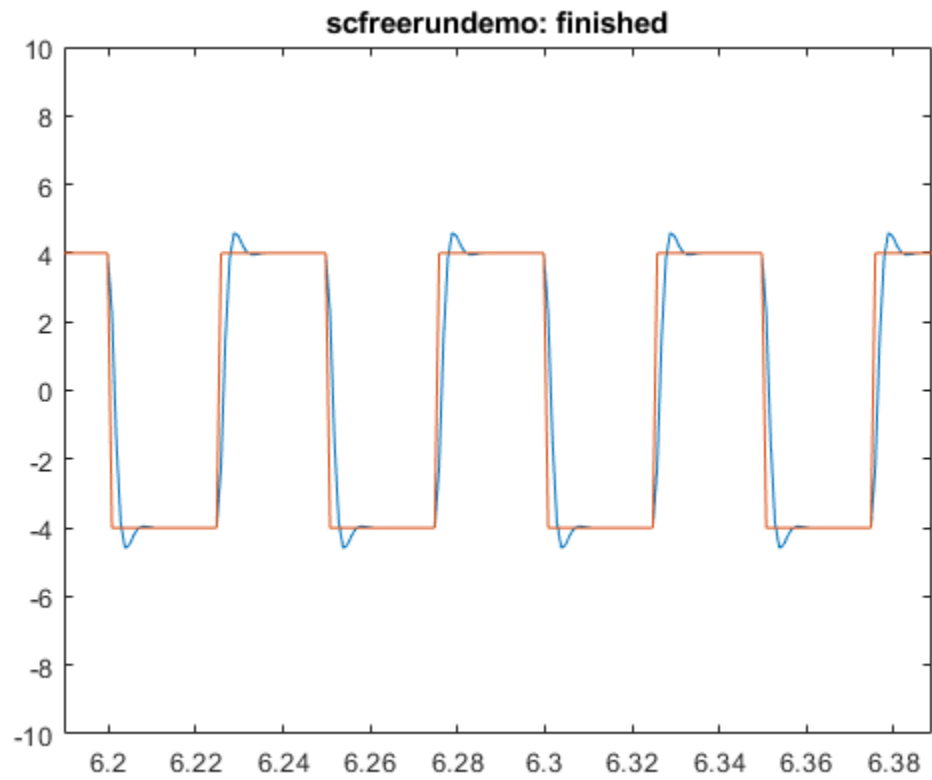
Task 5: Create time vector, upload scope data and display it.

- Upload time vector
- Upload acquired data and plot

```

tg = SimulinkRealTime.target; % create target object
tg.SampleTime = 0.000250;
tg.StopTime   = 10000;
start(tg);
tPar = getparamid(tg, 'Gain1', 'Gain'); % create host scope and plot
signals(1) = getsignalid(tg, 'Integrator1');
signals(2) = getsignalid(tg, 'Signal Generator');
sc = addscope(tg, 'host'); % define host scope object
addsignal(sc, signals);
sc.NumSamples = 200; % set number of samples and other settings
sc.Decimation = 4;
sc.TriggerMode = 'Freerun';
figh = findobj('Name', 'scfreerundemo'); % check for plot figure
if isempty(figh)
    figh = figure;
    set(figh, 'Name', 'scfreerundemo', 'NumberTitle', 'off');
else
    figure(figh);
end;
m = 1; flag = 0; % loop to acquire data
for n = 1 : 25
    if isempty(find(get(0, 'Children') == figh, 1)), flag = 1; break; end
    if ~m
        setparam(tg, tPar, 2*1000*rand); % change gain periodically
    end
    m = rem(m + 1, 5);
    start(sc); % start scope object
    while ~strcmpi(sc.Status, 'finished')
    end; % wait until scope is finished
    t = sc.Time; % create time vector and display it
    plot(t, sc.Data);
    title(['scfreerundemo: ', num2str(n), ' of 25 data packages']);
    set(gca, 'XLim', [t(1), t(end)]);
    set(gca, 'YLim', [-10, 10]);
    drawnow;
end
if ~flag, title('scfreerundemo: finished'); end

```

Stop and Close Model

When done, stop the application and close the model.

- Stop model
- Close model

```
stop(tg);  
close_system('xpcosc',0);
```

Signal Tracing Using Software Triggering

This example shows how to trace a signal using a software triggered Simulink® Real-Time™ host scope. After the script builds and downloads the oscillator model, `xpcosc`, to the target computer, it adds a scope of type 'host' to the real-time application and the signals 'Integrator1' and 'Signal Generator' to the scope. The scope is then configured in the software trigger mode.

Once the trigger is initiated, the scope is monitored to determine when its data acquisition is complete. Next, the scope data is uploaded to the development computer and plotted. This process repeats 25 times. The software trigger is re-enabled each run after a random pause (between 0 and 4 seconds). After every fifth run, the damping gain 'Gain1/Gain' is set to a new random value (between 0 and 2000).

Check Connection Between Development and Target Computers

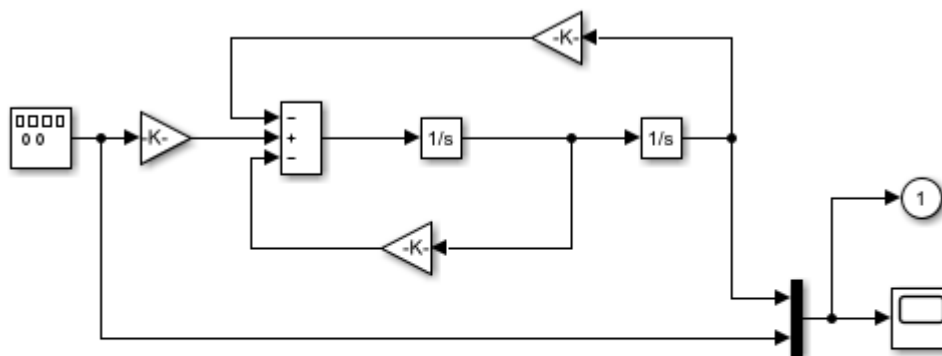
Use `slrtpingtarget` to test the connection between the development and target computers.

```
if ~strcmp(slrtpingtarget, 'success')
    error(message('xPCTarget:examples:Connection'));
end
```

Open, Build, and Download Model to the Target Computer

Open the oscillator model, `xpcosc`. Under the model's configuration parameter Simulink Real-Time option settings, the system target file has been set to `slrt.tlc`. Hence, building the model will create an executable image, `xpcosc.mldatx`, that can be run on a target computer booted with the Simulink Real-Time kernel.

```
open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcosc'));
```



Model `xpcosc`
Simulink Real-Time example model

Copyright 1999-2013 The MathWorks, Inc.

Build the model and download the image, `xpcosc.mldatx`, to the target computer.

- Configure for a non-Verbose build.
- Build and download application.

```

set_param('xpcosc', 'RTWVerbose', 'off');
rtwbuild('xpcosc');
tg = slrt('TargetPC1');
load(tg, 'xpcosc');

### Starting Simulink Real-Time build procedure for model: xpcosc
Warning: This model contains blocks that do not handle sample time
changes at runtime. To avoid incorrect results, only change
the sample time in the original model, then rebuild the model.
### Successful completion of build procedure for model: xpcosc
Warning: Unable to configure logging for model 'xpcosc' because it is already
configured.
### Created MLDATX ..\xpcosc.mldatx
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

```

Run Model, Randomize 'Gain' Parameter, Plot Host Scope Data

This code accomplishes a number of tasks.

Task 1: Create Target Object

Create the MATLAB® variable, `tg`, containing the Simulink Real-Time target object. This object allows you to communicate with and control the target computer.

- Create a Simulink Real-Time target object
- Set sample time to 250us
- Set stop time to a high value (10000s)
- Start model execution

Task 2: Create, configure, and plot to the host scope during each run.

- Get index of parameter 'Gain1/Gain'
- Get index of signal 'Integrator1'
- Get index of signal 'Signal Generator'
- Define (add) a host scope object
- Add signals to signal list of scope object
- Set number of samples
- Set decimation factor
- Set trigger mode

Task 3: Check for Plot Figure

Does the plot figure exist?

- If no, create figure
- If yes, make it the current figure

Task 4: Loop to acquire 25 data packages from the scope object.

Change parameter Gain1/Gain every fifth acquisition loop to a random value between 0 and 2000.

Task 5: Start scope object and trigger randomly

- Start scope object
- Wait until scope object has 'ready' state.
- Randomize the amount of wait time before triggering scope.
- Wait a random period (0..4s).
- Software trigger the scope object.
- Wait until scope-object has 'finished' state.

Task 6: Create time vector, upload scope data and display it.

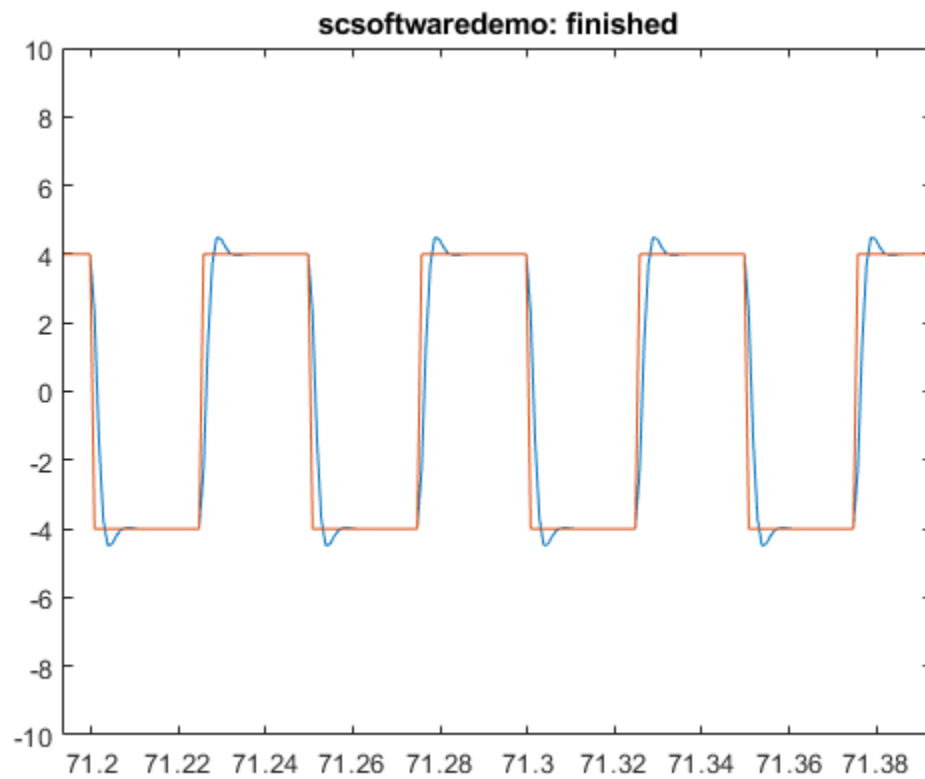
- Upload time vector
- Upload acquired data and plot

```

tg = slrt; % create target object
tg.SampleTime = 0.000250;
tg.StopTime = 10000;
start(tg);
tPar = getparamid(tg, 'Gain1', 'Gain'); % get indexes
signals(1) = getsignalid(tg, 'Integrator1');
signals(2) = getsignalid(tg, 'Signal Generator');
sc = addscope(tg, 'host'); % define scope object
addsignal(sc, signals);
sc.NumSamples = 200;
sc.Decimation = 4;
sc.TriggerMode = 'Software';
figh = findobj('Name', 'scsoftwaredemo'); % check for plot figure
if isempty(figh)
    figh = figure;
    set(figh, 'Name', 'scsoftwaredemo', 'NumberTitle', 'off');
else
    figure(figh);
end
m = 1; flag = 0; % loop to acquire data
for n = 1 : 25
    if isempty(find(get(0, 'Children') == figh, 1)), flag = 1; break; end
    if ~m
        setparam(tg, tPar, 2*1000*rand);
    end
    m = rem(m + 1, 5);
    start(sc); % start scope object
    while ~strcmp(sc.Status, 'Ready for being Triggered'), end
    ttrigger = rand * 4; % randomized trigger time
    title(['scsoftwaredemo: ', num2str(n), ...
        ' of 25 data packages, will be triggered in ', ...
        num2str(ttrigger), 's']);
    pause(ttrigger);
    if isempty(find(get(0, 'Children') == figh, 1)), flag = 1; break; end
    trigger(sc); % software trigger scope
    while ~strcmp(sc.Status, 'finished')
    end
    t = sc.Time; % create time vector and display it
    plot(t, sc.Data);
    set(gca, 'XLim', [t(1), t(end)], 'YLim', [-10, 10]);
    drawnow;

```

```
end  
if ~flag, title('scsoftwaredemo: finished'); end
```



Stop and Close Model

When done, stop the application and close the model.

- Stop model
- Close model

```
stop(tg);  
close_system('xpcosc',0);
```

Signal Tracing Using Signal Triggering

This example shows how to trace signals using a signal triggered Simulink® Real-Time™ host scope. After the script builds and downloads the oscillator model, `xpcosc`, to the target computer, it adds a scope of type 'host' to the real-time application and the signals 'Integrator1' and 'Signal Generator' to the scope. The scope is then configured to trigger on the signal 'Signal Generator' when it reaches 0.0 on a rising slope (signal value goes negative to positive).

Once the trigger condition is met, the scope is monitored to determine when its data acquisition is complete. Next, the scope data is uploaded to the development computer and plotted. This process repeats 25 times. After every fifth run, the damping gain 'Gain1/Gain' is set to a new random value (between 0 and 2000).

Check Connection Between Development and Target Computers

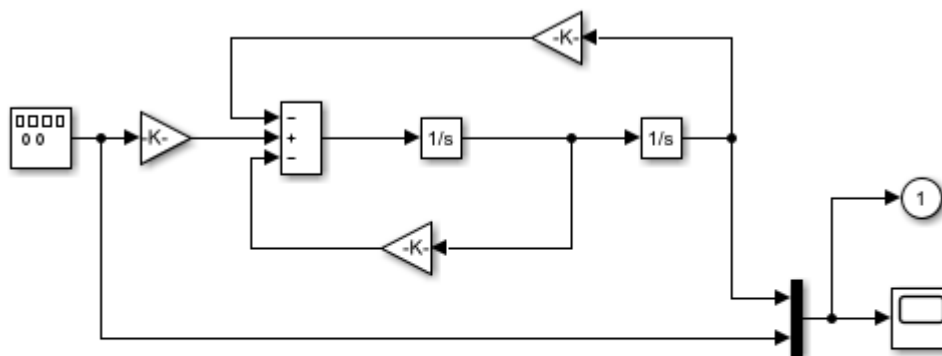
Use `slrtpingtarget` to test the connection between the development and target computers.

```
if ~strcmp(slrtpingtarget, 'success')
    error(message('xPCTarget:examples:Connection'));
end
```

Open, Build, and Download Model to the Target Computer

Open the oscillator model `xpcosc`. The model has been configured to build for Simulink Real-Time, and building the model creates an executable image, `xpcosc.mldatx`, that can be run on a target booted with the Simulink Real-Time kernel.

```
open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcosc'));
```



Model `xpcosc`
Simulink Real-Time example model
Copyright 1999-2013 The MathWorks, Inc.

Build the model and download the image, `xpcosc.mldatx`, to the target computer.

- Configure for a non-Verbose build.
- Build and download application.

```
set_param('xpcosc', 'RTWVerbose', 'off');
rtwbuild('xpcosc');
```

```
tg = slrt('TargetPC1');
load(tg, 'xpcosc');
```

```
### Starting Simulink Real-Time build procedure for model: xpcosc
Warning: This model contains blocks that do not handle sample time
changes at runtime. To avoid incorrect results, only change
the sample time in the original model, then rebuild the model.
### Successful completion of build procedure for model: xpcosc
### Created MLDATX ..\xpcosc.mldatx
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
```

Run Model, Randomize 'Gain' Parameter, Plot Host Scope Data

This code accomplishes a number of tasks.

Task 1: Create Target Object

Create the MATLAB® variable, `tg`, containing the Simulink Real-Time target object. This object allows you to communicate with and control the target computer.

- Create a Simulink Real-Time Object
- Set sample time to 250us
- Set stop time to a high value (10000s)
- Start model execution

Task 2: Create, configure, and plot to the host scope during each run.

- Get index of parameter 'Gain1/Gain'
- Get index of signal 'Integrator1'
- Get index of signal 'Signal Generator'
- Define (add) a host scope object
- Add signals to signal list of scope object
- Set number of samples
- Set decimation factor
- Set trigger mode
- Set trigger signal to 'Signal Generator'
- Set trigger level
- Set trigger slope

Task 3: Check for Plot Figure

Does the plot figure exist?

- If no, create figure
- If yes, make it the current figure

Task 4: Loop to acquire 25 data packages from the scope object.

Change parameter Gain1/Gain every fifth acquisition loop to a random value between 0 and 2000.

- Start scope object
- Wait until scope object has 'finished' state.

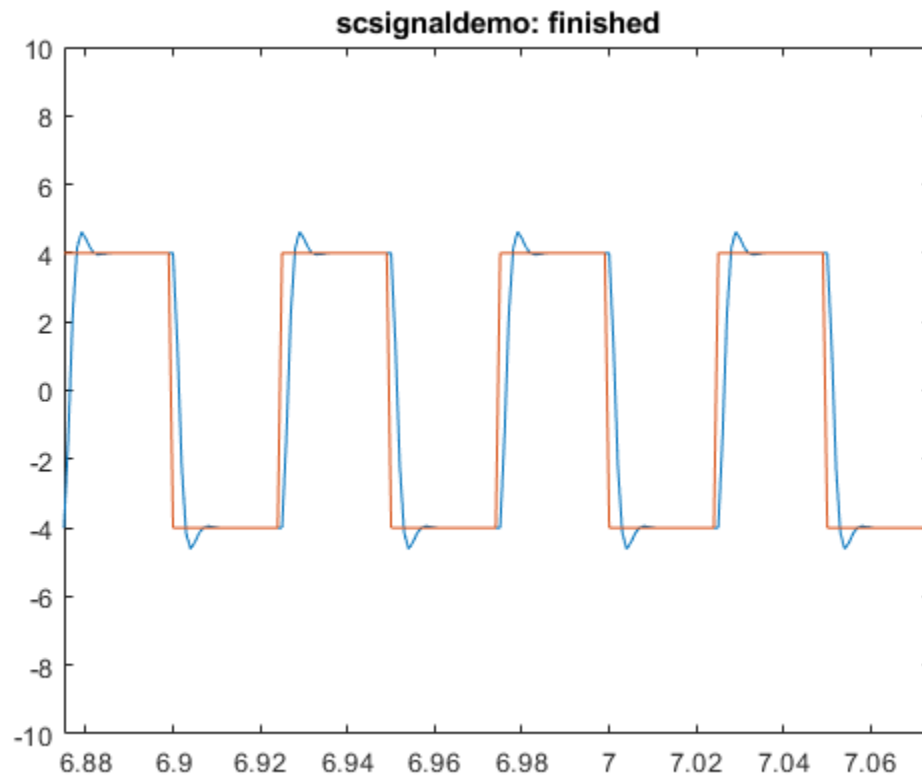
Task 5: Create time vector, upload scope data and display it.

- Upload time vector
- Upload acquired data and plot

```

tg = slrt; % create target object
tg.SampleTime = 0.000250;
tg.StopTime = 10000;
start(tg);
tPar = getparamid(tg, 'Gain1', 'Gain'); % create host scope and plot
signals(1) = getsignalid(tg, 'Integrator1');
signals(2) = getsignalid(tg, 'Signal Generator');
sc = addscope(tg, 'host'); % define scope object
addsignal(sc, signals);
sc.NumSamples = 200; % set number of samples and other settings
sc.Decimation = 4;
sc.TriggerMode = 'Signal';
sc.TriggerSignal = signals(2);
sc.TriggerLevel = 0.0;
sc.TriggerSlope = 'Rising';
figh = findobj('Name', 'scsignaldemo');
if isempty(figh)
    figh = figure; set(figh, 'Name', 'scsignaldemo', 'NumberTitle', 'off');
else
    figure(figh);
end
figh = findobj('Name', 'scsignaldemo'); % check for plot figure
if isempty(figh)
    figh = figure;
    set(figh, 'Name', 'scsignaldemo', 'NumberTitle', 'off');
else
    figure(figh);
end
m = 1; flag = 0; % loop to acquire data
for n = 1 : 25
    if isempty(find(get(0, 'Children') == figh, 1)), flag = 1; break; end
    if ~m
        setparam(tg, tPar, 2*1000*rand);
    end
    m = rem(m + 1, 5);
    start(sc); % start scope object
    while ~strcmpi(sc.Status, 'finished'), end;
    t = sc.Time; % create time vector and display it
    plot(t, sc.Data);
    title(['scsignaldemo: ', num2str(n), ' of 25 data packages']);
    set(gca, 'XLim', [t(1), t(end)], 'YLim', [-10, 10]);
    drawnow;
end
if ~flag, title('scsignaldemo: finished'); end

```

Stop and Close Model

When done, stop the application and close the model.

- Stop model
- Close model

```
stop(tg);  
close_system('xpcosc',0);
```

Signal Tracing Using Scope Triggering

This example shows how to trace signals with a scope triggered Simulink® Real-Time™ host scope. After the script builds and downloads the oscillator model, `xpcosc`, it adds two scopes of type 'host' to the real-time application. The first scope is configured to trigger on the signal 'Signal Generator' (the only signal added to this scope). The 'Integrator1' signal is also added to the second scope. Scope 2 is configured to be triggered by the first scope (i.e., it is triggered at the same time the first scope is triggered). This synchronizes the scopes.

Next, the scopes are started and monitored to determine when data acquisition is complete. Data from both scopes are then uploaded to the development computer and plotted. Although both scopes begin data acquisition at the same time, Scope 2 acquires data over a longer time record by increasing the decimation factor from 4 to 10. This process repeats 25 times. After every fifth run, the damping gain 'Gain1/Gain' is set to a new random value (between 0 and 2000).

Check Connection Between Development and Target Computers

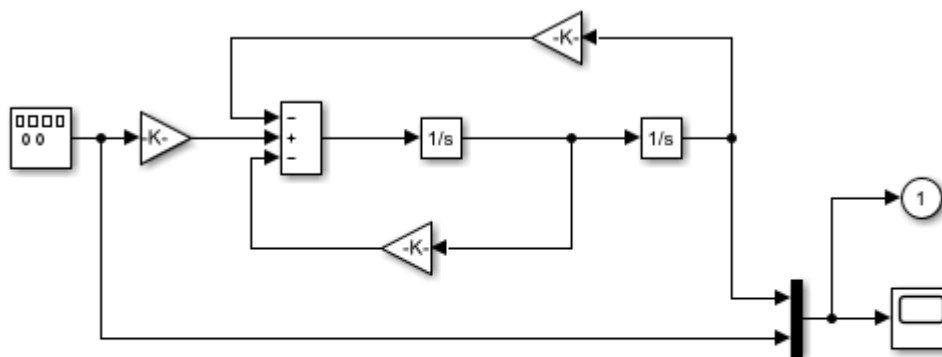
Use `slrtpingtarget` to test the connection between the development and target computers.

```
if ~strcmp(slrtpingtarget, 'success')
    error(message('xPCTarget:examples:Connection'));
end
```

Open, Build, and Download Model to the Target Computer

Open the oscillator model, `xpcosc`. Under the model's configuration parameter Simulink Real-Time option settings, the system target file has been set to `slrt.tlc`. Hence, building the model will create an executable image, `xpcosc.mldatx`, that can be run on a computer booted with the Simulink Real-Time kernel.

```
open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcosc'));
```



Model `xpcosc`
 Simulink Real-Time example model
 Copyright 1999-2013 The MathWorks, Inc.

Build the model and download the image, `xpcosc.mldatx`, to the target computer.

- Configure for a non-Verbose build.

- Build and download application.

```
set_param('xpcosc', 'RTWVerbose', 'off');
rtwbuild('xpcosc');
tg = slrt('TargetPC1');
load(tg, 'xpcosc');
```

```
### Starting Simulink Real-Time build procedure for model: xpcosc
Warning: This model contains blocks that do not handle sample time
changes at runtime. To avoid incorrect results, only change
the sample time in the original model, then rebuild the model.
### Successful completion of build procedure for model: xpcosc
### Created MLDATX ..\xpcosc.mldatx
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
```

Run Model, Randomize 'Gain' Parameter, Plot Host Scope Data

This code accomplishes a number of tasks.

Task 1: Create Target Object

Create the MATLAB® variable, `tg`, containing the Simulink Real-Time target object. This object allows you to communicate with and control the target computer.

- Create a Simulink Real-Time target object
- Set sample time to 250us
- Set stop time to a high value
- Start model execution

Task 2: Create, configure, and plot to the host scope during each run.

- Get index of parameter 'Gain1/Gain'
- Get index of signal 'Integrator1'
- Get index of signal 'Signal Generator'
- Define (add) first host scope object
- Define (add) second host scope object

Task 3: Set properties of first scope object

- Add 'Signal Generator' to signal list
- Set number of samples
- Set decimation factor
- Set trigger mode
- Set trigger signal to 'Signal Generator'
- Set trigger level
- Set trigger slope

Task 4: Set properties of second scope object

- Add both signals to signal list
- Set number of samples
- Set decimation factor
- Set trigger mode
- Set trigger scope to first scope object

Task 5: Check for Plot Figure

Does the plot figure exist?

- If no, create figure
- If yes, make it the current figure

Task 7: Loop to acquire 25 data packages from the scope object.

- Change parameter Gain1/Gain every fifth acquisition loop to a random value between 0 and 2000.
- Start second scope (waits until triggered by first scope)
- Start first scope
- Wait until both scope objects have 'finished' state.

Task 8: Scope Objects Display Time Vector

First scope object: create time vector, upload scope data and display it.

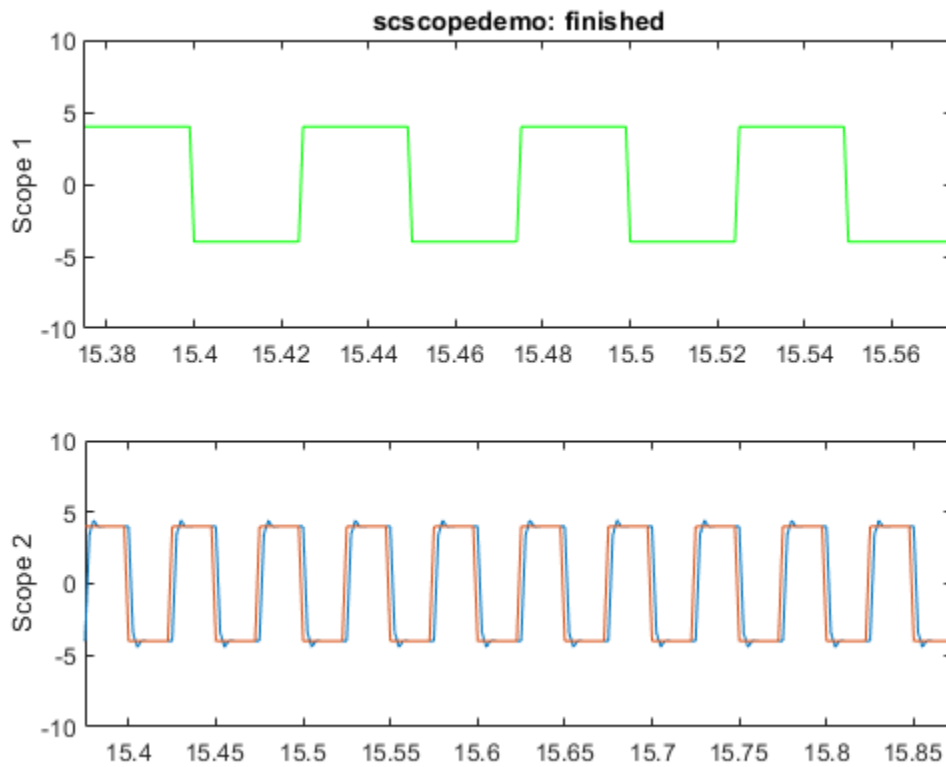
- Upload time vector
- Upload acquired data and plot

Second scope object: create time vector, upload scope data and display it.

- Upload time vector
- Upload acquired data and plot

```
tg = SimulinkRealTime.target; % create target object
tg.SampleTime = 0.000250;
tg.StopTime   = 10000;
start(tg);
tPar = getparamid(tg, 'Gain1', 'Gain'); % get indexes
signals(1) = getsignalid(tg, 'Integrator1');
signals(2) = getsignalid(tg, 'Signal Generator');
scs(1) = addscope(tg, 'host'); % add scopes
scs(2) = addscope(tg, 'host');
addsignal(scs(1), signals(2)); % set scope object properties
scs(1).NumSamples   = 200;
scs(1).Decimation   = 4;
scs(1).TriggerMode  = 'Signal';
scs(1).TriggerSignal = signals(2);
scs(1).TriggerLevel = 0.0;
scs(1).TriggerSlope = 'Rising';
addsignal(scs(2), signals(1)); % set scope object properties
scs(2).NumSamples   = 200;
scs(2).Decimation   = 10;
scs(2).TriggerMode  = 'Scope';
scs(2).TriggerScope = scs(1).ScopeId;
```

```
figh = findobj('Name', 'scscopedemo'); % check for plot figure
if isempty(figh)
    figh = figure;
    set(figh, 'Name', 'scscopedemo', 'NumberTitle', 'off');
else
    figure(figh);
end
m = 1; flag = 0; % loop to acquire data
for n = 1 : 25
    if isempty(find(get(0, 'Children') == figh, 1)), flag = 1; break; end
    if ~m
        setparam(tg, tPar, 2*1000*rand);
    end
    m = rem(m + 1, 5);
    scs(2).start;
    scs(1).start;
    while ~strcmpi(scs(1).Status, 'finished') || ...
        ~strcmpi(scs(2).Status, 'finished')
    end % wait for scope objects to finish
    subplot(2, 1, 1); % first scope object displays time vector
    t1 = scs(1).Time;
    plot(t1, scs(1).Data, 'g');
    set(gca, 'XLim', [t1(1), t1(end)], 'YLim', [-10, 10]); ylabel('Scope 1');
    title(['scscopedemo: ', num2str(n), ' of 25 data packages']);
    subplot(2,1,2); % second scope object displays time vector
    t2 = scs(2).Time;
    plot(t2, scs(2).Data);
    set(gca, 'XLim', [t2(1), t2(end)], 'YLim', [-10, 10]); ylabel('Scope 2');
    drawnow;
end
if ~flag
    subplot(2, 1, 1);
    title('scscopedemo: finished');
end
```



Stop and Close Model

When done, stop the application and close the model.

- Stop model
- Close model

```
stop(tg);  
close_system('xpcosc',0);
```


- Configure for a non-Verbose build.
- Build and download application.

```
set_param('xpcosc', 'RTWVerbose', 'off');  
rtwbuild('xpcosc');  
tg = slrt('TargetPC1');  
load(tg, 'xpcosc');
```

```
### Starting Simulink Real-Time build procedure for model: xpcosc  
Warning: This model contains blocks that do not handle sample time  
changes at runtime. To avoid incorrect results, only change  
the sample time in the original model, then rebuild the model.  
### Successful completion of build procedure for model: xpcosc  
### Created MLDATX ..\xpcosc.mldatx  
### Looking for target: TargetPC1  
### Download model onto target: TargetPC1
```

Run Model, Plot Target Scope Data

Create the MATLAB® variable, `tg`, containing the Simulink Real-Time target object. This object allows you to communicate with and control the target computer.

- Create a Simulink Real-Time target object
- Set sample time to 250us
- Set stop time to a high value (10000s)

```
tg.SampleTime = 0.000250;  
tg.StopTime   = 10000;
```

- Define target scope objects 1, 3, 6 and 7: vectorization is used.

```
scs = addscope(tg, 'target', [1,3,6,7]);
```

Get indices of signals 'Integrator1', 'Signal Generator'

- Get index of signal 'Integrator1'
- Get index of signal 'Signal Generator'
- Add signals to the scope objects

```
signals(1) = getsignalid(tg, 'Integrator1');  
signals(2) = getsignalid(tg, 'Signal Generator');  
addsignal(scs, signals);
```

Use the SET command to simultaneously set properties for each element of a scope vector

- Set decimation factor
- Set scope 1 properties.
- Set scope 3 properties.
- Set scope 6 properties.
- Set scope 7 properties.
- Set Y axis limits for each scope in the vector.

```
set(scs, 'Decimation', 1)  
set(scs(1), ...
```



```

    {'NumSamples', 'TriggerMode', 'Grid', 'DisplayMode', 'YLimit'}, ...
    {200, 'FreeRun', 'On', 'Redraw', [-10, 10]});
set(scs(2), ...
    {'NumSamples', 'TriggerMode', 'TriggerSignal', 'TriggerLevel', ...
    'TriggerSlope', 'Grid', 'DisplayMode'}, ...
    {500, 'Signal', getsignalid(tg, 'Signal Generator'), 0.0, ...
    'Rising', 'Off', 'Redraw'});
set(scs(3), 'NumSamples', 100, 'TriggerMode', 'Software', 'DisplayMode', 'Numerical');
set(scs(4), ...
    {'NumSamples', 'TriggerMode', 'TriggerScope', 'Grid', 'DisplayMode'}, ...
    {2000, 'Scope', 3, 'On', 'Redraw'});
set(scs([1,2,4]), 'YLimit', 'Auto');

```

- Start acquisition of every scope
- Start simulation
- Software trigger scope 6

```

start(scs);
start(tg);
trigger(scs(3));

```

Capture an Image of the Target Computer Video Display

- Wait for 1 sec after the run
- Snapshot of target computer video display

```

pause(1);
tg.viewTargetScreen;

```



Close Model

When done, close the model.

- Close model

```
close_system('xpcosc',0);
```

Pre- and Post-Triggering of a Host Scope

This example shows pre- and post-triggering of a signal-triggered Simulink® Real-Time™ host scope. After the script builds and downloads the oscillator model, `xpcosc`, to the target computer, it adds a scope of type 'host' to the real-time application and the signals 'Integrator1' and 'Signal Generator' to the scope. The scope is then configured to trigger on the signal 'Signal Generator' when it reaches 0.0 on a rising slope (signal value goes negative to positive). Pre-triggering is set to 12 samples, meaning that the 12 samples collected prior to the trigger are also stored in the host scope data vector.

Once the trigger condition is met, the scope is monitored to determine when its data acquisition is complete. Next, the scope data is uploaded to the development computer and plotted. This process repeats 25 times. After every fifth run, the damping gain 'Gain1/Gain' is set to a new random value (between 0 and 2000) and the scope toggles between a 12 sample pre-trigger and post-trigger mode.

Rationale for choosing 12 samples: The model sample time is 250 usec. Since the decimation factor for the scope is set to 4, the difference between two acquired samples is 1 ms. The 'Signal Generator' block outputs a square wave at 20 Hertz (50 msec per wave). Therefore, the acquisition will be shifted by 12 ms (approximately 1/4 of a square wave) each time the scope is updated.

Check Connection Between Development and Target Computers

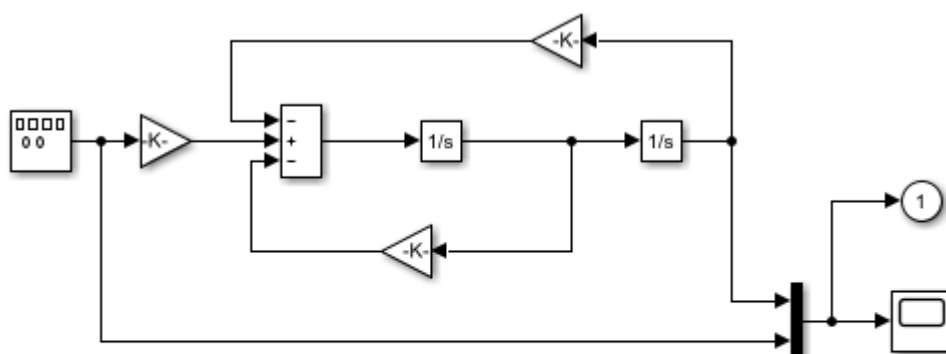
Use `slrtpingtarget` to test the connection between the development and target computers.

```
if ~strcmp(slrtpingtarget, 'success')
    error(message('xPCTarget:examples:Connection'));
end
```

Open, Build, and Download Model to the Target Computer

Open the oscillator model, `xpcosc`. Under the model's configuration parameter Simulink Real-Time options settings, the system target file has been set to `slrt.tlc`. Hence, building the model will create an executable image, `xpcosc.mldatx`, that can be run on a computer booted with the Simulink Real-Time kernel.

```
open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcosc'));
```



Model `xpcosc`
Simulink Real-Time example model

Copyright 1999-2013 The MathWorks, Inc.

Build the model and download the image, `xpcosc.mldatx`, to the target computer.

- Configure for a non-Verbose build.
- Build and download application.

```
set_param('xpcosc', 'RTWVerbose', 'off');  
rtwbuild('xpcosc');  
tg = slrt('TargetPC1');  
load(tg, 'xpcosc');
```

```
### Starting Simulink Real-Time build procedure for model: xpcosc  
Warning: This model contains blocks that do not handle sample time  
changes at runtime. To avoid incorrect results, only change  
the sample time in the original model, then rebuild the model.  
### Successful completion of build procedure for model: xpcosc  
### Created MLDATX ..\xpcosc.mldatx  
### Looking for target: TargetPC1  
### Download model onto target: TargetPC1
```

Run model, Randomize 'Gain' Parameter, Plot Host Scope Data

This code accomplishes a number of tasks.

Task 1: Create Target Object

Create the MATLAB® variable, `tg`, containing the Simulink Real-Time target object. This object contains the information required to communicate with and control the target computer.

- Create a Simulink Real-Time target object
- Set sample time to 250us
- Set stop time to a high value (10000s)
- Start model execution

Task 2: Create, configure, and plot to the host scope during each run.

- Get index of parameter 'Gain1/Gain'
- Get index of signal 'Integrator1'
- Get index of signal 'Signal Generator'
- Define (add) a host scope object
- Add signals to signal list of scope object
- Set number of samples
- Set decimation factor
- Set trigger mode
- Set trigger signal to 'Signal Generator'
- Set trigger level
- Set trigger slope
- Set pre-triggering to 12 samples

Task 3: Check for Plot Figure

Does the plot figure exist?

- If no, create figure
- If yes, make it the current figure

Task 4: Loop to acquire 25 data packages from the scope object.

- Change parameter Gain1/Gain every fifth acquisition loop to a random value between 0 and 2000.
- Toggle between pre- and post-triggering
- Start scope object

Task 5: Wait until scope object has 'finished' state.

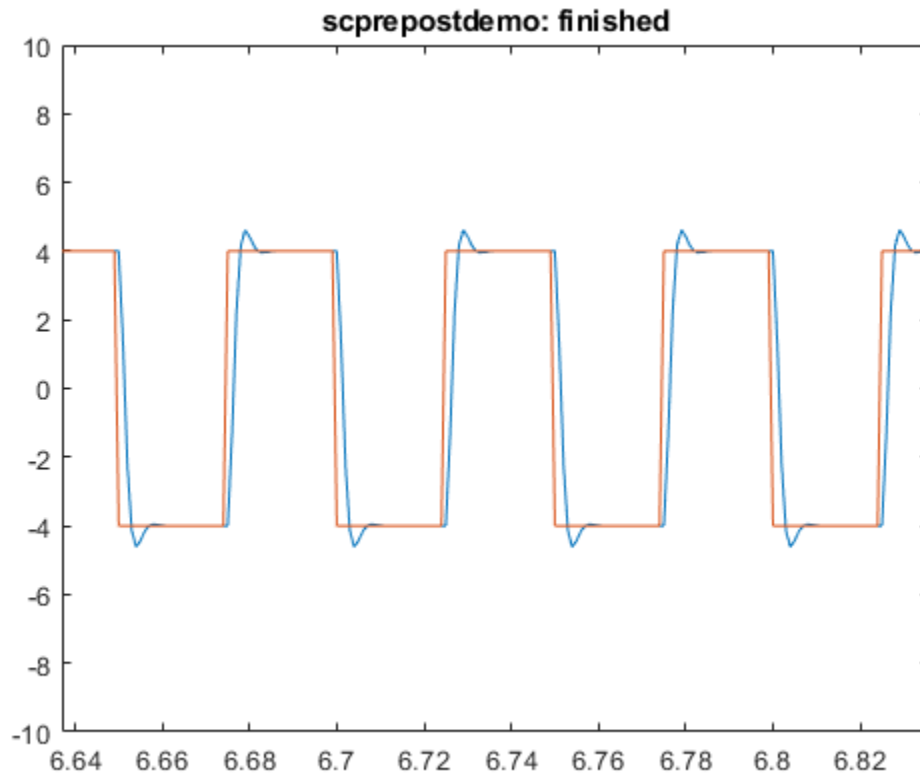
- Create time vector, upload scope data and display it.
- Upload time vector
- Upload acquired data and plot

```

tg = SimulinkRealTime.target; % create target object
tg.SampleTime = 0.000250;
tg.StopTime = 10000;
start(tg);
tPar = getparamid(tg, 'Gain1', 'Gain'); % get indexes
signals(1) = getsignalid(tg, 'Integrator1');
signals(2) = getsignalid(tg, 'Signal Generator');
sc = addscope(tg, 'host'); % define scope object
addsignal(sc, signals);
sc.NumSamples = 200;
sc.Decimation = 4;
sc.TriggerMode = 'Signal';
sc.TriggerSignal = signals(2);
sc.TriggerLevel = 0.0;
sc.TriggerSlope = 'rising';
sc.NumPrePostSamples = -12;
figh = findobj('Name', 'scprepostdemo'); % check for plot figure
if isempty(figh)
    figh = figure;
    set(figh, 'Name', 'scprepostdemo', 'NumberTitle', 'off');
else
    figure(figh);
end
m = 1; flag = 0; % loop to acquire data packages
for n = 1 : 25
    if isempty(find(get(0, 'Children') == figh, 1)), flag = 1; break; end
    if ~m
        setparam(tg, tPar, 2*1000*rand);
        sc.NumPrePostSamples = -sc.NumPrePostSamples;
    end
    m = rem(m + 1, 5);
    start(sc);
while ~strcmpi(sc.Status, 'finished'), end; % wait for scope finished
    t = sc.Time;
    plot(t, sc.Data);
    if (sc.NumPrePostSamples < 0)
        textString = '(Pre-Triggered)';
    else
        textString = '(Post-Triggered)';
    end

```

```
end
title(['scprepostdemo: ', num2str(n), ' of 25 data packages ' textString]);
set(gca,'XLim',[t(1), t(end)], 'YLim', [-10, 10]);
drawnow;
end
if ~flag, title('scprepostdemo: finished'); end
```



Stop and Close Model

When done, stop the application and close the model.

Stop model Close model

```
stop(tg);
close_system('xpcosc',0);
```

Time- and Value-Equidistant Data Logging

This example shows how to do time- and value-equidistant data logging with Simulink® Real-Time™. After the script builds and downloads the oscillator model, `xpcosc`, to the target computer, it runs the application and logs data for 0.2 sec. The option to log states is turned off for this example.

At the end of the first run, the time and output logs (`tg.TimeLog` and `tg.OutputLog`) are retrieved and plotted on the development computer. Initially the logging mode is time-equidistant and every sample is logged. Subsequently, the logging mode is set to value-equidistant with values between 0.02 and 0.2 in steps of 0.02. If the PARAM variable is set to 1, the damping gain 'Gain1/Gain' is randomly selected and set to a new value before starting each run. Otherwise, 'Gain1/Gain' is held constant.

Note: For the case with a random gain setting, the results may appear incorrect due to the combined effects of changing both the gain and the value-equidistant logging parameter.

Check Connection Between Development and Target Computers

Use `'slrtpingtarget'` to test the connection between the development and target computers.

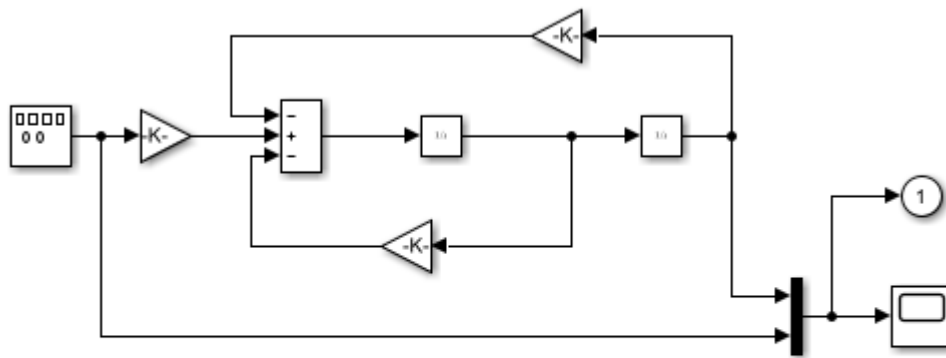
```
if ~strcmp(slrtpingtarget, 'success')
    error(message('xPCTarget:examples:Connection'));
end
```

Open, Build, and Download Model to the Target Computer

Open the oscillator model `xpcosc`. The model has been configured to build for Simulink Real-Time, and building the model creates an executable image, `xpcosc.mldatx`, that can be run on a target booted with the Simulink Real-Time kernel.

Determine Whether `xpcosc` is Open

```
systems = find_system('type', 'block_diagram');
if all(~strcmp('xpcosc', systems))
    mdlOpen = 0;
    open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcosc'));
else
    mdlOpen = 1;
end
if (mdlOpen), stateOption = get_param('xpcosc', 'SaveState'); end
```



Model xpcosc
 Simulink Real-Time example model
 Copyright 1999-2013 The MathWorks, Inc.

Turn State logging off for this example.

```
set_param('xpcosc', 'SaveState', 'off');
```

Build the model and download the image, xpcosc.mldatx, to the target computer.

- Configure for a non-Verbos build.
- Build and download application.

```
set_param('xpcosc', 'RTWVerbose', 'off');
rtwbuild('xpcosc');
tg = slrt('TargetPC1');
load(tg, 'xpcosc');
```

```
### Starting Simulink Real-Time build procedure for model: xpcosc
### Generated code for 'xpcosc' is up to date because no structural, parameter or code replacement
### Successful completion of build procedure for: xpcosc
### Created MLDATX ..\xpcosc.mldatx
```

Close or Reset Model

Close the model if we opened it or reset the state if it was already open.

```
if (mdlOpen)
    set_param('xpcosc', 'SaveState', stateOption);
else
    bdclose('xpcosc');
end
```

Run Model, Set Value-Equidistant Logging Parameter, Plot Logged Data

Create the MATLAB® variable, tg, containing the Simulink Real-Time target object. This object allows you to communicate with and control the target computer.

- Create a Simulink Real-Time Object

- Set sample time to 250us
- Set stop time to 0.2s
- Time-equidistant logging

```
tg.SampleTime = 0.000250;
tg.StopTime   = 0.2;
tg.LogMode    = 'normal';
```

- Start model execution
- Get index of parameter 'Gain1/Gain'

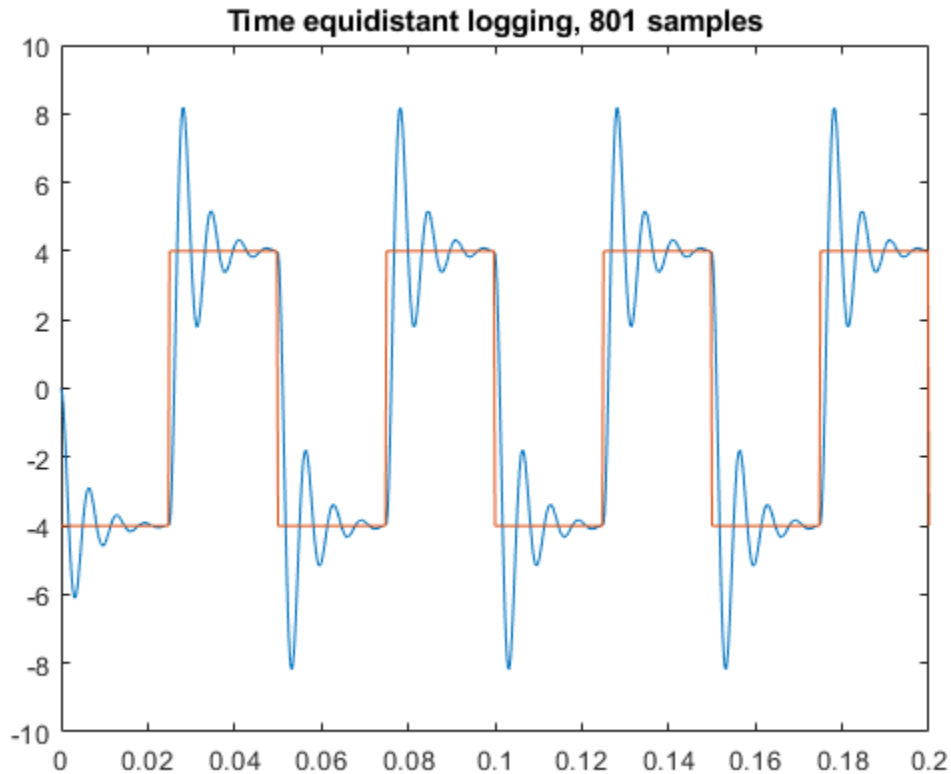
```
start(tg);
tPar = getparamid(tg, 'Gain1', 'Gain');
```

Does the plot figure exist?

- If no, create figure.
- If yes, make it the current figure.

Wait until the run is complete and then retrieve the logged data and plot it.

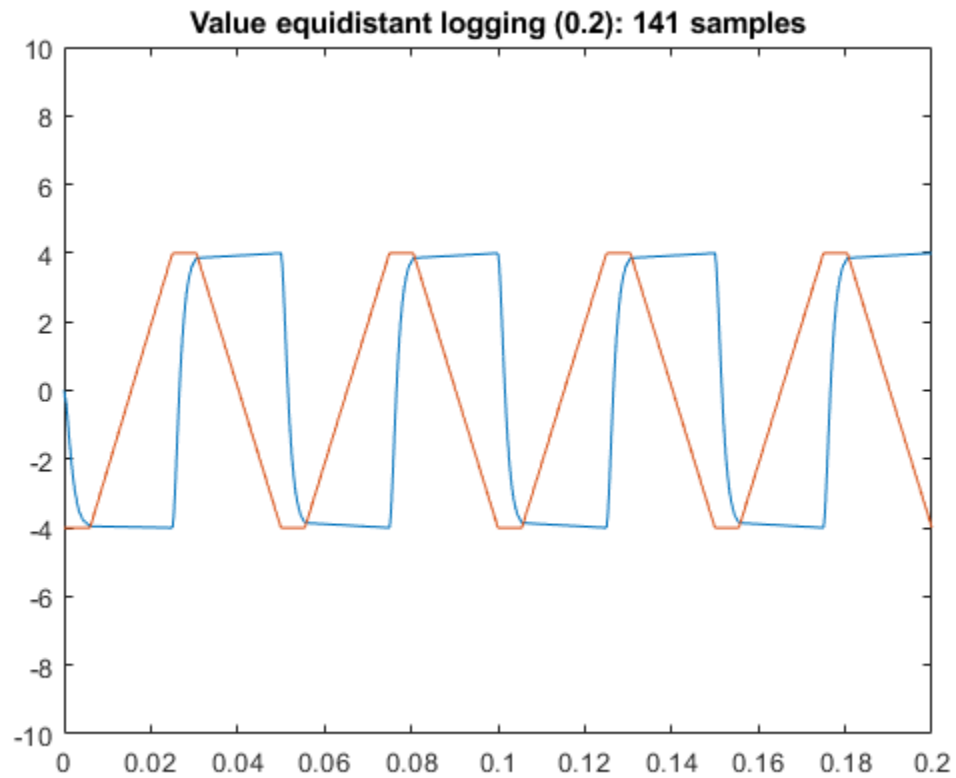
```
while strcmp(tg.Status, 'running')
    pause(0.05);
end
figh = findobj('Name', 'dataloggingdemo');
if isempty(figh)
    figh = figure;
    set(figh, 'Name', 'dataloggingdemo', 'NumberTitle', 'off');
else
    figure(figh);
end
tm = tg.TimeLog;
op = tg.OutputLog;
plot(tm, op);
set(gca, 'XLim', [tm(1), tm(end)], 'YLim', [-10, 10]);
title(['Time equidistant logging, ' num2str(length(tm)) ' samples']);
drawnow;
PARAM = 1; flag = 0;
```



Loop over the equidistant logging

- Change parameter Gain1/Gain to random value between 0 and 2000.
- Set value-equidistant logging parameter to n.
- Start model execution
- Wait until the application is complete.
- Retrieve the logged data and plot it.

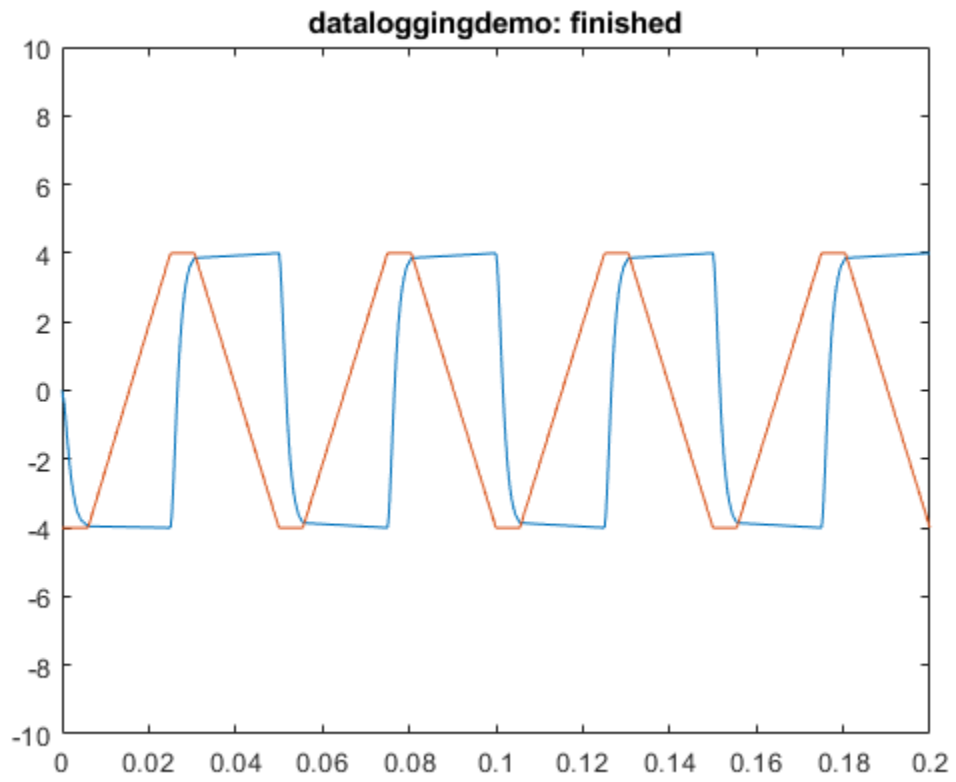
```
for vep = 0.02 : 0.02 : 0.2
    if isempty(find(get(0, 'Children') == figh, 1)), flag = 1; break; end
    if PARAM == 1
        setparam(tg, tPar, 2*1000*rand);
    end
    set(tg, 'LogMode', vep);
    start(tg);
    while strcmp(tg.Status, 'running')
        pause(0.05);
    end
    tm = tg.TimeLog;
    op = tg.OutputLog;
    plot(tm, op);
    set(gca, 'XLim', [tm(1), tm(end)], 'YLim', [-10, 10]);
    title(['Value equidistant logging (' num2str(vep) ') : ' ...
        num2str(length(tm)) ' samples']);
    drawnow;
end
```



Stop Application

When done, stop the application from running.

```
if ~flag, title('dataloggingdemo: finished'); end  
stop(tg);
```

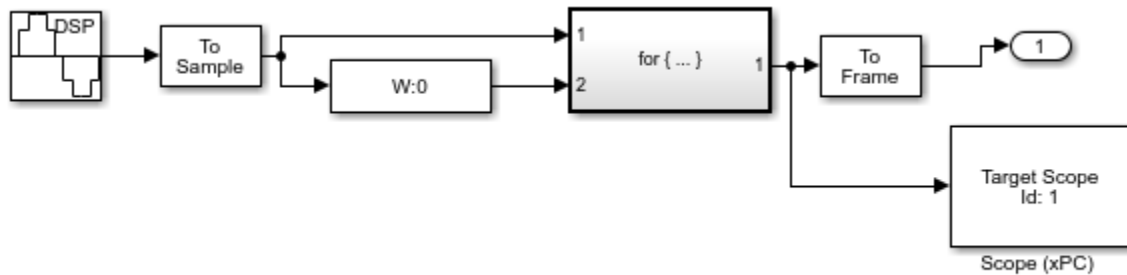


Frame Signal Processing

This example model shows how to use a `for()` loop to iterate through a frame one sample at a time when the minimum sample time is the frame completion time.

This example requires DSP System Toolbox™.

```
open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcFrameLoop'))
```



Copyright 2007-2012 The MathWorks, Inc.

Spectrum Analyzer

This example shows how to use Simulink® Real-Time™ as a real-time spectrum analyzer. The example uses the model `slrtex_dsp_spectrum`. To examine the design and implementation of the key block, 'Spectrum Analyzer', right-click the block and select **Mask > Look Under Mask**.

The example displays the Fast Fourier Transform (FFT) of the input signal using a buffer of 512 samples. The input signal is the sum of two sine waves, one with an amplitude of 0.6 and a frequency of 250 Hz, the other with an amplitude of 0.25 and a frequency of 600 Hz. The resulting spectrum is displayed in the Simulation Data Inspector and on a Simulink Scope block.

The example also shows how you can use MATLAB® language to change the amplitude and frequency of the input sine waves while the application is running.

To run the example, you must have installed DSP System Toolbox™ on your development computer and started the target computer.

Check Connection Between Development and Target Computers

Use `slrtpingtarget` to test the connection between the development and target computers.

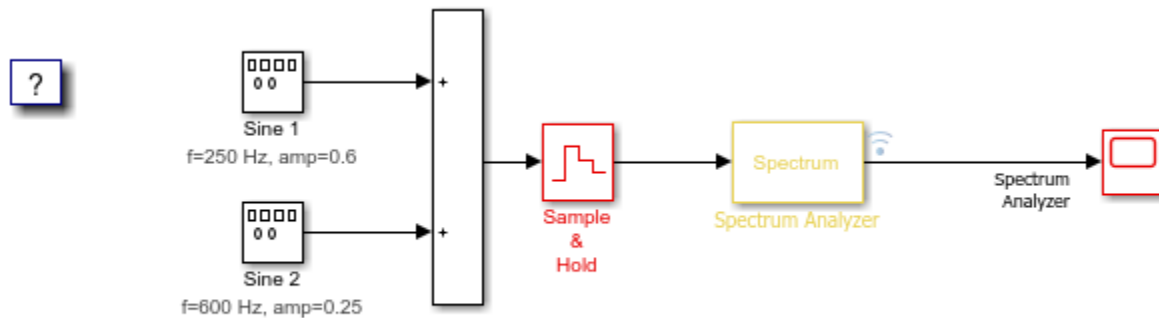
```
if ~strcmp(slrtpingtarget,"success")
    disp('No connection to target computer')
else
    disp('Successful connection to target computer')
end
```

```
Successful connection to target computer
```

Open, Build, and Download Model to the Target Computer

Open the model `slrtex_dsp_spectrum`. Under the model's configuration parameter Simulink Real-Time option settings, the system target file has been set to `slrt.tlc`. Hence, building the model will create an executable image, `slrtex_dsp_spectrum.mldatx`, that can be run on a computer booted with the Simulink Real-Time kernel.

```
open_system(fullfile(matlabroot,'toolbox','slrt','slrtexamples','slrtex_dsp_spectrum'));
```



Copyright 1996-2013 The MathWorks, Inc.

Build the model and download the image, `slrtex_dsp_spectrum.mldatx`, to the target computer.

- Configure for a non-Verbose build.
- Build and download application.

```
set_param('slrtex_dsp_spectrum', 'RTWVerbose', 'off');
rtwbuild('slrtex_dsp_spectrum');
```

```
### Starting Simulink Real-Time build procedure for model: slrtex_dsp_spectrum
Warning: This model contains blocks that do not handle sample time
changes at runtime. To avoid incorrect results, only change
the sample time in the original model, then rebuild the model.
### Successful completion of build procedure for model: slrtex_dsp_spectrum
### Created MLDATX ..\slrtex_dsp_spectrum.mldatx
```

Run Model and Plot Spectrum Data

Create the MATLAB® variable, `tg`, containing the Simulink Real-Time target object. This object allows you to communicate with and control the target computer. After starting the model, the spectrum will be displayed in the Simulation Data Inspector.

- Create an Simulink Real-Time Object
- Start model execution
- Wait for SDI to be updated
- View spectrum plot

```
tg = slrt('TargetPC1');
load(tg, 'slrtex_dsp_spectrum');
start(tg);
pause(1);
tg.StopTime = 30;
disp('Note: Model will continue to run for 30 seconds. To stop execution, type tg.stop')
```

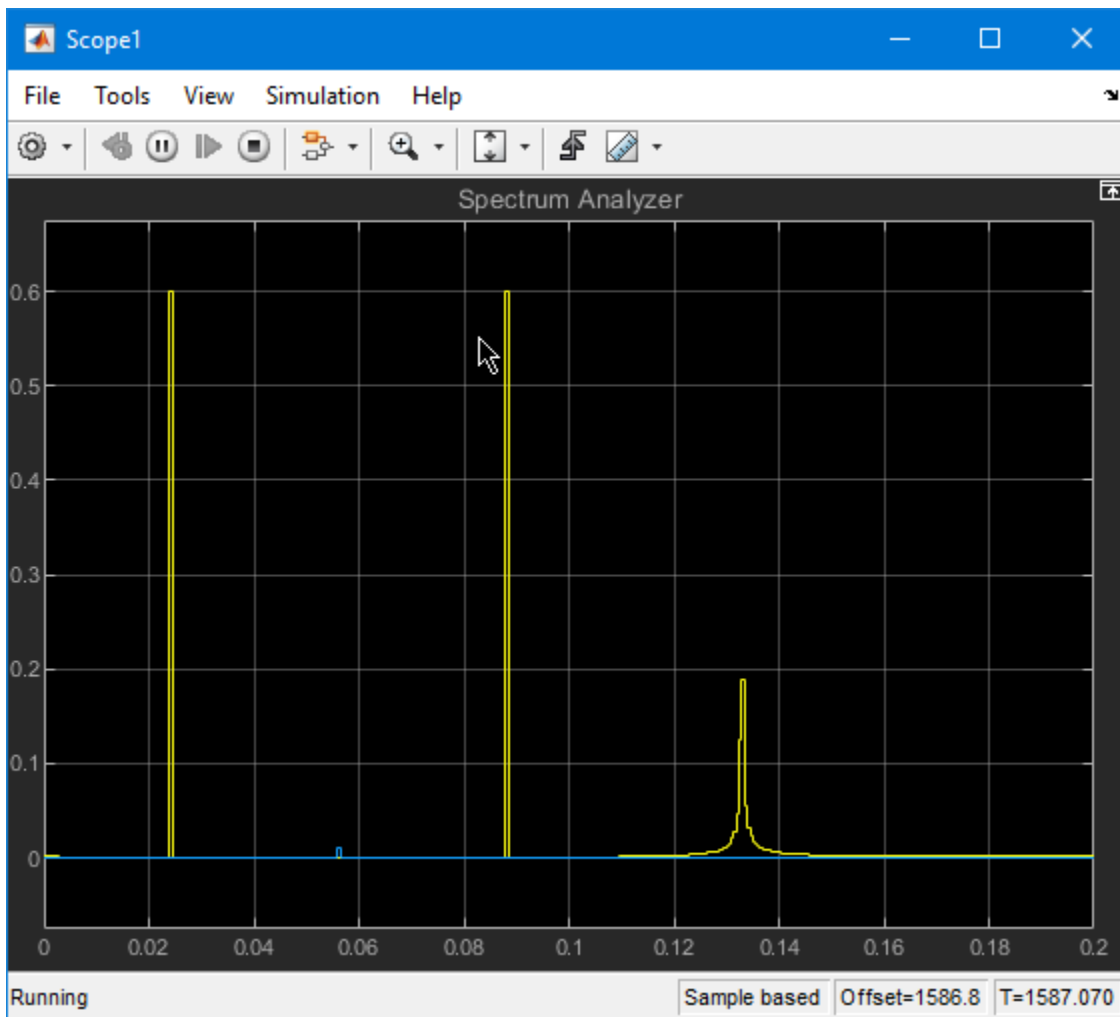
Note: Model will continue to run for 30 seconds. To stop execution, type `tg.stop`

Display the signals in the Simulation Data Inspector

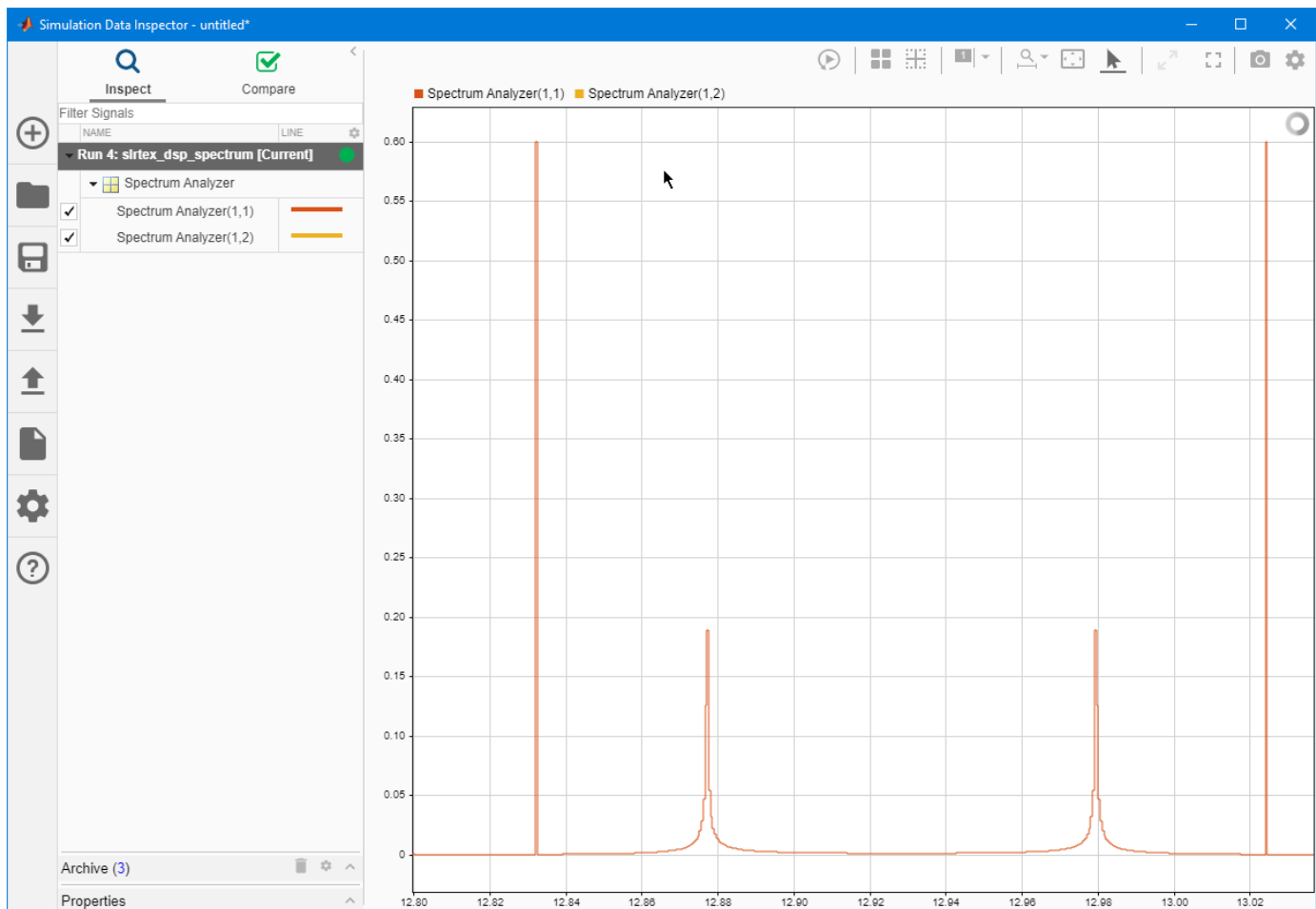
To view the plotted signal data, open the Simulation Data Inspector.

```
Simulink.sdi.view
```

This image shows an example view on the Simulink Scope.



This image shows an example view on the Simulation Data Inspector.



Changing Signal Characteristics

You can change the amplitude and frequency of the sine wave generators while the application is running. To do this, first call `getparamid` with the target object, the block name, and the parameter name to get the parameter object. Then, call `setparam` with the target object, the parameter object, and the new value.

```
slamp = getparamid(tg, 'Sine 1', 'Amplitude');
setparam(tg, slamp, 0.3);
```

By repeated use of the `getparamid` and `setparam` commands, you can monitor and vary the input signals in real time.

```
slfre = getparamid(tg, 'Sine 1', 'Frequency');
setparam(tg, slfre, 300);
s2amp = getparamid(tg, 'Sine 2', 'Amplitude');
setparam(tg, s2amp, 0.55);
s2fre = getparamid(tg, 'Sine 2', 'Frequency');
```

```
setparam(tg, s2fre, 500);
```

Simple Client Application With the .NET API

This example shows a C# client application that uses the Simulink® Real-Time™ API for Microsoft® .NET Framework to interface with the target computer. Developed using Microsoft® Visual Studio®, the client application loads, starts, and controls the oscillator real-time application, `xpcosc.mldatx`.

For more information about Simulink Real-Time API for Microsoft .NET Framework, see the Simulink Real-Time documentation.

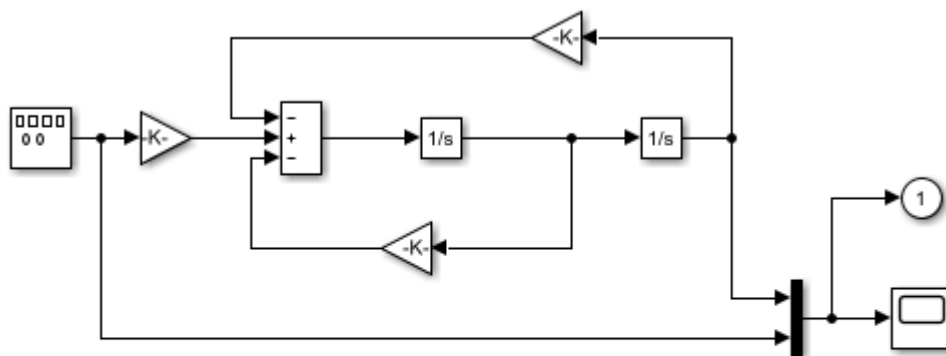
REQUIREMENT:

- To open the project files, you must have installed Microsoft Visual Studio.

Open the Model

Open the oscillator model.

```
addpath(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos'));
mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if ~any(strcmp('xpcosc', systems))
    mdlOpen = 1;
    open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcosc'));
end
```



Model xpcosc
Simulink Real-Time example model

Copyright 1999-2013 The MathWorks, Inc.

Run the Model

Run the model in Simulink® to observe its dynamic behavior.

Start the development computer model.

```
set_param(bdroot, 'SimulationCommand', 'start');
pause(3); % Wait for 3 sec.
```

View plot window.

```
open_system([bdroot, '/Scope']);
```



Close plot window.

```
close_system([bdroot, '/Scope']);
```

Build the Model

Build the model and download to the target computer

- Do not download after building.
- Configure for a non-Verbose build.
- Build and download application.
- Close the target connection.

```
set_param('xpcosc', 'xPCisDownloadable', 'off');
set_param('xpcosc', 'RTWVerbose', 'off');
rtwbuild('xpcosc');
tg = slrt('TargetPC1');
load(tg, 'xpcosc');
if exist('tg', 'var'), tg.close; end
```

```
### Starting Simulink Real-Time build procedure for model: xpcosc
Warning: This model contains blocks that do not handle sample time
changes at runtime. To avoid incorrect results, only change
the sample time in the original model, then rebuild the model.
### Successful completion of build procedure for model: xpcosc
```

```
### Created MLDATX ..\xpcosc.mldatx
### Download process is disabled.
```

Close the Model

Close the target model if we opened it.

```
if (mdlOpen)
    bdclose('xpcosc');
end
```

Open the Client Application (Example 1)

Open the client application: Oscillator Client (Example 1)

Set the **Target TCP/IP Address** and **Target TCP/IP Port** to match your target computer configuration.

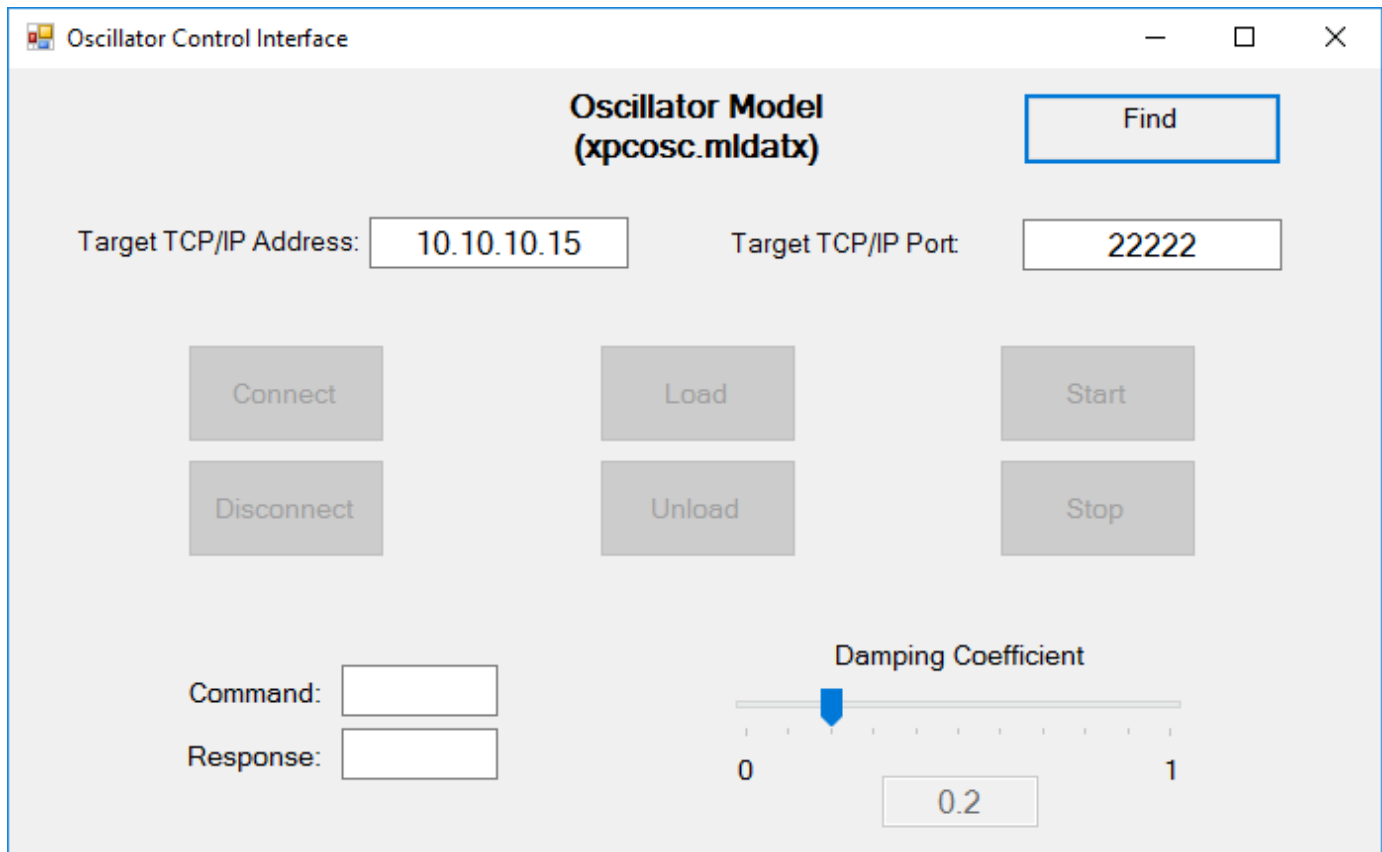


Figure 3: Oscillator Client Application (Example 1)

Find xpcosc.mldatx

Next, by clicking the **Find xpcosc.mldatx...** button, navigate to and select the application file xpcosc.mldatx. Click the **Open** button to close the **Select application file** window.

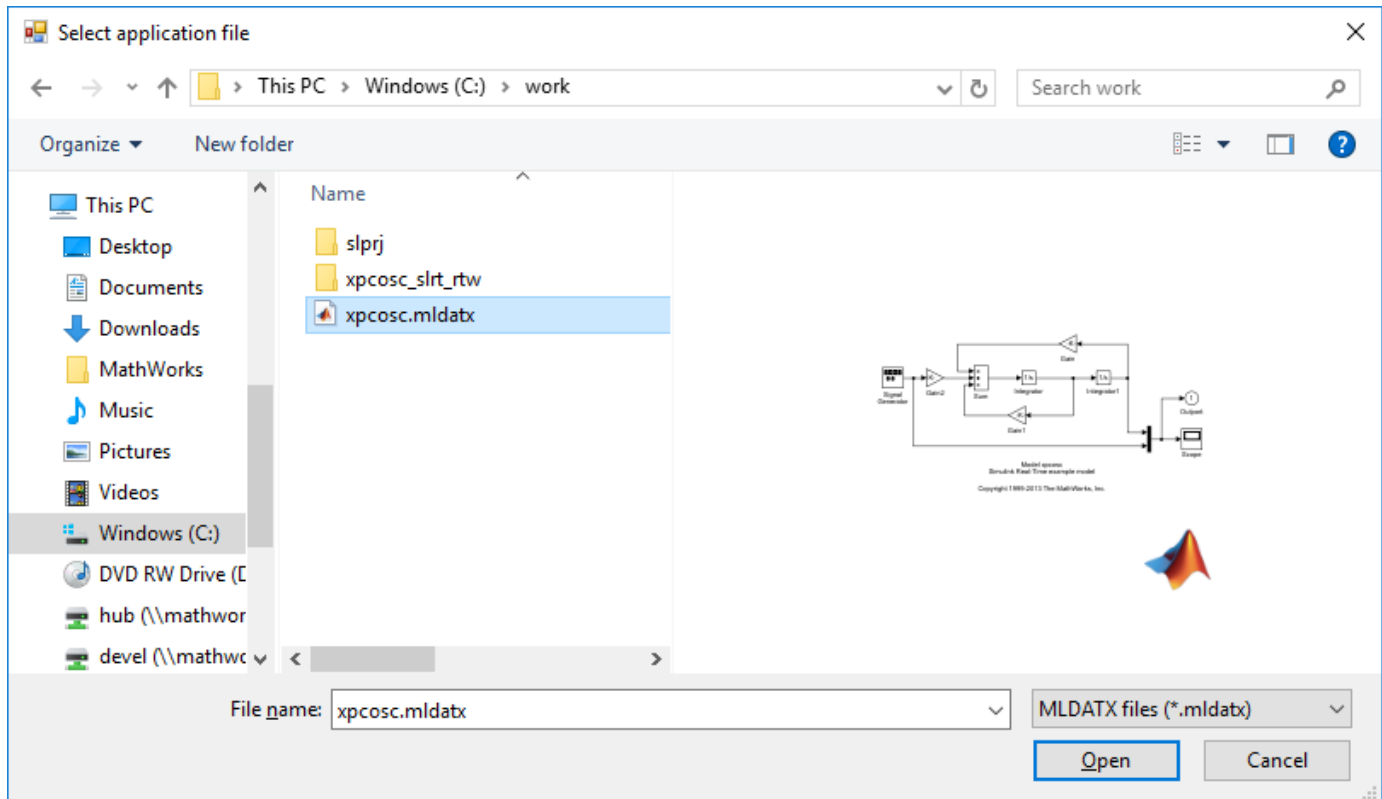


Figure 4: File Find dialog box

Confirm your target computer is booted and connected to the development computer, then:

- Click the **Connect** button to connect to the target computer
- Click the **Load** button to load the application on to the target computer
- Click the **Run** button to start the application running

If you have a monitor connected to the target computer, you will see the oscillator simulation results displayed in a target scope. You can change the damping parameter by moving the slider. When done:

- Click the **Stop** button to stop the application from running
- Click the **Unload** button to unload the application from the target computer
- Click the **Disconnect** button to disconnect from the target computer

Open the Client Application With Host Scope (Example 2)

Host scopes can be used to view signals on the development computer. This example uses host scopes to view the oscillator command and response within a client plot figure.

Open the client application: Oscillator Client (Example 2)

As before, set the **Target TCP/IP Address** and **Target TCP/IP Port** to match your target computer configuration and then use the **Find xpcosc.mldatx** button to navigate to and select the real-time application file.

Click the **Connect**, **Load**, and **Run** buttons. You should see the oscillator command and response signals in the plot figure.

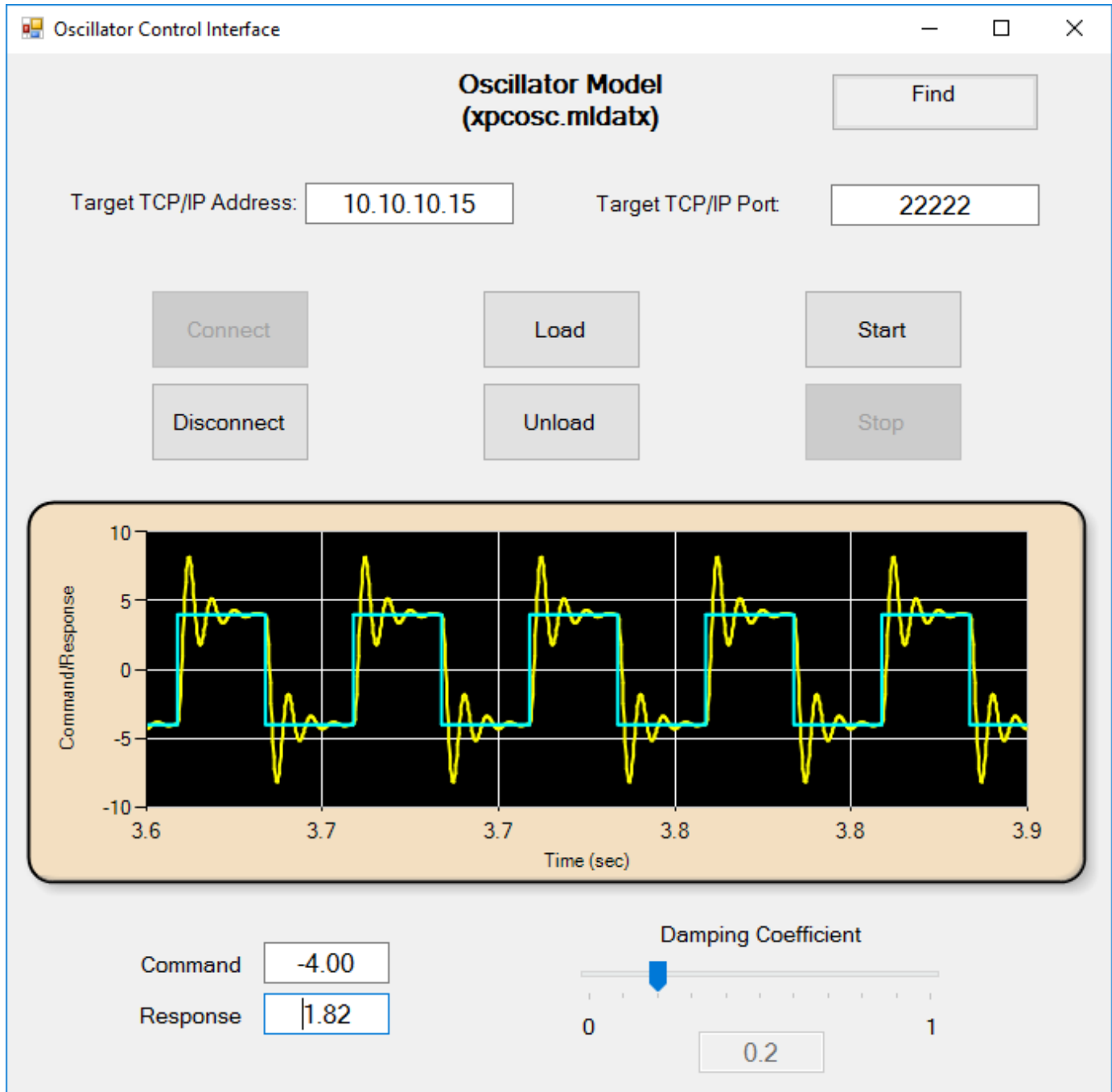


Figure 5: Oscillator Client Application With Host Scope (Example 2)

Project Files

The Microsoft Visual Studio project associated with these examples is located in the Simulink Real-Time API folder here: `Project Folder`

Concurrent Execution on Simulink® Real-Time™

This example shows how to apply explicit partitioning to enhance concurrent execution of a real-time application that you generate by using Simulink Real-Time.

Simulink Real-Time supports concurrent execution by using implicit partitioning or explicit partitioning of models. For explicit partitioning, Simulink Real-Time users partition the root-level model by using referenced models, Simulink subsystems, or other options that Simulink supports. For more information about model partitioning for concurrent execution, see:

- “Implicit and Explicit Partitioning of Models” (Simulink)
- “Implement Task Parallelism in Simulink” (Simulink)
- “Implement Data Parallelism in Simulink” (Simulink)
- “Partition Your Model Using Explicit Partitioning” (Simulink)

This example shows the relationship between the explicit partitioning of the tasks in the model subsystems and the execution of tasks by using the Simulink Real-Time profiling tool.

The example model `dxpcmds6t` runs at sample rate of 0.001 second.

To run the model with adjusted sample rate of 0.01 second, change the sample rate before running the example. In the MATLAB Command Window, type:

```
Ts = 0.01;
```

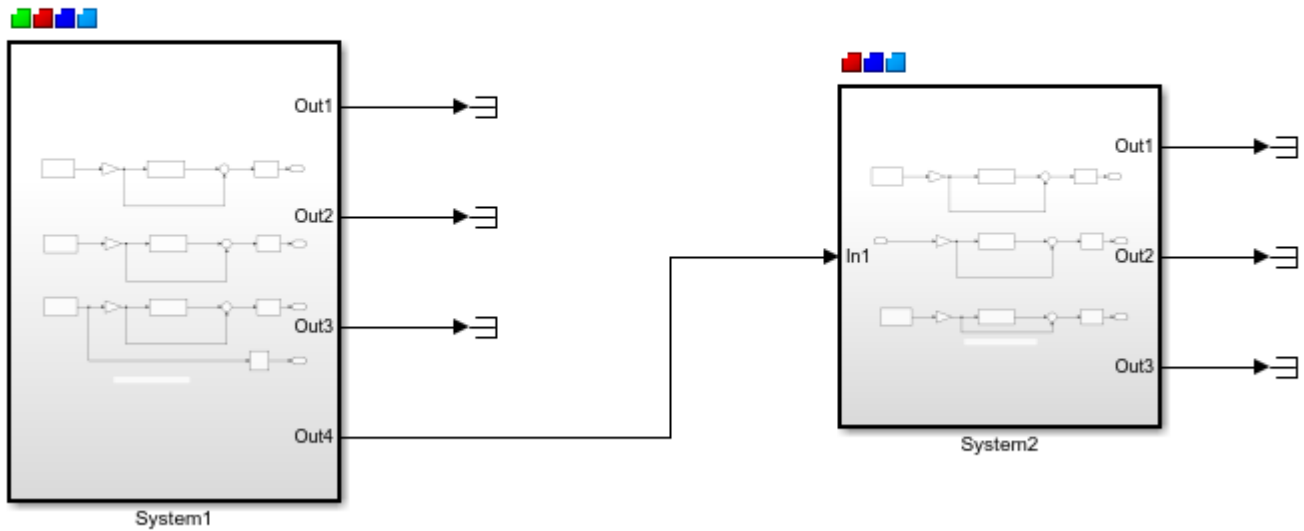
Open, Build, and Download the Model

Open the model `dxpcmds6t`. The model is mapped to seven threads: `Model1_R1`, `Model1_R2`, `Model1_R3`, `Model1_R4`, `Model2_R1`, `Model2_R3`, and `Model2_R4`.

These threads run at sample rates of T_s , $2*T_s$, $3*T_s$, $4*T_s$, T_s , $3*T_s$, and $4*T_s$.

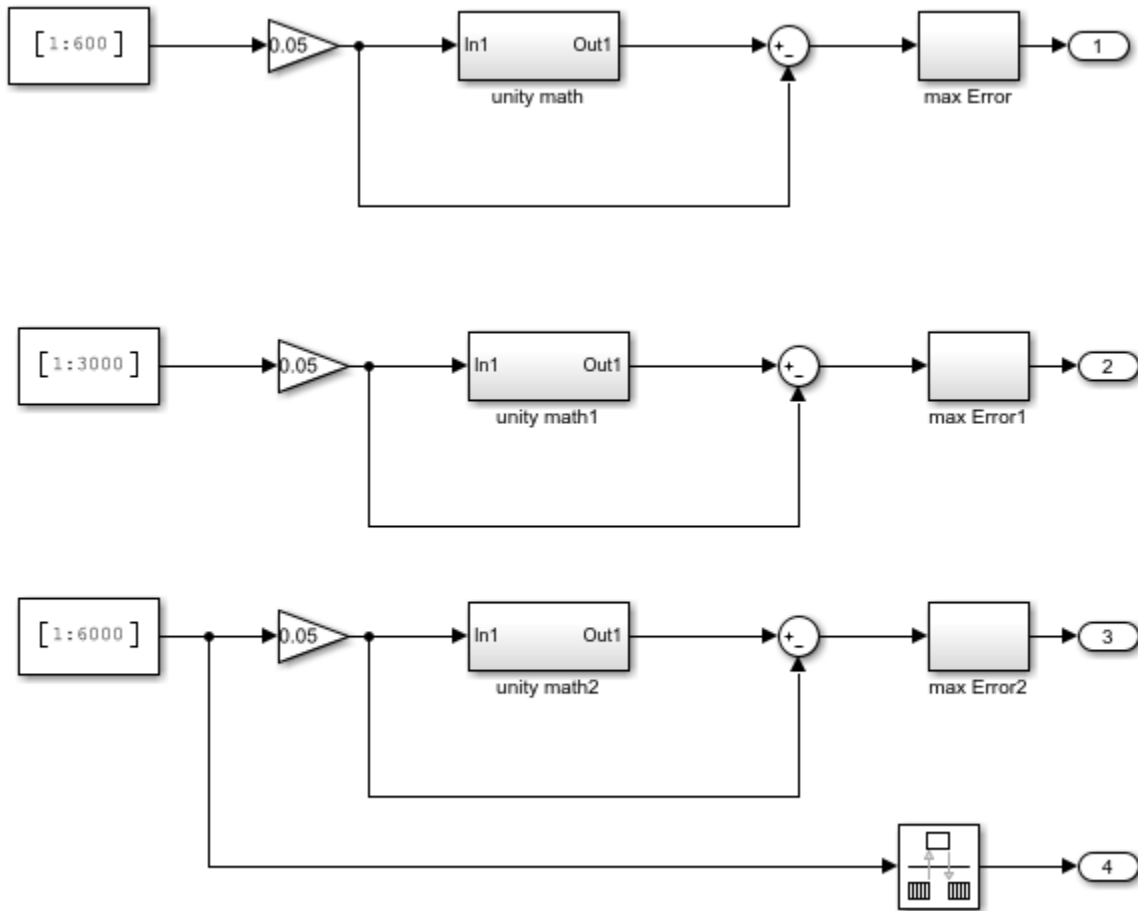
```
mdl='dxpcmds6t';  
open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', mdl));
```

Concurrent Execution on Simulink Real-Time Illustrated by Profiling Tool



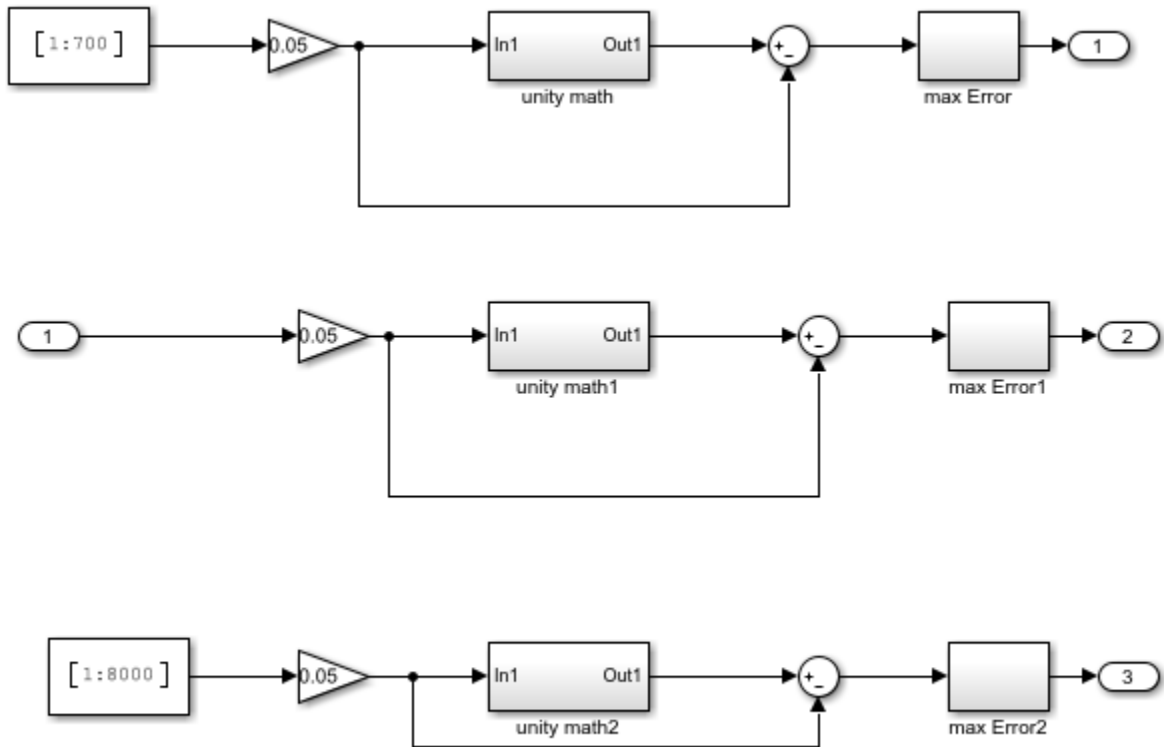
Copyright 2011 - 2013 The MathWorks, Inc.

The explicit partitioning in the top-level model occurs in subsystems System1 and System2.
`open_system([mdl, '/System1']);`



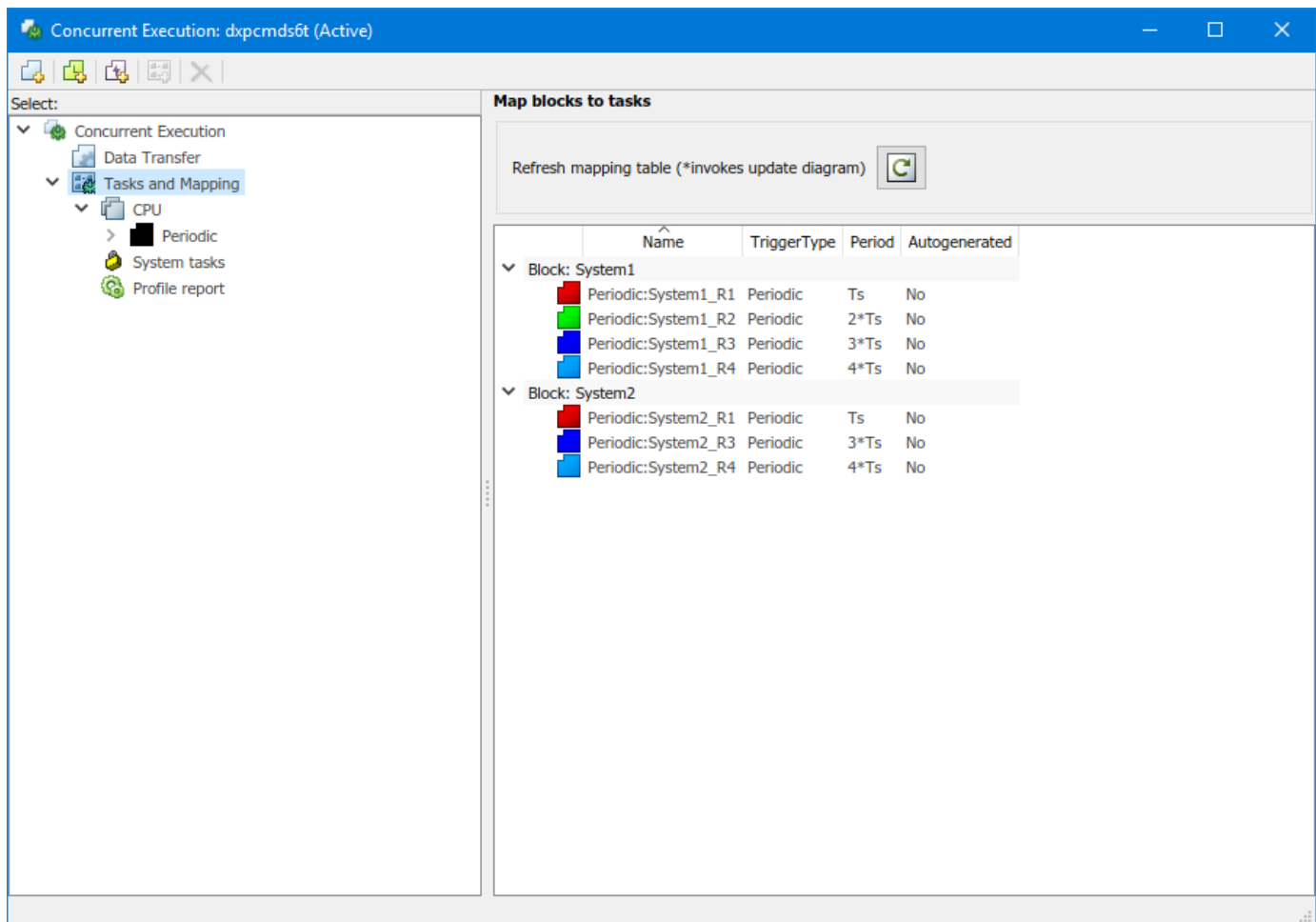
Copyright 2011 The MathWorks, Inc.

```
open_system([mdl, '/System2']);
```



Copyright 2011 The MathWorks, Inc.

To apply explicit partitioning, in the Simulink Editor, on the **Real-Time** tab, click **Hardware Settings**, and then select **Solver > Configure Tasks**. Select the Tasks and Mapping node.



Build, download, and run the model.

```
set_param mdl, 'RTWVerbose', 'off';
rtwbuild(mdl);
tg = slrt('TargetPC1');
load(tg, mdl);
startProfiler(tg);
start(tg);
pause(2);
stop(tg);
```

```
### Starting Simulink Real-Time build procedure for model: dxpcmds6t
### Generated code for 'dxpcmds6t' is up to date because no structural, parameter or code replacement
### Successful completion of build procedure for model: dxpcmds6t
### Created MLDATX ..\dxpcmds6t.mldatx
```

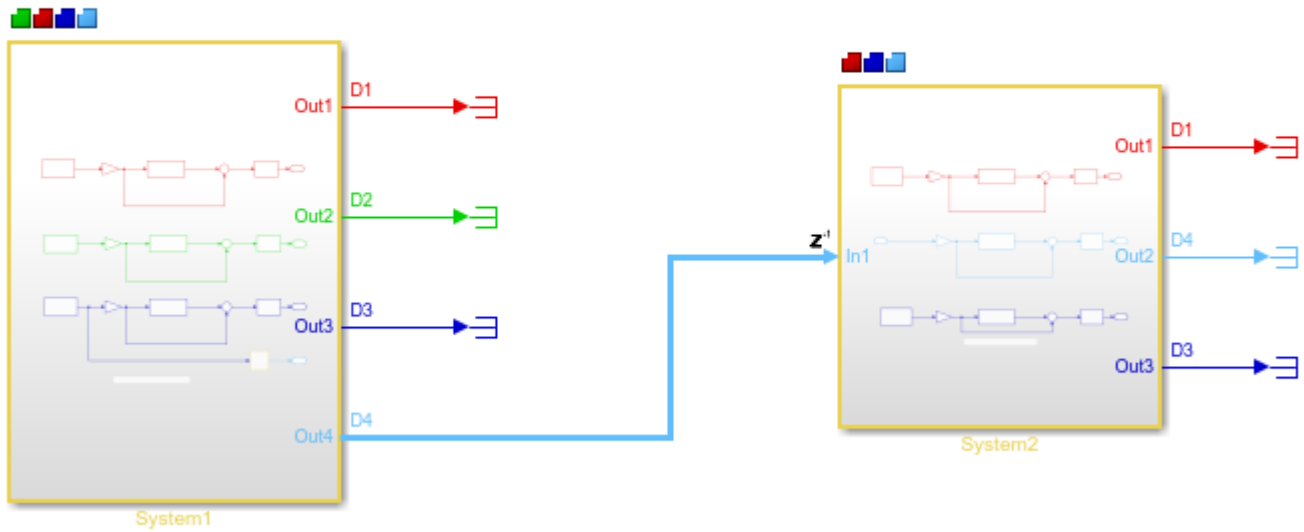
Display Profiling Data

The profiling data shows the execution time of each thread on a multi-core target computer

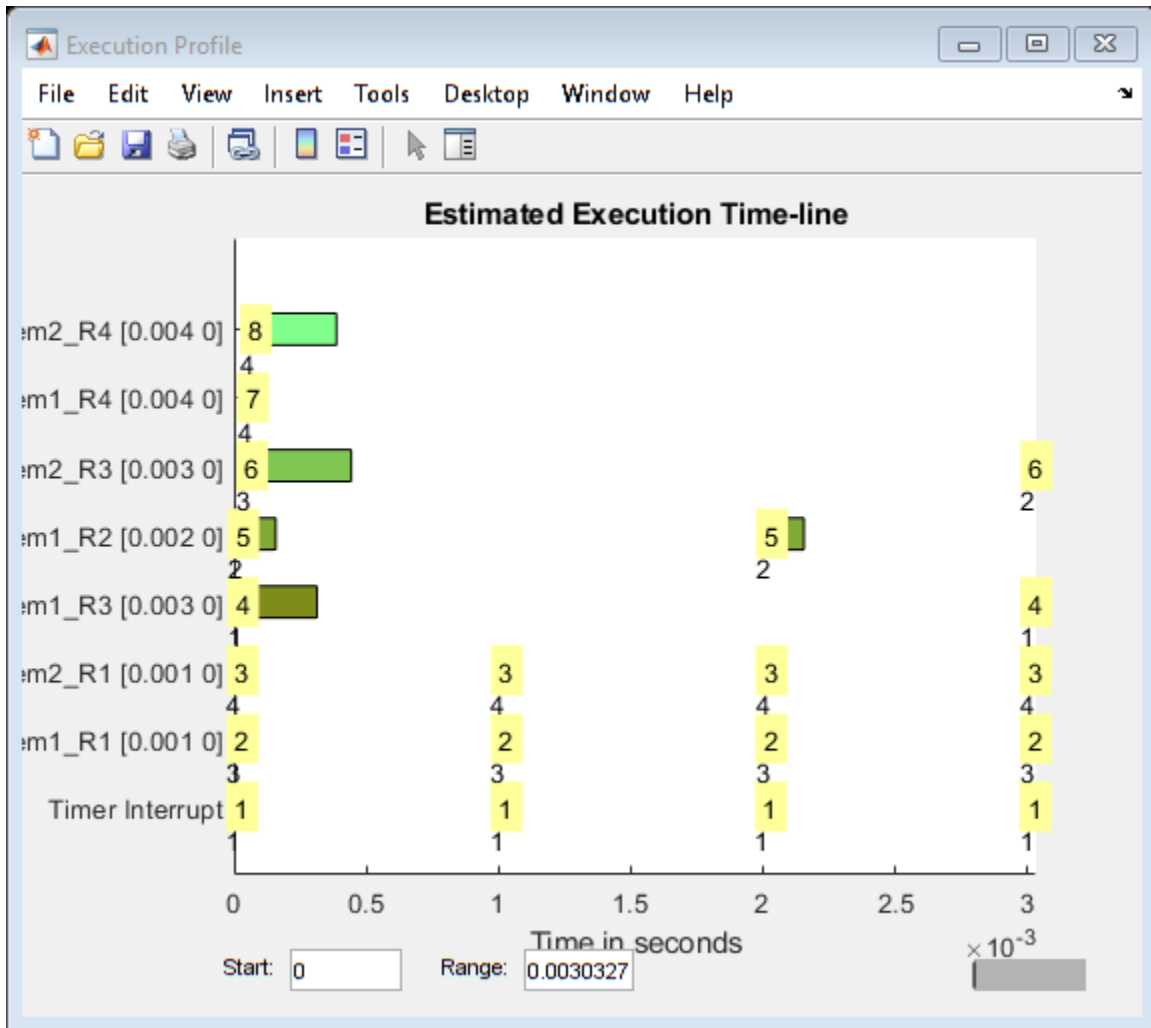
```
profData = tg.getProfilerData;
profData.plot;
```

Processing data, please wait ...

Concurrent Execution on Simulink Real-Time Illustrated by Profiling Tool



Copyright 2011 - 2013 The MathWorks, Inc.



Close the Model

```
bdclose('all');
```

Standalone User Interface using the MATLAB® Compiler™

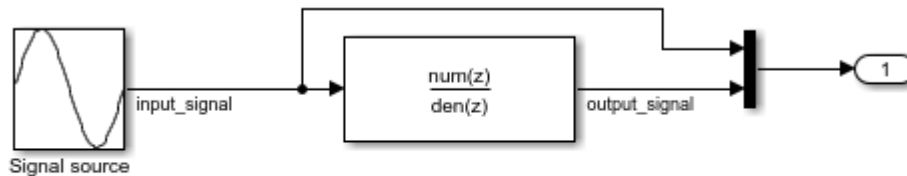
This example shows how to create a standalone user interface running on a Windows computer that interacts with a real-time application using the MATLAB API.

Open and build the model

Click here to open the model: `open_system('dApplicationDeploymentExample')`.

The model consists of a sine wave signal source, the output of which is filtered by a Discrete Filter. The model monitors the performance of the digital filter and compares it with theoretical results from the Bode plot of the filter transfer function. The frequency of the signal and the filter coefficients are defined by workspace variables that are created during model load.

```
model = 'dApplicationDeploymentExample';
open_system(model);
```



Copyright 2015 The MathWorks Inc.

The model is configured to build a real-time application for the default Simulink Real-Time target computer but not to automatically download the application to the target computer after building. Instead, we load it using the user interface.

```
rtwbuild(model);
```

```
### Starting Simulink Real-Time build procedure for model: dApplicationDeploymentExample
Warning: This model contains blocks that do not handle sample time
changes at runtime. To avoid incorrect results, only change
the sample time in the original model, then rebuild the model.
### Successful completion of build procedure for model: dApplicationDeploymentExample
### Created MLDATX ..\dApplicationDeploymentExample.mldatx
### Download process is disabled.
```

Set up the target computer

When we deploy a Simulink Real-Time function, we assume that the target computer has been correctly set up and is running the Simulink Real-Time kernel. We also expect that the Windows computer is able to communicate with the target computer over a reliable TCP/IP network.

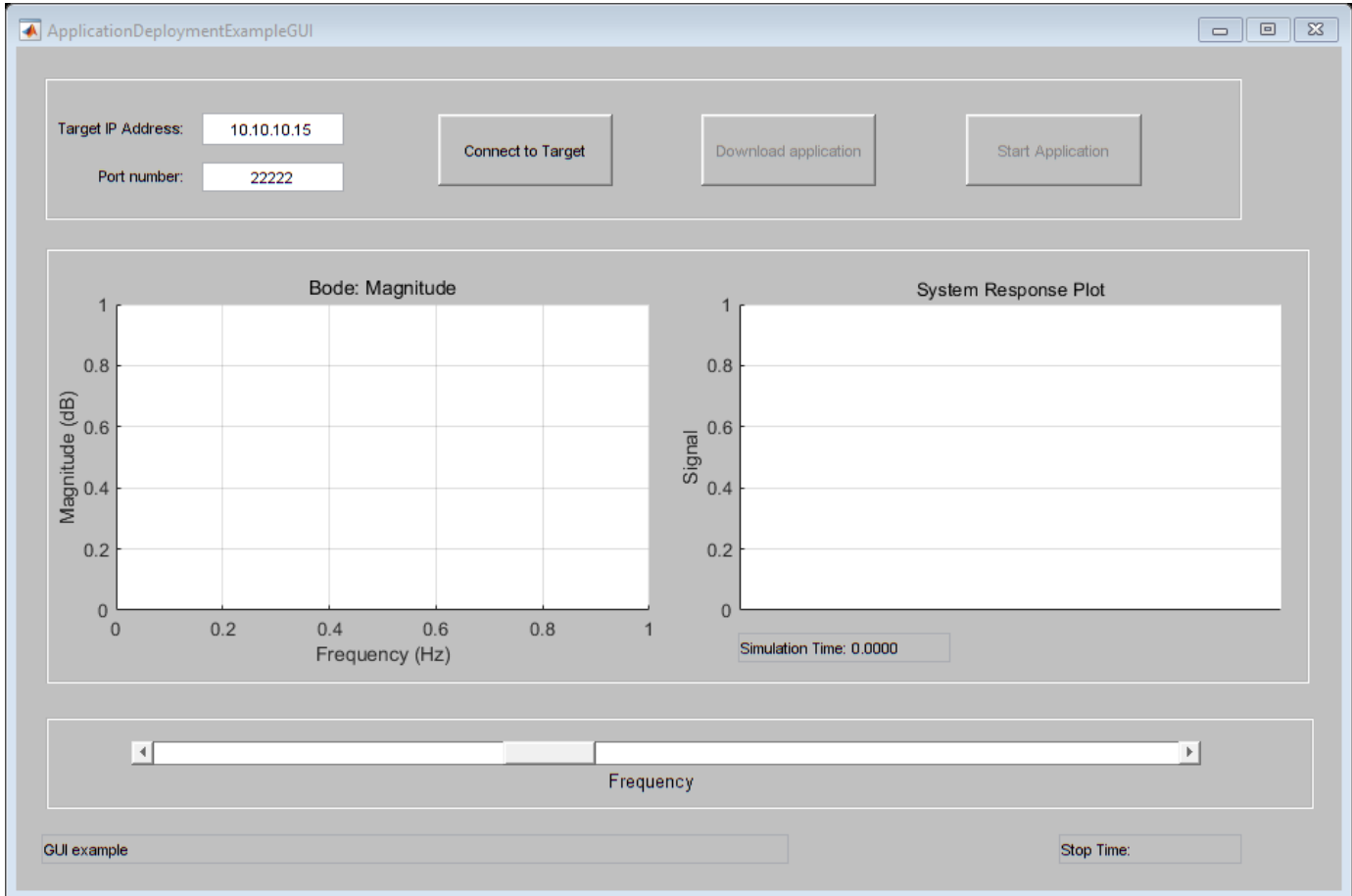
Open the application in MATLAB

We create the user interface using GUIDE. The specific tasks that it implements are:

- 1 Connect to target machine
- 2 Download application

- 3 Start and stop application
- 4 Tune a parameter
- 5 Monitor signals via a host scope
- 6 Integrate MATLAB data analysis with the Simulink Real-Time functions

ApplicationDeploymentExampleGUI



The example consists of the following:

- 1 In the top section, you enter the target IP address and port that will be used when you click 'Connect'. These should match the settings used when you set up the target computer. Once connected, you can download the real-time application to the target computer and start or stop it using the corresponding buttons.
- 2 The MATLAB figure on the left displays the Bode magnitude plot of the system calculated in MATLAB from the numerator and denominator coefficients of the discrete filter.
- 3 The 'System Response Plot' displays the filtered output signal acquired from the target computer using a host scope. The signal is plotted in blue and the expected maximum and minimum value of the signal is plotted in red. The expected value is calculated from the theoretically obtained Bode magnitude plot.
- 4 The 'Frequency' slider allows you to vary the input signal frequency for the real-time application. Every time the slider is changed, a blue '*' indicates the magnitude gain of the discrete filter calculated from the signal values obtained from the target computer.

Compile the User Interface

You use the following MATLAB Compiler command to create a standalone executable from the MATLAB file `ApplicationDeploymentExampleGUI.m` and the related figure `ApplicationDeploymentExampleGUI.fig`.

Note: To avoid issues with the generated EXE, copy `ApplicationDeploymentExampleGUI.m` and `ApplicationDeploymentExampleGUI.fig` to a temporary directory outside the MATLAB installation location, change directory to that location and then execute the following command.

```
mcc -m ApplicationDeploymentExampleGUI.m ApplicationDeploymentExampleGUI.fig
```

Run the executable

You can run the executable on a Windows computer different from the development computer. You must first install the MATLAB Compiler Runtime (MCR). Make sure that the MCR exists on the Windows PATH so that the generated application can find it. More information about distributing the application is present in the README file that is created by the MCC command. The standalone application can be invoked at the Windows command prompt by executing:

```
ApplicationDeploymentExampleGUI.exe
```

If you run the application on the same Windows computer as MATLAB, exit MATLAB before running the executable.

⌘

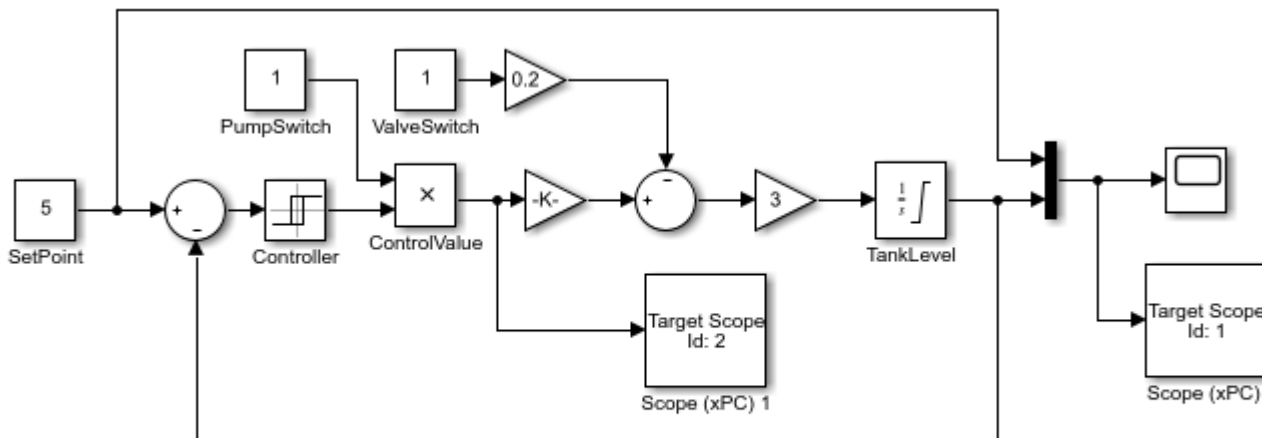
Add App Designer Instrument Panel App to Tank Model

This example shows how to create an App Designer instrument panel app for the Simulink Real-Time application that you build from the `xpctank` model. The instrument panel contains these App Designer components:

- Slider — To tune the required tank level (SetPoint).
- Linear Gauge — To display the actual tank level (TankLevel).
- Semicircular Gauge — To display the pump control status (ControlValue).
- Axes — To display signal output for SetPoint, TankLevel, and ControlValue.

This example also shows how to stream signal and parameter data between the real-time application and the instrument panel app by using the instrumentation object.

```
open_system(docpath(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpctank')));
```



Tank Level Control System

Start Target Computer and Build Real-Time Application

These operations generate the real-time application that streams data to the App Designer instrument panel app.

- 1 Start the target computer.
- 2 Open the model `xpctank`.
- 3 Connect the development computer to the target computer. Build the `xpctank` model. Deploy the real-time application to the target computer. In the MATLAB Command Window, type:

```
set_param('xpctank', 'RTWVerbose', 'off');
tg = slrt('TargetPC1');
rtwbuild('xpctank');
load(tg, 'xpctank');
```

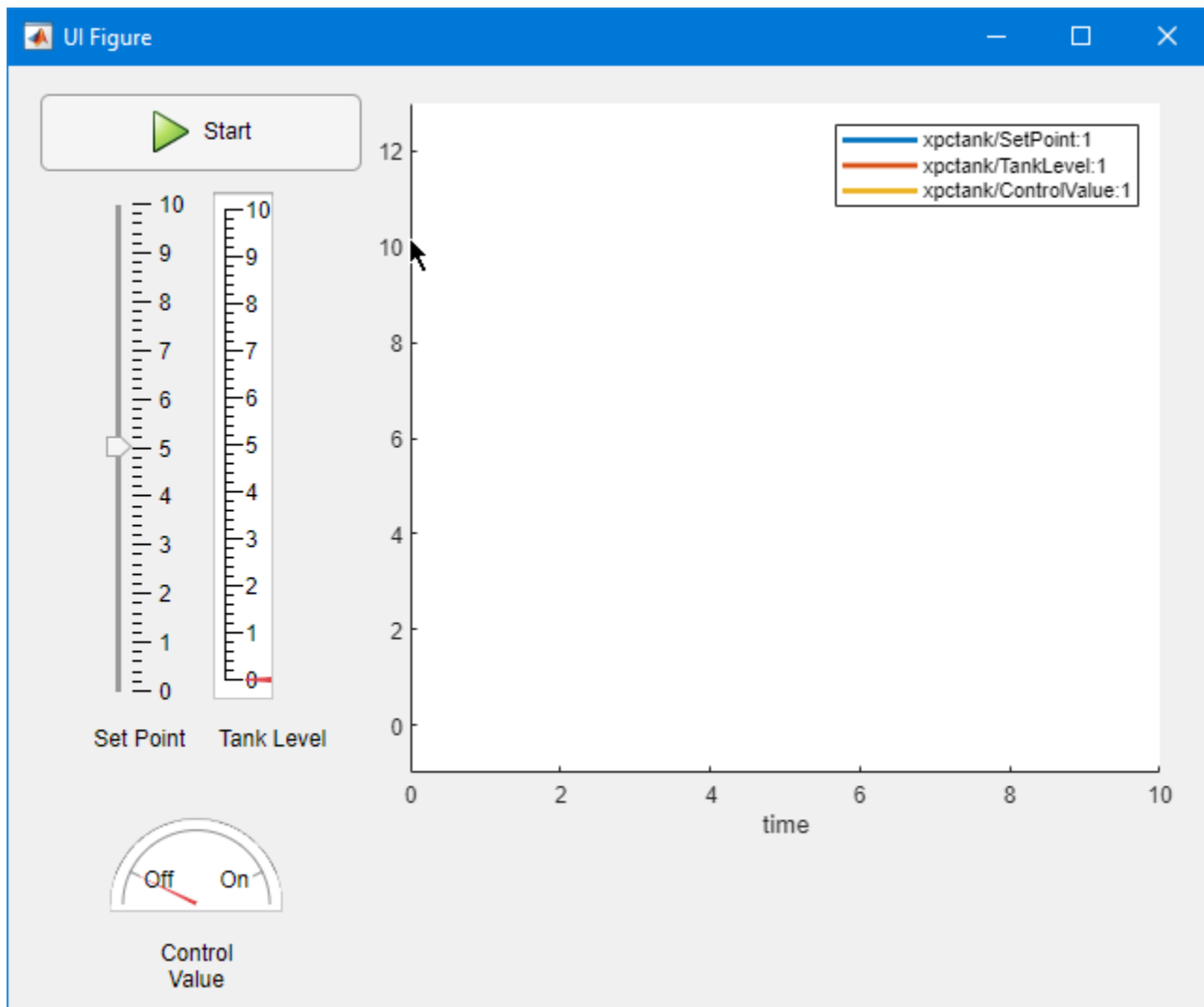
```
### Starting Simulink Real-Time build procedure for model: xpctank
```

```
Warning: This model contains blocks that do not handle sample time
changes at runtime. To avoid incorrect results, only change
the sample time in the original model, then rebuild the model.
### Successful completion of build procedure for model: xpctank
### Created MLDATX ..\xpctank.mldatx
```

Run App Designer Instrument Panel App

The App Designer instrument panel app `tankApp` provides controls to start and interact with the real-time application `xpctank`.

- 1 Run the app. Right-click the `tankApp.mlapp` file and click **Run**. Or, in the Command Window, type: `tankApp`
- 2 Start the real-time application and stream data. In the instrument panel app, click **Start**. The real-time application runs and streams data to the app. The axes display changes to signal values.
- 3 Adjust the SetPoint slider. The real-time application signals react to the set point change.
- 4 To stop the app, click **Stop**.



Create App Designer Instrument Panel App

To provide an interactive instrument panel display for simulation, create and save an App Designer instrument panel app for the xpctank real-time application.

To create an instrument panel in App Designer:

- 1 From the MATLAB Home tab, select **New > App**.
- 2 In the App Designer start page, click **Blank App**. Save the App Designer app as `my_tankApp.mlapp`.
- 3 From the App Designer **Design View**, click the app canvas. In the Component Browser, expand the **Position** tab and set the canvas position to `100 100 640 504`.

Note: The position settings are optional. This information is provided if you want your instrument panel layout to exactly match the example panel.

Add a Slider

Add a slider control from the component library. Label the slider as `Set Point`. Orient the slider as vertical. Set the slider Limit as `0 10`.

With the slider label selected on the canvas, in the Component Browser, expand the **Position** tab and set the slider label position to `41 136 54 22`. With the slider ruler selected, in the Component Browser, expand the **Position** tab and set the slider position to `58 172 3 259`.

With programming, you connect the `SetPoint` slider control to the `SetPoint` parameter in the real-time application. See **Add Callback Functions to App**.

Add a Semicircular Gauge

Add a semicircular gauge from the component library. Label the gauge as `Control Value` (with a line feed between the words). Set the gauge Limit as `-2 12`. Set the MajorTicks as `0 10`.

To label the two ticks as `Off` and `On`, set `MajorTicksMode`, `MajorTickLabelsMode`, and `MinorTicksMode` to `manual`. Then, set the `MajorTickLabels` to `Off On` (with a line feed between the words).

With the gauge label selected, in the Component Browser, expand the **Position** tab and set the label position to `79 12 44 28`. With the gauge dial selected, in the Component Browser, expand the **Position** tab and set the gauge position to `55 55 92 50`.

With programming, you connect the `ControlValue` gauge to the `ControlValue` signal in the real-time application. See **Add Callback Functions to App**.

Add a Linear Gauge

Add a linear gauge from the component library. Label the gauge as `Tank Level`. Orient the gauge as vertical. Set the gauge Limit as `0 10`.

With the gauge label selected, in the Component Browser, expand the **Position** tab and set the label position to `110 136 63 22`. With the gauge ruler selected, in the Component Browser, expand the **Position** tab and set the gauge position to `110 168 32 270`.

With programming, you connect the `TankLevel` gauge to the `TankLevel` signal in the real-time application. See **Add Callback Functions to App**.

Add Plot Axes

Add plot axes from the component library. Delete the `Title`. Set the `XLabel` string to `time`, and then delete the `YLabel` string. Set the `XLim` to `0 10` and the `YLim` to `-1 13`.

With the axes selected, in the Component Browser, expand the **Position** tab and set the axes position to `197 94 424 396`.

Add a Start / Stop Button

Add a button from the component library. Label the button as `StartStop`. This button starts and stops the real-time application on the target computer.

With the button selected, in the Component Browser, expand the **Position** tab and set the button position to `18 449 171 41`.

With programming, you direct the app to respond to button-push events. See **Add Callback Functions to App**.

Add Properties and Methods for Instrumentation Object to App

The App Designer adds UI code for each component that you add to the App Designer instrument panel app `my_tankApp`. The UI code defines the appearance of the component.

To add functionality to the UI components, you add properties, methods, and callbacks in the App Designer **Code View**.

Switch to **Code View**. In the Code Browser, click **Properties**, and then select **Add > Private Property**.

Replace `Property % Description` with these properties. These properties identify the target computer, model, instrumentation object and the start/stop button icons.

```
tg
mdl = 'xpctank';
hInst
stopIcon = 'stop_24.png';
startIcon = 'run_24.png';
```

In the Code Browser, click **Functions**, and then select **Add > Private Function**.

Replace the results function:

```
function results = func(app)
end
```

with these functions. These functions set up the instrumentation object and the callback to animate the plot axes.

For the tank level gauge and the control value gauge, the `HistoryFlag` is set to `'1'` because these signals have already been added to the instrumentation object as lines. Their history must be present even though the scalar display uses only the last value.

```
function setupInstrumentation(app)
    % create the instrumentation object
    app.hInst = SimulinkRealTime.prototype.Instrumentation(app.mdl);
```

```

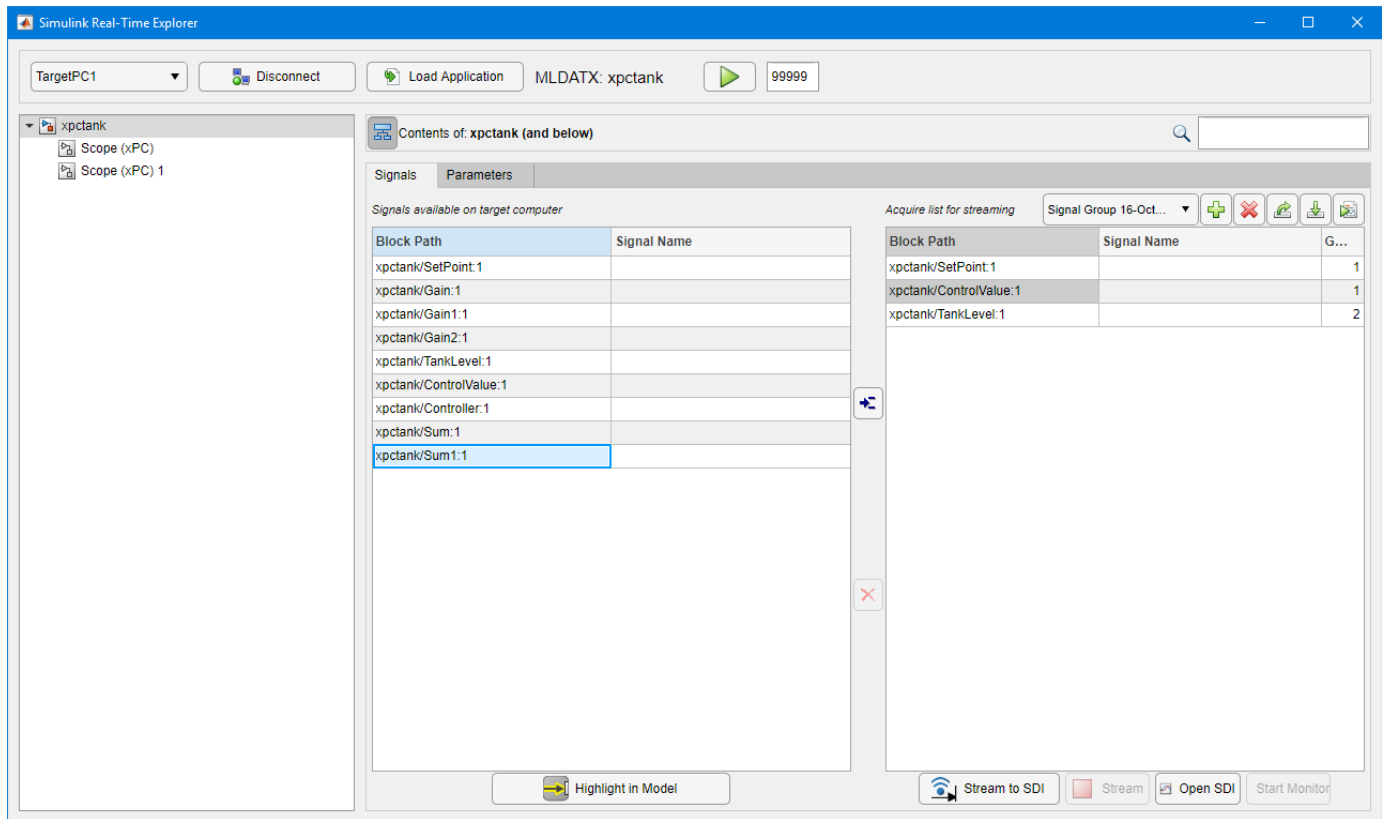
% connect signals to the axes
[~,line1] = app.hInst.connectLine(app.UIAxes,[app.mdl '/SetPoint'] ,1,'MaximumNumPoints',1000);
[~,line2] = app.hInst.connectLine(app.UIAxes,[app.mdl '/TankLevel'] ,1,'MaximumNumPoints',1000);
[~,line3] = app.hInst.connectLine(app.UIAxes,[app.mdl '/ControlValue'],1,'MaximumNumPoints',1000);
% add a legend
legend(app.UIAxes,{line1{1},line2{1},line3{1}})
% connect signals to the scalar displays
app.hInst.connectScalar(app.TankLevelGauge ,[app.mdl '/TankLevel'] ,1,'HistoryFlag',1);
app.hInst.connectScalar(app.ControlValueGauge,[app.mdl '/ControlValue'],1,'HistoryFlag',1);
% add a callback to wrap the x axis
app.hInst.connectCallback(@app.updateUIAxes);
app.UIAxes.XLim = [0 10];
app.UIAxes.YLim = [-1 13];
end

function updateUIAxes(app,instObj,eventData)
% this callback updates XLim so that the axis wrap
currTime = eventData.ExecTime;
if max(currTime)<max(app.UIAxes.XLim)
% do nothing
else
app.UIAxes.XLim = max(app.UIAxes.XLim) + [0 10];
end
end

```

When using the `connectLine` function to connect output signals from the real-time application to the instrumentation object, the Simulink Real-Time Explorer provides a useful view of the signals and their port numbers. To open the Explorer and view the signals, in the MATLAB Command Window, type:

```
SimulinkRealTime.prototype.Explorer
```



Add Callback Functions to App

The callback functions in the app respond to events. Events that are serviced by callback functions include app start up, button press, new signal data, and parameter value change.

Add Start-Up Function

The start-up function `startupFcn` in the app runs after component creation.

Switch to **Design View** and click on the app canvas. In the **Component Browser**, in the **Callbacks** tab, select `add StartupFcn` callback. Add this code in the `startupFcn` function.

```
% get target object
app.tg = slrt;
% if the target is running stop it
if strcmp(app.tg.status,'running')
    app.tg.stop;
end
% load the model on to the target
app.tg.load(app.mdl);

% update the slider based on the target value
value = app.tg.getparam('SetPoint','Value');
app.SetPointSlider.Value = value;

% setup the instrumentation object
app.setupInstrumentation;
```

Add Interface Close Request Function

The close request function `UIFigureCloseRequest` in the app runs when the app is closed. The instrumentation object uses callbacks to add streaming data to your UIAxes. If you delete the UIAxes by deleting and closing the `UIFigure` without first deleting the instrumentation object, the callbacks state leads to errors or warnings. The best practice is to add a close request callback to the app that stops the instrumentation object and deletes it.

Switch to **Design View** tab and click the app canvas. In the **Component Browser**, in the **Callbacks** tab, select add `CloseRequestFcn` callback. In the `UIFigureCloseRequest` function, replace `delete(app)` with this code:

```
app.hInst.stop;
app.tg.stop;
delete(app.hInst)
delete(app)
```

Add Set Point Change Value Function

The set point value change function `SetPointSliderValueChanged` in the app runs when the set point slider value changes.

Switch to **Design View** tab and click on the set point slider. In the **Component Browser**, in the **Callbacks** tab, select add `ValueChangedFcn` callback. In the `SetPointSliderValueChanged` function, replace `value = app.SetPointSlider.Value;` with this code:

```
value = app.SetPointSlider.Value;
app.tg.setparam('SetPoint','Value',value);
```

Add Start-Stop Button-Push Event Function

A callback that you add to the button definition toggles the text and icon on this button, depending on the state of the app. Click **Start** to run the real-time application on the target computer and stream data to the instrument panel app. Click **Stop** to stop the real-time application.

With the button selected, in the Component Browser, click **Callbacks**. Select `StartStopButtonPushed` and click the **Move cursor to function** button. Add this code in the `StartStopButtonPushed` function.

```
if strcmp(app.StartStopButton.Icon, app.startIcon)
    % START
    % clear old data
    app.hInst.clearData;
    % start instrumentation and target
    app.hInst.start(app.tg);
    % change the icon
    app.StartStopButton.Icon = app.stopIcon;
    app.StartStopButton.Text = 'Stop';
else
    % STOP
    app.hInst.stop;
    app.tg.stop;
    % change the icon
    app.StartStopButton.Icon = app.startIcon;
    app.StartStopButton.Text = 'Start';
end
```

Save and Run the App

The app `my_tankApp` app is complete. The slider, semicircular gauge, linear gauge, axes, and **Start-Stop** button have callback code that responds to their events. The app connects to the real-time application `xpctank` by using the instrumentation object to stream data to the app.

In App Designer, save the app `my_tankApp`.

Run the real-time application `xpctank` and run the App Designer app `my_tankApp`. For instructions, see **Start Target Computer and Build Real-Time Application** and **Run App Designer Instrument Panel App**.

```
bdclose ('all');
```

Add m-Script Instrument Panel App to Tank Model

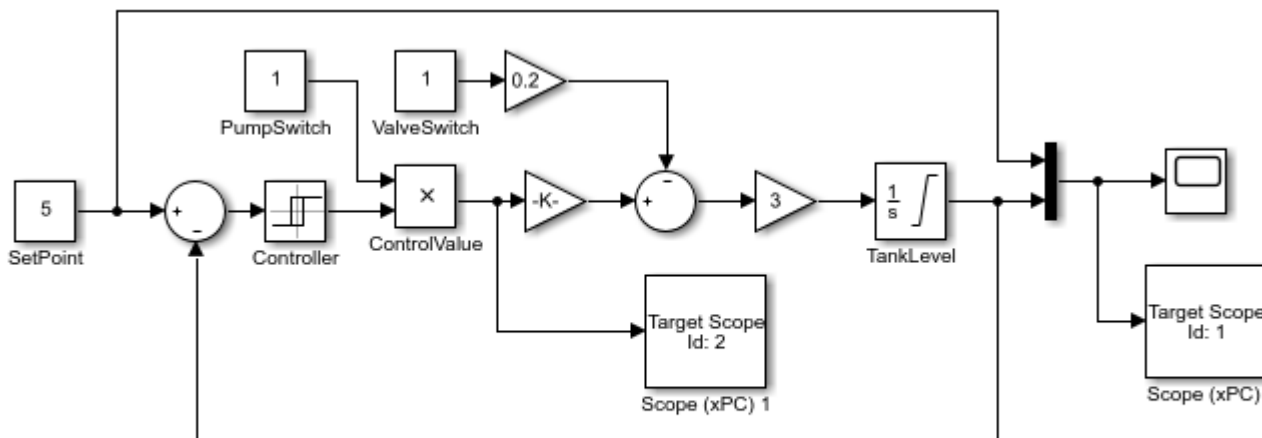
This example shows how a MATLAB m-script can create an instrument panel for the Simulink Real-Time application that you build from the model `xpctank`. The instrument panel contains these UI components:

- Slider — To tune the required tank level (SetPoint).
- Linear Gauge — To display the actual tank level (TankLevel).
- Semicircular Gauge — To display the pump control status (ControlValue).
- Axes — To display signal output for SetPoint, TankLevel, and ControlValue.

This example also shows how to stream signal and parameter data between the real-time application and the instrument panel by using the instrumentation object.

The example UI and callback code demonstrate the subtle syntax differences between applying the instrumentation object in an m-script versus applying the instrumentation object in an App Designer app. For the App Designer version of this example, see “Add App Designer Instrument Panel App to Tank Model” on page 15-59.

```
open_system(docpath(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpctank')));
```



Tank Level Control System

Start Target Computer and Build Real-Time Application

These operations generate the real-time application that streams data to the instrument panel app.

- 1 Start the target computer.
- 2 Open the model `xpctank`.
- 3 Connect the development computer to the target computer. Build the `xpctank` model. Deploy the real-time application to the target computer. In the MATLAB Command Window, type:

```
set_param('xpctank', 'RTWVerbose', 'off');
tg = slrt('TargetPC1');
```

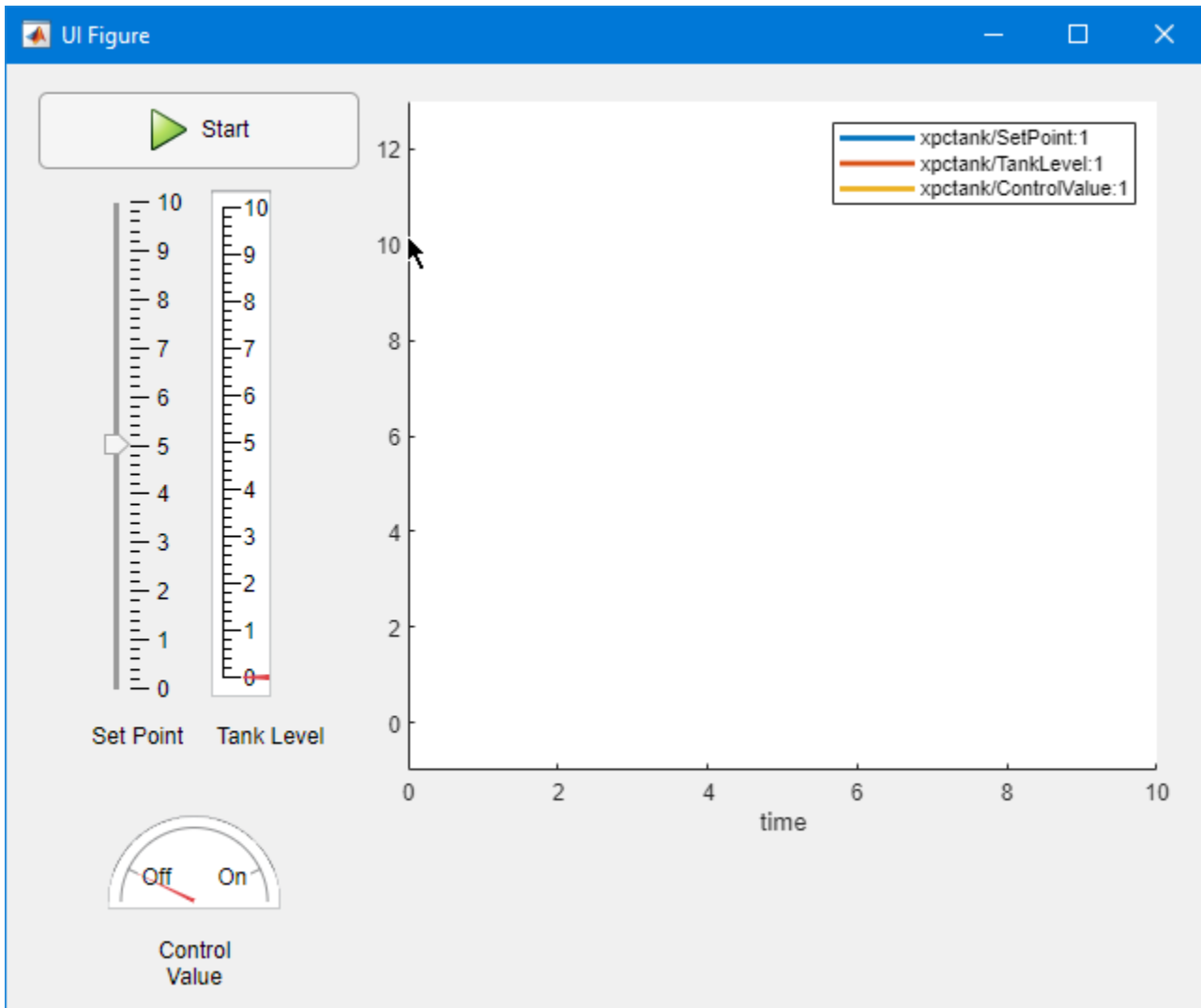
```
rtwbuild('xpctank');  
load(tg, 'xpctank');
```

```
### Starting Simulink Real-Time build procedure for model: xpctank  
### Generated code for 'xpctank' is up to date because no structural, parameter or code replacement  
### Successful completion of build procedure for model: xpctank  
### Created MLDATX ..\xpctank.mldatx
```

Run Instrument Panel App

The instrument panel app `tankAppScript` provides controls to start and interact with the real-time application `xpctank`.

- 1** Run the app. Right-click the `tankAppScript.m` file and click **Run**. Or, in the Command Window, type: `tankAppScript`
- 2** Start the real-time application and stream data. In the instrument panel app, click **Start**. The real-time application runs and streams data to the app. The axes display changes to signal values.
- 3** Adjust the SetPoint slider. The real-time application signals react to the set point change.
- 4** To stop the app, click **Stop**.



Set Up Instrument Panel App

To provide an interactive instrument panel display for simulation, create the instrument panel app as a UIFigure. Hide the app until all UI components are created.

```
hf = uifigure('Visible', 'off', ...
    'Position', [100 100 640 504], ...
    'CloseRequestFcn', @closeRequest);
```

Note: The Position settings are optional. This information is provided if you want your instrument panel layout to exactly match the example panel.

Create Plot Axes

Add a plot axe by using the UIAxes command. These axes use animation to display the signal data that streams from the real-time application.

```
ha = uiaxes(hf, 'Position', [197 94 424 396]);
xlabel(ha, 'time')
```

Create Tank Level Gauge

Add a linear gauge by using the `UIGauge` command and `'linear'` argument. This gauge displays the tank level signal data.

```
tlGauge = uigauge(hf, 'linear', ...
    'Limits', [0 10], ...
    'Orientation', 'vertical', ...
    'Position', [110 168 32 270]);
```

Add a label for the tank level gauge.

```
uilabel(hf, ...
    'HorizontalAlignment', 'center', ...
    'Position', [110 136 63 22], ...
    'Text', 'Tank Level');
```

Create Set Point Slider

Add a slider by using the `UISlider` command. This slider enables changes to the set point parameter value.

```
spSlider = uislider(hf, ...
    'Limits', [0 10], ...
    'Orientation', 'vertical', ...
    'Position', [58 172 3 259], ...
    'ValueChangedFcn', @SetPointSliderValueChanged);
```

Update the slider from the set point parameter value.

```
value = tg.getparam('SetPoint','Value');
spSlider.Value = value;
```

Add a label to the set point slider.

```
uilabel(hf, ...
    'HorizontalAlignment', 'right', ...
    'Position', [41 136 54 22], ...
    'Text', 'Set Point');
```

Create Control Value Gauge

Add a semicircular gauge by using the `UIGauge` command and `'semicircular'` argument. This gauge displays the control value as `'Off'` or `'On'`.

```
cvGauge = uigauge(hf, 'semicircular', ...
    'Limits', [-2 12], ...
    'MajorTicks', [0 10], ...
    'MajorTickLabels', {'Off', 'On'}, ...
    'MinorTicks', [], ...
    'Position', [55 55 92 50]);
```

Add a label to the control value gauge.

```
uilabel(hf, ...
    'HorizontalAlignment', 'center', ...
    'Position', [79 12 44 28], ...
    'Text', {'Control'; 'Value'});
```

Create Start-Stop Button

Add a start or stop button by using the `UIButton` command. This button starts the real-time application and begins streaming data to the instrument panel. This button also stops the real-time application.

```
ssButton = uibutton(hf, 'push', ...
    'Icon', 'run_24.png', ...
    'Position', [18 449 171 41], ...
    'Text', 'Start', ...
    'ButtonPushedFcn', @StartStopButtonPushed);
```

Show Instrument Panel App

After all the UI instruments are created, display the instrument panel.

```
hf.Visible = 'on';
```

Create Instrumentation Object

Creates the instrumentation object for the real-time application:

```
hInst = SimulinkRealTime.prototype.Instrumentation mdl;
```

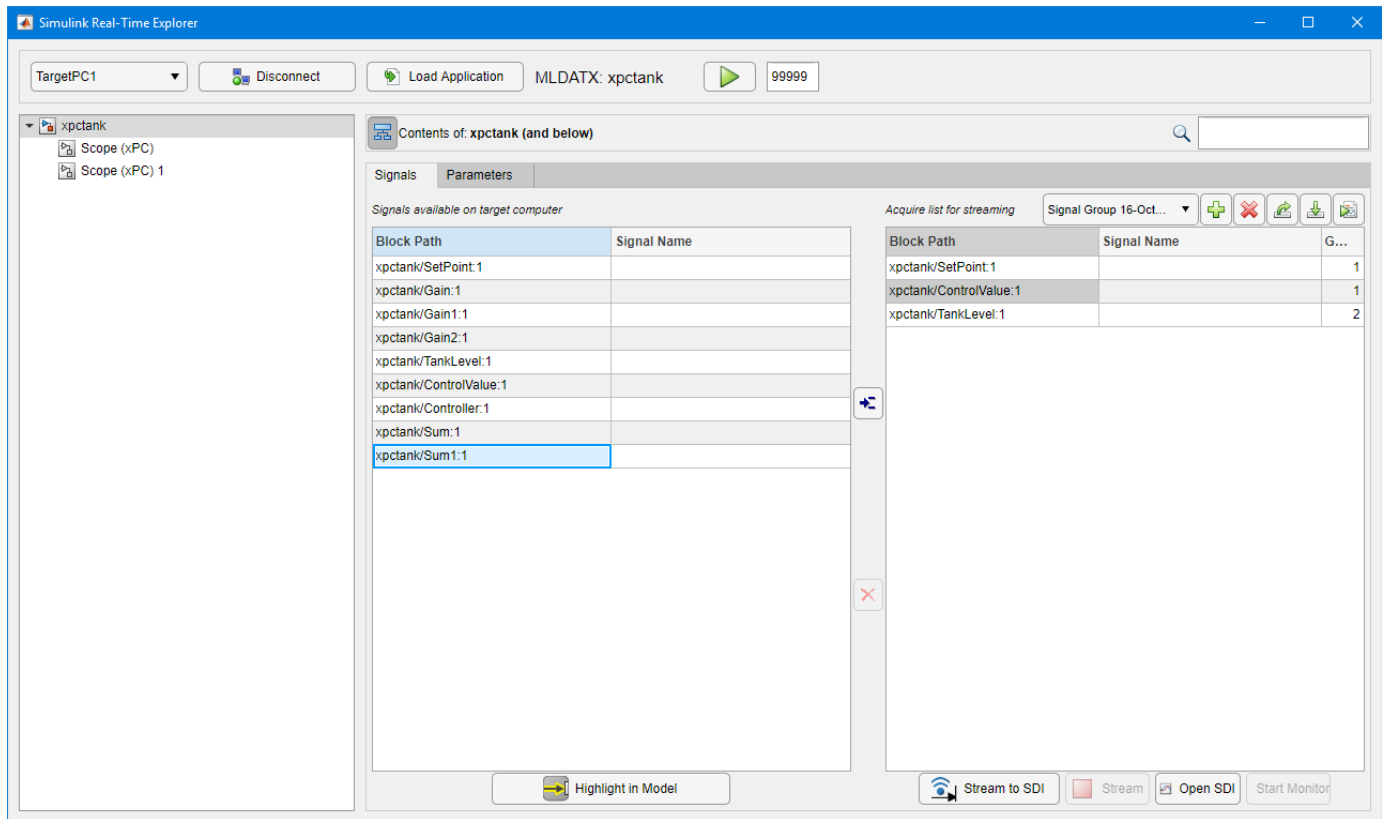
Connect Signals to Axes

The `connectLine` commands connect the signals to lines that are plotted on the axes.

```
[~,line1] = hInst.connectLine(ha,[mdl '/SetPoint'] ,1,'MaximumNumPoints',10000);
[~,line2] = hInst.connectLine(ha,[mdl '/TankLevel'] ,1,'MaximumNumPoints',10000);
[~,line3] = hInst.connectLine(ha,[mdl '/ControlValue'] ,1,'MaximumNumPoints',10000);
```

When using the `connectLine` function to connect output signals from the real-time application to the instrumentation object, the Simulink Real-Time Explorer provides a useful view of the signals and their port numbers. To open the Explorer and view the signals, in the MATLAB Command Window, type:

```
SimulinkRealTime.prototype.Explorer
```



Add Axes Legend

The legend command labels each line in the legend with the signal name.

```
legend(ha, {line1{1}, line2{1}, line3{1}})
```

Connect Signals to Scalar Displays

The connectScalar commands connects the parameter values to the scalar displays. The HistoryFlag is set to '1' because these signals have already been added to the instrumentation object as lines. Their history must be present even though the scalar display only uses the last value.

```
hInst.connectScalar(tlGauge, [mdl '/TankLevel'] ,1, 'HistoryFlag',1);
hInst.connectScalar(cvGauge, [mdl '/ControlValue'] ,1, 'HistoryFlag',1);
```

Add Callback to Wrap x Axis

This connectCallback command helps animate plotting signals on the axes. The command wraps the display of signal data by changing the x origin value to keep the signal data within the plot UI.

```
hInst.connectCallback(@(instObj, eventData)handleAxes(instObj, eventData, ha));
ha.XLim = [0 10];
ha.YLim = [-1 13];
```

Store UI Object Handles in User Data

Add object handles for the UI object. These handles enable adding callbacks to process object events.

```
handle.ha = ha;
handle.tlGauge = tlGauge;
```



```

handle.spSlider = spSlider;
handle.cvGauge  = cvGauge;
handle.ssButton = ssButton;
handle.tg       = tg;
handle.hInst    = hInst;
handle.startIcon = 'run_24.png';
handle.stopIcon  = 'stop_24.png';
hf.UserData = handle;

```

Add Callback to Update XLim and Prevent Axis Wrap

The `handleAxes` callback processes signal data that plots signals on the axes.

```

function handleAxes(instObj, eventData, ha)
currTime = eventData.ExecTime;
if max(currTime)<max(ha.XLim) % do nothing
else
    ha.XLim = max(ha.XLim) + [0 10];
end
end

```

Add Callback for Clean Up on Instrument Panel Close

The `closeRequest` callback processes UI close requests, including stop of the real-time application. The instrumentation object uses callbacks to add streaming data to your UIAxes. If you delete the UIAxes by deleting and closing the UIFigure without first deleting the instrumentation object, the callback state can produce errors or warnings. It is best practice to add a close request callback to your app that stops the instrumentation object and deletes it.

```

function closeRequest(hfigure, event)
handle = hfigure.UserData;
if ~isempty(handle.hInst)
    handle.hInst.stop;
    delete(handle.hInst)
end
if ~isempty(handle.tg)
    handle.tg.stop;
end
delete(hfigure)
end

```

Add Callback to Handle Slider Value Changes

The `SetPointSliderValueChanged` callback processes changes to the slider value by streaming the parameter value data to the real-time application.

```

function SetPointSliderValueChanged(hslider, event)
hfigure = hslider.Parent;
handle = hfigure.UserData;
if ~isempty(handle.tg)
    value = handle.spSlider.Value;
    handle.tg.setparam('SetPoint', 'Value', value);
end
end

```

Add Callback to Handle Button-Push Events

The `StartStopButtonPushed` callback processes button-push events. For **Start** button push, the callback starts the real-time application and toggles the button label to **Stop**. For **Stop** button push, the callback stops the real-time application and toggles the button label to **Start**.

```
function StartStopButtonPushed(hbutton, event)
hfigure = hbutton.Parent;
handle = hfigure.UserData;
if strcmp(handle.ssButton.Icon, handle.startIcon) % START
    handle.hInst.clearData; % clear old data
    handle.hInst.start(handle.tg); % start instrumentation and target
    handle.ssButton.Icon = handle.stopIcon; % change the icon
    handle.ssButton.Text = 'Stop';
else % STOP
    handle.hInst.stop;
    handle.tg.stop;
    handle.ssButton.Icon = handle.startIcon; % change the icon
    handle.ssButton.Text = 'Start';
end
end

bdclose('all')
```

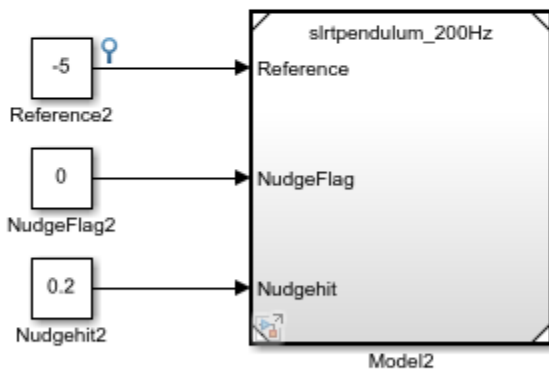
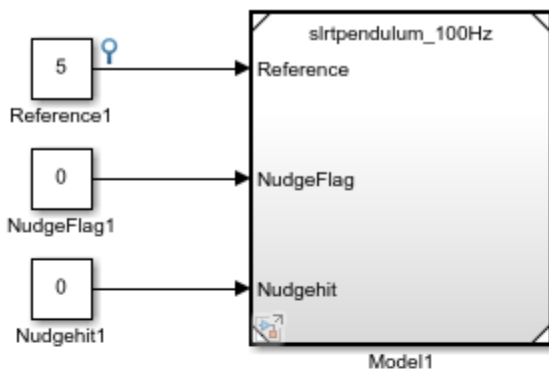
Add App Designer App to Inverted Pendulum Model

This example shows how to stream signal signals to an App Designer instrument panel app from a Simulink Real-Time application. The example builds the real-time application from the model `slrtpendulum`. This model contains referenced models that produce the signals that are streamed and plotted. The instrument panel contains these App Designer components:

- Axes — To display an animation for the two inverted pendulum and cart systems.
- Axes — To display signal output for responses to disrupting the pendulums.
- Nudge buttons — To apply input (nudges) to the carts that hold the pendulums.

To stream signal and parameter data between the real-time application and the instrument panel app, the app uses the instrumentation object.

```
open_system(docpath(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'slrtpendulum
```



Model slrtpendulum
 Simulink Real-Time example model
 Copyright 2019 The MathWorks, Inc.

```
load_system(docpath(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'slrtpendulum
```

```
load_system(docpath(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'slrtpendulum_
```

Start Target Computer and Build Real-Time Application

These tasks generate the real-time application that streams data to the App Designer instrument panel app.

- 1 Start the target computer.
- 2 Open the model `slrtpendulum`.
- 3 Connect the development computer to the target computer. Build the `slrtpendulum` model.
- 4 Deploy the real-time application to the target computer.

In the MATLAB Command Window, type:

```
set_param('slrtpendulum', 'RTWVerbose', 'off');
tg = slrt('TargetPC1');
rtwbuild('slrtpendulum');
load(tg, 'slrtpendulum');

### Starting Simulink Real-Time build procedure for model: slrtpendulum_100Hz
### Starting Simulink Real-Time build procedure for model: slrtpendulum_200Hz
### Starting Simulink Real-Time build procedure for model: slrtpendulum
### Generated code for 'slrtpendulum' is up to date because no structural, parameter or code rep
### Successful completion of build procedure for model: slrtpendulum
### Created MLDATX ..\slrtpendulum.mldatx
```

Run App Designer Instrument Panel App

The App Designer instrument panel app `Pendulum` provides controls to start and interact with the real-time application `slrtpendulum`.

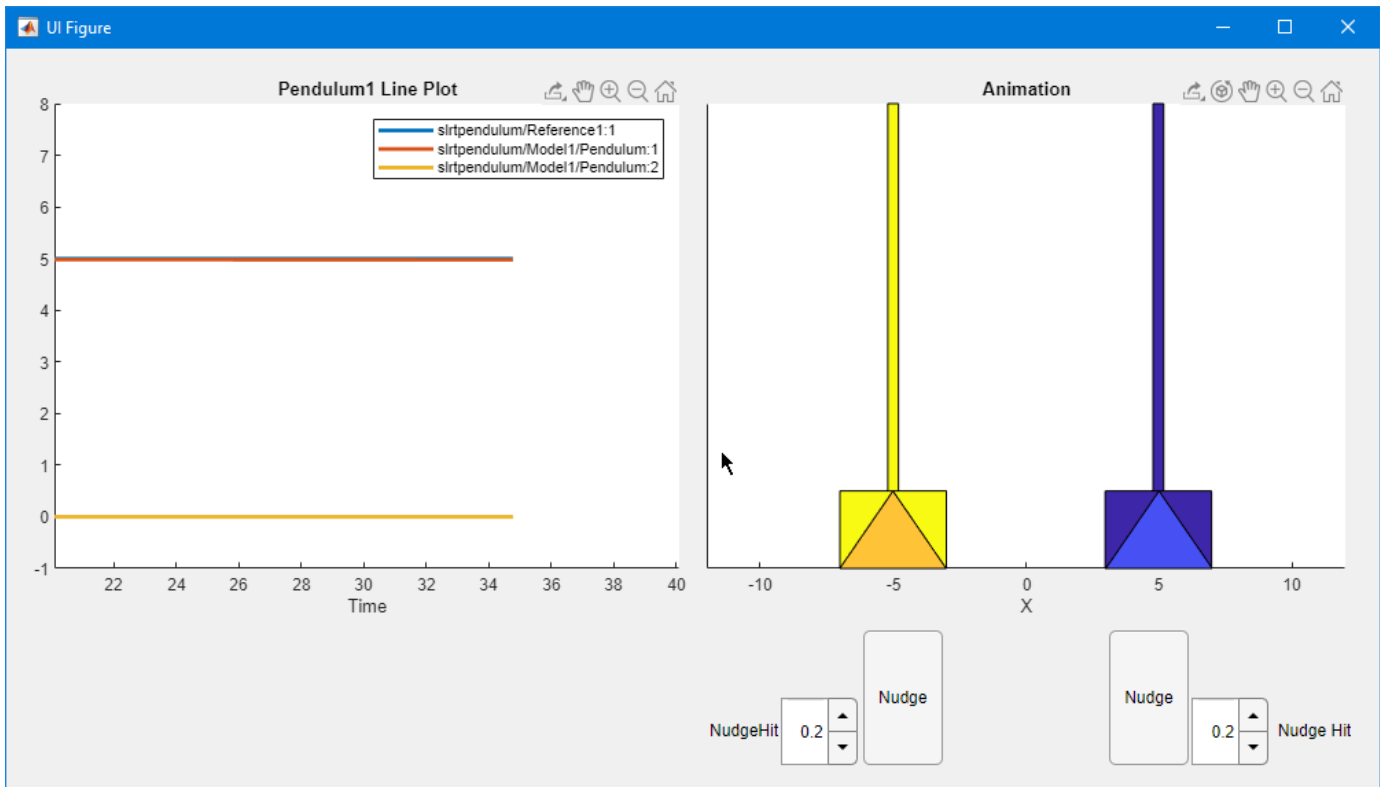
1. Run the app. To start the App Designer app `Pendulum.mlapp` and create the handle `app`, in the MATLAB Command Window, type:

```
app = Pendulum;

Acquire Group 1
  DiscreteInterval = 0.010
  SampleTimeString = 0.01
  HistoryFlag      = 1
  signals:
    slrtpendulum/Reference1:1
    cartposition1
    pendposition1
Acquire Group 2
  DiscreteInterval = 0.005
  SampleTimeString = 0.005
  HistoryFlag      = 1
  signals:
    slrtpendulum/Reference2:1
    cartposition2
    pendposition2
```

As the app starts, it displays the output of the view `AcquiredList` command. This view displays the signal hierarchy in the instrumentation object in the app.

2. To disrupt the equilibrium of the pendulum on each cart, click the **Nudge** buttons . You can adjust the nudge size by using the value selection next to each button.
3. Observe the plot reactions to each nudge. When the nudge value gets too large, the pedulum cannot recover its equilibrium.



App Callback Code

The instrument panel app functionality is provided by callback code. For more information about adding app components and inserting callback code, see “Add App Designer Instrument Panel App to Tank Model” on page 15-59.

Comments in the callback code in the instrument panel app `Pendulum.mlapp` describe the callback operations and programming suggestions. To view the callback code, open `Pendulum.mlapp` in the App Designer, and then click the **Code View** tab. In the Command Window, type:

```
edit Pendulum
```

Specify Block Paths for Signals in Referenced Models

To stream data from signals in referenced models, the `connectLine` and `addSignal` functions for the instrumentation object use a cell array to pass the block path.

For examples, see the `setupInstrumentation(app)` function in the app.

updatePlotAxes Function

This function is a callback with three arguments.

The function uses `eventData.ExecTime` to get the current time on the target computer and use that time to wrap the XLims of the PlotAxes.

updateAnimationCallback Function

For each AcquireGroup, this function checks whether there is fresh data since the last time the callback was called. If there is data, the function updates the animation objects.

Update Axes and Animation by Using Acquire Groups

Often, models have multiple sample rates.

To update plot data and plot animation, the instrumentation object groups data by AcquireGroups. For the `slrtpendulum` example, the two AcquireGroups are at different sample rates.

In the callback code, this processing is visible as `AcquireGroupData` signal groups in the `updateAnimationCallback` function. The app displays these groups in the instrumentation object by using the view command.

```
app.hInst.AcquireList.view
```

Close the App and Models

The instrument panel app handle `app` provides access to close the app.

Close the app. In the MATLAB Command Window, type:

```
close(app.UIFigure)
```

Close the open models. In the Command Window, type:

```
bdclose ('all');
```

Triggering Scope Instruments

An instrument panel that uses scope instruments to trace signals.

This example shows how to trace signals using an instrument panel. The instrument panel contains four scope instruments positioned in a two by two grid. Each scope instrument is configured with different triggering sources: freerun, signal, manual, and scope.

To run this example, click `run example`. This command runs a setup script that builds, downloads, and starts the real-time application, opens Simulink Real-Time Explorer, and loads the instrument panel.

You can then run the instrument panel and interact with the scope instruments.

The screenshot displays the Simulink Real-Time Explorer interface for a real-time application named `scopeInstDemo.slrtp`. The main workspace is divided into four quadrants, each containing a scope instrument with a different triggering method:

- Top-Left: Freerun - Repeated** (ACQUIRING, Offset = 193.471250 s). Shows a signal trace with the description: "Scope1 configured to start another trace as soon as a trace is finished."
- Top-Right: Signal Triggering** (ACQUIRING, Offset = 193.525000 s). Shows a signal trace with the description: "Scope2 configured to trigger from trigger signal (Signal Generator) with rising edge and level 4 to display repeatedly. Any value reaching below the trigger level 4 of the trigger signal will prevent the scope from triggering."
- Bottom-Left: Manual Triggering** (WAITFORTRIG, Offset = 30.408750 s). Shows a step function trace with the description: "Scope3 configured for manual triggering. Click the 'Force Trigger' button to manually start a trace."
- Bottom-Right: Scope Triggering** (WAITFORTRIG, Offset = 30.408750 s). Shows a signal trace with the description: "Scope4 triggers from Scope3."

Below the top two scopes is a horizontal slider labeled "Slider for adjusting amplitude level of trigger signal (Signal Generator)." with a scale from 0 to 20. The slider is currently set to approximately 4.

The interface includes a menu bar (File, Edit, View, Window, Help), a toolbar, and several panels on the left (Targets, Applications, Scopes) and right (Palette, Panels, Properties). The status bar at the bottom indicates "Ready".

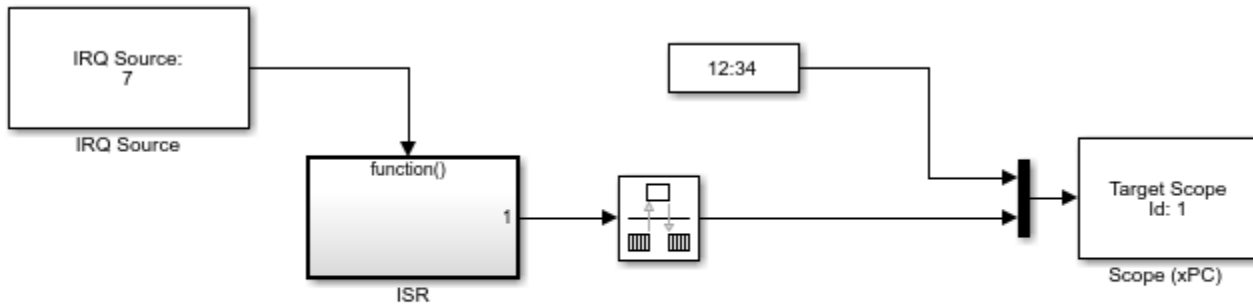
Asynchronous Events

This example model shows how to use asynchronous event support with an external TTL signal firing an interrupt on the parallel port.

This event invokes a function-call subsystem that serves as an interrupt service routine (ISR).

The data exchange between the asynchronous task and the rest of the model (rate monotonic task) is accomplished by using a Rate Transition block.

```
open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcasynctrans'))
```



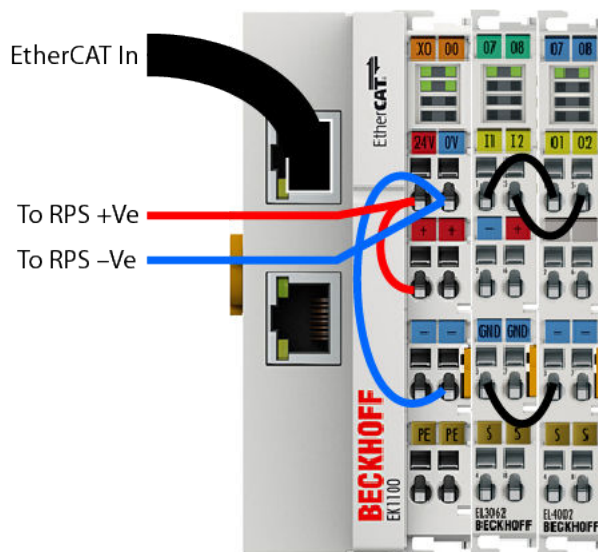
Copyright 2007-2017 The MathWorks, Inc.

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

This example shows how to communicate with EtherCAT devices using the Beckhoff® analog I/O terminals EL3062 and EL4002.

Requirements

To run this example, you need an EtherCAT network that consists of the target computer as EtherCAT Master device and two analog input/output terminals EL3062 and EL4002 as EtherCAT Slave devices. This example requires a dedicated network card that is installed and available on the target computer. Use the dedicated card for the EtherCAT communication. The dedicated card is in addition to the card used for the Ethernet link between the development and target computers.



To test this model:

- 1 Connect the network port of the dedicated card in the target computer to the network IN port of the Beckhoff® EK1100 coupler.
- 2 Assemble Terminals EL3062 and EL4002 with Coupler EK1100.
- 3 Loop back the I/O ports: Connect each output port of Terminal EL4002 to a corresponding input port of Terminal EL3062.
- 4 Make sure that the terminals are supplied with the required 24-volt power supply.
- 5 Build and download the model onto the target.

For a complete example that configures the EtherCAT network, configures the EtherCAT master node model, and builds then runs the real-time application, see the Simulink Real-Time EtherCAT documentation.

Open the Model

To open the model, in the Command Window, type:

```
xpcEthercatBeckhoffAI0
```

This model creates two sine wave signals and sends the signals to the EL4002 terminal. Then, the model receives input signal values from the EL3062 terminal.

The EtherCAT initialization block requires that the configuration ENI file is present in the current folder. Copy the example configuration file from the example folder to the current folder. Then, open the model.

```
copyfile(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'BeckhoffAI0config.xml')
copyfile(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcEthercatBeckhoffAI0.
mdl = 'xpcEthercatBeckhoffAI0';
mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if isempty( strcmp(systems, mdl ) )
    mdlOpen = 1;
    open_system(mdl);
end
```

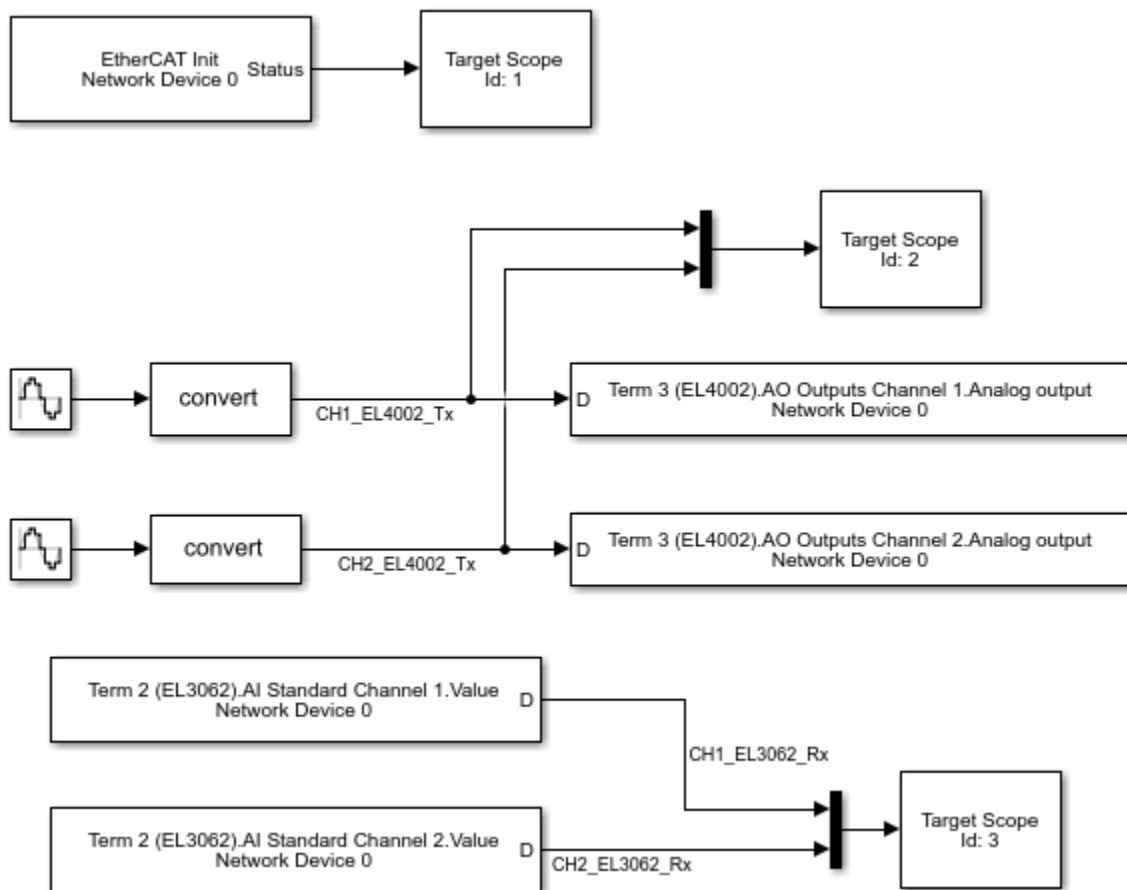


Figure 1: EtherCAT model using Beckhoff® analog I/O slave devices EL3062 and EL4002.

Configure the Model

Open the mask for the EtherCAT Init block and provide the required values for the PCI bus and slot numbers for the network card being used for EtherCAT communication. To get these values, in

the Command Window, type `tg.getPCIInfo('ethernet')`. An example command to set configuration parameters for the EtherCAT Init block is:

```
set_param('xpcEthercatBeckhoffAI0/EtherCAT Init ','pci_bus','5','pci_slot','0','pci_function','0')
```

Describe Network with Configurator

Using a third-party EtherCAT configurator that you install on a development computer, generate an EtherCAT configuration file `BeckhoffAI0config.xml`. This file describes the network to the master. An overview of the process for creating the configuration file in the EtherCAT configurator is:

- 1 Connect the network (consisting of terminals EK1100, EL3062, and EL4002 in this example) to the computer where the EtherCAT configurator is installed and scan the network to discover the connected devices.
- 2 Select the transmit and receive variables to be accessed as signals from the IO terminals.
- 3 Define at least one cyclic task, select a task execution rate, and associate the desired input/output variables to the task. These input/output variables must belong to previously defined transmit/receive PDOs (for example, PDOs defined in step 2) and be linked to the required terminals. You only must to select one variable from each PDO to make every variable in that PDO accessible.
- 4 Export the configuration file into an XML file. Make sure the name of the XML file is different from the name of your Simulink® model.

Each EtherCAT configuration file is specific to the exact network setup for which it has been created (for example, the network discovered in step 1 of the configuration file creation process). The configuration file provided for this example is valid if and only if the EtherCAT network consists of terminals EK1100, EL3062, and EL4002.

The configuration file defines a set of transmit and receive variables. For this example, a set of receive variables are defined for each input channel of terminal EL3062. Make sure the variables for channel 1 and channel 2 of terminal EL3062 are selected respectively in the two EtherCAT PDO Receive blocks. These two variables are 'Term 2 (EL3062).AI Standard Channel 1.Value' and 'Term 2 (EL3062).AI Standard Channel 2.Value'. In the same way, a set of transmit variables are defined for the two output channels of terminal EL4002. Make sure the variables for channel 1 and channel 2 of terminal EL4002 are selected in the two EtherCAT PDO Transmit blocks. These two variables are 'Term 3 (EL4002).AO Outputs Channel 1.Analog Output' and 'Term 3 (EL4002).AO Outputs Channel 2.Analog Output'.

Build, Download, and Run the Model

Build the model and download to the target computer. Let the model run for 10 seconds

```
set_param mdl, 'RTWVerbose', 'off';
rtwbuild(mdl);
tg = slrt('TargetPC1');
load(tg, mdl);
tg.CommunicationTimeOut=20;
start(tg);
pause(10);
```

```
### Starting Simulink Real-Time build procedure for model: xpcEthercatBeckhoffAI0
Warning: This model contains blocks that do not handle sample time
changes at runtime. To avoid incorrect results, only change
the sample time in the original model, then rebuild the model.
```

```
### Successful completion of build procedure for model: xpcEthercatBeckhoffAI0
### Created MLDATX ..\xpcEthercatBeckhoffAI0.mldatx
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
```

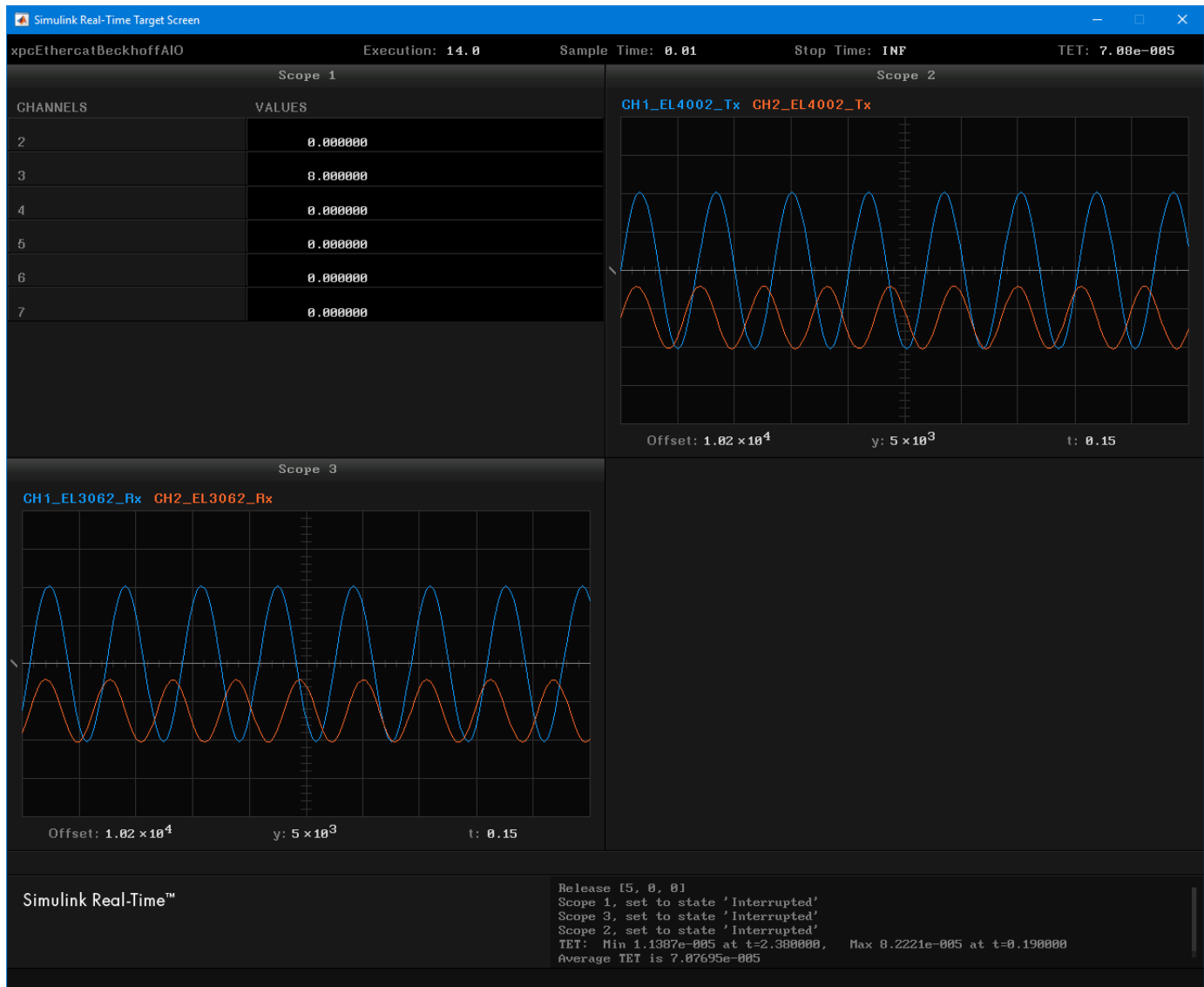
Display the Target Computer Scopes

Take a snapshot of the target computer video display. Plotted are the signals transmitted to Terminal EL4002 and received from Terminal EL3062. As expected, the transmitted and received signals displayed on the two scopes are identical, except the offset introduced by the transmission delay.

- **Scope 1** displays the outputs of the Ethercat Init block. See the documentation of this block for the meaning of the displayed values.
- **Scope 2** displays the sine waves generated by the application and sent to channels 1 and 2 of Terminal EL4002 by the Master.
- **Scope 3** displays the sine waves received from channels 1 and 2 of Terminal EL3062 via the external wire connections.

To take a snapshot of the target scopes, type:

```
tg.viewTargetScreen
```



Stop and Close the Model

When the example completes its run, stop and close the model.

```

stop(tg);
if (mdlOpen)
    save_system(mdl);
    close_system(mdl);
end

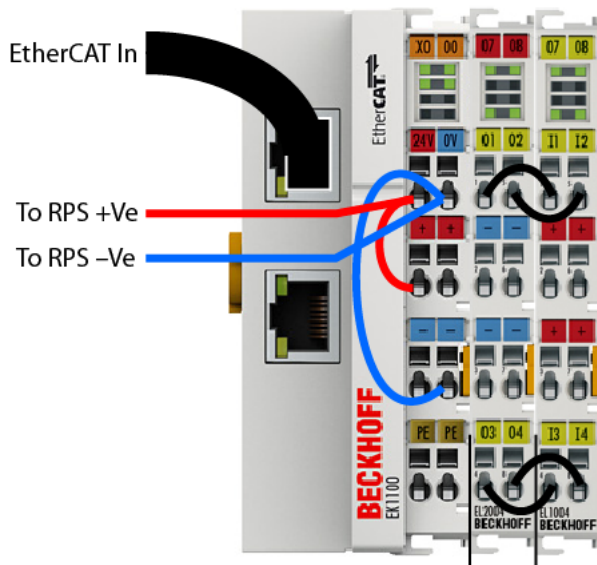
```

EtherCAT® Communication with Beckhoff® Digital IO Slave Devices EL1004 and EL2004

This example shows how to communicate with EtherCAT devices using the Beckhoff digital I/O terminals EL1004 and EL2004.

Requirements

To run this example, you need an EtherCAT network that consists of the target computer as EtherCAT Master device and two analog input/output terminals EL1004 and EL2004 as EtherCAT Slave devices. This example requires a dedicated network card that is installed and available on the target computer. Use the dedicated card for the EtherCAT communication. The dedicated card is in addition to the card used for the Ethernet link between the development and target computers.



To test this model:

- 1 Connect the network port of the dedicated card in the target computer to the network IN port of the Beckhoff® EK1100 coupler.
- 2 Assemble Terminals EL1004 and EL2004 with Coupler EK1100.
- 3 Loop back the first two I/O ports: Connect ports numbered 1 and 5 of Terminal EL2004 to ports numbered 1 and 5 of Terminal EL1004.
- 4 Make sure that the terminals are supplied with the required 24-volt power supply.
- 5 Build and download the model onto the target.

For a complete example that configures the EtherCAT network, configures the EtherCAT master node model, and builds then runs the real-time application, see the Simulink Real-Time EtherCAT documentation.

Open the Model

To open the model, in the Command Window, type:

```
xpcEthercatBeckhoffDIO
```

This model drives a pulse wave signal and transmits the signal and its inverse as Boolean values to the EL2004 terminal, and receives the input signal transmitted by the EL1004 terminal.

The EtherCAT initialization block requires that the configuration ENI file is present in the current folder. Copy the example configuration file from the example folder to the current folder. Then, open the model.

```
copyfile(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'BeckhoffDI0config.xml'),
copyfile(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcEthercatBeckhoffDI0.
mdl = 'xpcEthercatBeckhoffDI0';
mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if isempty( strcmp(systems, mdl) )
    mdlOpen = 1;
    open_system(mdl);
end
```

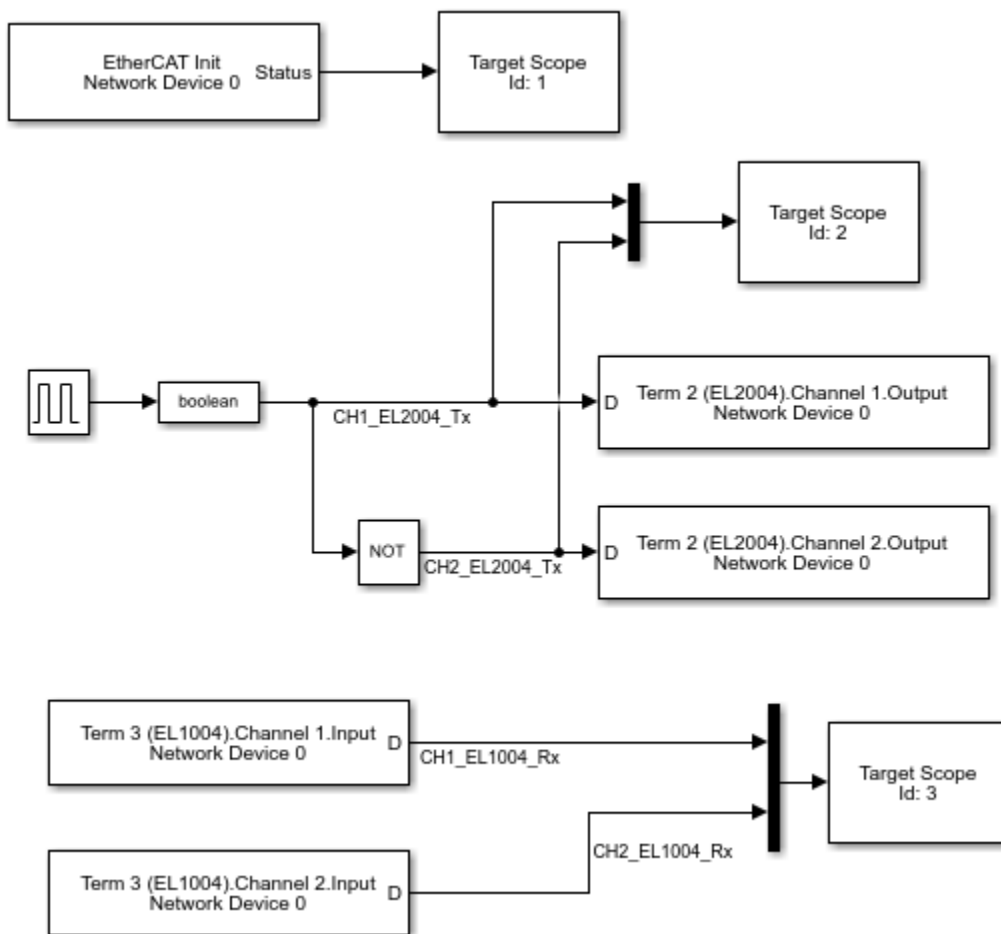


Figure 1: EtherCAT model using Beckhoff® digital I/O terminals EL1004 and EL2004.

Configure the Model

Open the mask for the EtherCAT Init block and provide the required values for the PCI bus and slot numbers for the network card being used for EtherCAT communication. To get these values, in

the Command Window, type `tg.getPCIInfo('ethernet')`. An example command to set configuration parameters to for the EtherCAT Init block is:

```
set_param('xpcEthercatBeckhoffDIO/EtherCAT Init ','pci_bus','5','pci_slot','0','pci_function','0')
```

Describe Network with Configurator

Using a third-party EtherCAT configurator that you install on a development computer, generate an EtherCAT configuration file `BeckhoffDIOconfig.xml`. This file describes the network to the master. An overview of the process for creating the configuration file in the EtherCAT configurator is:

- 1 Connect the network (consisting of terminals EK1100, EL1004, and EL2004 in this example) to the computer where the EtherCAT configurator is installed and scan the network to discover the connected devices.
- 2 Select the transmit and receive variables to be accessed as signals from the IO terminals.
- 3 Define at least one cyclic task, select a task execution rate, and associate the selected variables to the task. You only must select one variable from each PDO to make every variable in that PDO accessible.
- 4 Export the configuration file into an XML file. Make sure the name of the XML file is different from the name of your Simulink® model.

Each EtherCAT configuration file is specific to the exact network setup from which it was created (for example, the network discovered in step 1 of the configuration file creation process). The configuration file provided for this example is valid if and only if the EtherCAT network consists of Terminals EK1100, EL1004, and EL2004 from Beckhoff®.

The configuration file defines a set of transmit and receive variables. For this example, four receive variables are defined for the four input channels of Terminal EL1004. Only the first two channels of Terminal EL1004 are used in this example. Make sure the receive variables for channel 1 and channel 2 of terminal EL1004 are selected respectively in the two EtherCAT PDO Receive blocks. These two variables are 'Term 3 (EL1004).Channel 1.Input' and 'Term 3 (EL1004).Channel 2.Input'. In the same way, four transmit variables are defined for the four output channels of terminal EL2004, but only the first two channels are tested in this example. Make sure the transmit variables for channel 1 and channel 2 of terminal EL2004 are selected respectively in the two EtherCAT PDO Transmit blocks. These two variables are 'Term 2 (EL2004).Channel 1.Output' and 'Term 2 (EL2004).Channel 2.Output'.

Build, Download, and Run the Model

Build the model and download to the target computer. Let the model run for 10 seconds

```
set_param mdl, 'RTWVerbose', 'off';
rtwbuild(mdl);
tg = slrt('TargetPC1');
load(tg, mdl);
tg.CommunicationTimeOut=20;
start(tg);
pause(10);
```

```
### Starting Simulink Real-Time build procedure for model: xpcEthercatBeckhoffDIO
### Generated code for 'xpcEthercatBeckhoffDIO' is up to date because no structural, parameter or
### Successful completion of build procedure for: xpcEthercatBeckhoffDIO
### Created MLDATX ..\xpcEthercatBeckhoffDIO.mldatx
```

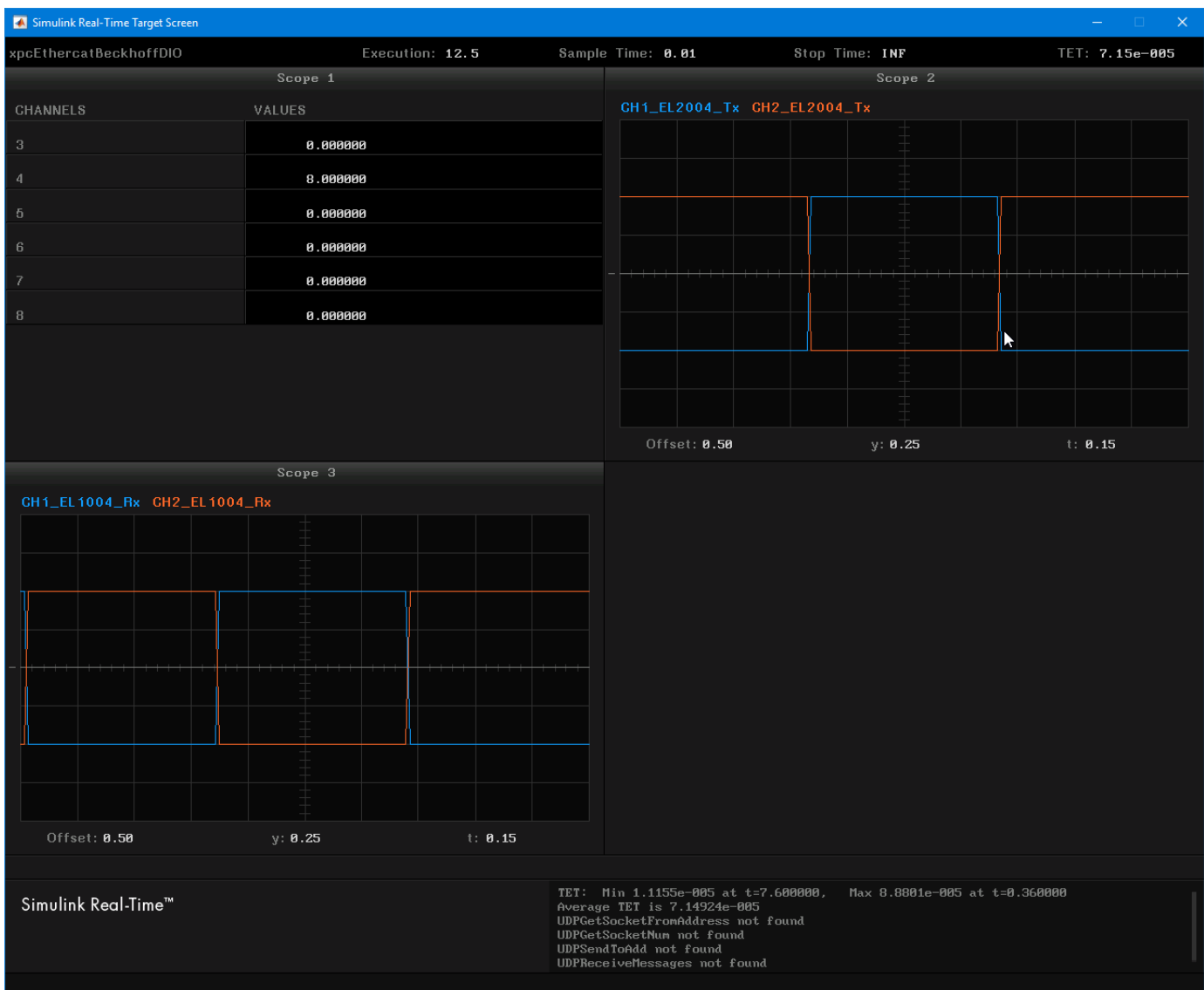

Display the Target Computer Scopes

Take a snapshot of the target computer video display. Plotted are the signals transmitted to terminal EL2004 and received from terminal 1004. As expected, the transmitted and received signals displayed on the two scopes are identical.

- **Scope 1** displays the outputs of the Ethercat Init block. See the documentation of this block for the meaning of the displayed values.
- **Scope 2** displays the pulse wave and its inverse generated by the application and sent to Terminal EL2004 by the EtherCAT Master.
- **Scope 3** displays the signals received at the two inputs ports of Terminal EL1004.

To take a snapshot of the target scopes, type:

```
tg.viewTargetScreen
```



Stop and Close the Model

When the example completes its run, stop and close the model.

```
stop(tg);  
if (mdlOpen)  
    save_system(mdl);  
    close_system(mdl);  
end
```

EtherCAT® Communication - Motor Velocity Control with Accelnet™ Drive

This example shows how to control the velocity of a motor using EtherCAT communication.

Requirements

To run this example, you need an EtherCAT network that consists of the target computer as EtherCAT Master device and an Accelnet™ AEP 180-18 drive from Copley Controls as EtherCAT Slave device. Connect a supported brushless or brush motor to the drive. An example motor that works with this example is the SM231BE-NFLN from PARKER.

This example requires a dedicated network card that is installed and is available on the target computer. Use the dedicated card for the EtherCAT communication. The dedicated card is in addition to the card used for the Ethernet link between the development and target computers.

To test this model:

- 1 Connect the network port of the dedicated card in the target computer to the EtherCAT IN port of the Accelnet™ drive.
- 2 Connect a motor to the Accelnet™ drive.
- 3 Make sure that the Accelnet™ drive is supplied with a 24-volt power supply.
- 4 Build and download the model onto the target.

For a complete example that configures the EtherCAT network, configures the EtherCAT master node model, and builds then runs the real-time application, see the Simulink Real-Time EtherCAT documentation.

Open the Model

To open the model, in the Command Window, type:

```
open_system('xpcEthercatVelocityControl')
```

This model sends a varying velocity command to the drive.

The EtherCAT initialization block requires that the configuration ENI file is present in the current folder. Copy the example configuration file from the example folder to the current folder. Then, open the model.

```
copyfile(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'CopleyMotorVelocityConf', 'eni'),
fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcEthercatVelocityControl', 'eni'));
mdl = 'xpcEthercatVelocityControl';
mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if isempty( strcmp(systems, mdl) )
    mdlOpen = 1;
    open_system(mdl);
    set_param('xpcEthercatVelocityControl/EtherCAT Init ', 'pci_bus', '5', 'pci_slot', '0', 'pci_function', '0');
end
```

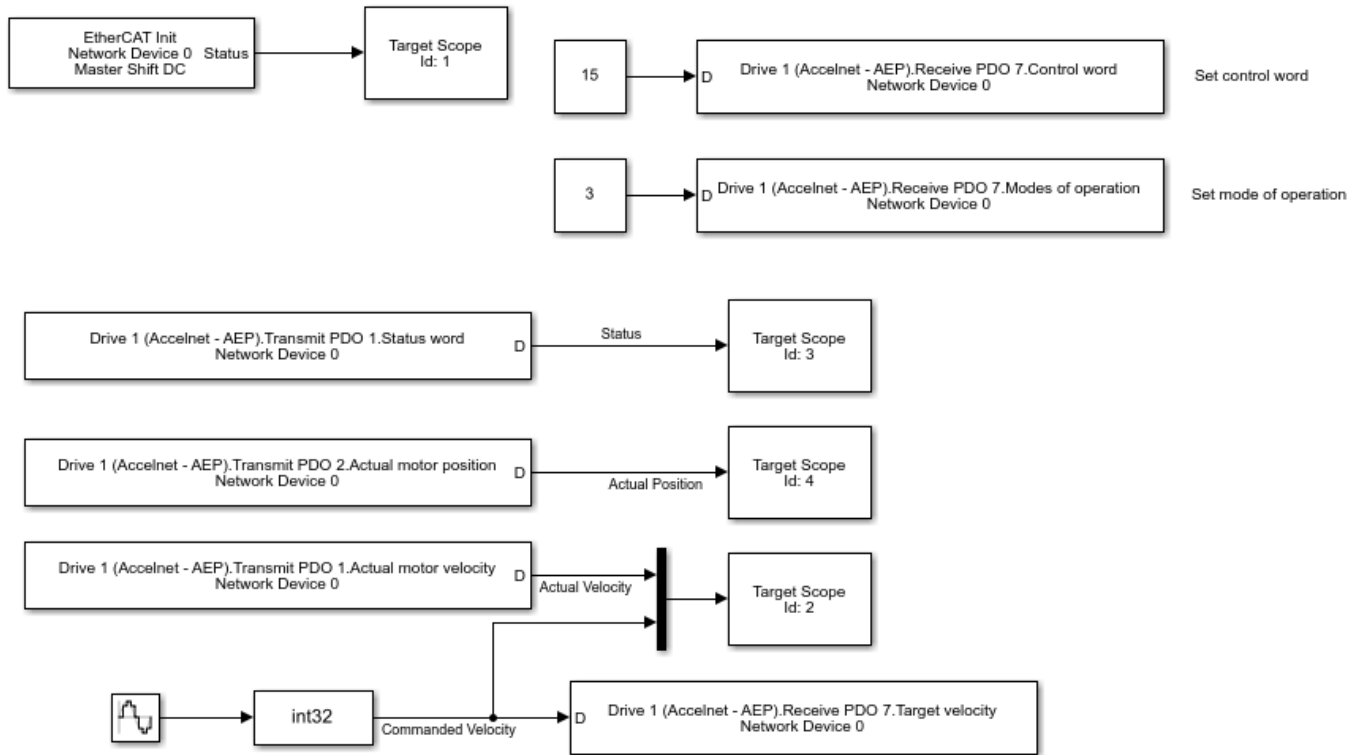


Figure 1: EtherCAT model for motor velocity control.

Configure the Model

Open the mask for the EtherCAT Init block and provide the required values for the PCI bus and slot numbers for the network card being used for EtherCAT communication. To get these values, in the Command Window type `tg.getPCIInfo('ethernet')`. An example command to set configuration parameters for the EtherCAT Init block is:

```
set_param('xpcEthercatVelocityControl/EtherCAT Init','pci_bus','5','pci_slot','0','pci_function')
```

Describe Network with Configurator

Using a third-party EtherCAT configurator that you install on a development computer, generate an EtherCAT configuration file `CopleyMotorVelocityConfig.xml`.

This file describes the network to the master. For more information, see the Simulink Real-Time EtherCAT documentation.

An overview of the process for creating the configuration file in the EtherCAT configurator is:

- 1 Connect the EtherCAT network (consisting of the Accelnet™ drive in this example) to the computer where the EtherCAT configurator is installed and scan the network to discover the connected slave devices.
- 2 Select the transmit and receive Process Data Object (PDO) variables to be accessed from the Accelnet™ drive.
- 3 Define at least one cyclic task and associate the selected PDO variables to the task.

- 4 Export the configuration file into an XML file. Make sure the name of the XML file is different from the name of your Simulink® model.

Each EtherCAT configuration file is specific to the exact network setup for which it was created (for example, the network discovered in step 1 of configuration file creation). The configuration file provided for this example is valid if and only if the EtherCAT network consists of an Accelnet™ AEP drive from Copley Controls.

For this example, three receive PDO variables are defined and used in the three EtherCAT PDO Transmit blocks: Control Word, Modes of Operation and Target Velocity.

- The Control Word PDO variable serves to control the state of the drive. The constant value 15 is given as input to the block to set the first 4 bits to 1 to enable the drive. Refer to the CANOpen manual from Copley Controls for details on the bits mapping of this variable.
- The Modes of Operation PDO variable serves to set the drive operating mode. The constant value 3 is given as input to the block to set the mode of the drive to 'Profile Velocity mode'. Refer to the CANOpen manual from Copley Controls for details on supported modes of operation.
- The Target Velocity PDO variable serves to set the desired velocity. In this example, the velocity command at the input of the block can be tuned through the gain block.

Three transmit PDO variables are also defined in the configuration file and used in the three EtherCAT PDO Receive blocks: Status Word, Actual Motor Velocity, and Actual Motor Position.

- The Status Word PDO variable indicates the current state of the drive.
- The Actual Motor Velocity and Actual Motor Position PDO variables indicate the current values of the motor velocity and position as read in the drive.

Make sure that the required transmit and receive PDO variables are selected in the blocks as illustrated in Figure 1 before running the example. (You could need to refresh these variables.)

Build, Download, and Run the Model

Build the model and download to the target computer. Then, run the model.

```
set_param mdl, 'RTWVerbose', 'off';
rtwbuild(mdl);
tg = slrt('TargetPC1');
load(tg, mdl);
start(tg);
```

```
### Starting Simulink Real-Time build procedure for model: xpcEthercatVelocityControl
Warning: This model contains blocks that do not handle sample time
changes at runtime. To avoid incorrect results, only change
the sample time in the original model, then rebuild the model.
### Successful completion of build procedure for model: xpcEthercatVelocityControl
### Created MLDATX ..\xpcEthercatVelocityControl.mldatx
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
```

The velocity command for the motor is a low frequency sine wave. The actual velocity read back from the controller is delayed by one sample time and the actual position is out of phase by 90 degrees from the velocity, as expected.

```
pause(20);
```

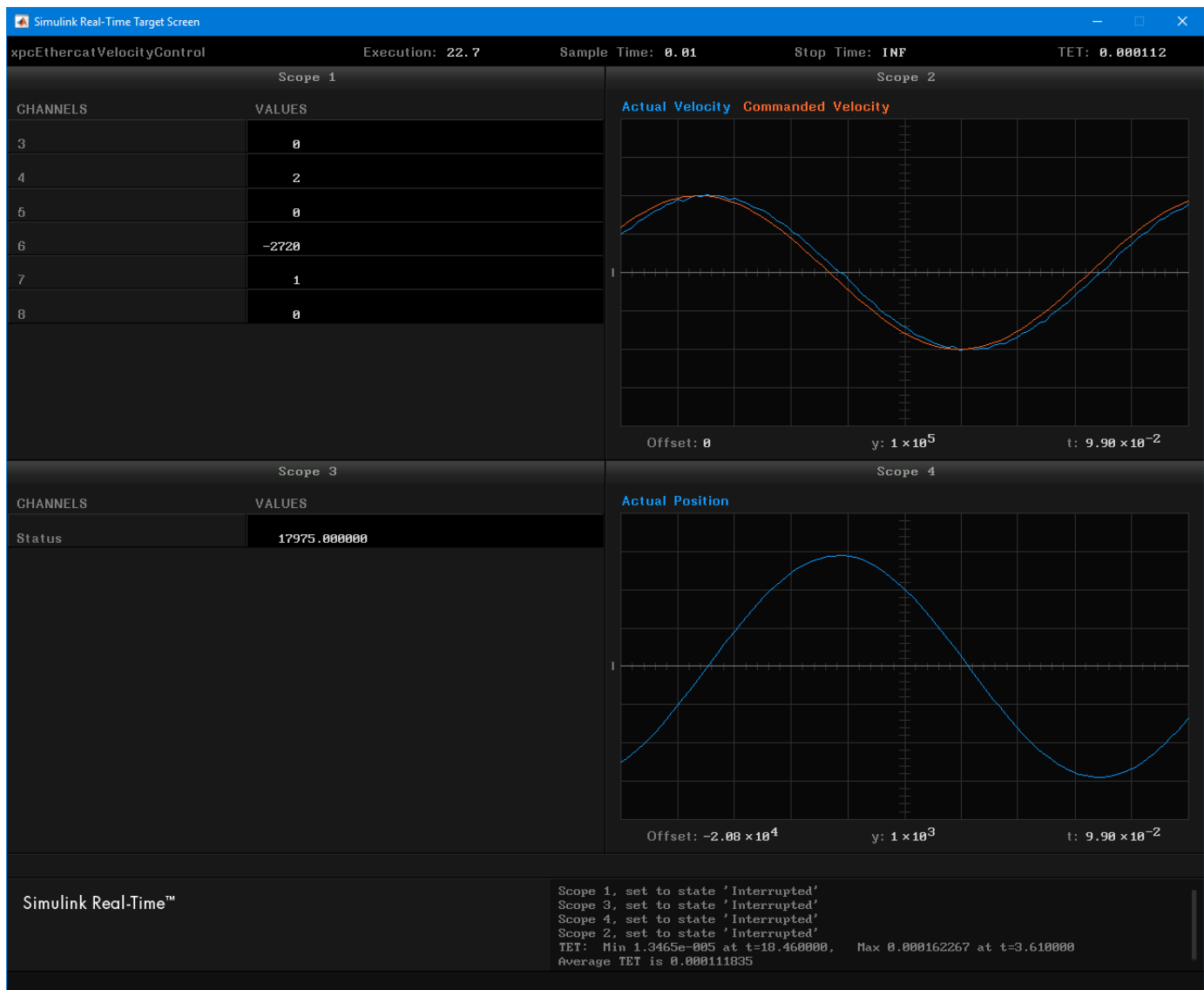
Display the Target Computer Scopes

Take a snapshot of the target computer video display.

- **Scope 1** displays the outputs of the Ethercat Init block. See the documentation of this block for the meaning of the displayed values.
- **Scope 2** displays the PDO variables received and transmitted to the drive, once the drive initializes and the state goes to Op state (=8)

To take a snapshot of the target scopes, type:

```
tg.viewTargetScreen
```



Stop and Close the Model

When the example completes its run, stop and close the model.

```
stop(tg);  
if (mdlOpen)  
    save_system(mdl);  
    close_system(mdl);  
end
```

EtherCAT® Communication - Motor Position Control with an Accelnet™ Drive and Beckhoff® Analog IO Devices

This example shows how to control the position of a motor using EtherCAT communication.

Requirements

To run this example, you need an EtherCAT network that consists of the target computer as EtherCAT Master device and an Accelnet™ AEP 180-18 drive from Copley Controls as EtherCAT Slave device. Connect a supported brushless or brush motor to the drive. An example motor that works with this example is the SM231BE-NFLN from PARKER.

This example requires a dedicated network card that is installed and available on the target computer. Use the dedicated card for the EtherCAT communication. The dedicated card is in addition to the card used for the Ethernet link between the development and target computers.

To test this model:

- 1 Connect the network port of the dedicated card in the target computer to the EtherCAT IN port of the Accelnet™ drive.
- 2 Connect the EtherCAT OUT port of the Accelnet™ drive to the EtherCAT IN port of the Beckhoff® EK1100 coupler.
- 3 Assemble the Beckhoff® EK1100 coupler and the Beckhoff® IO terminals EL3062 and EL4002.
- 4 Connect a motor to the Accelnet™ Drive.
- 5 Connect a variable power supply to the Input port 1 of the EL3062 terminal.
- 6 Make sure the Accelnet™ drive and the Beckhoff® terminals are supplied with a 24-volt power source.
- 7 Build and download the model onto the target.

For a complete example that configures the EtherCAT network, configures the EtherCAT master node model, and builds then runs the real-time application, see the Simulink Real-Time EtherCAT documentation.

Open the Model

To open the model, in the Command Window, type:

```
slrtex_ethercat_position_control
```

This model creates a sine wave, and modulates it by multiplying by the value of the signal present at the first input port of Terminal EL3062. The modulated signal is sent as motor position command to the drive.

The EtherCAT initialization block requires that the configuration ENI file is present in the current folder. Open the model.

```
mdl = 'slrtex_ethercat_position_control';
mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if isempty( strcmp(systems, mdl) )
    mdlOpen = 1;
    open_system(fullfile(matlabroot, 'toolbox', 'slrt', 'slrtexamples', 'slrtex_ethercat_position_cont
end
```

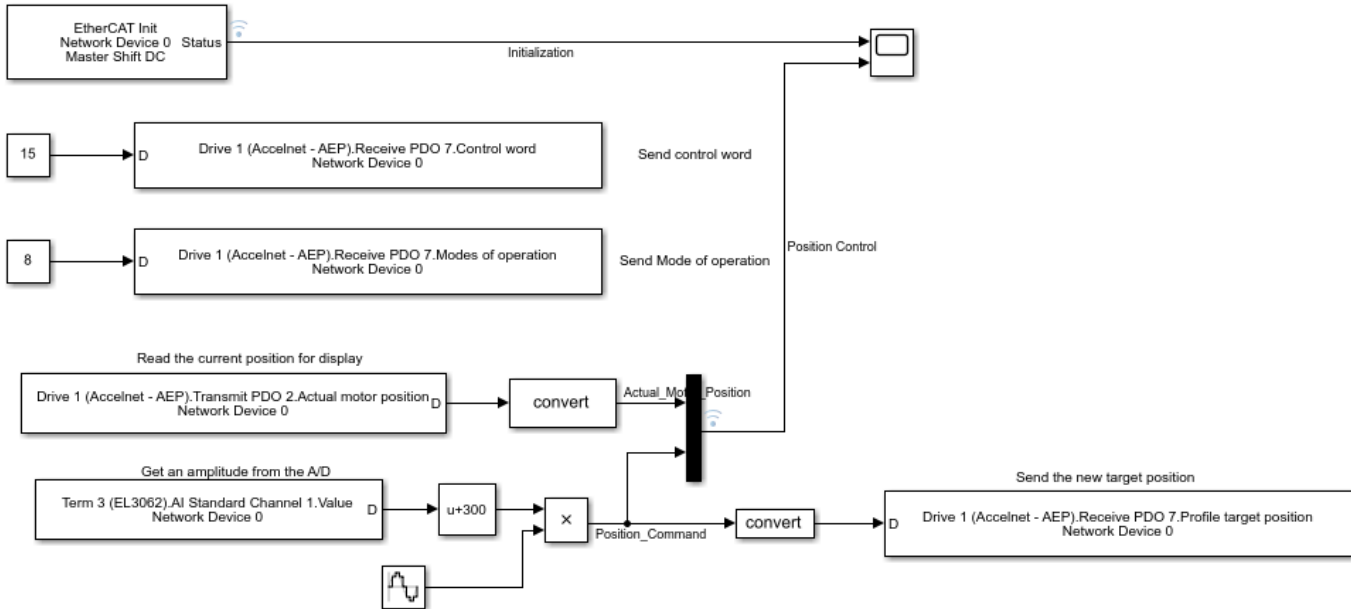



Figure 1: EtherCAT model for controlling the position of a motor through an analog input terminal.

Configure the Model

Open the mask for the EtherCAT Init block and provide the required values for the PCI bus and slot numbers for the network card being used for EtherCAT communication. To get these values, in the Command Window, type `tg.getPCIInfo('all')`. An example command to set configuration parameters for the EtherCAT Init block is:

```
set_param('slrtex_ethercat_position_control/EtherCAT Init ', 'pci_bus', '5', 'pci_slot', '0', 'pci_fu
```

Describe Network with Configurator

Using a third-party EtherCAT configurator that you install on a development computer, generate an EtherCAT configuration file `CopleyMotorPositionConfig.xml`.

This file describes the network to the master. For more information, see the Simulink Real-Time EtherCAT documentation.

An overview of the process for creating the configuration file in the EtherCAT configurator is:

- 1 Connect the EtherCAT network (consisting of the Accelnet drive, terminal EK1100, EL3062, and EL4002 in this example) to the computer where the EtherCAT configurator is installed and scan the network to discover the connected slave devices.
- 2 Select the transmit and receive variables to be accessed as signals from the Beckhoff® IO terminals and the Process Data Objects (PDOs) variables to be accessed from the Accelnet™ drive.
- 3 Define at least one cyclic task, select a task execution rate, and associate the selected IO and PDO variables to the task. You only must select one variable from each PDO to make every variable in that PDO accessible.
- 4 Export the configuration file into an XML file. Make sure the name of the XML file is different from the name of your Simulink® model.

- 5 Close or disconnect configurator from the EtherCAT network or you could get interference between Simulink Real-Time and configurator.

Each EtherCAT configuration file is specific to the exact network setup for which it was created. (For example, the network discovered in step 1 of the configuration file creation process.) The configuration file provided for this example is valid if and only if the EtherCAT network consists of an Accelnet™ drive from Copley Controls and terminals EK1100, EL3062, and EL4002 from Beckhoff®.

For this example, five receive PDO variables are defined in the configuration file and three are used in the three EtherCAT PDO Transmit blocks: Control Word, Modes of Operation, and Profile Target Position.

- The Control Word PDO variable serves to control the state of the drive. The constant value 15 is given as input to the block to set the first 4 bits to 1 to enable the drive. Refer to the CANOpen manual from Copley Controls for details on the bits mapping of this variable.
- The Modes of Operation PDO variable serves to set the operating mode of the drive. The constant value 8 is given as input to the block to set the mode of the drive to 'Cyclic Synchronous Position mode'. Refer to the CANOpen manual from Copley Controls for details on supported modes of operation.
- The Profile Target Position PDO variable serves to set the desired position. In this example, the position command given as input to the block is a sine wave modulated by the signal read at the first input channel of terminal EL3062.

Transmit PDO variables are also defined in the configuration file and two are used in the two EtherCAT PDO Receive blocks: 'Actual Motor Position' for the drive and 'Channel 1.Value' for the EL3062 terminal. The Actual Motor Position PDO variable indicates the current value of the motor position as read in the drive. Make sure the required transmit and receive PDO variables are selected in the blocks before running the example (you could need to refresh these variables).

Build, Download, and Run the Model

Build the model and download to the target computer. Then, run the model. Let the model run for 20 seconds.

```
set_param mdl, 'RTWVerbose', 'off';
rtwbuild(mdl);
tg = slrt('TargetPC1');
load(tg, mdl);
start(tg);
pause(20);
```

```
### Starting Simulink Real-Time build procedure for model: slrtex_ethercat_position_control
### Generated code for 'slrtex_ethercat_position_control' is up to date because no structural, p
### Successful completion of build procedure for model: slrtex_ethercat_position_control
### Created MLDATX ..\slrtex_ethercat_position_control.mldatx
```

Display the signals in the Simulation Data Inspector

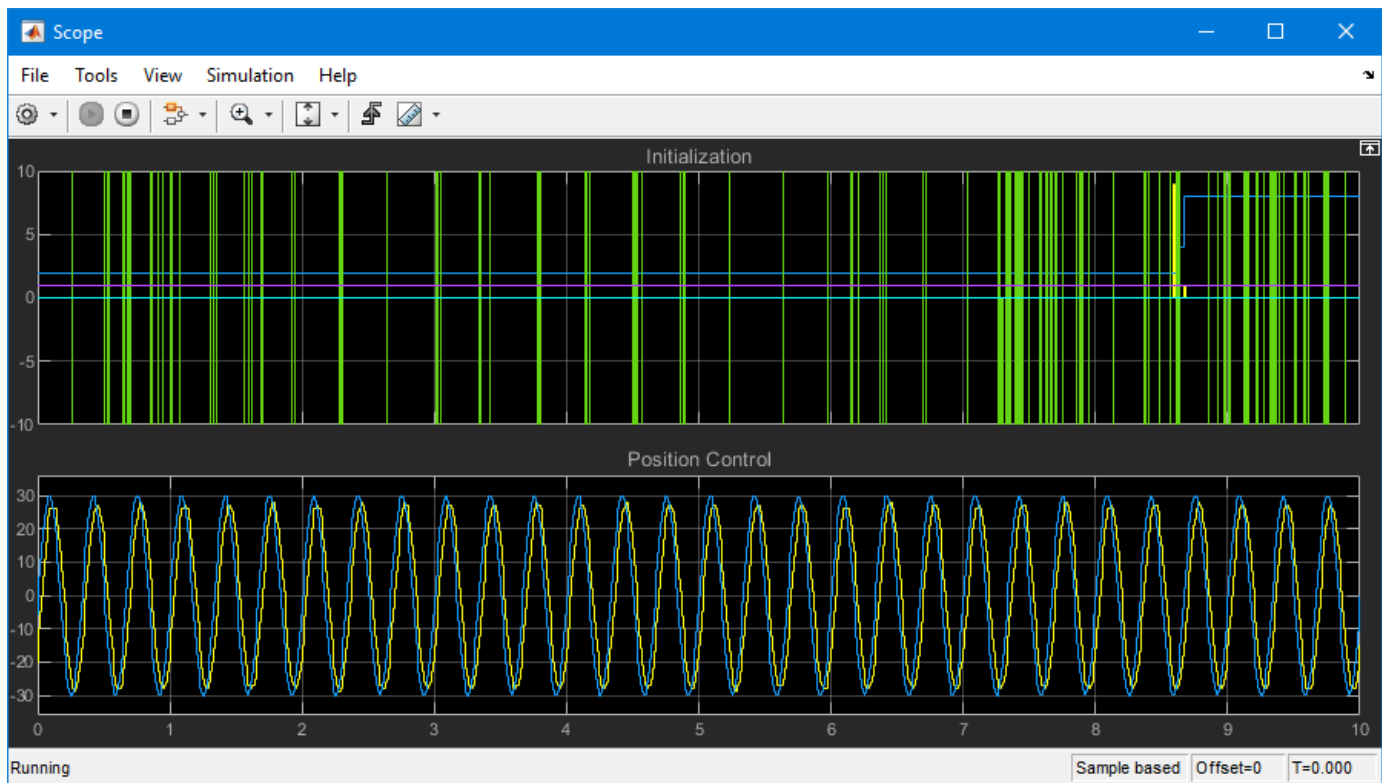
The position command for the motor is varied following the modulated sine wave. The motor turns alternatively in one direction and its opposite. By varying the amplitude of the voltage at Input port 1 of terminal EL3062 between 0 and 10 volts, the amplitude by which the position of the motor changes increases or decreases in same proportion.

- The **Initialization** trace displays the outputs of the Ethercat Init block. See the documentation of this block for the meaning of the displayed values.
- **Position Control** trace displays the PDOs received from and sent to the Drive. Those PDOs are the position command (sent to the Drive by the Master) and the actual motor position (sent to the Master by the Drive). As expected, these two signals coincide, after the small offset introduced by the transmission delay.

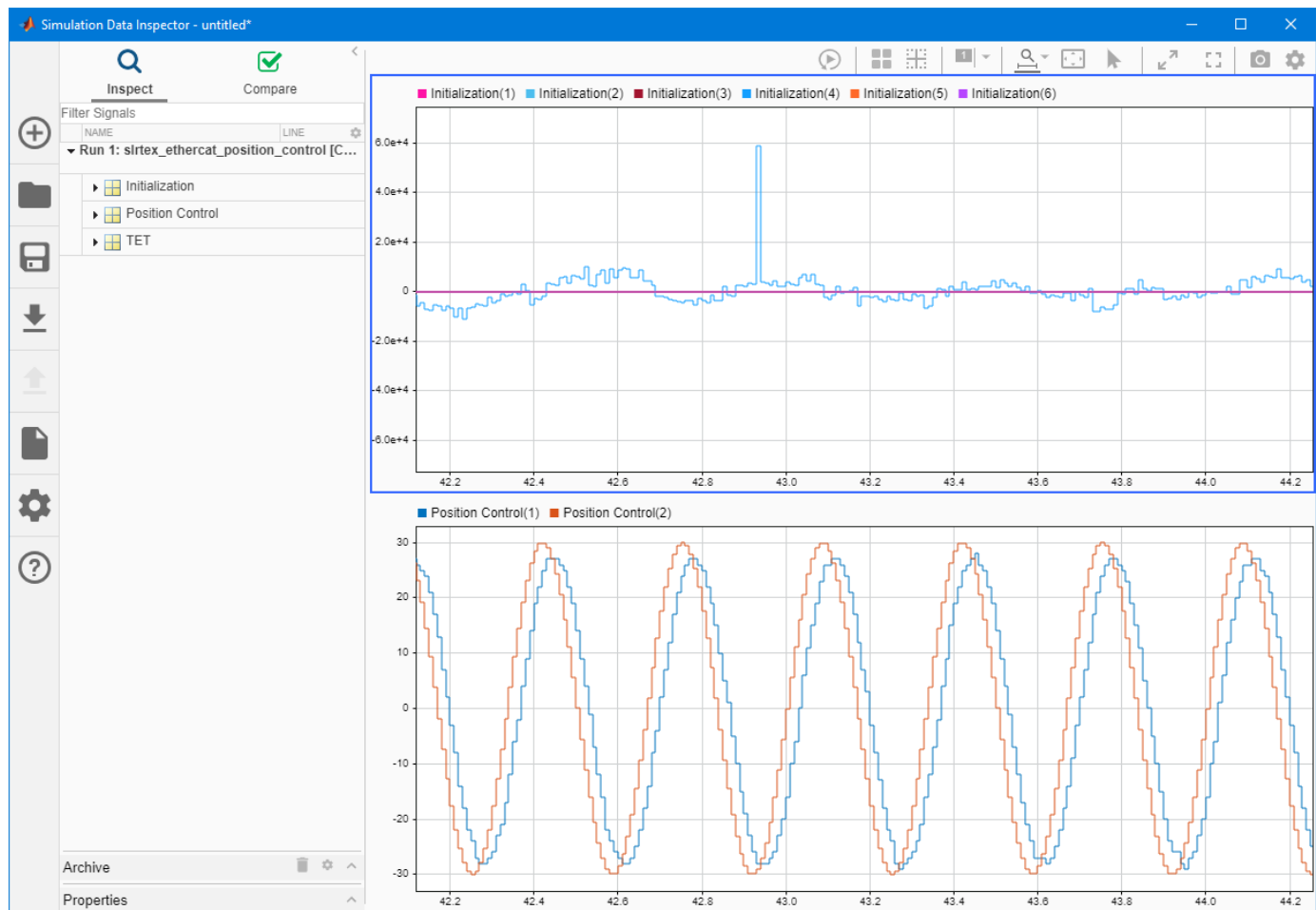
To view the plotted signal data, open the Simulation Data Inspector.

```
Simulink.sdi.view
```

This image shows an example view on the Simulink Scope.



This image shows an example view on the Simulation Data Inspector.



Stop and Close the Model

When the example completes its run, stop and close the model.

```
stop(tg);
if (mdlOpen)
    close_system(mdl);
end
```

Generate ENI Files for EtherCAT Devices

This example shows how to generate EtherCAT network information (ENI) files to use in Simulink Real-Time with EtherCAT devices.

The example shows the generation process steps in EtherCAT Configurator and the process steps in the TwinCAT XAE plugin for Microsoft Visual Studio®.

The hardware connections are:

- EK1100 -- EtherCAT coupler
- EL3062 -- EtherCAT terminal
- EL4002 -- EtherCAT terminal
- EL9011 -- Bus End terminal

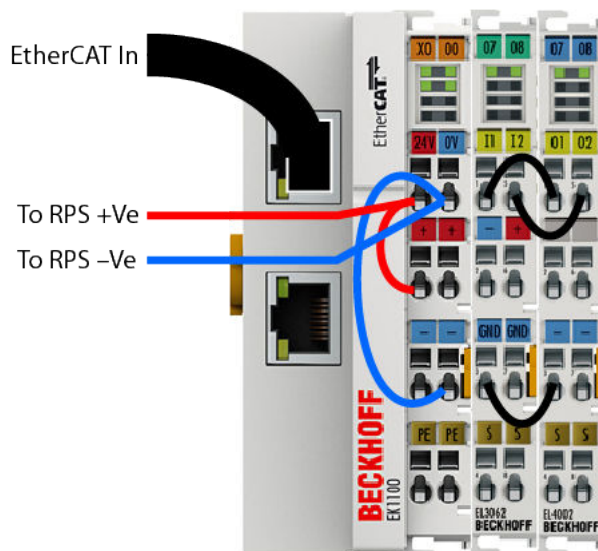
The EK1100 coupler connects EtherCAT with the EtherCAT terminals (ELxxxx). One station consists of an EK1100 coupler, any number of EtherCAT terminals, and a bus end terminal.

To provide power connections, connect the 24 V and 0 V terminals of the EK1100 to a 24 V regulated power supply (RPS) +Ve and -Ve terminals.

The EL3062 analog input terminal processes signals in the range of 0-10 V. The voltage is digitized to a resolution of 12 bits and is transmitted.

The EL4002 analog output terminal generates signals in the range of 0 and 10 V.

To configure the EtherCAT network, connect the EtherCAT devices to the development computer on which the EtherCAT configurator is running. This connection permits scanning and discovery of the EtherCAT devices. After the configurator generates the XML file, you can reconnect the EtherCAT devices to the target computer. This diagram shows the suggested connections.



Install and Run EtherCAT Configurator ET9000

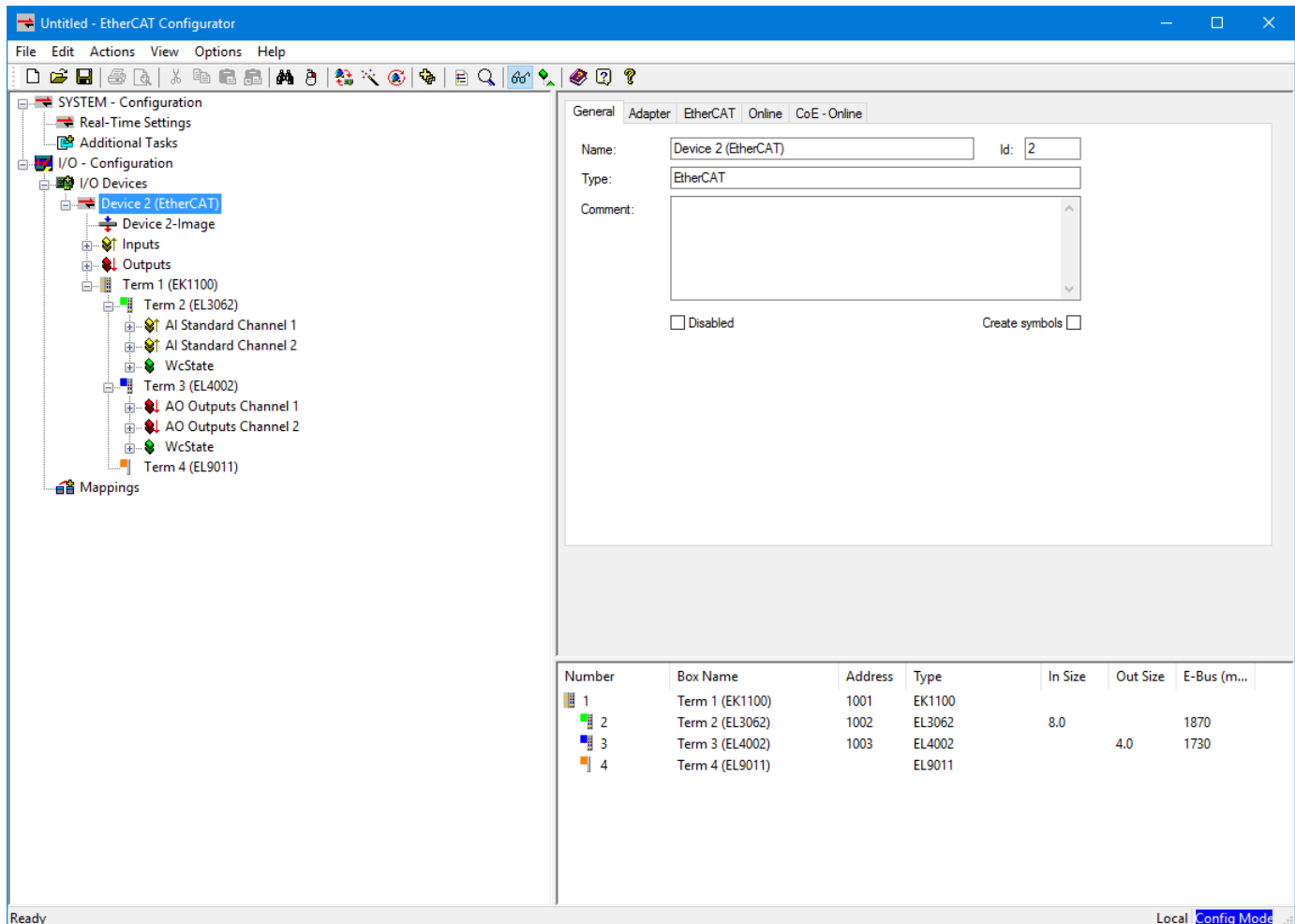
To install the EtherCAT configurator ET9000:

- 1 Go to <https://www.beckhoff.com/> and select **Download**.
- 2 Select ET9000 and download the setup.
- 3 Install the ET9000 configurator and start the software.
- 4 Run the configurator and select the correct license keys or select the evaluation option.

The EtherCAT configurator creates an EtherCAT network information (ENI) file from the standardized slave description files (ESI - EtherCAT slave information). To generate the ESI files for the slaves:

- 1 Start the ET9000 software.
- 2 Right-click **I/O Devices** and select **Scan Devices**. Click **OK**.
- 3 Select the correct network interface card (NIC) in your system and click **OK**.
- 4 When the dialog box asks whether to scan for boxes, select **Yes**. As the EtherCAT devices in your network are scanned, they appear in the **System Pane**.
- 5 When the dialog box asks whether to activate free run mode, select **No**.

When the scan is complete, expand the **Device Hierarchy** under **I/O Devices** in the **System Pane**. The EL3062 and the EL4002 devices appear under the EK1100 device.



| Number | Box Name | Address | Type | In Size | Out Size | E-Bus (m... |
|--------|-----------------|---------|--------|---------|----------|-------------|
| 1 | Term 1 (EK1100) | 1001 | EK1100 | | | |
| 2 | Term 2 (EL3062) | 1002 | EL3062 | 8.0 | | 1870 |
| 3 | Term 3 (EL4002) | 1003 | EL4002 | | 4.0 | 1730 |
| 4 | Term 4 (EL9011) | | EL9011 | | | |

Configure EtherCAT Master Node Data with Configurator

The configurator uses the ESI to configure the EtherCAT master node. This operation includes creating a task, configuring the task, and adding the I/O to the task.

To create an EtherCAT task:

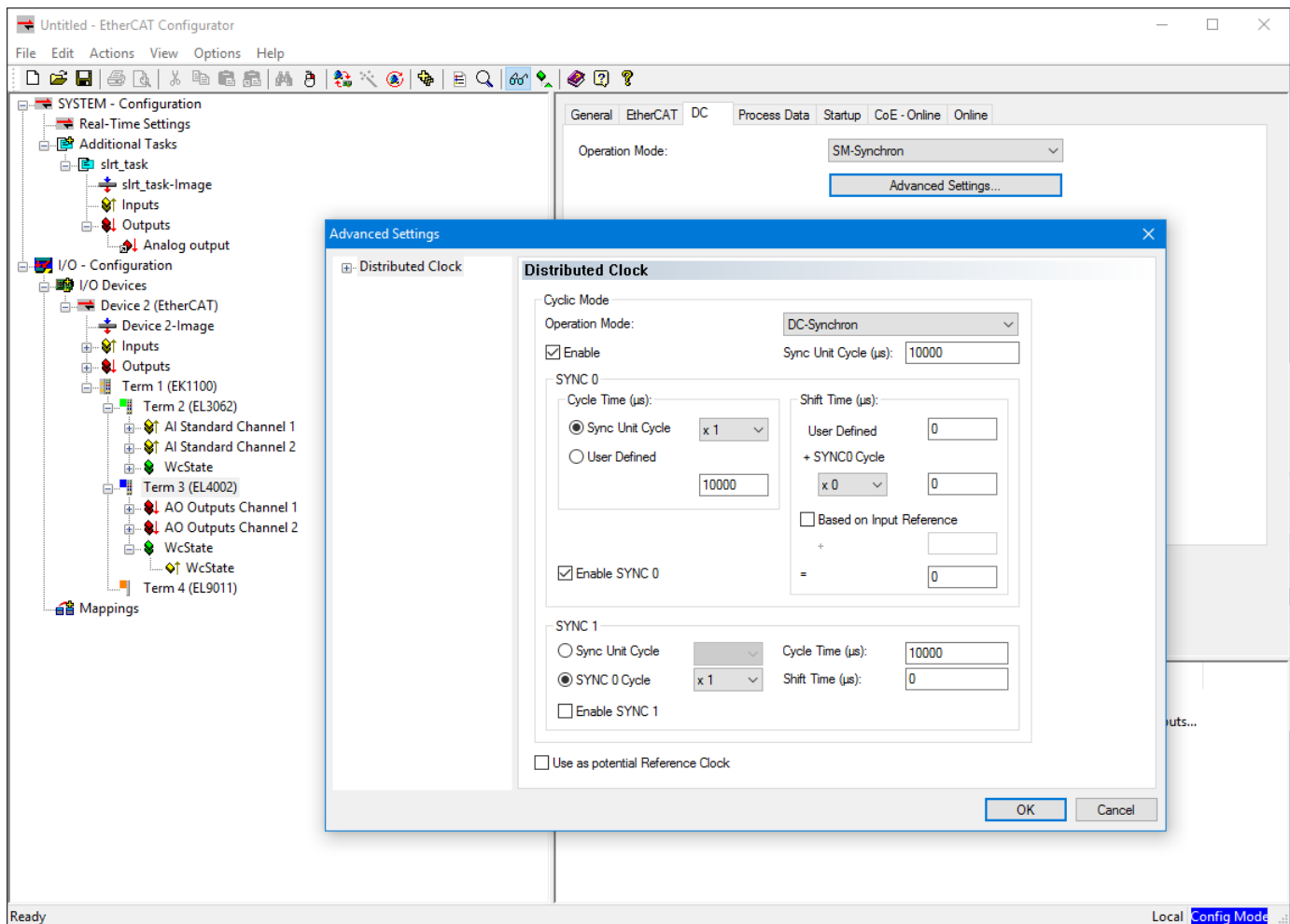
- 1 Under **SYSTEM - Configuration**, right-click **Additional Tasks > Append Task**.
- 2 Provide a name for the task and click **OK**. In this example, the name of the task is `slrt_task`.
- 3 Click the task. The value **Cycle Ticks** determines the cycle time. In the settings, it is set to 10ms.

To configure an EtherCAT task outputs:

- 1 Click and drag the node **Analog output** (under **AO Outputs Channel 1**) to **Outputs** (under `slrt_task`).
- 2 Select **Analog output** (under **Outputs**) and select the **Variable** tab.
- 3 Click the **Linked to** button and select the corresponding entry (**Analog Output** under **AO Outputs Channel 1** under **Term 3**).

By using distributed clocks (DC), the EtherCAT real-time Ethernet protocol can synchronize the time in all local bus devices within a narrow tolerance range. Only some EtherCAT devices support DC. When the device supports DC, it is important to configure a device for DC. For example, in the example configuration, the EL4002 supports DC. To configure the EL4002 for DC:

- 1 Click **Term 3 (EL4002)** in the **System Pane** and select the **DC** tab. By default, the **Operation Mode** is set to **SM-Synchron**. Change Operation Mode to **DC-Synchron**.
- 2 Click the **Advanced Settings** button and set the **Distributed Clock** options as shown.



Import a Device with the Configurator

Device import is often part of the workflow for third-party (different manufacturer) devices. Use this process to configure a device that is not present in the Beckhoff system. Numerous motors and their drives fall under this category. Sometimes, you must configure a device that is not present in the Beckhoff system. The TwinCAT EtherCAT master or System Manager uses the device description files for the devices to generate the configuration in online or offline mode.

The device descriptions are contained in ESI files (EtherCAT Slave Information) in XML format. These files can be requested from the respective manufacturer and are made available for download. An XML file can contain several device descriptions.

The ESI files for Beckhoff EtherCAT devices are available on the Beckhoff website and are stored in the TwinCAT installation folder. The default for TwinCAT2 is C:\TwinCAT\IO\EtherCAT. The files are read (once) when you open a new System Manager window and if they have changed since the last time that you opened the System Manager window.

If using a TwinCAT configurator, the TwinCAT installation includes the set of Beckhoff ESI files which were current at the time when the TwinCAT build was created. For TwinCAT 2.11, TwinCAT 3, and later, you can update the ESI folder from the System Manager if the programming PC is connected to the Internet (**Option > Update EtherCAT Device Descriptions**).

To import a device from an ESI file:

- 1 For the ET9000 Configurator, the ESI folder is C:\Program Files (x86)\EtherCAT Configurator\EtherCAT. Paste the file from the manufacturer into this location.
- 2 After adding the XML file, restart your configurator and select **Actions > Reload Devices**.
- 3 If the device is connected, you can scan again to add the devices. (See **Install and Apply Beckhoff EtherCAT Configurator ET9000**.)
- 4 If the device is not connected, you can also add the device in Offline mode. If you want to add the device to the same term, right-click your device in the hierarchy and select **Append Box**. A dialog box appears asking which device to add.
- 5 Click the square icon next to Beckhoff Automation GmbH to collapse the hierarchy. You now see the manufacturer whose devices you added.
- 6 Select the device that you want to add and click **OK**. Your device should now appear in the **System Pane** on the left.
- 7 Repeat the steps under **Configure EtherCAT Master Node Data** to add the Outputs to your task. In this example, drag the available outputs under your drive to **Outputs** under `slrt_task`. Remember to make the appropriate DC Configurations for your device.
- 8 DC configuration information is available from the manufacturer. In this case, enable DC.
- 9 Continue with configuration of the terminals.

Export and Save the EtherCAT Configuration with the Configurator

To generate the ENI file and save the configuration:

- 1 Click the node for your EtherCAT device, then click the **EtherCAT** tab.
- 2 Click **Export Configuration File**.
- 3 In the file save dialog box, enter an XML file name, such as `BeckhoffAI0config.xml` for this example, and then click **Save**. This XML file is the ENI file. The ENI file and the Simulink® Real-Time™ model that uses the ENI file cannot have the same name. They must have different names.
- 4 Save the configuration as an ESM file. Click **File > Save**. If the ESM file corresponding to the ENI file is not present, the Beckhoff ET9000 program cannot open the ENI file.
- 5 In the **File Save** dialog box, enter an ESM file name, such as `et9000config.esm`, and then click **Save**.

Install TwinCAT 3.1 XAE and Run Microsoft Visual Studio® with TwinCAT

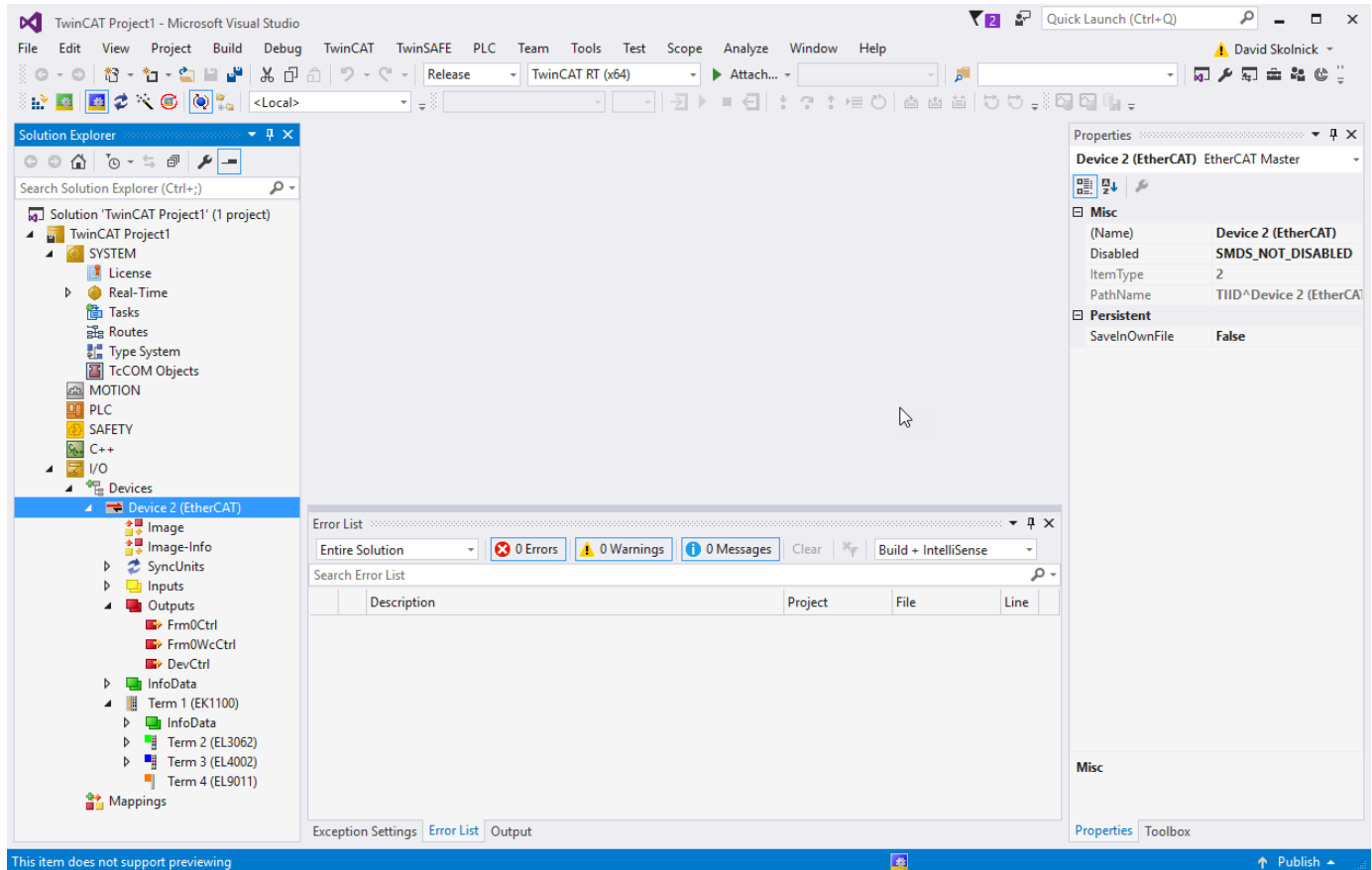
When you install TwinCAT 3.1 eXtended Automation Engineering (XAE), you can use the integration of this software with Microsoft Visual Studio to program automation objects with IEC 61131-3 and C/C++ languages.

To install the TwinCAT 3.1 XAE:

- 1 Go to <https://www.beckhoff.com/> and select **Download**.
- 2 Select TwinCAT 3 and download the setup.
- 3 Install TwinCAT 3 and start Microsoft Visual Studio.
- 4 From the **TwinCAT** menu, select **Show Realtime Ethernet Compatible Devices**.
- 5 Select the Ethernet adapter for your EtherCAT device, then select **Install**.

To open a new TwinCAT project in Visual Studio:

- 1 Start Visual Studio. Go to **File > New > Project**.
- 2 Under **Templates**, select **TwinCAT Project**.
- 3 Verify whether the project has been created successfully in the status bar of Microsoft Visual Studio.
- 4 Enter your license if this instance is the first time that you are using TwinCAT. If you are using TwinCAT in evaluation mode, fill in the Captcha.
- 5 Observe the **Solution Explorer** pane the right side of Visual Studio.
- 6 Go to **TWINCAT** in the menu and select **Scan**. You can also right click **Solution Explorer > your TwinCAT project > I/O > Devices > Scan**.
- 7 A dialog box opens with the message **All devices may not automatically be found**. Click **OK** and wait for the scan to complete. You now see a dialog box saying **New I/O devices have been found**.
- 8 Ensure that the check box is selected, then click **OK**. A dialog box appears with a **Scan for boxes?** message. Click **Yes**. The EtherCAT devices in your network are scanned, and the devices appear.
- 9 You see a dialog box that asks whether to activate free run mode. Select **No**.
- 10 Observe the Solution Explorer and verify that the devices were scanned correctly.



Configure EtherCAT Master Node Data with TwinCAT

To configure the EtherCAT master node, create and configure a task, then add the inputs and outputs to the task.

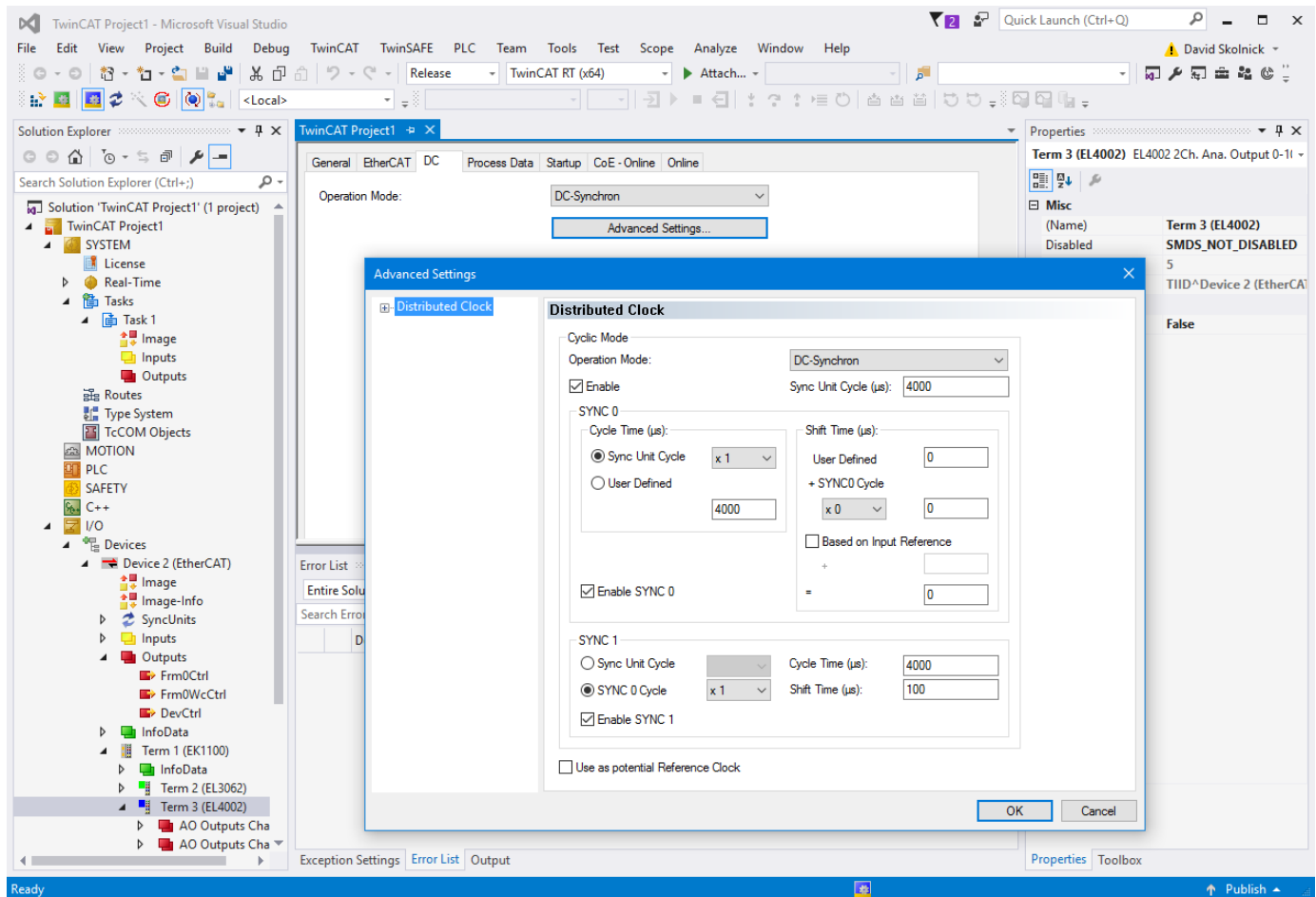
To create an EtherCAT Task:

- 1 In the **Solution Explorer**, right-click the **Task** node.
- 2 In the **Insert Task** dialog box, select **TwinCAT Task With Image**, provide a name for the task, and click **OK**.
- 3 Double-click the task that you created. The value **Cycle Ticks** determines the cycle time. In this example, it is set to 10 ms.
- 4 Create at least one cyclic input/output task. Link this task to at least one input channel and one output channel on each slave device.

By using distributed clocks (DC), the EtherCAT real-time Ethernet protocol can synchronize the time in all local bus devices within a narrow tolerance range. Only some EtherCAT devices support DC. It is important that if a device supports DC, you configured it accordingly. For example, in the example configuration, the EL4002 supports DC.

To configure EtherCAT DC:

- 1 Double-click the node **Term 3 (EL4002)** and select the **DC** tab.
- 2 By default, the **Operation Mode** is set to **SM-Synchron**. Change the **Operation Mode** to **DC-Synchron**.
- 3 Click **Advanced Settings** and set the **Distributed Clock** options as shown.



To export and save the EtherCAT configuration, generate the ENI file:

- 1 Double-click the node for your EtherCAT device and click the **EtherCAT** tab.
- 2 Click **Export Configuration File**.
- 3 In the **Save As** dialog box, enter an XML file name, such as `twincatconfig.xml`, then click **Save**. This XML file is the ENI file. The ENI file and the Simulink® Real-Time™ model that uses the ENI file cannot have the same name. They must have different names.
- 4 If the Solution file corresponding to the ENI file is not saved, the TwinCat XAE program cannot open the ENI XML file. Save the Solution file as an archive (zip file). Select **File > Save Project As Archive**.
- 5 In the **Save As** dialog box, enter an ESM file name, such as `twincatproject`, and click **Save**. The project is saved as a `.tnzip` archive.

Related Information

- “EtherCAT® Communication with Beckhoff® Analog IO Slave Devices EL3062 and EL4002” on page 15-81
- “EtherCAT® Communication with Beckhoff® Digital IO Slave Devices EL1004 and EL2004” on page 15-86
- “EtherCAT® Communication - Motor Velocity Control with Accelnet™ Drive” on page 15-91

- “EtherCAT® Communication - Motor Position Control with an Accelnet™ Drive and Beckhoff® Analog IO Devices” on page 15-96

Digital I/O with Speedgoat FPGA Board

This example shows a workflow that uses HDL Coder™ to deploy a Simulink® subsystem to a Speedgoat FPGA I/O board that resides in the target computer. A Simulink® Real-Time™ application runs on the target computer and communicates with the FPGA over the PCI bus.

In this case, the FPGA algorithm maps Simulink Real-Time generated pulse trains to I/O channels on the FPGA. Over the PCI bus, the FPGA receives pulses generated on the target computer. It then writes the signals to eight output channels. The FPGA reads eight input channels and sends them over the PCI bus to the target computer for graphical display. You accomplish loopback by wiring inputs to outputs (output channel 0 to input channel 0, output channel 1 to input channel 1, and so on).

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.

The default FPGA clock rate is 75 MHz. The Simulink Real-Time simulation is set to 1 kHz.

Requirements and Preconditions

HDL Coder™

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).

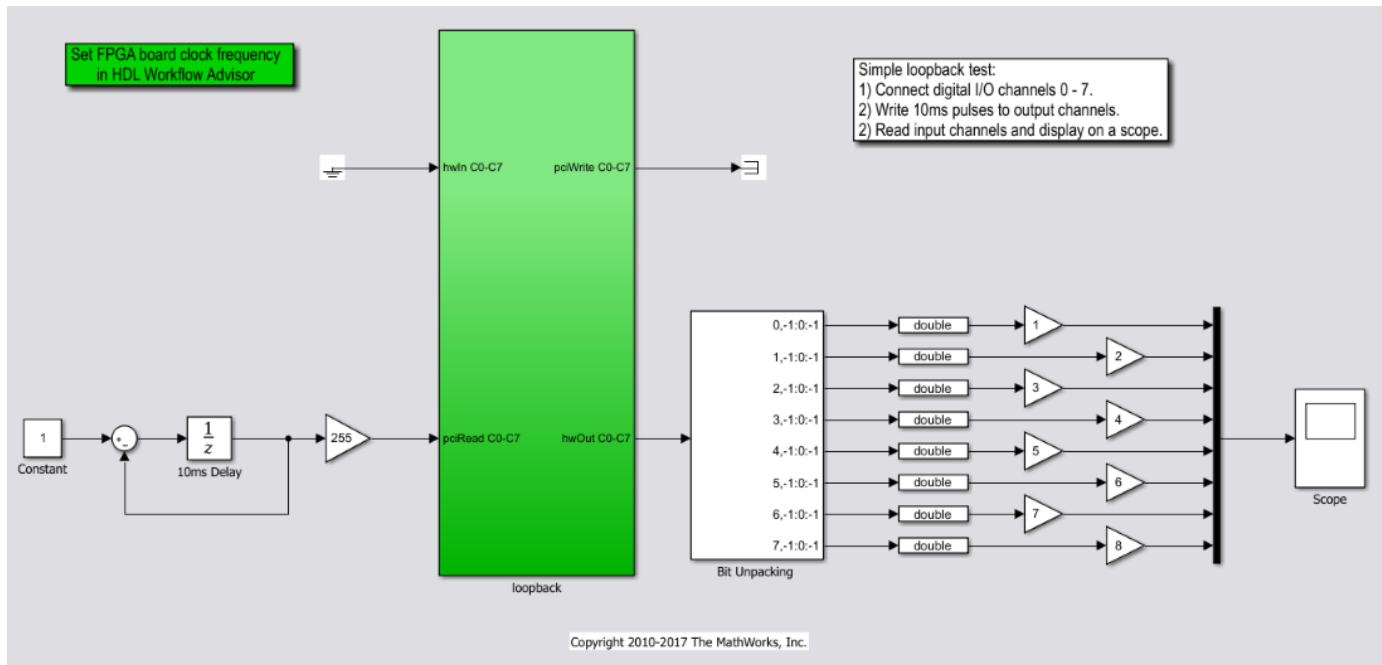
Open the FPGA Domain Model

Click here to open the FPGA model: [dslrtSGFPGAloopback_fpga](#).

This model contains the algorithm (green subsystem) that will eventually run on the FPGA. It also contains some test blocks to verify, in simulation, the algorithm is working as expected before synthesizing the FPGA bitstream. Note that this model is an "FPGA domain" model, meaning the simulation sample rate is representative of the clock rate of the FPGA (75 MHz). Hence, 1 second of simulation requires 75e6 iterations of the model.

Once the algorithm is complete, use the **HDL Workflow Advisor** to:

- Select the FPGA I/O board
- Map the subsystem inports and outports
- Synthesize the FPGA bitstream
- Generate the Simulink Real-Time interface subsystem



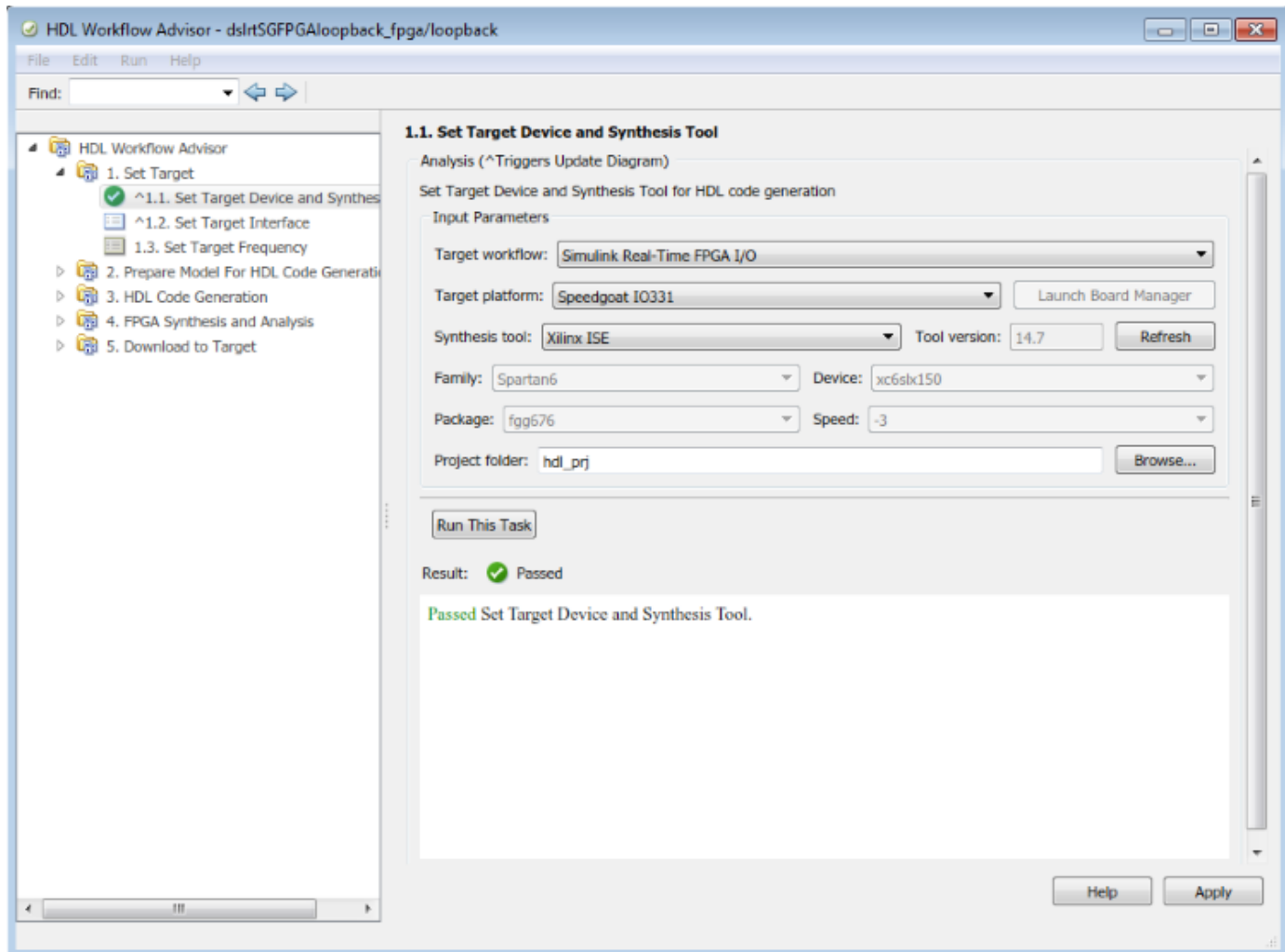
Using the HDL Workflow Advisor

To invoke the HDL Workflow Advisor, right-click on the "loopback" subsystem and select **HDL Code > HDL Workflow Advisor**.

Task 1.1. Set Target Device and Synthesis Tool

Select Simulink Real-Time by choosing **Simulink Real-Time FPGA I/O** for Target workflow. Set the target platform to the Speedgoat IO331 and check that HDL Workflow advisor sets the synthesis tool to the Xilinx® ISE Design Suite. This setting configures the board characteristics and synthesis tool used in subsequent tasks.

When done, click **Run This Task** and continue with Task 1.2.

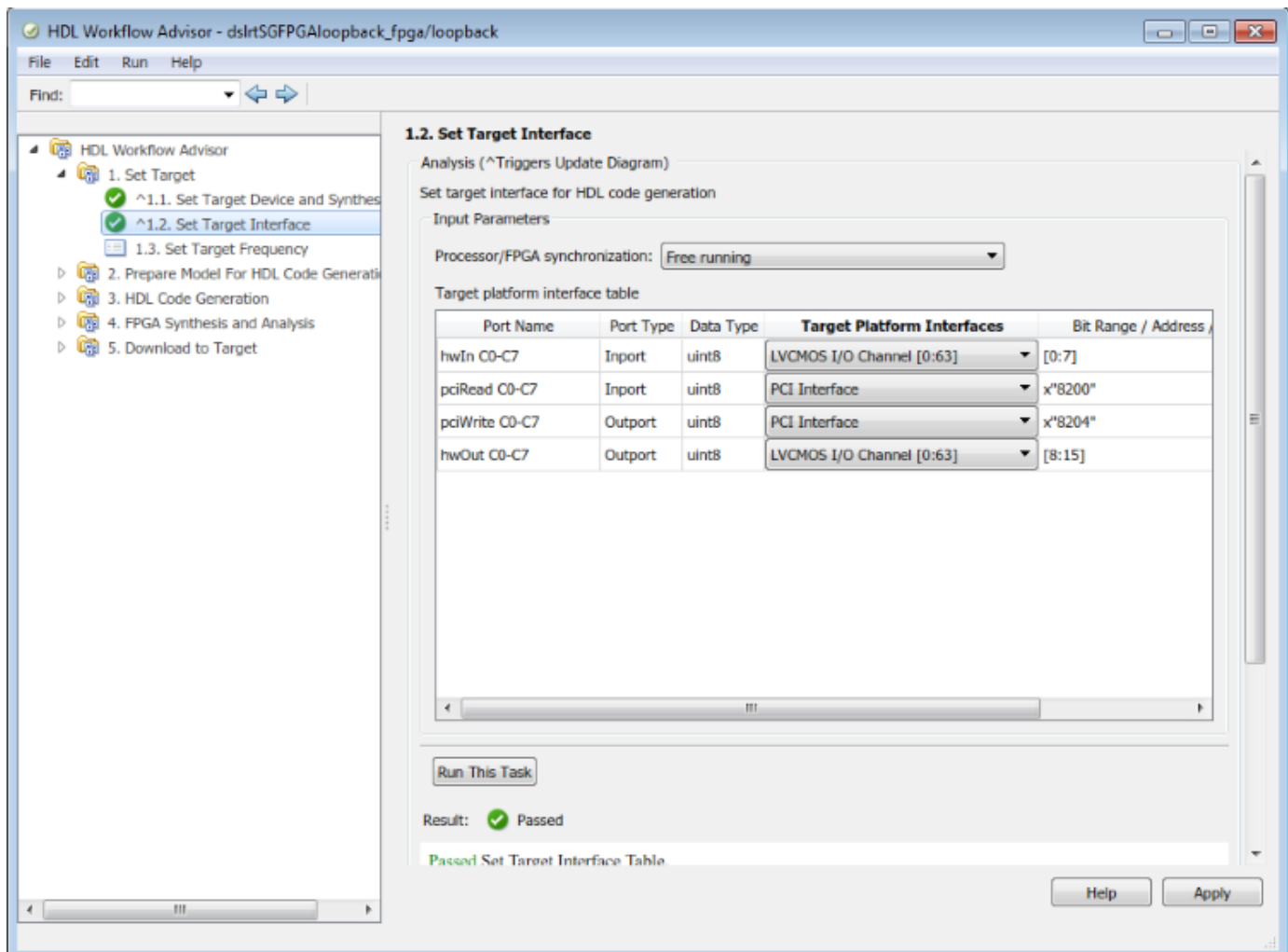


Task 1.2. Set Target Interface

Use the **Target Platform Interface Table** to specify and map the inports and outports. For this example,

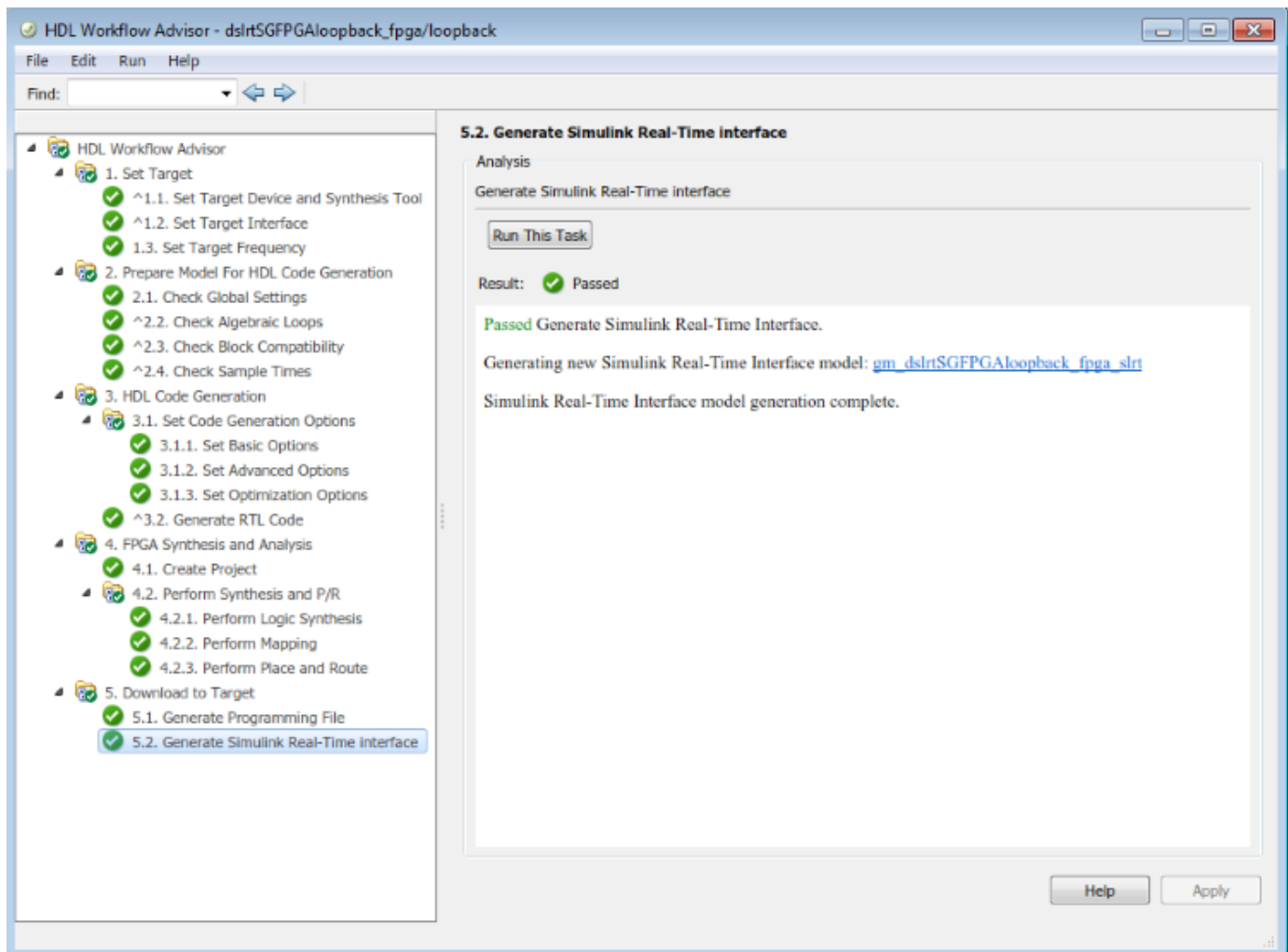
- hwIn C0-7 are signals read from the LVCMOS I/O channels 0-7 (input channels)
- hwOut C0-7 are signals written to the LVCMOS I/O channels 8-15 (output channels)
- pciRead C0-7 are signals read from the target computer over the PCI bus (input channels)
- pciWrite C0-7 are signals written to the target computer over the PCI bus (output channels)

When done, click **Run This Task** and continue with Task 5.2.

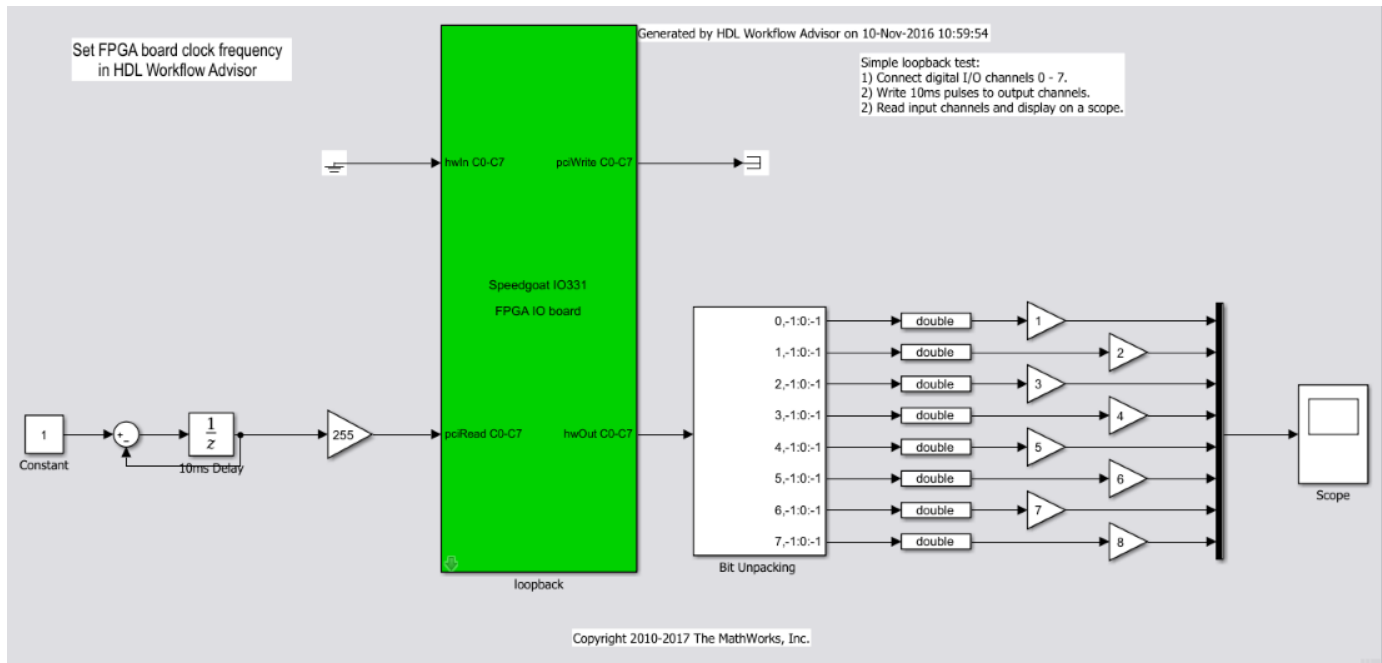


Task 5.2. Generate Simulink Real-Time Interface

Right-click Task 5.2 and click **Run to Selected Task**. The HDL Workflow Advisor will run the tasks, synthesize the FPGA bitstream, and generate a new model which contains a Simulink Real-Time interface subsystem.



When the tasks complete, a newly generated model containing the Simulink Real-Time interface subsystem appears. On the surface, this subsystem looks like the FPGA subsystem. However, inside, the Simulink algorithm has been removed and replaced with blocks that the real-time application will use to communicate with the FPGA during simulation execution.



Open the Simulink Real-Time Model

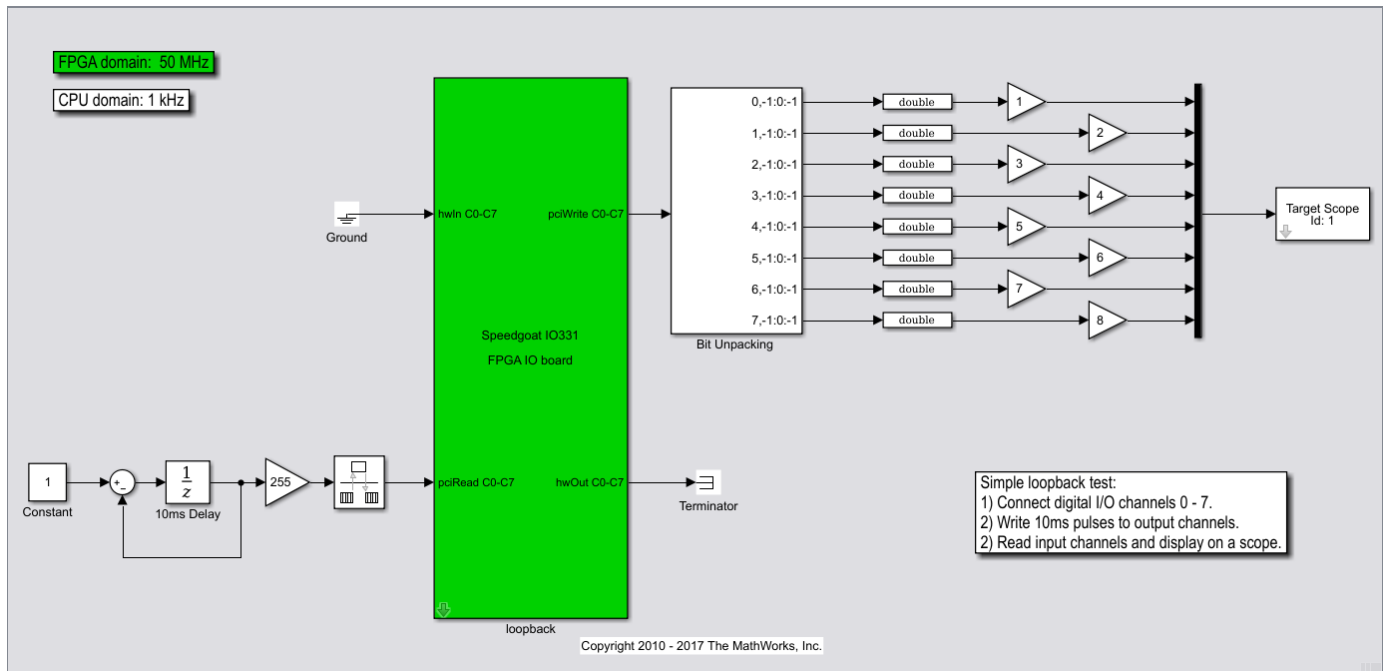
Create or open a Simulink Real-Time model that will run on the target computer while the bitstream algorithm runs on the FPGA.

Click here to open the Simulink Real-Time model: [dslrtSGFPGAloopback_slrt](#).

Add the Simulink Real-Time Interface Subsystem

From the generated model, copy the Simulink Real-Time interface subsystem and paste it in the Simulink Real-Time model. Connect inports and outports. Double-click the interface subsystem and set mask parameters as required.

Click here to open the Simulink Real-Time model with the interface subsystem included: [dslrtSGFPGAloopback_slrt_wiss](#).



Test the Model

Build the model. When the build is complete, the Simulink Real-Time application is downloaded to the target computer and the bitstream is downloaded to the FPGA.

Run the application. The eight pulse train signals sent and received through the FPGA and the LVCMOS I/O channels are displayed on the target scope.

PLL-Based Interrupt Generation from FPGA Input

This example shows how Simulink® Real-Time™ can drive a target application not only with interrupts based on its internal timer, but also with interrupts based on an external signal.

In the simplest use case, we would like to run the real-time application synchronized 1-1 to the external signal; for instance, every time we receive a rising edge from an external signal, an interrupt is generated which then runs one step of the application. Using a phase-locked loop (PLL), we can drive the real-time application at a frequency that is different from the input frequency, but at the same time synchronized to the external signal. For instance, we may have an external signal at 600 Hz, but we would like to run the application at 2 kHz.

This example uses a Speedgoat FPGA based I/O module (the Speedgoat IO331) and a digital PLL modeled in Simulink®. From the model, HDL Coder™ generates and synthesizes HDL code. 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™
- DSP System Toolbox™

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.

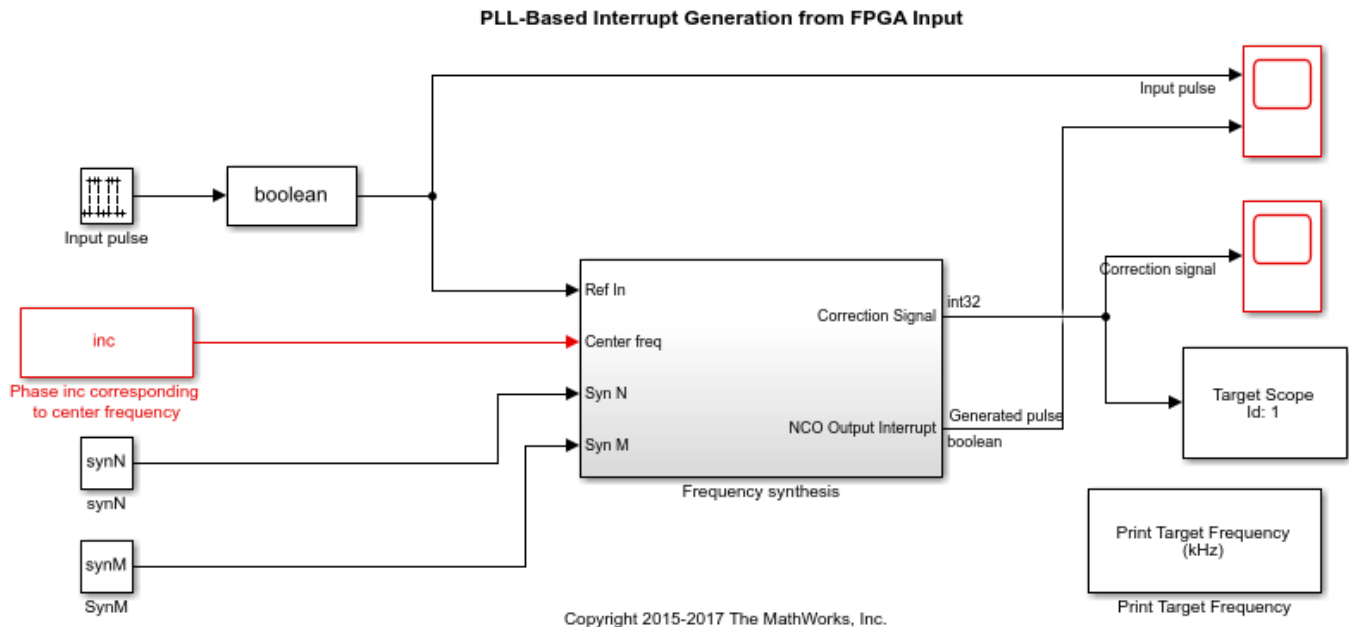
For the toolset requirements of other boards, see Supported Third-Party Tools and Hardware (HDL Coder).

PLL and frequency synthesis

A PLL is required to lock the generated frequency since the desired frequency is any arbitrary ratio (**synN/synM**, within some range) of the input frequency. We will first create a model with the PLL in the 'Frequency synthesis' subsystem. The model contains top level blocks that aid the simulation of the model on the desktop as well as provide inputs to the FPGA in order to tune parameters while running in real-time on the target.

An external signal of frequency **Fr** is input into one of the TTL I/O lines configured as input. We wish to drive the model at a frequency $Fd = Fr * (synN/synM)$. **synN** and **synM** are integers that may be varied to obtain frequencies different from **Fr**. The model `ds1rtExtDIOInt` uses a phase-locked loop (PLL) to lock the frequency generated by the **NCO HDL Optimized** block to the desired frequency **Fd**.

```
open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'ds1rtExtDIOInt'))
```



Frequency synthesis

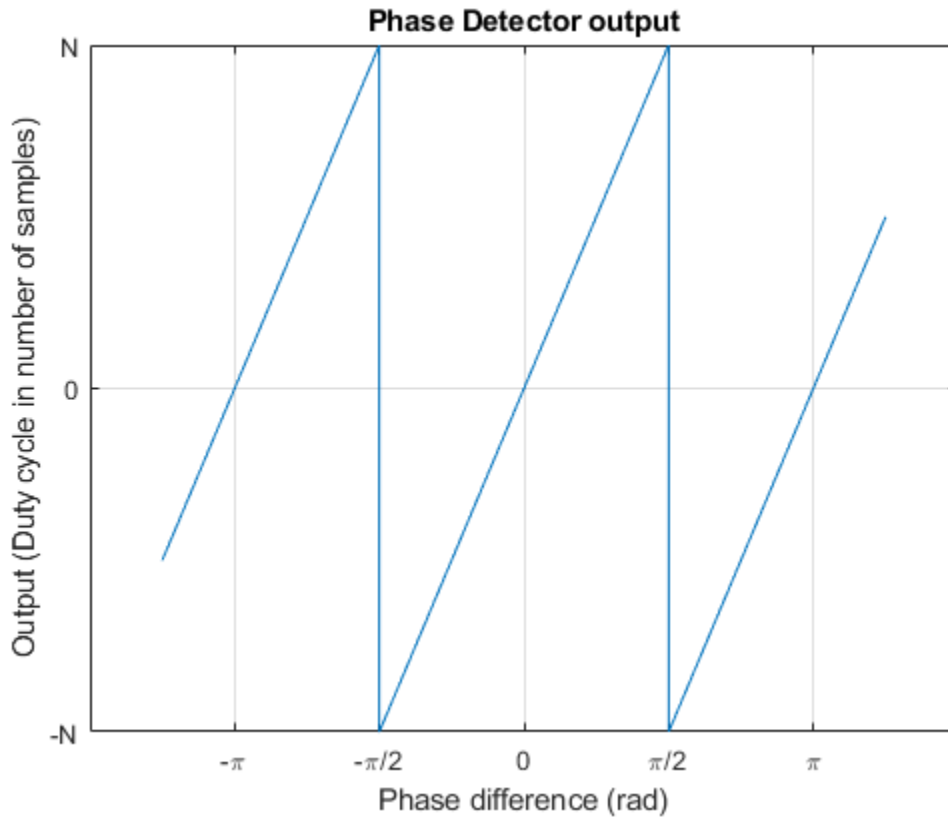
The Frequency synthesis subsystem consists of a Phase detector subsystem, a numerically controlled oscillator (NCO) and frequency dividers. HDL code will be generated for this subsystem.

Phase Detector

The phase detector subsystem compares the phase of two signals by using an XOR block. The 'countDutyCycle' function block implements a loop filter in the form of counting the number of high samples between two rising edges of the XOR output. It also compares the occurrence of the rising edges of the two input signals to the subsystem in order to provide a sign to the phase difference.

Transfer function of the 'countDutyCycle' function:

```
x = [-5/4 -1 -(1/2+eps) -1/2 0 1/2 (1/2+eps) 1 5/4];
y = [ -.5 0 1 -1 0 1 -1 0 .5];
h = plot(x,y);
a=get(h, 'parent');
a.XTickLabel = { '-\pi' '-\pi/2' '0' '\pi/2' '\pi' ''};
a.YTick = [-1 0 1];
a.YTickLabel = {'-N' '0' 'N'};
a.XGrid = 'on';
a.YGrid = 'on';
xlabel('Phase difference (rad)');
ylabel('Output (Duty cycle in number of samples)');
title('Phase Detector output');
snapnow;
```



The phase detector output wraps around outside of the range $[-\pi/2, \pi/2]$ indicating the working range of the XOR detector. Due to this the input frequency should be within a certain working range of the PLL. The value **N** on the y-axis depends on the instantaneous frequency of the input signal and the center frequency **F_c**. The range is derived in the section 'PLL working frequency range'.

NCO HDL Optimized: The numerically controlled oscillator generates a quiescent frequency as determined by the input 'Center freq'. When the input signal 'Ref In' varies from the center frequency, the 'Phase Detector' output generates a signal to make up for the difference in frequency. The generated correction signal is added to the value for the center frequency to produce the appropriate frequency. This subsystem is used only for debugging and monitoring and can be safely left out of the application.

Frequency Dividers: These divide the frequency of the square wave input into 'Ref In' and the generated signal to produce the required frequency transformation.

'Print Target Frequency' subsystem

This subsystem contains blocks which determine the time interval between consecutive sample hits and prints the instantaneous model execution frequency in kilohertz. For accurate measurement the block 'CPU Clk Freq' should be set to the CPU clock frequency of the target divided by 1e3 (for conversion to kHz).

PLL working frequency range

The following is a list of variables and default values used by the model (defined in the PreLoadFcn model callback):

```

synN = 3;          % frequency division ratio
synM = 2;          % frequency division ratio
Fs = 33e6;         % Sampling Frequency
Fr = 16.5e3;       % Input frequency = 16.5 kHz
df = .5;          % Frequency resolution = .5 Hz
minSFDR = 96;     % Spurious free dynamic range >= 96 dB

Ts = 1/Fs;        % Sample period = 3.0303e-08 seconds
Fc = Fr * synN / synM; % Oscillator Center frequency = 24.75 kHz

% For simulation only:
inputPeriod = round(Fs/Fr);
% Ensure period is even for the pulse generator to produce 50% duty cycle
inputPeriod = inputPeriod + mod(inputPeriod, 2);
% 'Period' parameter of Pulse Generator block = 1334

% Calculate number of accumulator bits required for the frequency resolution.
Nacc = ceil(log2(1/(df*Ts))); % NCO Accumulator word length = 26
actdf = 1/(Ts*2^Nacc); % Actual frequency resolution achieved = 0.4917 Hz

% Calculate number of quantizer accumulator bits required from the SFDR requirement.
Nqacc = ceil((minSFDR-12)/6); % number of quantizer accumulator bits = 14

% Calculate the phase increment.
inc = round(Fc*Ts*2^Nacc); % inc = 75497

```

The working range of this PLL may be derived as follows:

The limits at $-\pi/2$ and $\pi/2$ are reached when the delay in the generated signal corresponds to half the number of samples in one period of the signals. This can occur at a frequency F_h and F_l above and below the center frequency, respectively. Consider the lower frequency F_l . To simulate this frequency using the pulse generator block the variable 'inputPeriod' needs to be increased by a value 'x'. As a result:

$$F_l = \frac{F_s}{(\text{inputPeriod} + x)}$$

The Phase Detector produces an output equal to the duty cycle within the range $[-\pi/2, \pi/2]$ or one period of the square wave being compared. The correction factor = $-(\text{inputPeriod} + x)$. This is combined with the phase increment value corresponding to the center frequency to produce the frequency equal to the input. Equating the two previous results gives:

$$\frac{F_s}{(\text{inputPeriod} + x)} \cdot \frac{\text{synN}}{\text{synM}} = (\text{inc} - (\text{inputPeriod} + x)) \cdot \frac{F_s}{2^{N_{\text{acc}}}}$$

Solving this quadratic equation and taking the smaller root gives x from which F_l is calculated:

```

x_l = min(roots([1 -(inc - (2*inputPeriod)) - (inputPeriod*(inc - inputPeriod)) + (2^Nacc*(synN/synM))
% The lower bound for the frequency range is:
Fl = Fs / (inputPeriod + x_l);

```

Similarly the upper frequency bound F_h may be calculated:

```

x_h = min(roots([1 -(inc + (2*inputPeriod)) + (inputPeriod*(inc + inputPeriod)) - (2^Nacc*(synN/synM))
% The upper bound for the frequency range is:
Fh = Fs / (inputPeriod - x_h);

```


Open the model

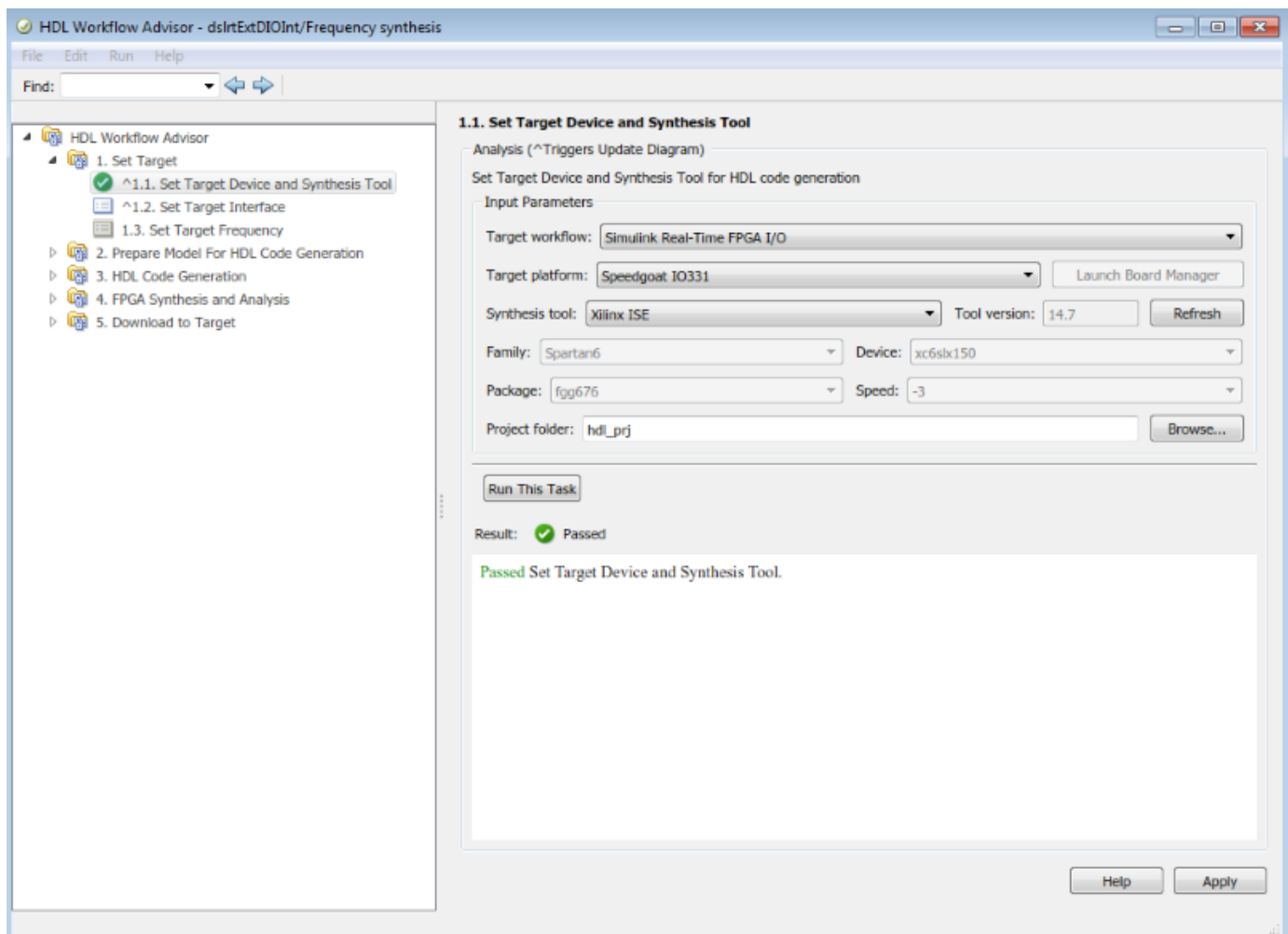
Click here to open the model: [dslrtExtDIOInt](#).

Generate HDL Code using HDL Workflow Advisor

For this example the device used is a Speedgoat IO331 FPGA board. The HDL code generation procedure is as described in the example “Digital I/O with Speedgoat FPGA Board” on page 15-110. The differences include the board used and the input and output pin mappings which are described in this section.

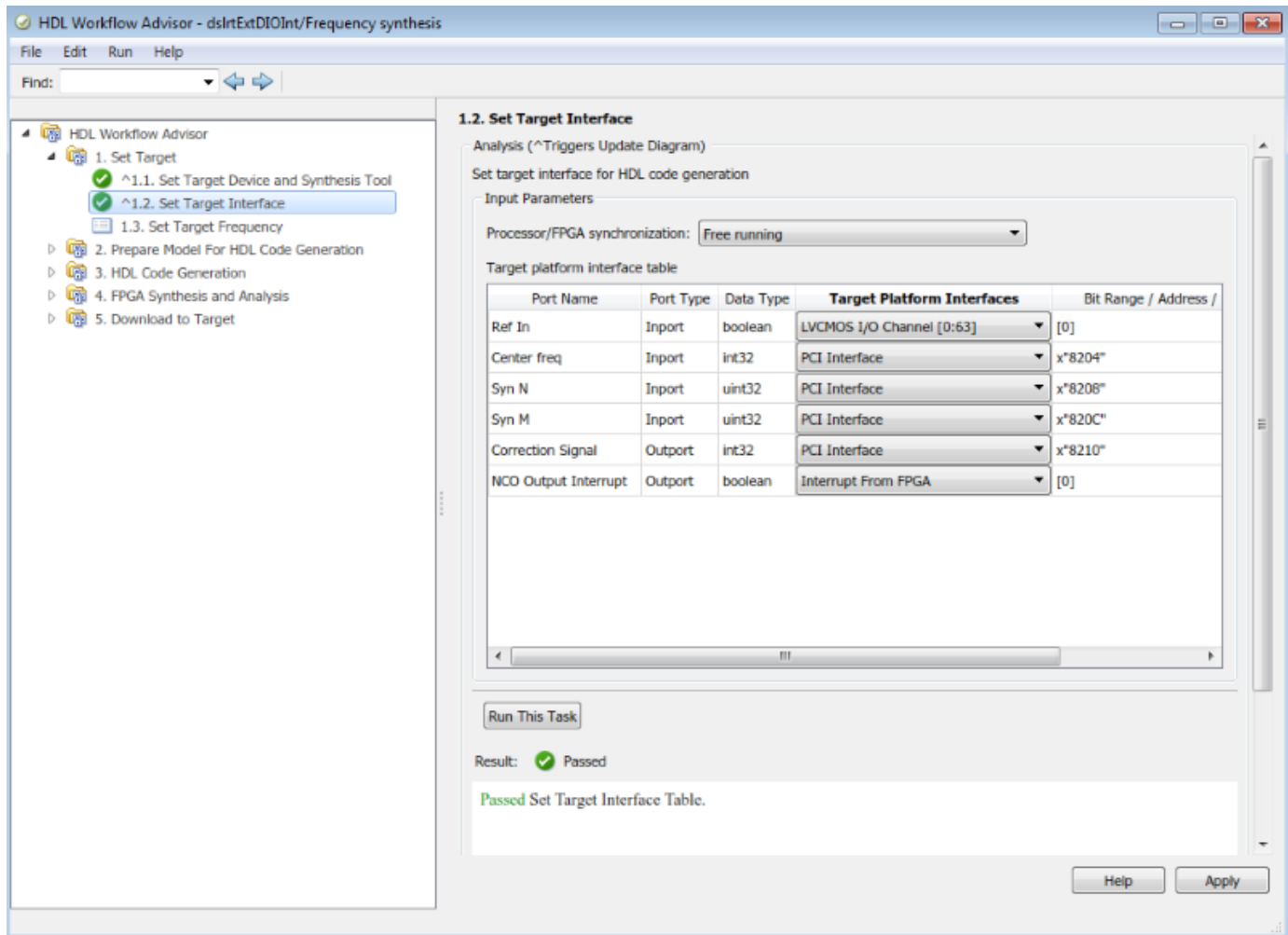
Target Device

Set the target platform to the Speedgoat IO331 FPGA and check that HDL Workflow advisor sets the synthesis tool to the Xilinx® ISE Design Suite. This setting configures the board characteristics and synthesis tool used in subsequent tasks.



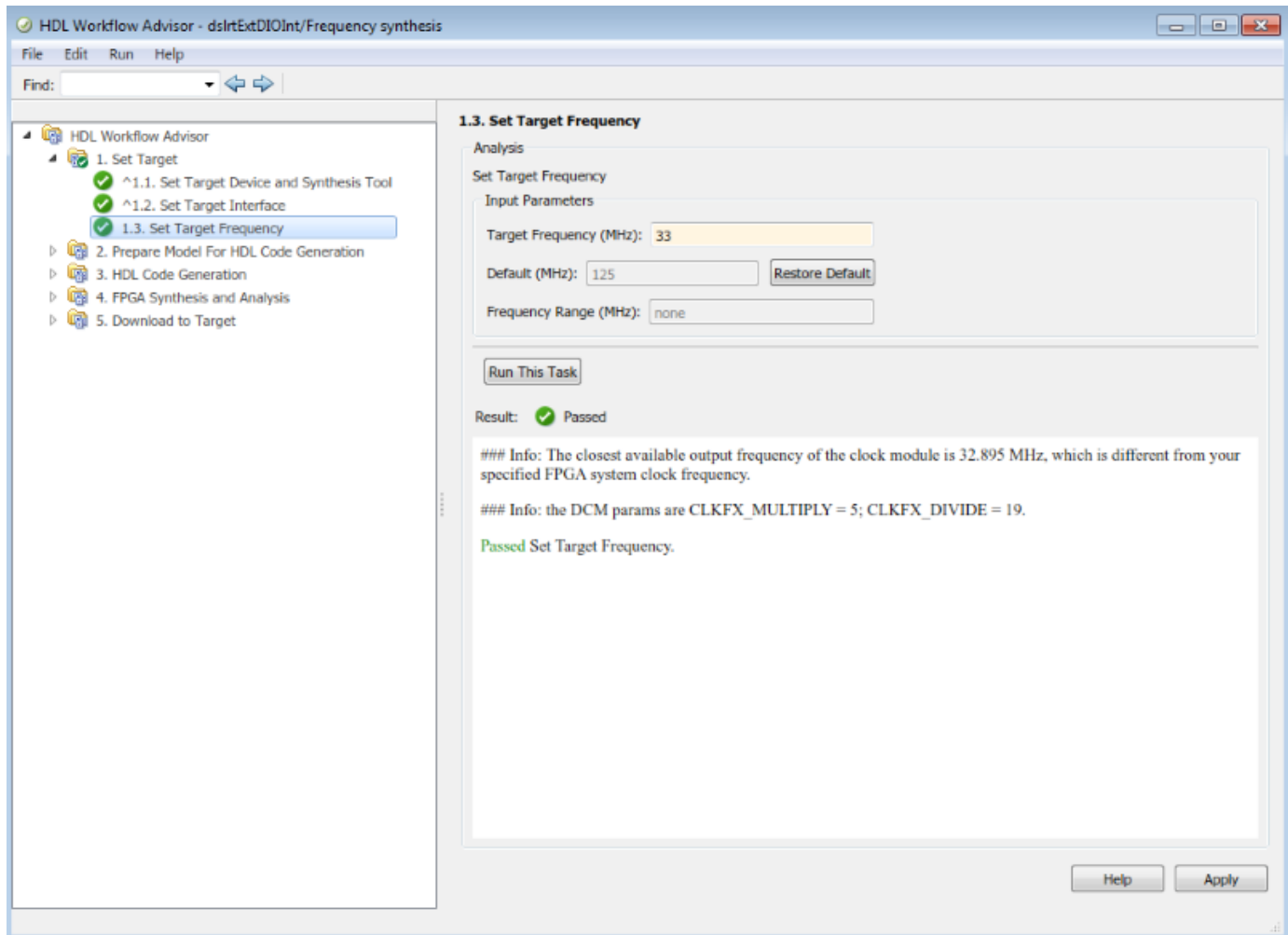
Target interface

The reference signal is input at the first pin (pin 0). The NCO output is mapped as the 'Interrupt from FPGA'. This signal becomes the interrupt for driving the model execution on the CPU.



Target Frequency

The model is designed using a fixed step size corresponding to a sampling frequency of 33 MHz. Set 'Target Frequency' to this value. This is the frequency at which the 'Frequency synthesis' subsystem will run on the FPGA.



Generate FPGA Model

The model generated by HDL Workflow Advisor is now ready to be built and downloaded to the target. However, since the intention is to run the real-time application around the center frequency F_c it is appropriate to change the sample time of the generated model to $1/F_c$ before building and downloading. This may be done by executing:

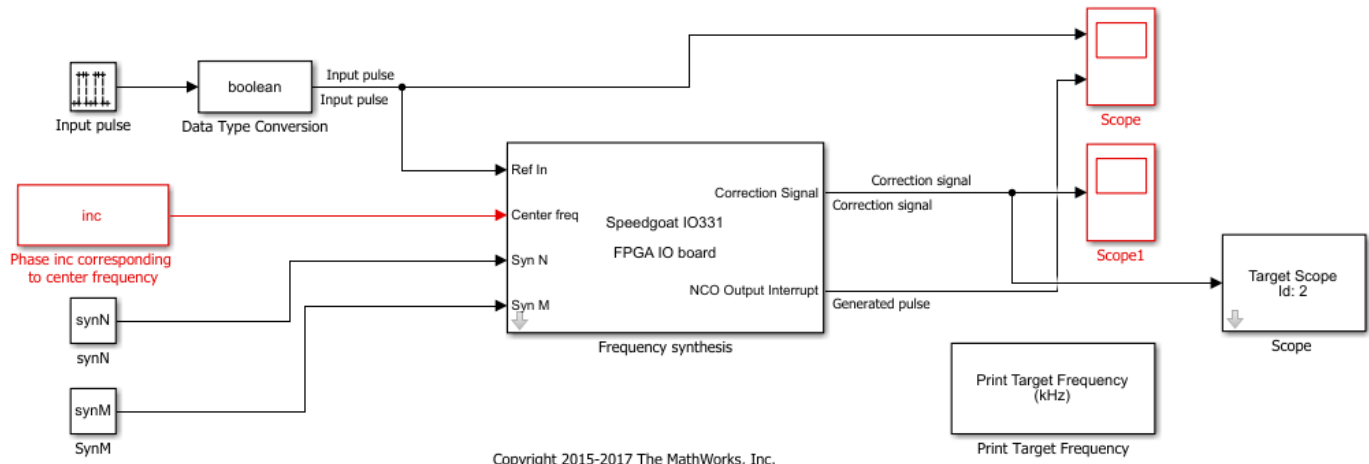
$$T_s = 1/F_c;$$

After you have set T_s :

- 1 In the Simulink Editor, open the Configuration Parameters. On the **Real-Time** tab, click **Hardware Settings**.
- 2 Select **Code Generation > System target file** to `grt.tlc`.
- 3 Right-click **Download to Target > Generate Simulink Real-Time Interface**, and then click **Run to selected task**.

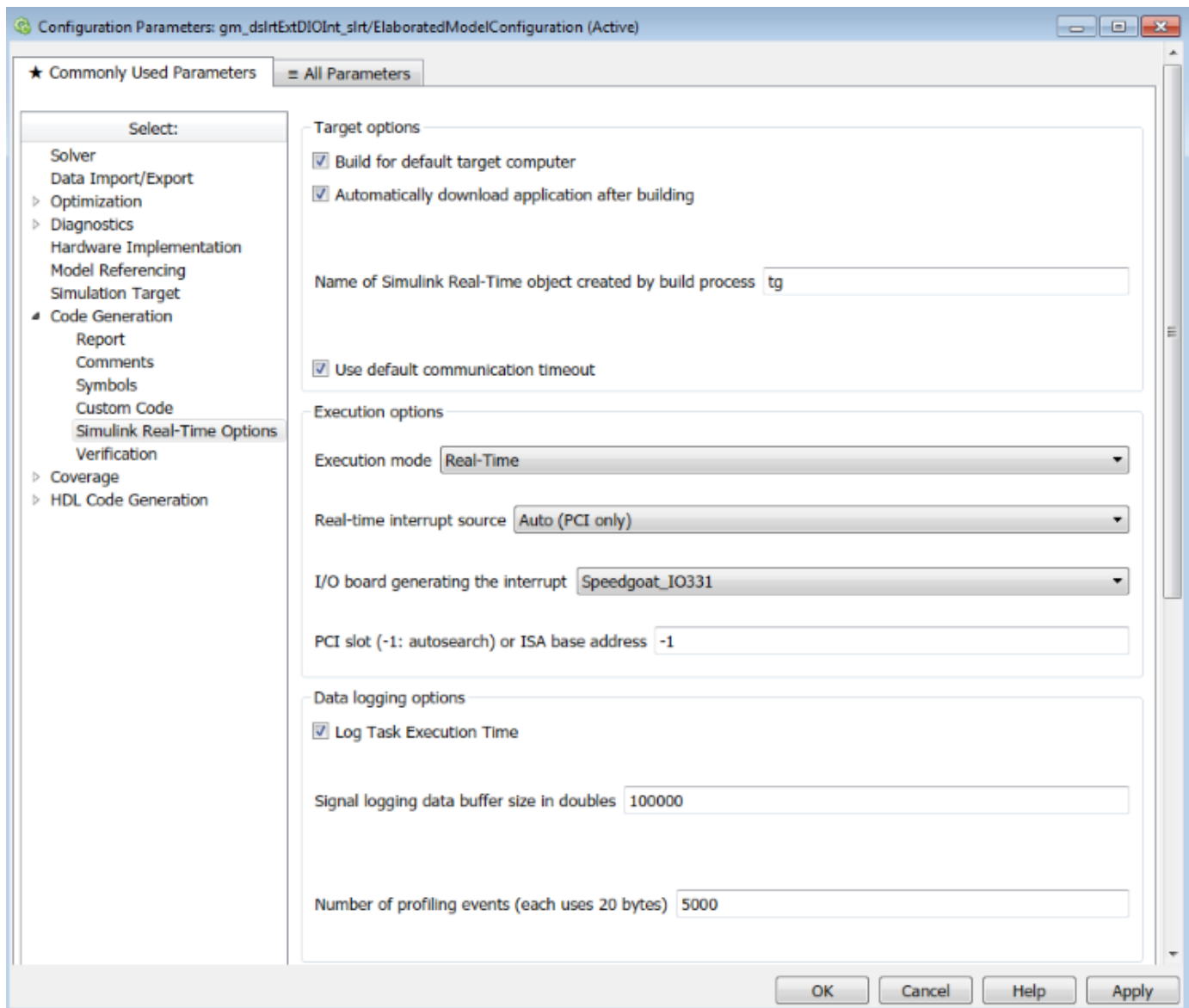
This action produces a generated model, `gm_dslrtExtDIOInt_slrt`.

Generated by HDL Workflow Advisor on 10-Nov-2016 14:17:01

PLL-Based Interrupt Generation from FPGA Input**Model Configuration Parameters**

In the Configuration Parameters dialog box of the generated model, configure the target computer to be driven by the FPGA board interrupt:

- 1 In the Simulink Editor, open the Configuration Parameters. On the **Real-Time** tab, click **Hardware Settings**.
- 2 Select **Code Generation > System target file** to `slrt.tlc`.
- 3 Set **Simulink Real-Time Options > Real-time Interrupt source** to Auto (PCI only).
- 4 Set **I/O board generating the interrupt** to Speedgoat IO331.



You can now build and download the generated model to the target computer as a real-time application.

Model Execution

The values used in this example cause the input frequency to be centered at 16.5 kHz. The target scopes show the correction signal and the instantaneous frequency when the input signal frequency is increased from 16 kHz to 17 kHz. The real-time application is being executed at a frequency that is close to $F_d = F_r * (synN / synM)$.

- This model is able to handle only square pulses that have an approximate duty cycle of 50%.
- The operation of the PLL is valid only in the frequency range as derived in the earlier sections.

```
%  
bdclose all;
```

IEEE® 1588™ Precision Time Protocol - Execution Synchronization

This example shows Execution synchronization between target computers using the Precision Time Protocol (PTP) with Raw Ethernet as transport protocol.

Requirements

To run this example, you need to:

- Acquire two Speedgoat target computers equipped with the Intel® 82574 Ethernet card from www.speedgoat.com.
- Acquire a PTP transparent clock switch, such as the EDS-405A-PTP from www.moxa.com. (In the next section, see an alternative setup in case you do not have a PTP switch.)
- Acquire a standard network switch that has at least 3 ports, such as the LINKSYS® SD2005 5-ports switch.

Connect and configure the devices in the network

In this example, the two target computers are named TargetPC1 and TargetPC2.

- 1 Connect the network port of the development computer to a port of the LINKSYS® Ethernet switch.
- 2 Connect the network port of each target computer that is dedicated to communicating with the development computer to a port of the LINKSYS® Ethernet switch.
- 3 Connect the network port of the Intel® 82574 Ethernet card in TargetPC1 and TargetPC2 to ports of the EDS-405A-PTP switch respectively. If you do not have a PTP switch, either connect directly the port of the Intel® 82574 Ethernet card of TargetPC1 to the port of the same card in TargetPC2, or connect these two ports to ports of the LINKSYS® Ethernet switch.
- 4 Configure the EDS-405A-PTP switch as an end-to-end transparent clock.
- 5 Enable PTP for the ports connected to TargetPC1 and TargetPC2.

Open and configure the models

Click on the following links to open the two models:

- Master node model: `dPTPMasterEthernet.slx`.
- Slave node model : `dPTPSlaveEthernet.slx`.

The two models are configured for download on target computers TargetPC1 and TargetPC2 respectively. If one of these target computers, for example TargetPC2, does not exist in the Simulink® Real-Time™ environment configuration on your development computer, you can create and configure it by using Simulink® Real-Time™ Explorer or by typing the following command at the MATLAB® command line:

```
tg2 = SimulinkRealTime.addTarget('TargetPC2');
```

For each model, open the mask for the IEEE 1588 Ethernet block and insert the required values for the PCI bus and slot numbers assigned to the Intel 82574 Ethernet card. For example, model `dPTPSlaveEthernet` is configured to run on target computer TargetPC2. To obtain the bus and slot numbers, type the following commands at the MATLAB® command line and look for the information for the Intel® 82574 Ethernet card:

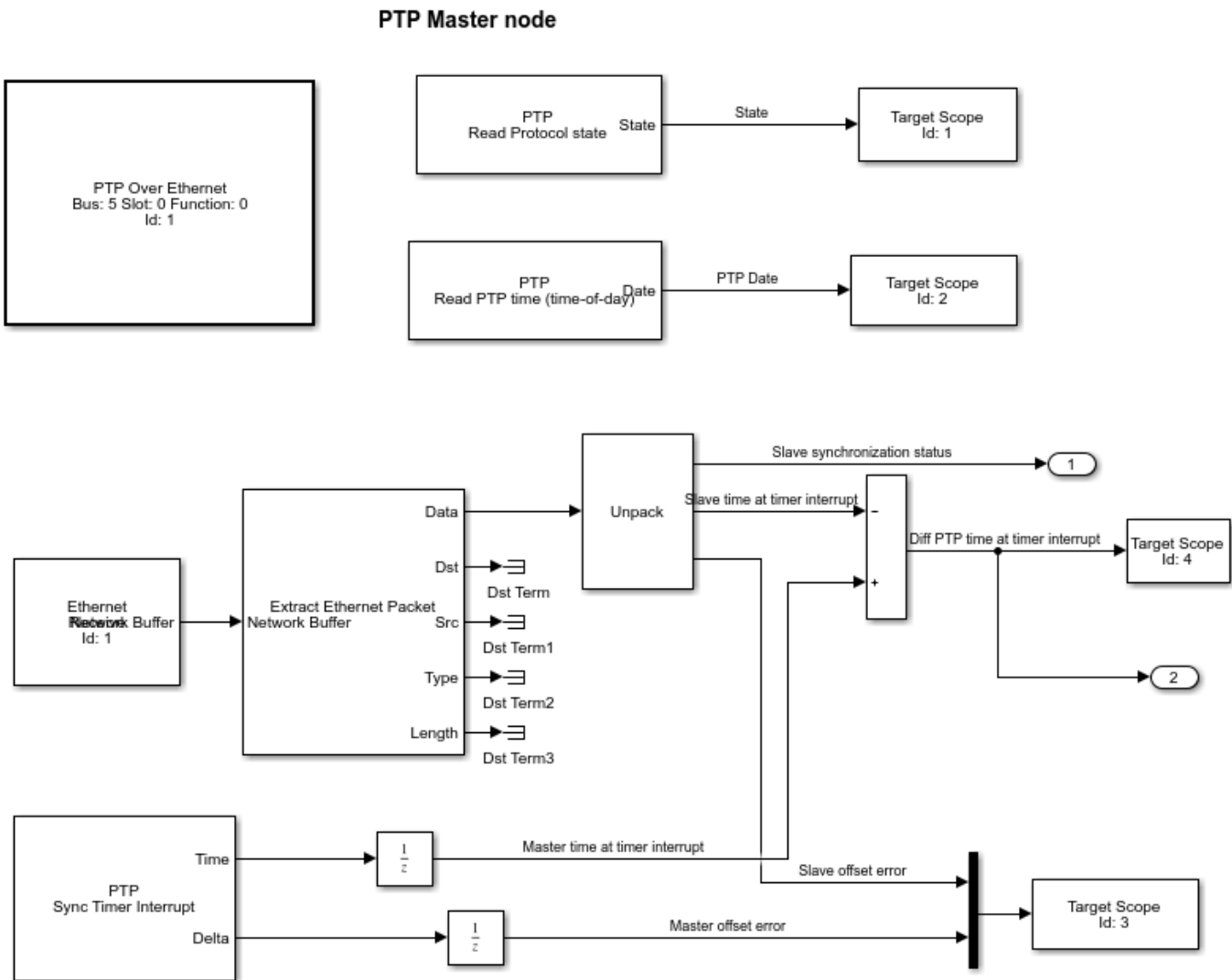
```
tg2 = SimulinkRealTime.target('TargetPC2');  
getPCIInfo(tg2, 'ethernet');
```

Model dPTPMasterEthernet is the PTP Master. It displays status and synchronization errors on target computer scopes and receives data from model dPTPSlaveEthernet.

```
openedMdl = find_system('type', 'block_diagram');  
masterMdl = fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'dPTPMasterEthernet')  
masterMdlOpen = 0;
```

Check if the model is already opened. If not, open it.

```
if ~any(strcmp(masterMdl, openedMdl))  
    masterMdlOpen = 1;  
    open_system(masterMdl);  
end
```

Copyright 2015 The MathWorks, Inc.

Models dPTPSlaveEthernet is configured as PTP Slave node. It displays status and synchronization errors on target scopes and sends three signals to model dPTPMasterEthernet:

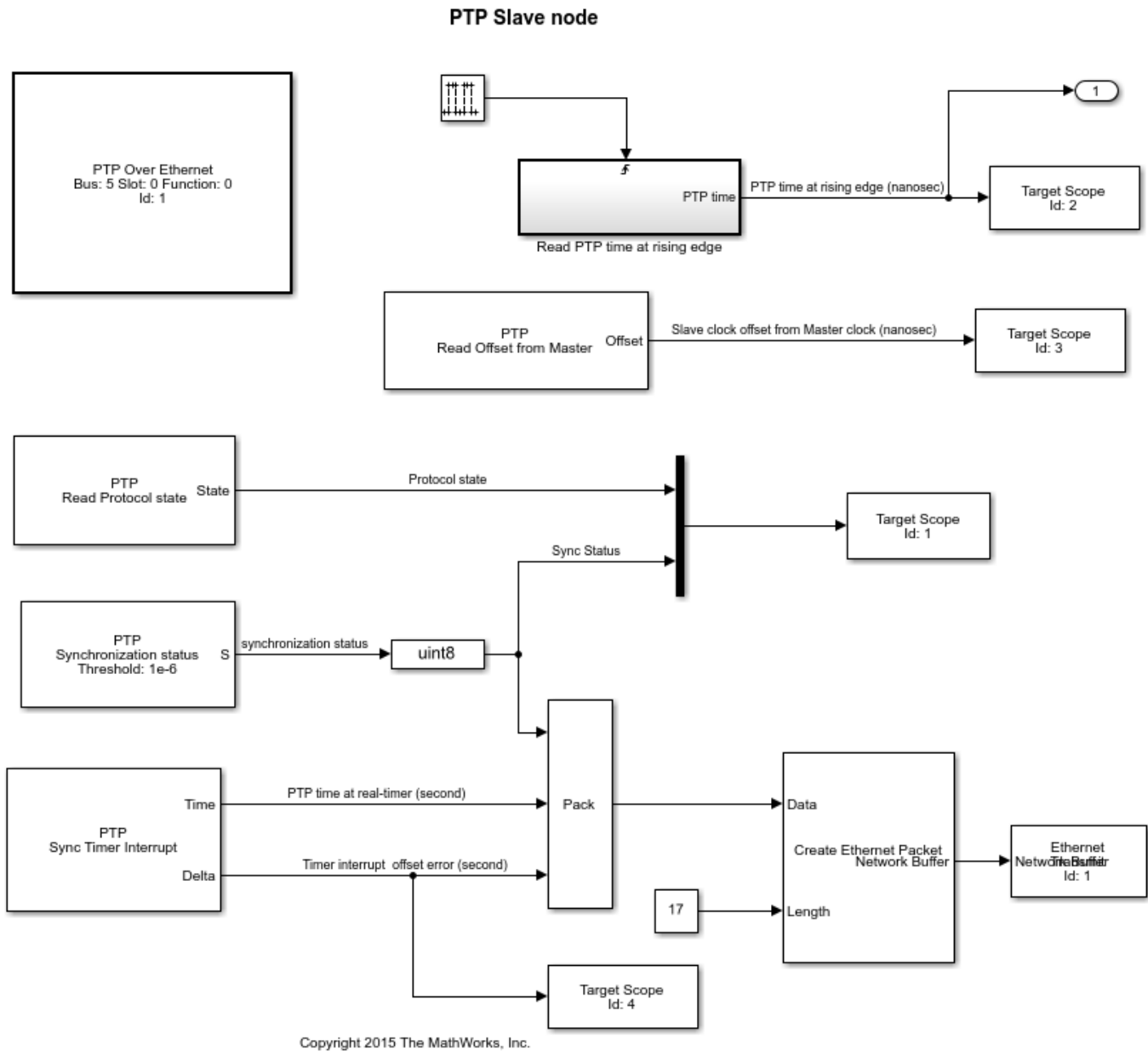
- Synchronization status: Indicates when the slave PTP clock is synchronized to the master PTP clock within the configured threshold of one microsecond.
- PTP time at timer interrupt: Indicates the value in seconds of the PTP time when the real-time interrupt occurs.
- Offset error: Indicates the synchronization error between the PTP time and the kernel clock that generates real-time interrupts.

The Read PTP time at rising edge subsystem shows how you can log timestamps of an event with PTP. The IEEE 1588 Read Parameter block in the subsystem is configured to read the PTP time when the block execution starts.

```
slaveMdl = fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'dPTPSlaveEthernet');
slaveMdlOpen = 0;
```

Check if the model is already opened. If not, open it.

```
if ~any(strcmp(slaveMdl, openedMdl))
    slaveMdlOpen = 1;
    open_system(slaveMdl);
end
```



Build and download the models onto the target computer

- Configure for a non-verbose build.

- Set Ethernet configuration to match target computer settings.
- Build and download the models onto the target computers.

```
set_param(masterMdl, 'RTWVerbose', 'off');
set_param('masterMdl/IEEE 1588 Ethernet', 'PciBus', '5', 'PciSlot', '0');
rtwbuild(masterMdl);
tg1 = slrt('TargetPC1');
load(tg1, 'masterMdl');
```

```
set_param(slaveMdl, 'RTWVerbose', 'off');
set_param('slaveMdl/IEEE 1588 Ethernet', 'PciBus', '8', 'PciSlot', '10');
rtwbuild(slaveMdl);
tg2 = slrt('TargetPC1');
load(tg2, 'slaveMdl');
```

Run and stop application

Run the two models for 50 seconds

```
tg1.start;
tg2.start;
pause(50);
```

Stop the models

```
tg1.stop;
tg2.stop;
```

Display the target computer scopes and stop the execution

View the target computer display.

For model dPTPSlaveEthernet on TargetPC2, use the command:

```
tg2.viewTargetScreen;
```

- Scope 1 shows the protocol state and the synchronization status. The protocol state value is 9 for a Slave node. The synchronization status is 1 if the Slave PTP clock is synchronized to the Master PTP clock within the specified threshold of 1 microsecond. Otherwise, it is 0.
- Scope 2 displays the PTP time when the Read PTP time at rising edge triggered subsystem is triggered.
- Scope 3 shows the current time offset between the Slave PTP clock and the Master PTP clock in nanoseconds.
- Scope 4 shows the current time offset between the real-time interrupt clock and the PTP clock in seconds.

For model dPTPMasterEthernet on TargetPC1, use the command:

```
tg1.viewTargetScreen;
```

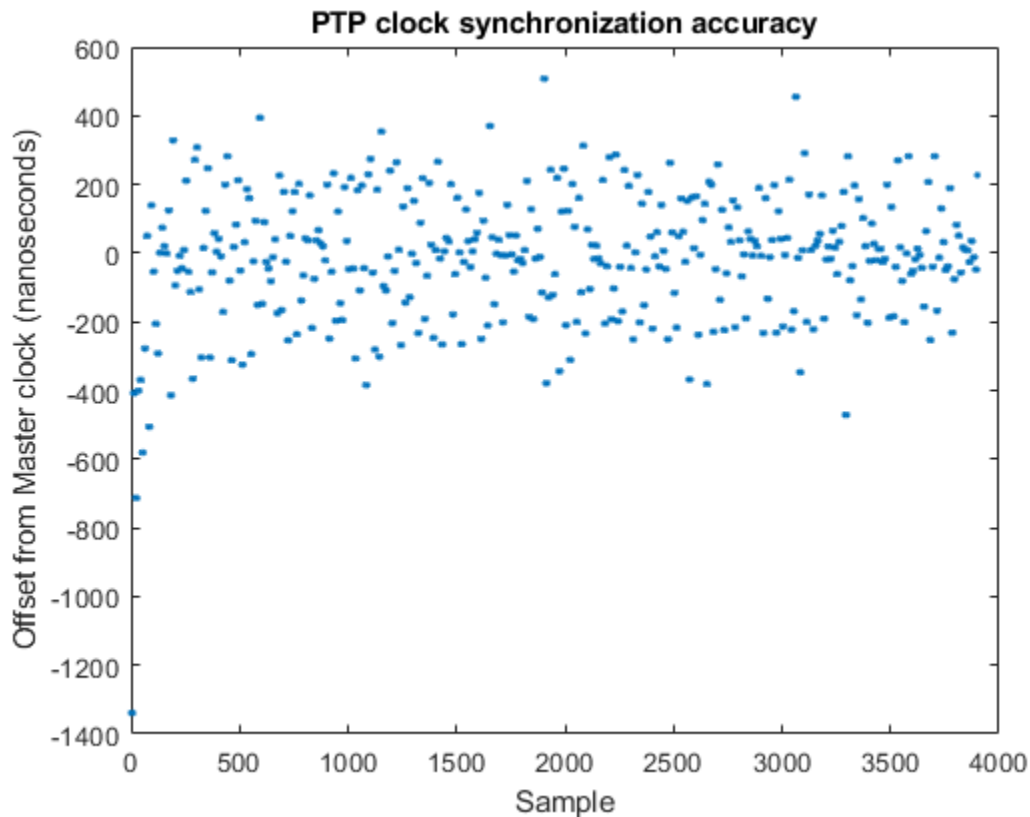
- Scope 1 shows the protocol state. The protocol state value is 6 for the Master node.
- Scope 2 displays the current PTP time in date format.
- Scope 3 shows the current time offset between the real-time interrupt clock and the PTP clock in seconds, for the Master node and the Slave node (received from model dPTPSlaveEthernet).

- Scope 4 shows the difference between the PTP time when the real-time interrupt occurs on the Master node and the Slave node respectively.

When the Slave PTP clock is synchronized to the Master PTP clock and both the Slave and Master real-time interrupt clocks are synchronized to their respective PTP clocks, the signal on Scope 4 indicates the precision to which the execution of the two models is synchronized.

Obtain and plot logged data

The following figure shows the value displayed on Scope 4 of model dPTPMasterEthernet when the two nodes have their PTP clocks and kernel clocks synchronized.



Retrieve logged data for slave nodesdPTPMa

```
logDataMaster = tg1.OutputLog;
syncIndex = find(logDataMaster(:,1) ~= 0, 1, 'first');
clockDiff = logDataMaster(syncIndex +100:end,2);
```

Does the plot figure exist?

- If no, create figure.
- If yes, make it the current figure.

```
figh = findobj('Name', 'ptpexample');
if isempty(figh)
    figh = figure;
    set(figh, 'Name', 'ptpexample', 'NumberTitle', 'off');
```

```
else
    figure(figh);
end
```

Plot difference of PTP time at real-time interrupt for the two models

```
plot(clockDiff, '.');
xlabel('Sample');
ylabel('Time difference (seconds)');
title('Execution synchronization precision');

drawnow;
```

Close the models

Close the model if we opened them.

```
if (masterMdlOpen)
    bdclose(masterMdl);
end
if (slaveMdlOpen)
    bdclose(slaveMdl);
end
```

IEEE® 1588™ Precision Time Protocol - Clock Synchronization

This example shows clock synchronization between target computers using the Precision Time Protocol (PTP) with UDP as transport protocol

Requirements

To run this example, you need to:

- 1 Acquire two Speedgoat target computers equipped with the Intel® i210 Ethernet card from www.speedgoat.com. Alternatively, you can use the Intel 82574L Ethernet card.
- 2 Acquire a standard network switch that has at least 4 ports, such as the Cisco® SG110D-08 8-ports network switch.

Connect and configure the devices in the network

In this example, the two target computers are named TargetPC1 and TargetPC2. TargetPC1 has one Intel® i210 network card that is used for host-target connection and PTP synchronization. TargetPC2 has a dedicated Intel® i210 card for PTP synchronization.

- 1 Connect the network port of the development computer to a port of the Cisco® Ethernet switch.
- 2 Connect the network port of each target computer that is dedicated to communicating with the development computer to a port of the Cisco® SG110D-08 Ethernet switch.
- 3 Connect the network port of the Intel® i210 Ethernet card in TargetPC2 to a port of the the Cisco® Ethernet switch.

Note: For better PTP synchronization accuracy, consider using a network switch that support PTP in place of the Cisco® SG110D-08 switch. An example of PTP capable network swicth is the EDS-405A-PTP from www.moxa.com.

Open the models

Click on the following links to open the two models:

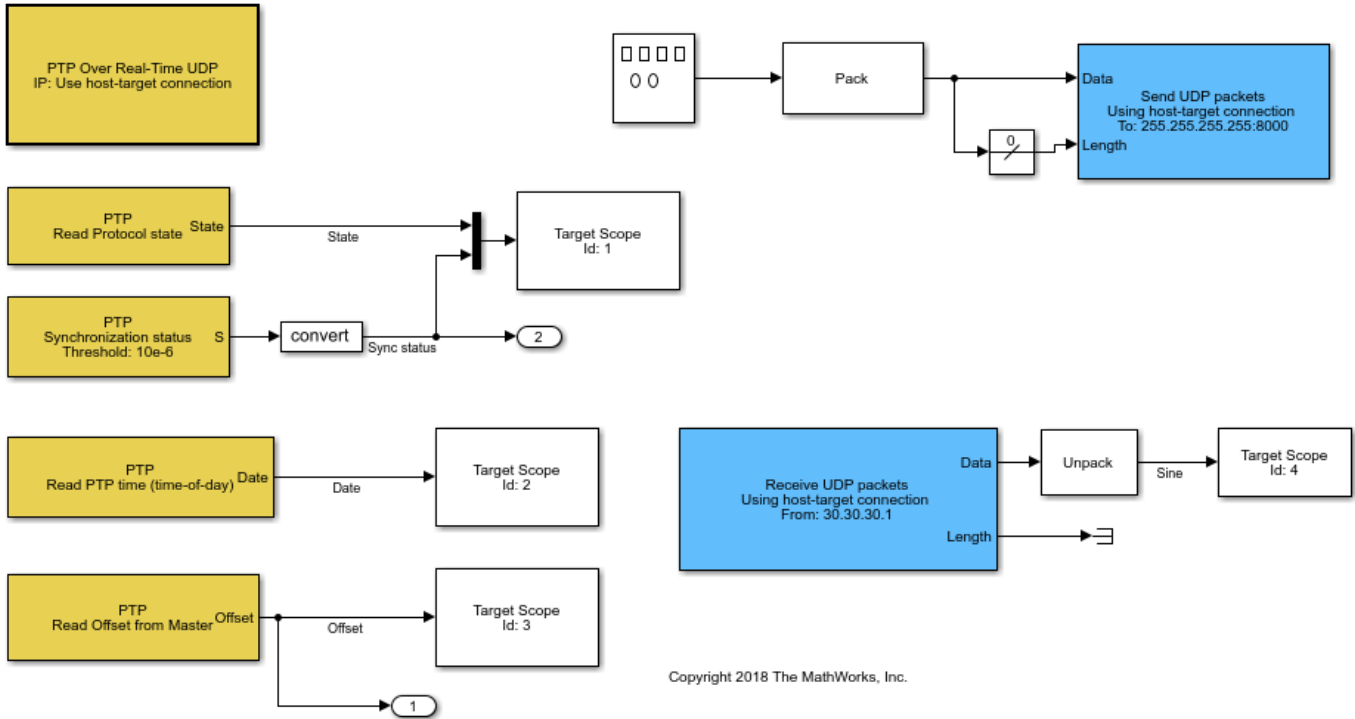
- Node 1 model: [dPTPUDPNode1.slx](#).
- Node 2 model : [dPTPUDPNode2.slx](#).

Open the dPTPUDPNode1 model:

```
openedMdl = find_system('type', 'block_diagram');
mdl1 = fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'dPTPUDPNode1');
mdl1open = 0;
mdlName1 = 'dPTPUDPNode1';
```

Check if the model is already opened. If not, open it.

```
if ~any(strcmp(mdl1, openedMdl))
    mdl1open = 1;
    open_system(mdl1);
end
```

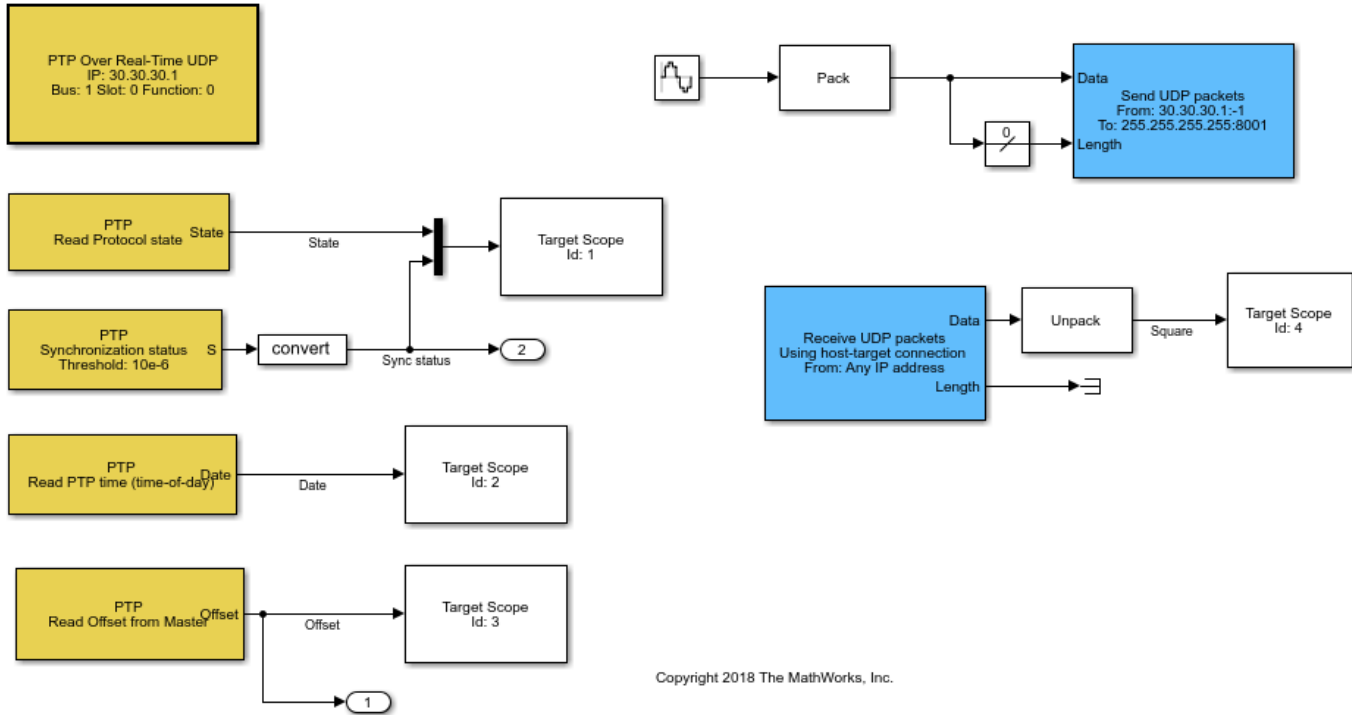


Open the dPTPUDPNode2 model:

```
model2 = fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'dPTPUDPNode2');
model2open = 0;
modelName2 = 'dPTPUDPNode2';
```

Check if the model is already opened. If not, open it.

```
if ~any(strcmp(model2, openedMdl))
    model2open = 1;
    open_system(model2);
end
```



Configure the models

The two models are configured for download on target computers **TargetPC1** and **TargetPC2** respectively. If one of these target computers setting, does not exist in the Simulink® Real-Time™ environment configuration on your development computer, you can create and configure it by using Simulink® Real-Time™ Explorer or MATLAB® command line. For example, to add **TargetPC2** to your environment, type the following command at the MATLAB® command line:

```
tg2 = SimulinkRealTime.addTarget('TargetPC2');
```

Model **dPTPUDPNode1** is configured to use the host-target connection for PTP. Configure **TargetPC1** to boot using the Intel® i210 card for host-target communication. If you are using network boot, you can configure the target settings using Simulink® Real-Time™ Explorer or by typing the following commands at the MATLAB® command line:

```
env = SimulinkRealTime.getTargetSettings('TargetPC1');
env.TargetBoot = 'NetworkBoot';
env.TcpIpTargetDriver = 'I210';
SimulinkRealTime.createBootImage(env);
```

For model **dPTPUDPNode2**, open the mask for the IEEE 1588 UDP block and insert the required values for the PCI bus, slot and function numbers assigned to the Intel® i210 card dedicated to PTP synchronization. To obtain the bus, slot and function numbers, type the following commands at the MATLAB® command line and look for the information for the Intel® i210 Ethernet card (or Intel 82574L card if that is the card you are using for PTP):

```
tg2 = SimulinkRealTime.target('TargetPC2');
getPCIInfo(tg2, 'ethernet');
```

Each model displays on target computer scopes:

- The PTP protocol state.
- The PTP clock synchronization status (relevant for Slave clock only).
- The PTP time in date format.
- The offset from the Master (relevant for Slave clock only).
- A signal received from the other model.

The PTP hardware clock running on the Intel® i210 Ethernet card is initialized with the target computer system time. Before you run the example, you can check the target computer system time by typing the following command at the MATLAB® command line:

```
tg1 = SimulinkRealTime.target('TargetPC1');
d = SimulinkRealTime.utils.getTargetSystemTime(tg1)
```

If the target computer system time is incorrect, you can set it to your desired value. For example, to set the target computer system time to be the same as the development computer system time in UTC time zone, type the following command at the MATLAB® command line:

```
SimulinkRealTime.utils.setTargetSystemTime(tg1);
```

Build and download the models onto the target computer

- Configure for a non-verbose build.
- Build and download the models onto the target computers.

```
set_param(modelName1, 'RTWVerbose', 'off');
rtwbuild(modelName1);
tg1 = slrt('TargetPC1');
load(tg1, modelName1);
set_param(modelName2, 'RTWVerbose', 'off');
set_param([modelName2, '/IEEE 1588 Real-Time UDP'], 'PciBus', '8', 'PciSlot', '10');
rtwbuild(modelName2);
tg2 = slrt('TargetPC1');
load(tg2, modelName2);
```

Run The application

Run the two models for 50 seconds `tg1.start; tg2.start; pause(50);`

One of The models will be the PTP Master clock and the other one the Slave clock. Since the PTP clock parameters for the two models are identical, this state decision will be made based on the MAC address of the Ethernet cards. In our setup, `dPTPUDPNode1` is selected as the Slave clock.

Stop the application

```
tg1.stop;
tg2.stop;
```

Display the target computer scopes

View the target computer display.

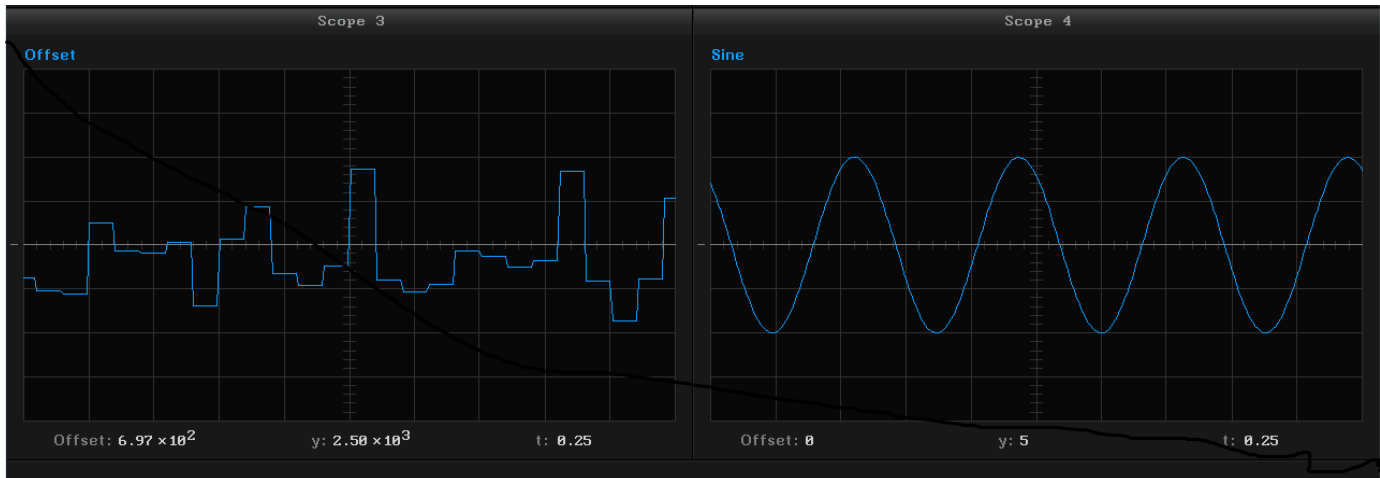
- Scope 1 shows the protocol state and the synchronization status. The protocol state value is equal to 9 for the Slave clock and equal to 6 for the Master clock. The synchronization status is always equal to 1 for the Master clock after the state switch to 6. For the Slave clock, the synchronization status is initially equal to 0, and then switch to 1 when the clock servo locks to

the Master clock within the threshold specified in the block parameter. Note that when the model starts and the protocol state switches to Slave clock, a couple of seconds elapse before the clock servo locks to the master clock.

- The offset from master value displayed on Scope 3 is relevant only for the Slave clock. For the Master clock it is always equal to 0.

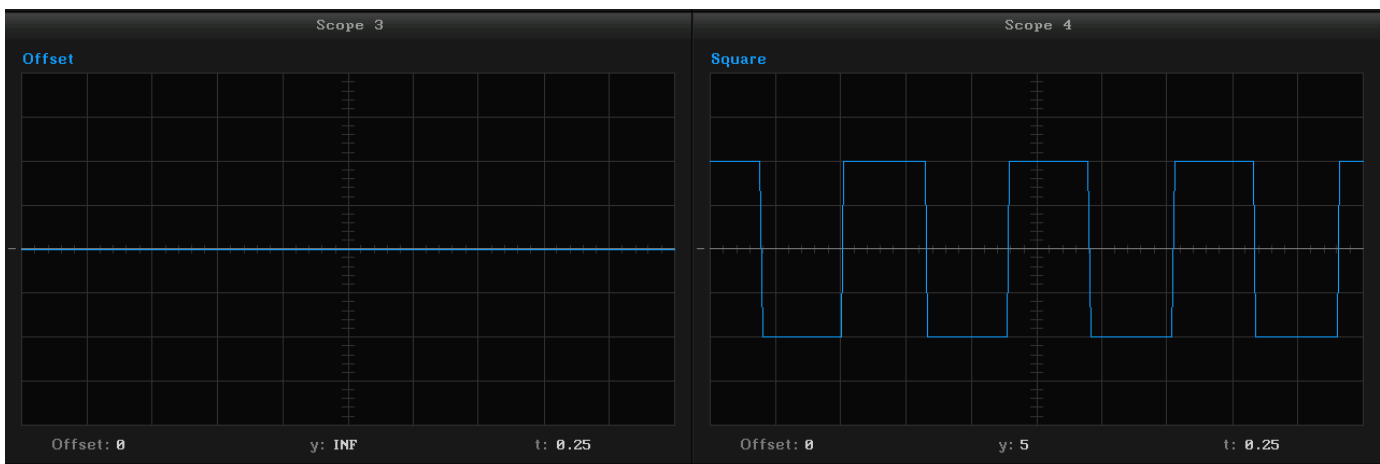
For model dPTPUDPNode1 on TargetPC1, use the command:

```
tg1.viewTargetScreen;
```



For model dPTPUDPNode2 on TargetPC2, use the command:

```
tg2.viewTargetScreen;
```



Obtain and plot the offset from Master.

The following figure shows the value displayed on Scope 3 for the Slave clock when the clock servo lock to the Master clock.

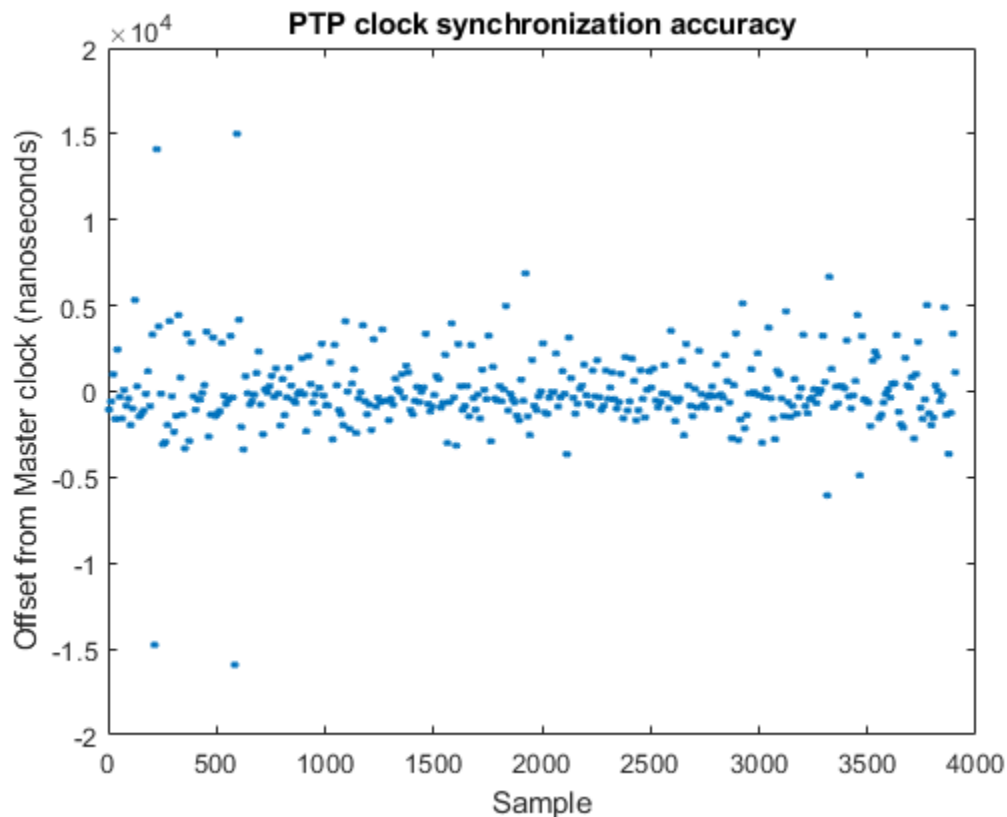
The accuracy of clock synchronization depends on the network switch that you use to connect with the target computers, because the switch can introduce delay in the packet transmission. To achieve

more accurate clock synchronization, acquire a PTP transparent clock switch, such as the EDS-405A-PTP from www.moxa.com.

```

figh = findobj('Name', 'PTPExampleUDP');
if isempty(figh)
    figh = figure;
    set(figh, 'Name', 'PTPExampleUDP', 'NumberTitle', 'off');
else
    figure(figh);
end
data = tg1.OutputLog;
if ~any(data(:, 1))
    data = tg2.OutputLog;
end
Find index when clock enter Slave state and servo locked
offsetIndex = find(data(:,1) ~= 0, 1, 'first');
syncIndex = offsetIndex + find(data(offsetIndex:end,2) ~= 0, 1, 'first');
offset = data(syncIndex:end, 1);
plot(offset, '.');
xlabel('Sample');
ylabel('Offset from Master clock (nanoseconds)');
title('PTP clock synchronization accuracy');
drawnow limitrate;%

```



Close the models

Close the model if we opened them.

```
if (model1open)
    bdclose(model1);
end
if (model2open)
    bdclose(model2);
end
```

Real-Time Transmit and Receive over Ethernet

Communicate between two Simulink® Real-Time™ models over Ethernet.

This example shows how to use blocks in the library `xpcethernetlib` to communicate between two target computers over Ethernet. Signal data is sent by the transmitter model, `xpcEnetDemo1Tx`, running on one target computer, TargetPC1, to the receiver model, `xpcEnetDemo1Rx`, running on the second target computer, TargetPC2. The blocks in `xpcethernetlib` enable "raw" Ethernet for real-time IO.

Requirements

To run this example, you will need two target computers, each with an installed and configured **dedicated** Ethernet card (in addition to the Ethernet card used for the Ethernet link between the development and target computers). Refer to the Simulink Real-Time documentation on model-based Ethernet communications for details. Once configured, set the PCI Bus and Slot in the "Real-Time Ethernet Configuration" block of `xpcEnetDemo1Tx` and `xpcEnetDemo1Rx` to that of the Ethernet card installed in TargetPC1 and TargetPC2 respectively.

Open, Build, and Download the Tx Model to TargetPC1

Click [here](#) to open the Tx model: `xpcEnetDemo1Tx`. This model drives an oscillator with a square wave signal and sends the oscillator input and output signals to the Rx target computer using raw Ethernet.

Open the model.

```
mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if ~any(strcmp(systems, 'xpcEnetDemo1Tx'))
    mdlOpen = 1;
    open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcEnetDemo1Tx'))
end
```

Build the model and download to the Tx target computer, TargetPC1.

- Configure for a non-Verbose build.
- Set Ethernet configuration to match target computer settings
- Build and download application.
- Close the model if we opened it.

```
set_param('xpcEnetDemo1Tx', 'RTWVerbose', 'off');
set_param('xpcEnetDemo1Tx/Real-time Ethernet Configuration', 'Driver', 'Intel Gigabit', 'Bus', '5', 'Slot', '1');
evalc('rtwbuild(''xpcEnetDemo1Tx'')');
tgTx = slrt('TargetPC1');
load(tgTx, 'xpcEnetDemo1Tx');
if (mdlOpen)
    bdclose('xpcEnetDemo1Tx');
end
```

Open, Build, and Download the Rx Model to TargetPC2

Click [here](#) to open the Rx model: `xpcEnetDemo1Rx`. This model receives data sent by `xpcEnetDemo1Tx` and unpacks the data for display in a target scope. Open the model.

```
mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if ~any(strcmp(systems, 'xpcEnetDemo1Rx'))
    mdlOpen = 1;
    open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcEnetDemo1Rx'))
end
```

Build the model and download to the Rx target computer, TargetPC2.

- Configure for a non-Verbose build.
- Set Ethernet configuration to match target computer settings
- Build and download application.
- Close the model if we opened it.

```
set_param('xpcEnetDemo1Rx', 'RTWVerbose', 'off');
set_param('xpcEnetDemo1Rx/Real-time Ethernet Configuration', 'Driver', 'Intel Gigabit', 'Bus', '8', '1');
evalc('rtwbuild(''xpcEnetDemo1Rx'')');
tgRx = slrt('TargetPC2');
load(tgRx, 'xpcEnetDemo1Rx');
if (mdlOpen)
    bdclose('xpcEnetDemo1Rx');
end
```

Run both Models

Using the Simulink Real-Time object variables `tgTx` and `tgRx`, start the models.

- Start the Tx model.
- Start the Rx model.
- Let the models run for at least 5 sec.

```
start(tgTx);
start(tgRx);
pause(5);
```

Display the Tx Target Computer Scope

View the Tx target computer video display. It displays a plot of the signal data that is sent to the Rx target computer via raw Ethernet. Use command:

```
tgTx.viewTargetScreen
```

Display the Rx Target Computer Scopes

View the Rx target computer video display. It displays a plot of the signal data that is received from the Tx target computer via raw Ethernet. Use command:

```
tgRx.viewTargetScreen
```

Stop both Models

When done, stop the models from running.

- Stop the Tx model.
- Stop the Rx model.

```
stop(tgTx);  
stop(tgRx);
```

Filtering on MAC Address

Filtering Ethernet data using MAC addresses.

This example shows how to use blocks in the library `xpcethernetlib` to filter Ethernet data based on the sender's MAC address. Signal data is sent by the transmitter model, `xpcEnetDemo2Tx`, running on one target computer, TargetPC1, to the receiver model, `xpcEnetDemo2Rx`, running on the second target computer, TargetPC2. The "Filter Address" block is used to specify the source MAC addresses that will be accepted. In this simple example, data packets containing one of three source MAC addresses will pass through the Filter Address block in the receiver model:

- 1 MAC Address : 40:41:42:43:44:45
- 2 MAC Address : 20:21:22:23:24:25
- 3 MAC Address : 50:51:52:53:54:55

Note: Only packets with a MAC address 20:21:22:23:24:25 are actually transmitted, received, and plotted.

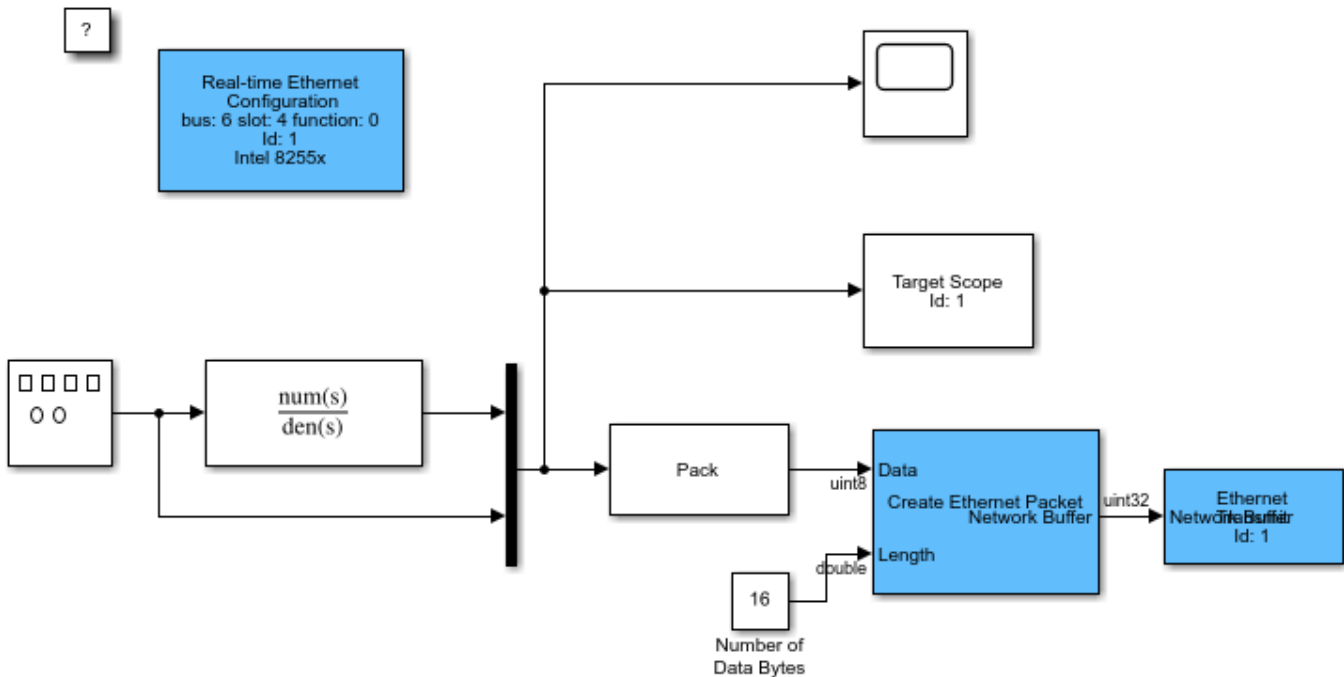
Requirements

To run this example, you will need two target computers each with, each with an installed and configured **dedicated** Ethernet card (in addition to the Ethernet card used for the Ethernet link between the development and target computers). Refer to the Simulink® Real-Time™ documentation on model-based Ethernet communications for details. Once configured, set the PCI Bus and Slot in the "Real-Time Ethernet Configuration" block of `xpcEnetDemo2Tx` and `xpcEnetDemo2Rx` to that of the Ethernet card installed in TargetPC1 and TargetPC2 respectively.

Open, Build, and Download the Tx Model to TargetPC1

Click here to open the Tx model: `xpcEnetDemo2Tx`. This model drives an oscillator with a square wave signal and sends the oscillator input and output signals to the Rx target computer using raw Ethernet. Transmitted data packets have a specified MAC address of 20:21:22:23:24:25. Open the model.

```
mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if ~any(strcmp(systems, 'xpcEnetDemo2Tx'))
    mdlOpen = 1;
    open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcEnetDemo2Tx'));
end
```

Copyright 2008-2015 The MathWorks, Inc.

Build the model and download to the Tx target computer, TargetPC1.

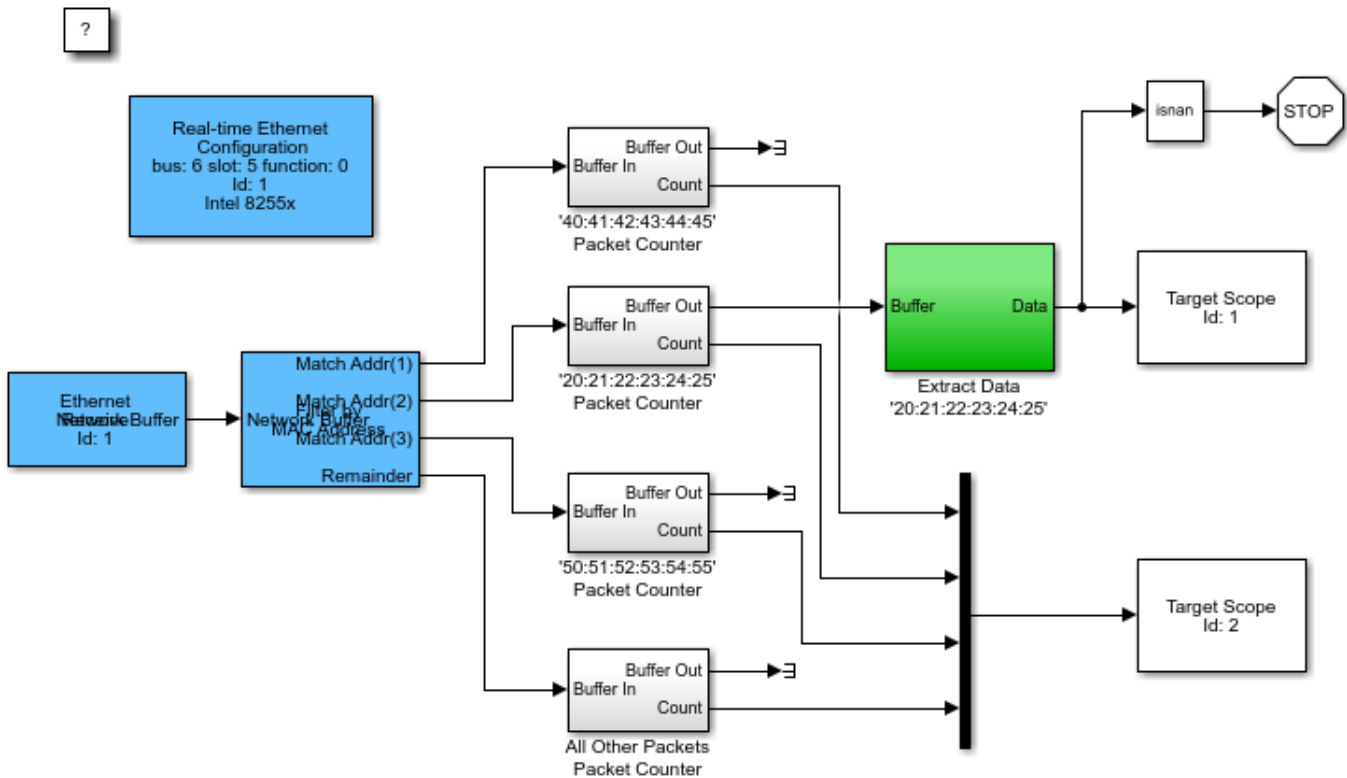
- Configure for a non-Verbose build.
- Set Ethernet configuration to match target computer settings.
- Build and download application.
- Close the model if we opened it.

```
set_param('xpcEnetDemo2Tx','RTWVerbose','off');
set_param('xpcEnetDemo2Tx/Real-time Ethernet Configuration','Driver','Intel Gigabit','Bus','5','');
evalc('rtwbuild(''xpcEnetDemo2Tx'')');
tgTx = slrt('TargetPC1');
load(tgTx,'xpcEnetDemo2Tx');
if (mdlOpen)
    bdclose('xpcEnetDemo2Tx');
end
```

Open, Build, and Download the Rx Model to TargetPC2

Click here to open the Rx model: xpcEnetDemo2Rx. This model filters packets based on specified MAC addresses and unpacks the received data for display in target scopes. Open the model.

```
mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if ~any(strcmp(systems, 'xpcEnetDemo2Rx'))
    mdlOpen = 1;
    open_system(fullfile(matlabroot,'toolbox','rtw','targets','xpc','xpcdemos','xpcEnetDemo2Rx'));
end
```



Copyright 2008-2015 The MathWorks, Inc.

Build the model and download to the Rx target computer, TargetPC2.

- Configure for a non-Verbose build.
- Set Ethernet configuration to match target computer settings.
- Build and download application.
- Close the model if we opened it.

```
set_param('xpcEnetDemo2Rx', 'RTWVerbose', 'off');
set_param('xpcEnetDemo2Rx/Real-time Ethernet Configuration', 'Driver', 'Intel Gigabit', 'Bus', '8', ' ');
evalc('rtwbuild(''xpcEnetDemo2Rx'')');
tgRx = slrt('TargetPC2');
load(tgRx, 'xpcEnetDemo2Rx');
if (mdlOpen)
    bdclose('xpcEnetDemo2Rx');
end
```

Run both Models

Using the Simulink Real-Time object variables tgTx and tgRx, start the models.

- Start the Tx model.
- Start the Rx model.
- Let the models run for at least 5 sec.

```
start(tgTx);  
start(tgRx);  
pause(5);
```

Display the Tx Target Computer Scope

View the Tx target computer video display. It displays a plot of the signal data that's sent to the Rx target computer via raw Ethernet. Use command:

```
tgTx.viewTargetScreen
```

Display the Rx Target Computer Scopes

View the Rx target computer video display. It displays a plot of the signal data received from the Tx target computer via raw Ethernet. Note that data passes through only one of the MAC address filters. Use command:

```
tgRx.viewTargetScreen
```

Stop both Models

When done, stop the models from running.

- Stop the Tx model.
- Stop the Rx model.

```
stop(tgTx);  
stop(tgRx);
```

Filtering on EtherType

Filtering Ethernet data using EtherType.

This example shows how to use blocks in the library `xpcethernetlib` to filter Ethernet data based on the sender's EtherType. EtherType is a field in the Ethernet networking standard and indicates which protocol is being transported in an Ethernet packet. Signal data is sent by the transmitter model, `xpcEnetDemo3Tx`, running on one target computer, TargetPC1, to the receiver model, `xpcEnetDemo3Rx`, running on the second target computer, TargetPC2. The "Filter Type" block is used to specify the EtherTypes that will be accepted. In this simple example, data packets containing one of three different EtherTypes are allowed to pass through respective Filter Type blocks in the receiver model. The accepted Ether Types are:

- 1 EtherType : 88CD (SERCOS-III)
- 2 EtherType : 88A4 (EtherCAT®)
- 3 EtherType : 0800 (IPv4)

Notes:

- 1 In this example, only packets with EtherTypes 88CD (SERCOS-III) and 88A4 (EtherCAT) are transmitted, received, and plotted.
- 2 Real SERCOS-III and EtherCAT packets are not being transmitted. Only the EtherType field in the Ethernet packet is set to emulate these types.

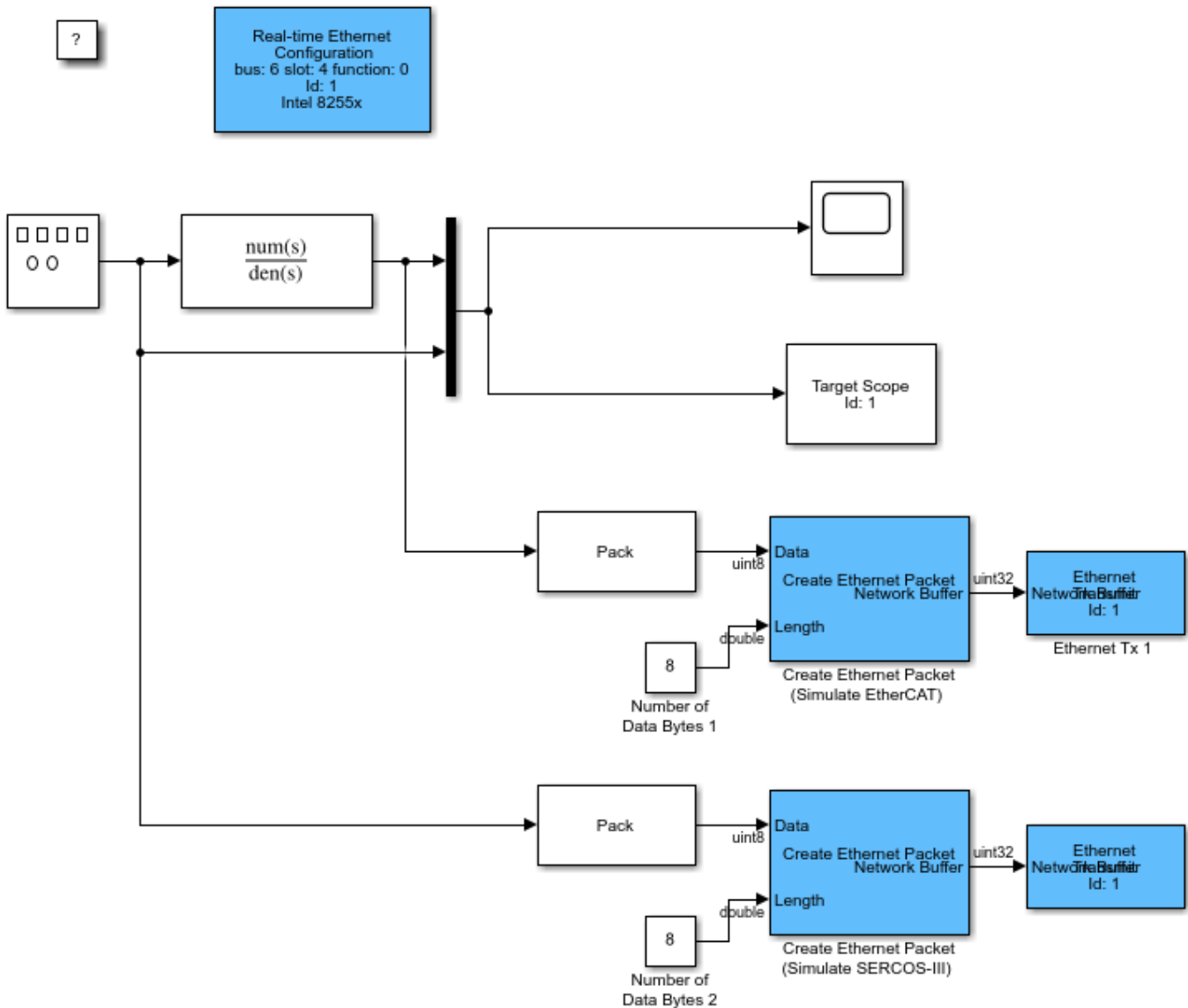
Requirements

To run this example, you will need two target computers, each with an installed and configured **dedicated** Ethernet card (in addition to the Ethernet card used for the Ethernet link between the development and target computers). Refer to the Simulink® Real-Time™ documentation on model-based Ethernet communications for details. Once configured, set the PCI Bus and Slot in the "Real-Time Ethernet Configuration" block of `xpcEnetDemo3Tx` and `xpcEnetDemo3Rx` to that of the Ethernet card installed in TargetPC1 and TargetPC2 respectively.

Open, Build, and Download the Tx Model to TargetPC1

Click here to open the Tx model: `xpcEnetDemo3Tx`. This model drives an oscillator with a square wave signal and sends the oscillator input and output signals to the Rx target computer. The oscillator input (square wave) is transmitted as a SERCOS-III packet and the oscillator output is transmitted as an EtherCAT packet. Open the model.

```
mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if ~any(strcmp(systems, 'xpcEnetDemo3Tx'))
    mdlOpen = 1;
    open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcEnetDemo3Tx'));
end
```



Copyright 2008-2015 The MathWorks, Inc.

Build the model and download to the Tx target computer, TargetPC1.

- Configure for a non-Verbose build.
- Set Ethernet configuration to match target computer settings.
- Build and download application.
- Close the model if we opened it.

```
set_param('xpcEnetDemo3Tx', 'RTWVerbose', 'off');
set_param('xpcEnetDemo3Tx/Real-time Ethernet Configuration', 'Driver', 'Intel Gigabit', 'Bus', '5', ' ');
evalc('rtwbuild(''xpcEnetDemo3Tx'')');
tgTx = slrt('TargetPC1');
load(tgTx, 'xpcEnetDemo3Tx');
```

```

if (mdlOpen)
    bdclose('xpcEnetDemo3Tx');
end

```

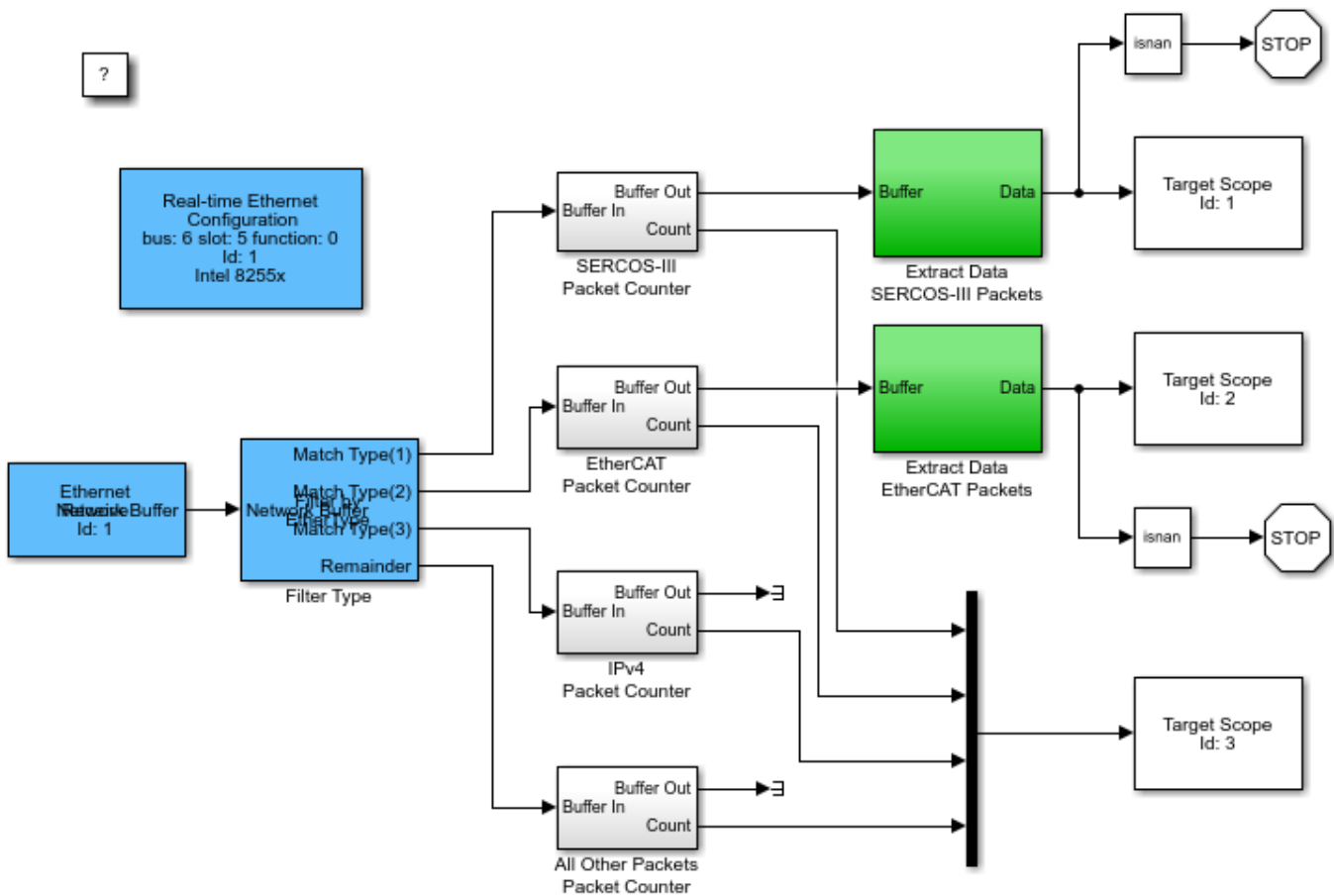
Open, Build, and Download the Rx Model to TargetPC2

Click here to open the Rx model: xpcEnetDemo3Rx. This model filters packets based on specified EtherType and unpacks the received data for display in target scopes. Open the model.

```

mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if ~any(strcmp(systems, 'xpcEnetDemo3Rx'))
    mdlOpen = 1;
    open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcEnetDemo3Rx'));
end

```



Copyright 2008-2015 The MathWorks, Inc.

Build the model and download to the Rx target computer, TargetPC2.

- Configure for a non-Verbose build.
- Set Ethernet configuration to match target computer settings.

- Build and download application.
- Close the model if we opened it.

```
set_param('xpcEnetDemo3Rx', 'RTWVerbose', 'off');
set_param('xpcEnetDemo3Rx/Real-time Ethernet Configuration', 'Driver', 'Intel Gigabit', 'Bus', '8', '9');
evalc('rtwbuild(''xpcEnetDemo3Rx'')');
tgRx = slrt('TargetPC2');
load(tgRx, 'xpcEnetDemo3Rx');
if (mdlOpen)
    bdclose('xpcEnetDemo3Rx');
end
```

Run both Models

Using the Simulink Real-Time object variables `tgTx` and `tgRx`, start the models.

- Start the Tx model.
- Start the Rx model.
- Let the models run for at least 5 sec.

```
start(tgTx);
start(tgRx);
pause(5);
```

Display the Tx Target Computer Scope

View the Tx target computer video display. It displays a plot of the signal data that is sent to the Rx target computer via raw Ethernet. Use command:

```
tgTx.viewTargetScreen;
```

Display the Rx Target Computer Scopes

View the Rx target computer video display. It displays a plot of the signal data received from the Tx target computer via raw Ethernet. Note that data with EtherTypes SERCOS-III and EtherCAT pass through the filters. Use command:

```
tgRx.viewTargetScreen;
```

Stop both Models

When done, stop the models from running.

- Stop the Tx model.
- Stop the Rx model.

```
stop(tgTx);
stop(tgRx);
```

Ethernet Rx Block Filtering

Filtering Ethernet data using the Ethernet Rx block.

This example shows how to use blocks in the library `xpcethernetlib` to filter Ethernet data based on the packet's EtherType. EtherType is a field in the Ethernet networking standard that designates which protocol is being transported in the Ethernet packet. Packets of signal data are sent by the transmitter model, `xpcEnetDemo4Tx`, running on one target computer, TargetPC1, to the receiver model, `xpcEnetDemo4Rx`, running on the second target computer, TargetPC2. The "Ethernet Rx" block is used to specify the EtherType filter criteria. In this simple example, two separate Ethernet Rx blocks are used to receive and filter data packets. One Ethernet Rx block accepts SERCOS-III packets (EtherType - 88CD), the second Ethernet Rx block accepts EtherCAT® packets (EtherType - 88A4). Go to the Filter tab in the respective Ethernet Rx blocks to see EtherType specification options.

Notes:

- 1 Real SERCOS-III and EtherCAT packets are not being transmitted. Only the EtherType field in the Ethernet packet is set to emulate these types.
- 2 This model shows how to receive and process packets using multiple Ethernet Rx blocks. Compare this example with `xpcEnetDemo3` which uses one Ethernet Rx block and then parses data packets with the Network Buffer Filter block. Use the method shown in this example, `xpcEnetDemo4`, if you want more than one Ethernet Rx block - perhaps each contained in separate subsystems.

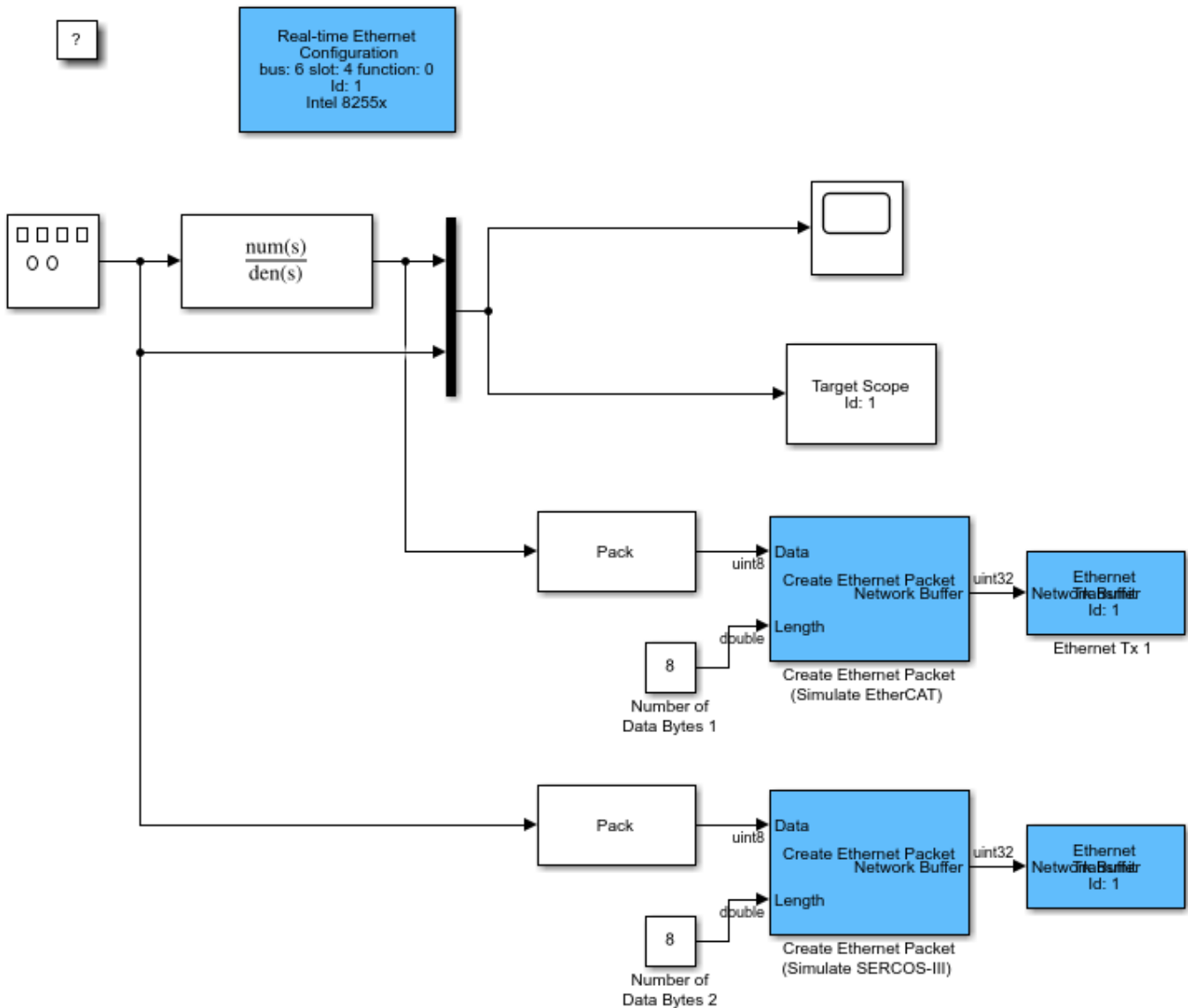
Requirements

To run this example, you will need two target computers, each with an installed and configured **dedicated** Ethernet card (in addition to the Ethernet card used for the Ethernet link between the development and target computers). Refer to the Simulink® Real-Time™ documentation on model-based Ethernet communications for details. Once configured, set the PCI Bus and Slot in the "Real-time Ethernet Configuration" block of `xpcEnetDemo4Tx` and `xpcEnetDemo4Rx` to that of the Ethernet card installed in TargetPC1 and TargetPC2 respectively.

Open, Build, and Download the Tx Model to TargetPC1

Click here to open the Tx model: `xpcEnetDemo4Tx`. This model drives an oscillator with a square wave signal and sends the oscillator input and output signals to the Rx target computer. The oscillator input (square wave) is transmitted as a SERCOS-III packet and the oscillator output is transmitted as an EtherCAT packet. Open the model.

```
mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if ~any(strcmp(systems, 'xpcEnetDemo4Tx'))
    mdlOpen = 1;
    open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcEnetDemo4Tx'));
end
```

Copyright 2008-2013 The MathWorks, Inc.

Build the model and download to the Tx target computer, TargetPC1.

- Configure for a non-Verbose build.
- Set Ethernet configuration to match target computer settings.
- Build and download application.
- Close the model if we opened it.

```
set_param('xpcEnetDemo4Tx', 'RTWVerbose', 'off');
set_param('xpcEnetDemo4Tx/Real-time Ethernet Configuration', 'Driver', 'Intel Gigabit', 'Bus', '5', ' ');
evalc('rtwbuild(''xpcEnetDemo4Tx'')');
tgTx = slrt('TargetPC1');
load(tgTx, 'xpcEnetDemo4Tx');
```

```

if (mdlOpen)
    bdclose('xpcEnetDemo4Tx');
end

```

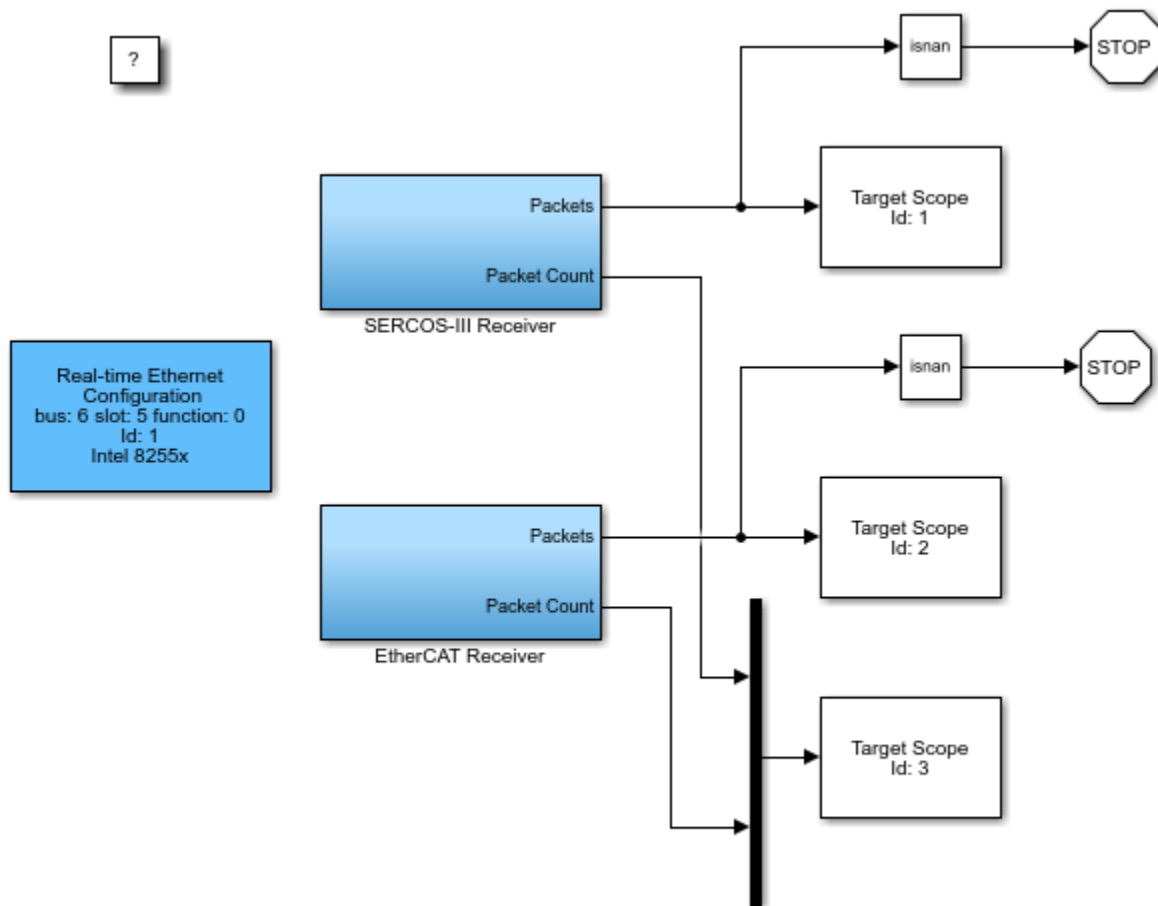
Open, Build, and Download the Rx Model to TargetPC2

Click [here](#) to open the Rx model: xpcEnetDemo4Rx. This model filters packets based on EtherType and unpacks the received data for display in target scopes. Open the model.

```

mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if ~any(strcmp(systems, 'xpcEnetDemo4Rx'))
    mdlOpen = 1;
    open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcEnetDemo4Rx'));
end

```



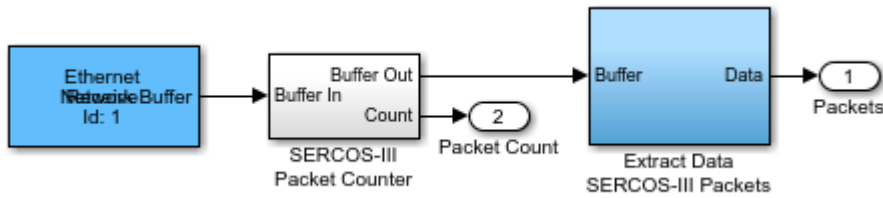
Copyright 2008-2015 The MathWorks, Inc.

SERCOS-III Receiver Subsystem

```

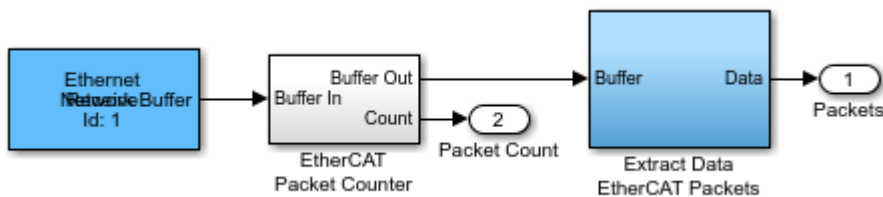
open_system('xpcEnetDemo4Rx/SERCOS-III Receiver');

```

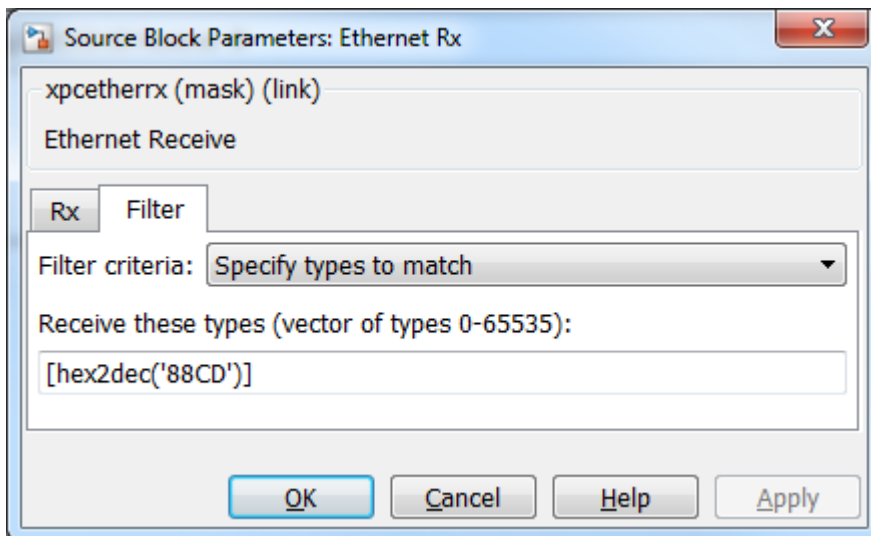


EtherCAT Receiver Subsystem

```
open_system('xpcEnetDemo4Rx/EtherCAT Receiver');
```



Select the Ethertype(s) of interest on the Filter tab of the Ethernet Rx block. As example, SERCOS-III packet are selected here:



Build the model and download to the Rx target computer, TargetPC2.

Configure for a non-Verbose build. * Set Ethernet configuration to match target computer settings. Build and download application. Close the model if we opened it.

```
set_param('xpcEnetDemo4Rx', 'RTWVerbose', 'off');
set_param('xpcEnetDemo4Rx/Real-time Ethernet Configuration', 'Driver', 'Intel Gigabit', 'Bus', '8', '1');
evalc('rtwbuild(''xpcEnetDemo4Rx'')');
tgRx = slrt('TargetPC2');
load(tgRx, 'xpcEnetDemo4Rx');
if (mdlOpen)
```

```
bdclose('xpcEnetDemo4Rx');  
end
```

Run both Models

Using the Simulink Real-Time object variables `tgTx` and `tgRx`, start the models.

- Start the Tx model.
- Start the Rx model.
- Let the models run for at least 5 sec.

```
start(tgTx);  
start(tgRx);  
pause(5);
```

Display the Tx Target Computer Scope

View the Tx target computer video display. It displays a plot of the signal data that is sent to the Rx target computer via raw Ethernet. Use command:

```
tgTx.viewTargetScreen;
```

Display the Rx Target Computer Scopes

View the Rx target computer video display. It displays a plot of the signal data received from the Tx target computer via raw Ethernet. Note that only data with SERCOS-III and EtherCAT EtherTypes are pass through the respective Ethernet Rx blocks. Use command:

```
tgRx.viewTargetScreen;
```

Stop both Models

When done, stop the models from running.

- Stop the Tx model.
- Stop the Rx model.

```
stop(tgTx);  
stop(tgRx);
```

Simple ASCII Encoding/Decoding Loopback Test (With Baseboard Blocks)

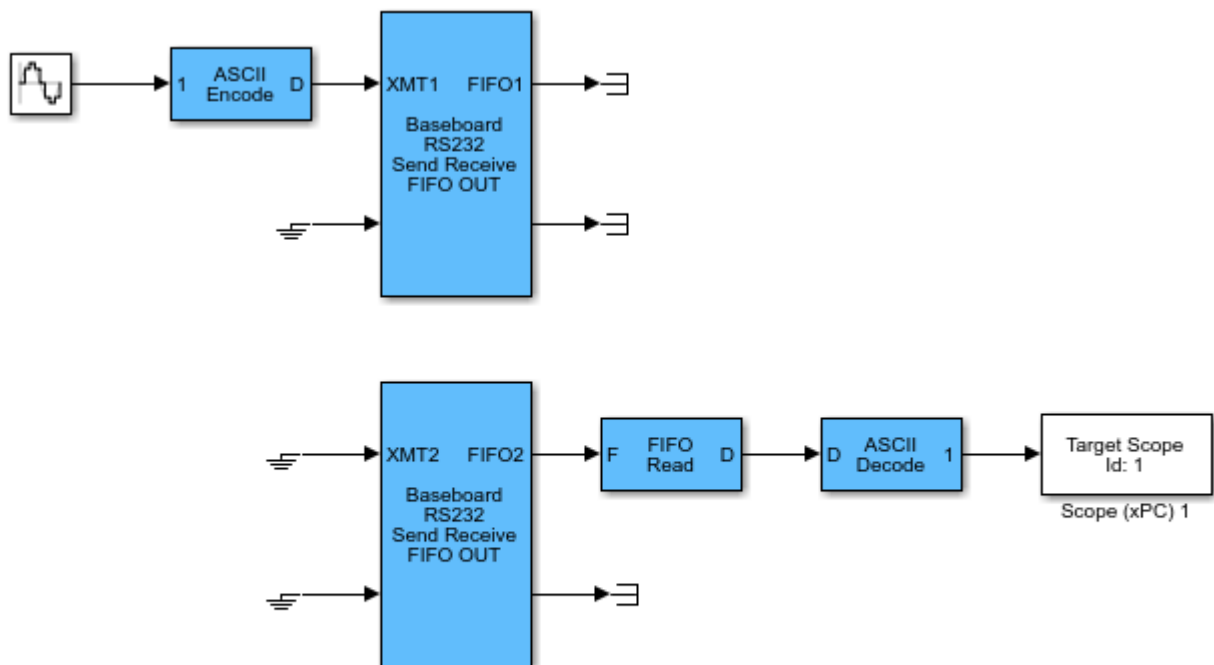
This example model shows how a single floating point number can be converted to ASCII and transmitted over a serial link. The sending serial port and receiving serial port can be in the same system or in different systems.

To test this model:

- 1 The target computer must have two COM ports.
- 2 Connect COM1 to COM2 with a null modem cable.

This example is configured to use baseboard serial ports (COM1 and COM2). You can also use COM3 and COM4 by changing the board setup in the Baseboard blocks. Other serial blocks could be used in place of the Baseboard blocks. For instance, a single Quatech® 4-port block could be used whereby you send on port 1 and receive on port 2.

`open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcserialbaseboardsi`



Copyright 2004-2012 The MathWorks, Inc.

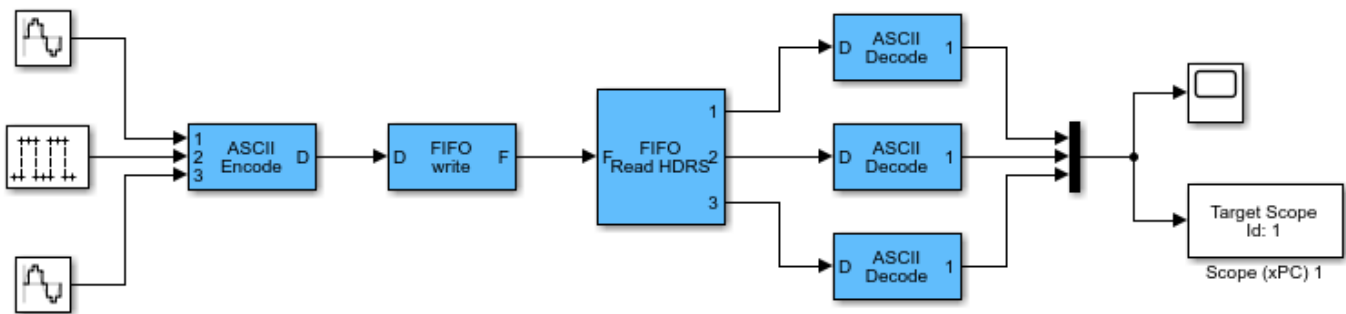
ASCII Encoding/Decoding Loopback Test

This model shows how to send ASCII data over a serial link.

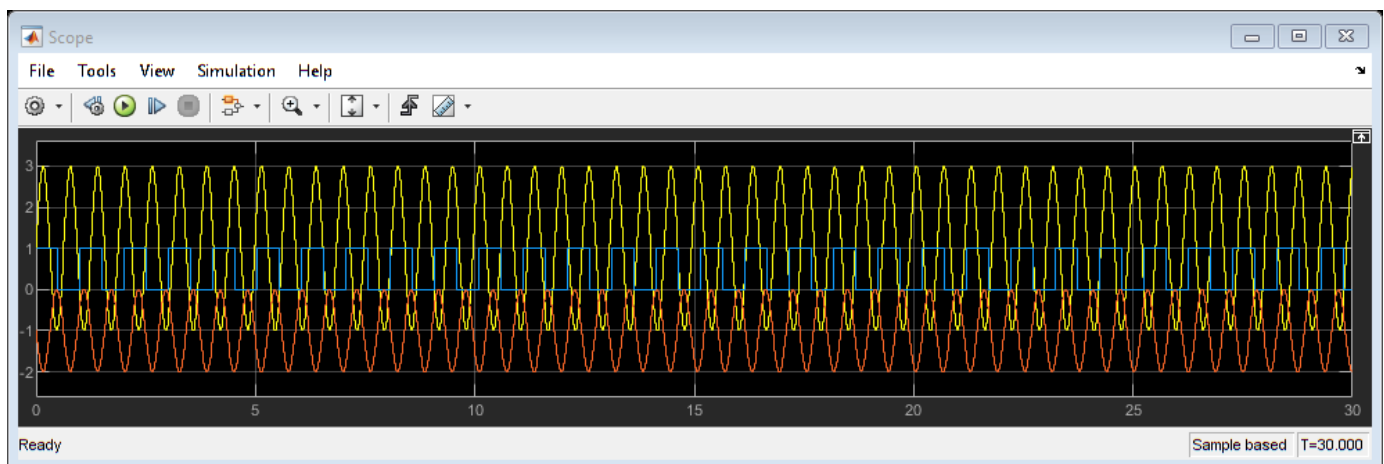
The ASCII Encode block generates a message with three different sub messages along with some extraneous 'junk' to show how the FIFO Read HDRS block can remain synchronized to the valid byte stream even in the presence of transmission errors.

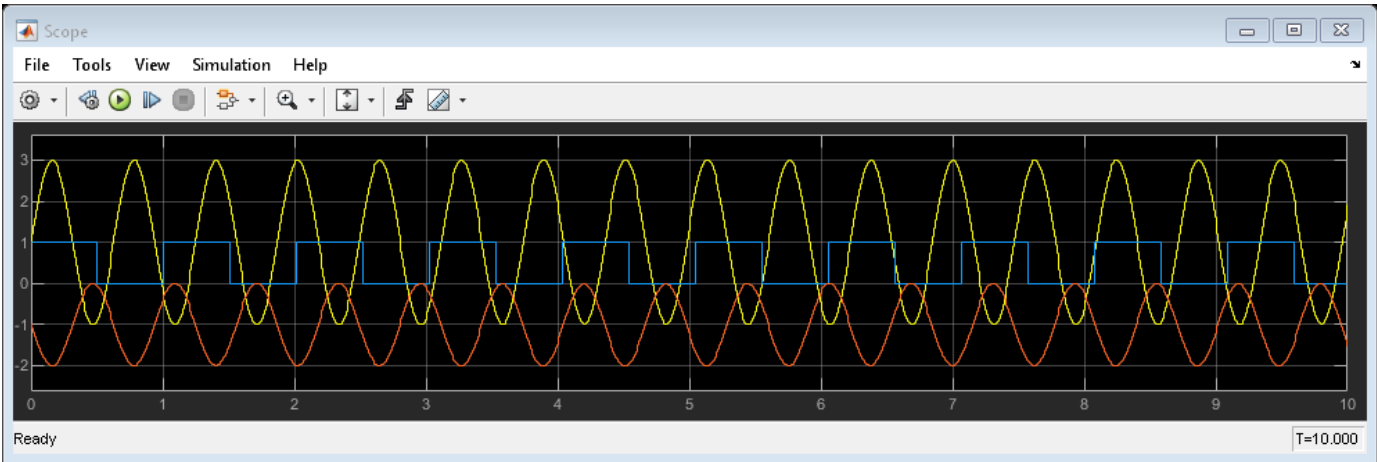
The FIFO Read HDRS block can handle an arbitrary number of headers; just add them as strings to the cell array in the block parameters dialog box. The messages must share the same termination string. In this example, it is a carriage return followed by a line feed: "\r\n".

```
open_system(fullfile(matlabroot,'toolbox','rtw','targets','xpc','xpcdemos','xpcserialasciitest')
set_param('xpcserialasciitest','StopTime','30');
sim('xpcserialasciitest')
```



Copyright 2004-2012 The MathWorks, Inc.





ASCII Encoding/Decoding Loopback Test (With Baseboard Blocks)

This example model shows how to send ASCII data over a serial link.

The ASCII Encode block generates a message with three different sub messages along with some extraneous 'junk' to show how the FIFO Read HDRS block can remain synchronized to the valid byte stream even in the presence of transmission errors.

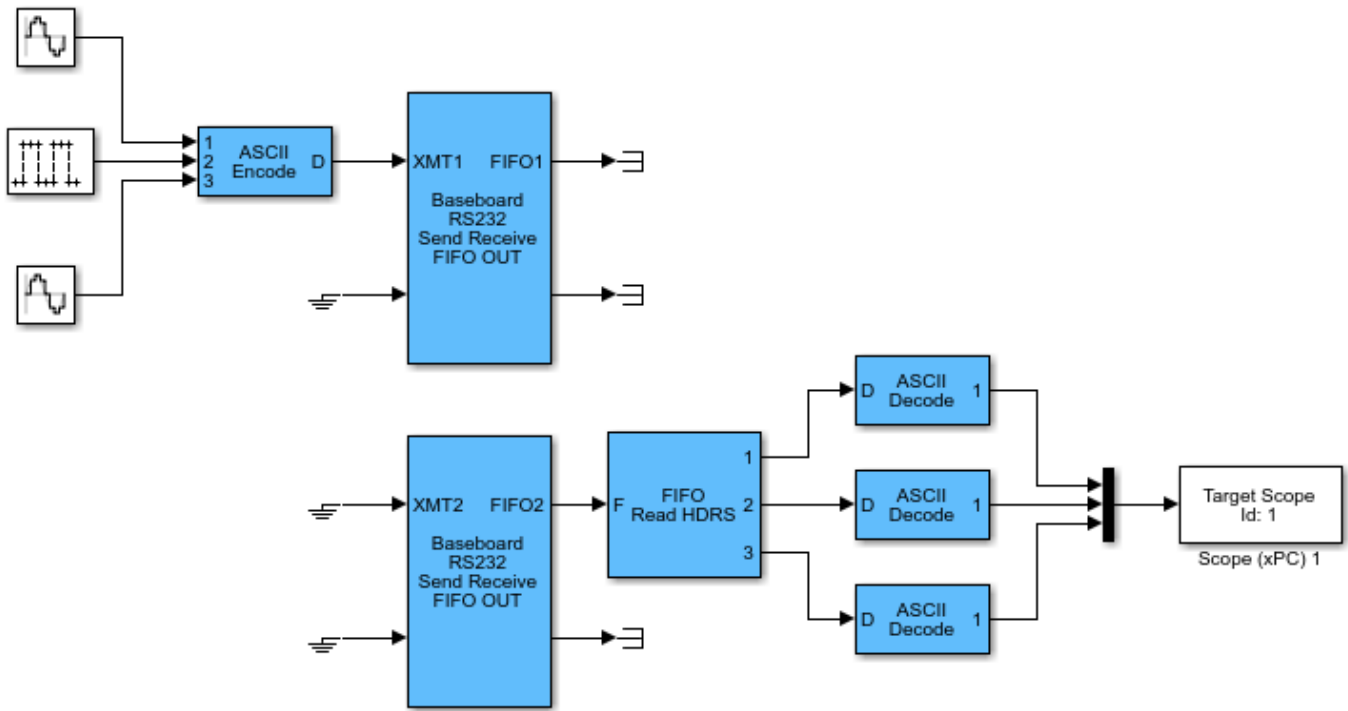
The FIFO Read HDRS block can handle an arbitrary number of headers; just add them as strings to the cell array in the block parameters dialog box. The messages must share the same termination string. In this example, it is a carriage return followed by a line feed: "\r\n".

To test this model:

- 1 The target computer must have two COM ports.
- 2 Connect COM1 to COM2 with a null modem cable.

This example is configured to use baseboard serial ports (COM1 and COM2). You can also use COM3 and COM4 by changing the board setup in the Baseboard blocks. Other serial blocks could be used in place of the Baseboard blocks. For instance, a single Quatech® 4-port block could be used whereby you send on port 1 and receive on port 2.

```
open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcserialbaseboardas
```

Copyright 2004-2012 The MathWorks, Inc.

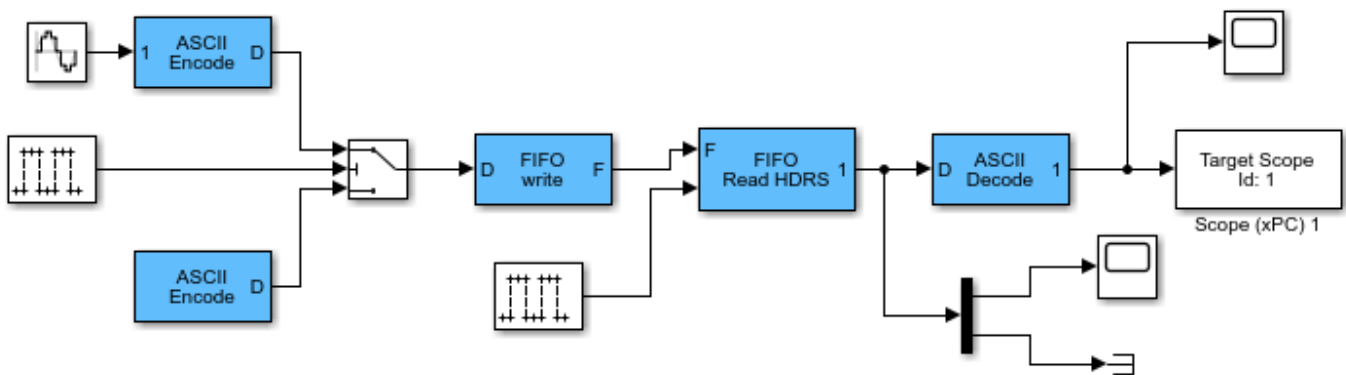
ASCII Encoding/Decoding Resync Loopback Test

This example model shows the ability of the FIFO Read HDRS block to resynchronize after being repeatedly disabled as well as the ability to resolve errors such as when a message is only partially complete at the time the read is attempted.

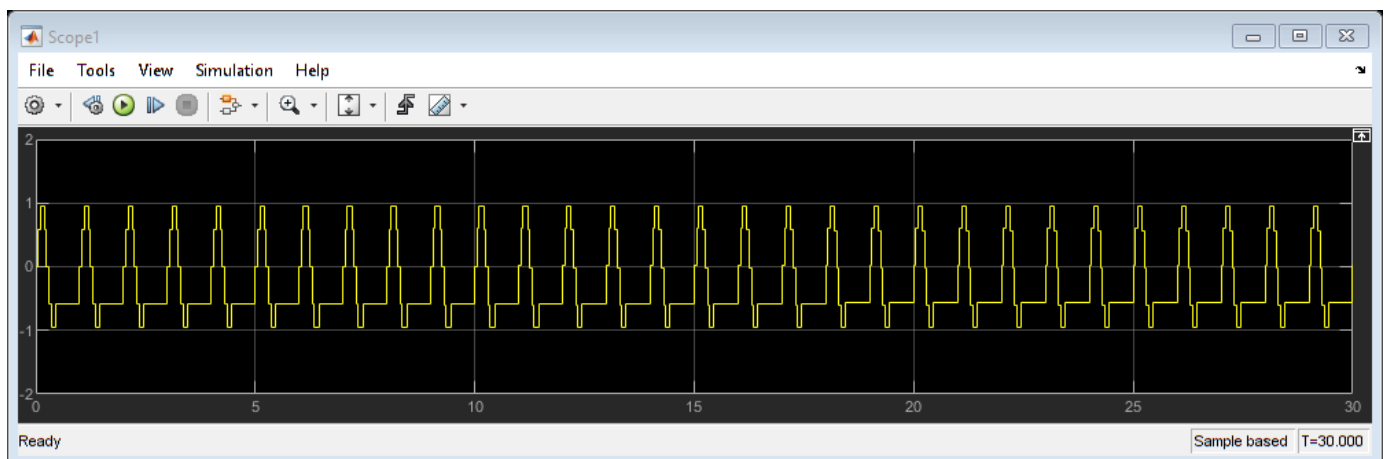
The Switch block alternates between the first and last parts of the message on successive sample times. This mimics a worst case scenario where the model updates before the message construction is complete. As a result, sometimes only part of the message is received. The second pulse generator alternately enables and disables the FIFO Read HDRS block.

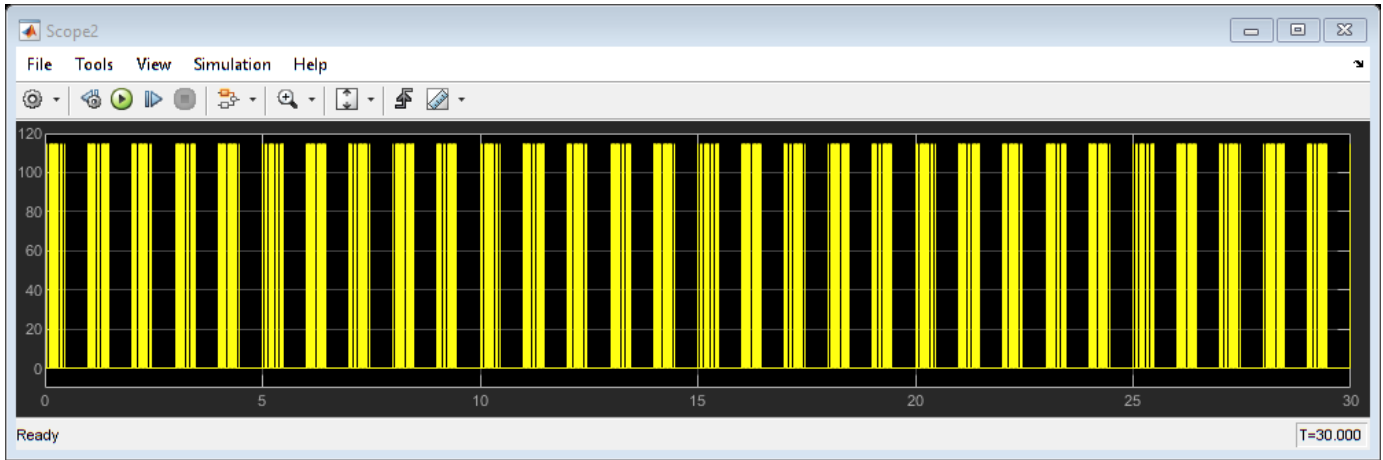
Scope 1 graphs the decoded sine wave data received at each time step. When the Pulse Generator1 block outputs a 0, the count from the FIFO Read HDRS block is 0. When it outputs a 1, the read catches up by throwing away extra data and returns the last complete value found in the FIFO. Scope 2 indicates when new data is present.

```
open_system(fullfile(matlabroot,'toolbox','rtw','targets','xpc','xpcdemos','xpcserialasciisplit')
set_param('xpcserialasciisplit','StopTime','30');
sim('xpcserialasciisplit')
```



Copyright 2004-2012 The MathWorks, Inc.





ASCII Encoding/Decoding Resync Loopback Test (With Baseboard Blocks)

This model shows the ability of the FIFO Read HDRS block to resynchronize after being repeatedly disabled as well as the ability to resolve errors such as when a message is only partially complete at the time the read is attempted.

The Switch block alternates between the first and last parts of the message on successive sample times. This mimics a worst case scenario where the model updates before the message construction is complete. As a result, sometimes only part of the message is received. The second pulse generator alternately enables and disables the FIFO Read HDRS block.

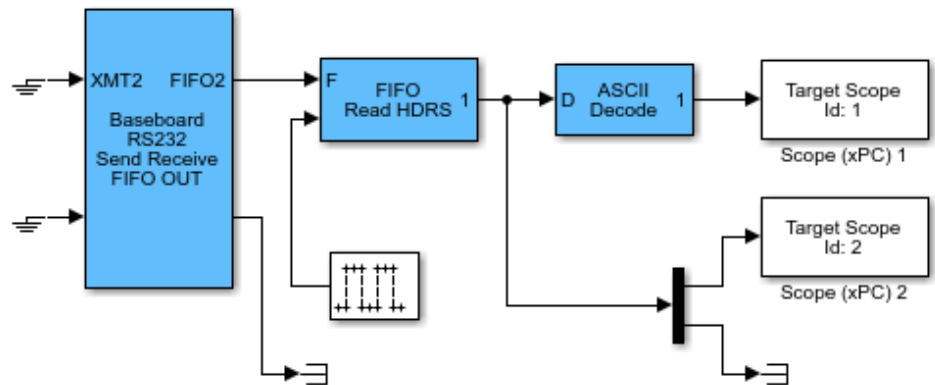
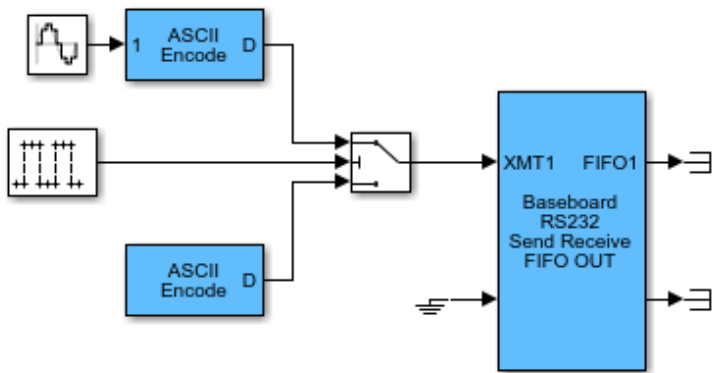
Scope 1 graphs the decoded sine wave data received at each time step. When the Pulse Generator1 block outputs a 0, the count from the FIFO Read HDRS block is 0. When it outputs a 1, the read catches up by throwing away extra data and returns the last complete value found in the FIFO. Scope 2 indicates when new data is present.

To test this model:

- 1 The target computer must have two COM ports.
- 2 Connect COM1 to COM2 with a null modem cable.

This example is configured to use baseboard serial ports (COM1 and COM2). You can also use COM3 and COM4 by changing the board setup in the Baseboard blocks. Other serial blocks could be used in place of the Baseboard blocks. For instance, a single Quatech® 4-port block could be used whereby you send on port 1 and receive on port 2.

```
open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcserialbaseboardas
```



Copyright 2004-2012 The MathWorks, Inc.

Binary Encoding/Decoding Loopback Test

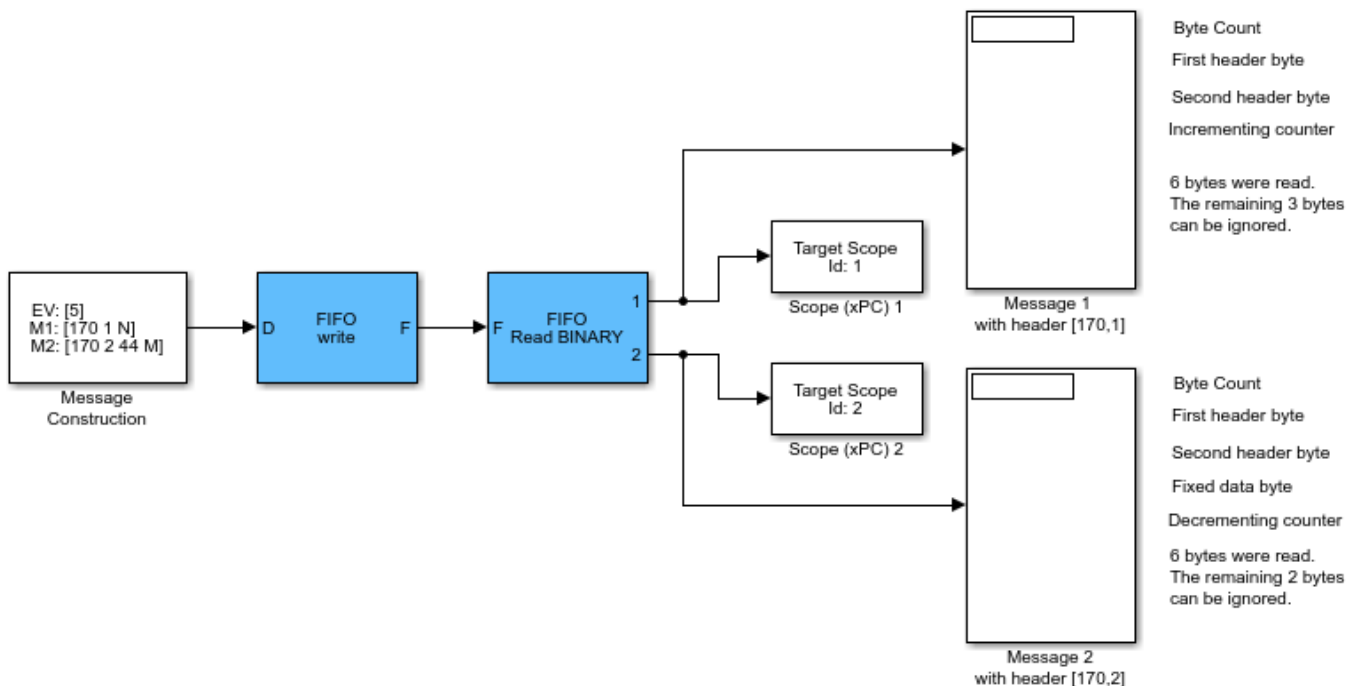
This model shows how to send Binary data over a serial link.

The transmitted data are: [8, 5, 170, 1, N, 170, 2, 44, M]. This byte stream contains two "messages" along with other elements as defined below.

- The first byte, 8, is a count of the remaining number of bytes in the stream.
- The second byte, 5, is an extraneous value (EV).
- [170, 1, N] is message 1 (M1).
- [170, 2, 44, M] is message 2 (M2).
- N and M are numbers between 0 and 255 that are incrementing and decrementing, respectively.

Even though the data stream includes extraneous bytes (5 in this case), the FIFO Read BINARY block can handle and ignore this extra information. Scope 1 displays the received message 1 data. Scope 2 displays the received message 2 data.

```
open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcserialbinarytest')
```



Copyright 2004-2012 The MathWorks, Inc.

Binary Encoding/Decoding Loopback Test (With Baseboard Blocks)

This model shows how to send Binary data over a serial link.

The transmitted data are: [8, 5, 170, 1, N, 170, 2, 44, M]. This byte stream contains two "messages" along with other elements as defined below.

- The first byte, 8, is a count of the remaining number of bytes in the stream.
- The second byte, 5, is an extraneous value (EV).
- [170, 1, N] is message 1 (M1).
- [170, 2, 44, M] is message 2 (M2).
- N and M are numbers between 0 and 255 that are incrementing and decrementing, respectively.

Notice that when the data contains extraneous bytes (5 in this case) the FIFO Read BINARY block can handle and ignore this extra information.

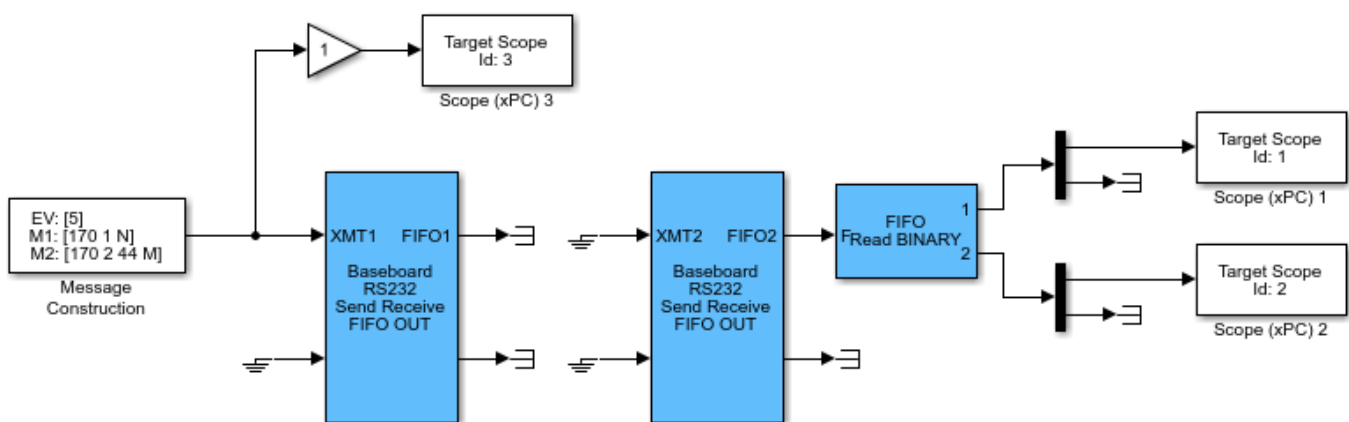
Scope 1 displays the received message 1 data. Scope 2 displays the received message 2 data. Scope 3 shows the transmitted byte stream. The gain block on the signal to Scope 3 makes the elements of the vector non-virtual so the scope can see them.

To test this model:

- 1 The target computer must have two COM ports.
- 2 Connect COM1 to COM2 with a null modem cable.

This example is configured to use baseboard serial ports (COM1 and COM2). You can also use COM3 and COM4 by changing the board setup in the Baseboard blocks. Other serial blocks could be used in place of the Baseboard blocks. For instance, a single Quatech® 4-port block could be used whereby you send on port 1 and receive on port 2.

```
open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcserialbaseboardbi
```



Copyright 2004-2012 The MathWorks, Inc.

Binary Encoding/Decoding Resync Loopback Test

This model shows the ability of the FIFO Read BINARY block to handle messages that are interrupted and only partially complete. This is a 'worst case' example where every message is interrupted.

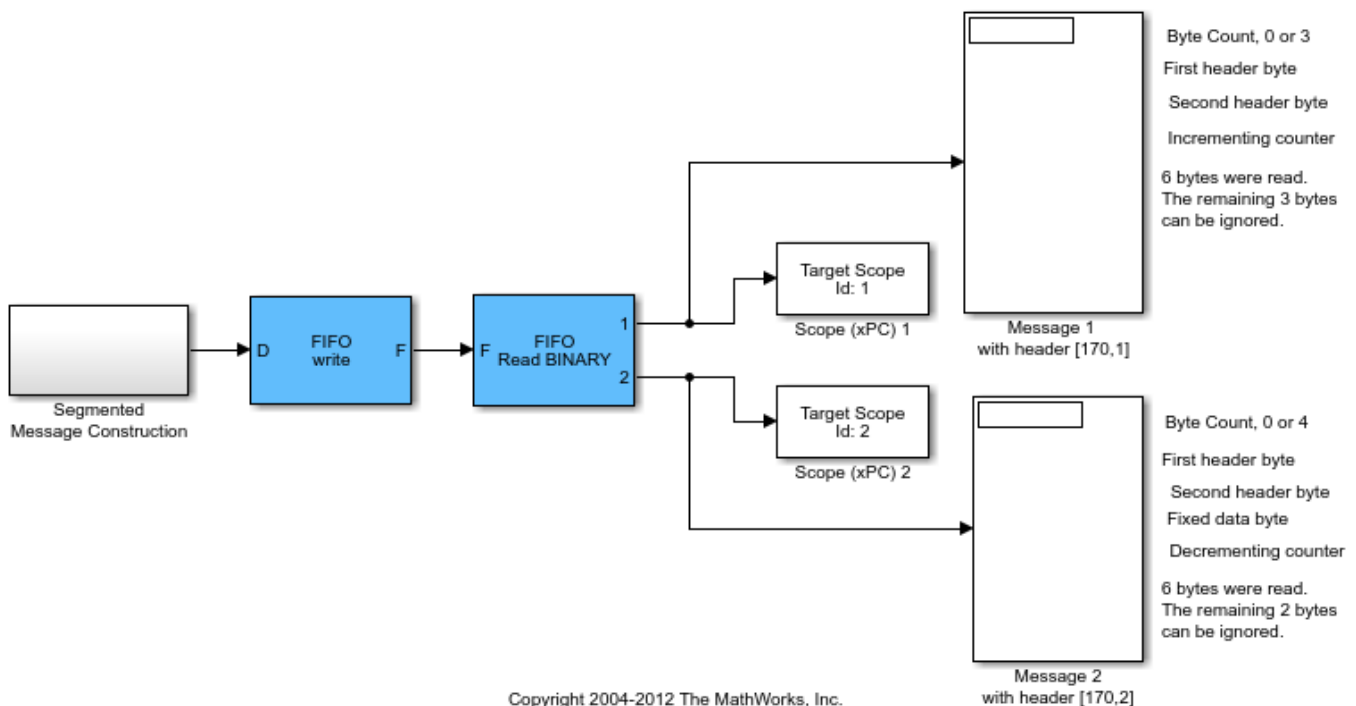
The Segmented Message Constructor subsystem contains blocks that prepare and send only parts of messages on each time step.

On the receive side, the FIFO read BINARY block is looking for two different two-character headers. If it finds [170,1] it outputs [3,170,1,N] on port 1. If it finds [170,2], it outputs [4,170,2,44,M] to port 2. N and M are numbers between 0 and 255 that are incrementing and decrementing, respectively.

If a message header is not found in the FIFO on a given time step, then that port will output 0. The outputs are padded to the maximum vector size specified in the FIFO Read BINARY block. In this example output vectors are 6 in width. The count in the first element tells how many elements are significant.

Scope 1 displays the received message 1 data. Scope 2 displays the received message 2 data.

```
open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcserialbinarysplit
```



Binary Encoding/Decoding Resync Loopback Test (With Baseboard Blocks)

This model shows the ability of the FIFO Read BINARY block to handle messages that are interrupted and only partially complete. This is a 'worst case' example where every message is interrupted.

The Segmented Message Constructor subsystem contains blocks that prepare and send only parts of messages on each time step.

On the receive side, the FIFO read BINARY block is looking for two different two-character headers. If it finds [170, 1] it outputs [3, 170, 1, N] on port 1. If it finds [170, 2], it outputs [4, 170, 2, 44, M] to port 2. N and M are numbers between 0 and 255 that are incrementing and decrementing, respectively.

If a message header is not found in the FIFO on a given time step, then that port will output 0. The outputs are padded to the maximum vector size specified in the FIFO Read BINARY block. In this example output vectors are 1024 in width. The count in the first element tells how many elements are significant. The Demux blocks discard the uninteresting parts of the signal.

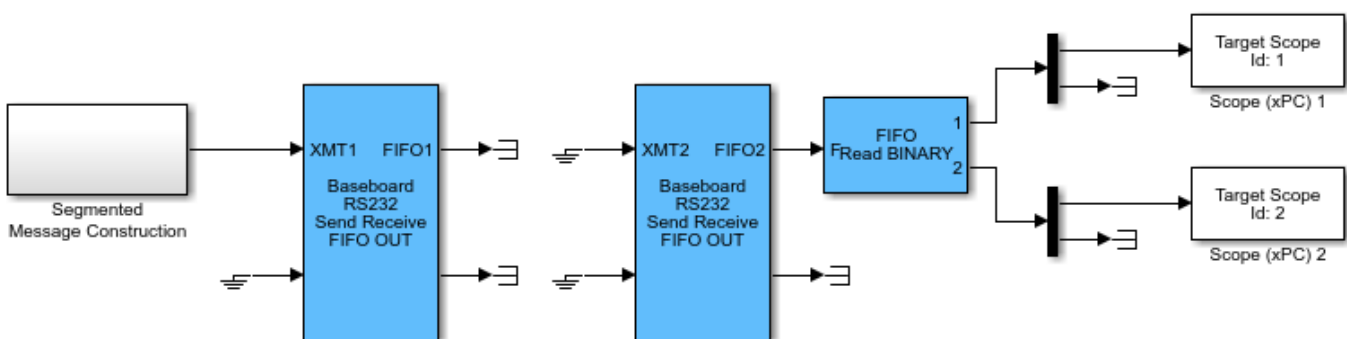
Scope 1 displays the received message 1 data. Scope 2 displays the received message 2 data.

To test this model:

- 1 The target computer must have two COM ports.
- 2 Connect COM1 to COM2 with a null modem cable.

This example is configured to use baseboard serial ports (COM1 and COM2). You can also use COM3 and COM4 by changing the board setup in the Baseboard blocks. Other serial blocks could be used in place of the Baseboard blocks. For instance, a single Quatech® 4-port block could be used whereby you send on port 1 and receive on port 2.

```
open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'xpcserialbaseboardbi
```



Copyright 2004-2014 The MathWorks, Inc.

Read CPU Temperature on Simulink® Real-Time™

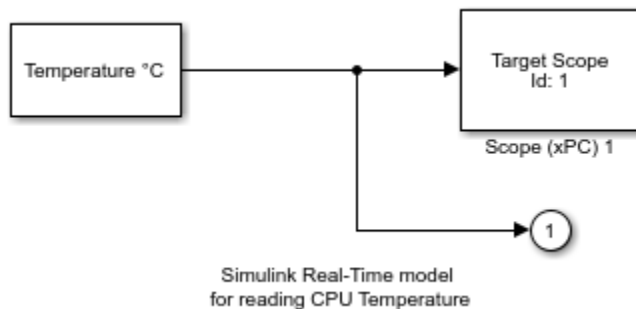
This example shows how to read CPU temperature in degrees Celsius (°C).

Requirements:

- 1 Boot the target computer with the Simulink Real-Time real-time kernel.
- 2 At the MATLAB® command prompt, type `dslrtCPUtemperatureDemo` to download and run the model on the target computer.

Select and Open the Model

```
mdl='dslrtCPUtemperature';
open_system(fullfile(matlabroot,'toolbox','rtw','targets','xpc','xpcdemos',mdl));
```



Build, Download, and Run the Model

```
slrtpingtarget;
set_param(mdl,'RTWVerbose','off');
rtwbuild(mdl);
tg = slrt('TargetPC1');
load(tg,mdl);
start(tg);
pause(20);
stop(tg);
```

```
### Starting Simulink Real-Time build procedure for model: dslrtCPUtemperature
Warning: This model contains blocks that do not handle sample time
changes at runtime. To avoid incorrect results, only change
the sample time in the original model, then rebuild the model.
### Successful completion of build procedure for model: dslrtCPUtemperature
### Created MLDATX ..\dslrtCPUtemperature.mldatx
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
```

Close the Model

```
close_system(mdl,0);
```

Target to Target communication using TCP

This example shows how to use TCP blocks to send data between two target computers. This example also describes the effects of a server and a client running at different sample times.

The server model TargetToTargetTCPServer runs on TargetPC1 with sample time .02 second. This model contains a sine wave source. The client model TargetToTargetTCPClient runs on TargetPC2 with sample time .01 second. This model contains a sawtooth wave source. Both models send and receive signal data packets.

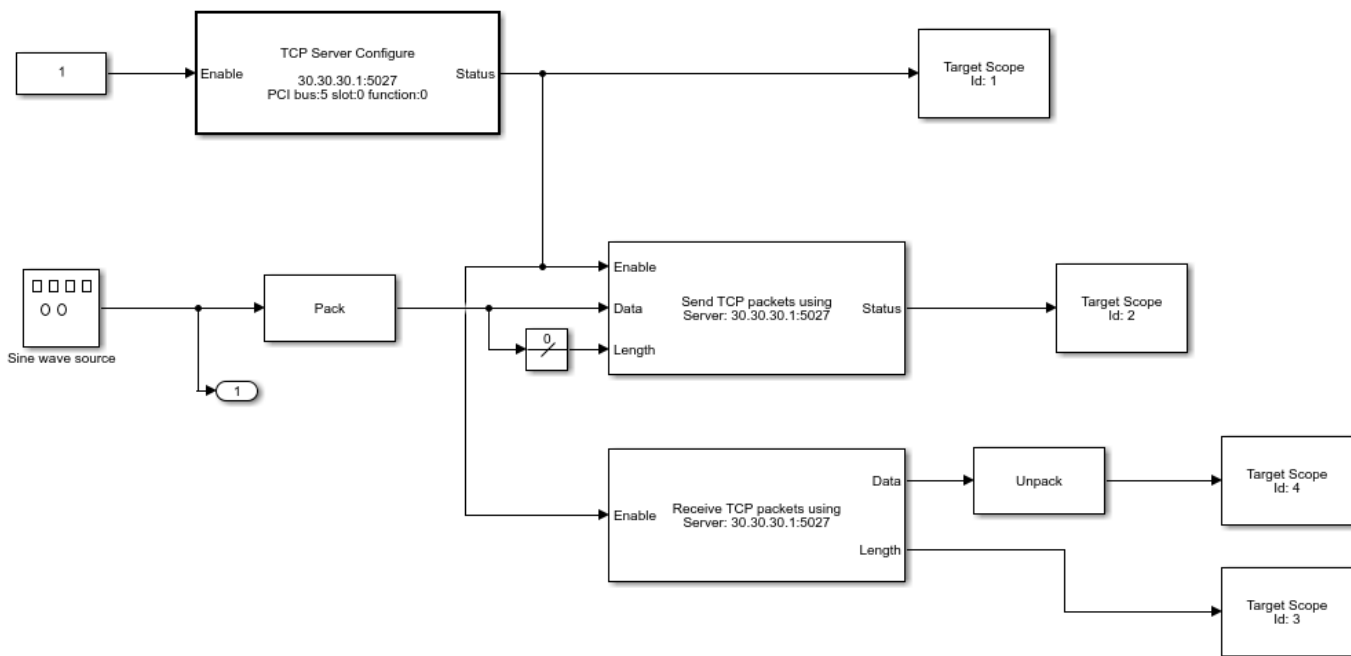
Click here to open this example: TargetToTargetTCP.

Open, build, and download the server model

Click here to open model 1: TargetToTargetTCPServer.

Open the model.

```
mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if ~any(strcmp('TargetToTargetTCPServer', systems))
    mdlOpen = 1;
    open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'TargetToTargetTCPServer'));
end
```



Copyright 2016 The MathWorks, Inc.

Build the model and download to the target computer.

- Configure server TCP settings.
- Build and download application.

```

set_param('TargetToTargetTCPServer/TCP Server Configure','PCIBus','5');
set_param('TargetToTargetTCPServer/TCP Server Configure','PCISlot','0');
set_param('TargetToTargetTCPServer/TCP Server Configure','PCIFunction','0');
rtwbuild('TargetToTargetTCPServer');
tgl = slrt('TargetPC1');
load(tgl,'TargetToTargetTCPServer');

```

```

### Starting Simulink Real-Time build procedure for model: TargetToTargetTCPServer
### Generated code for 'TargetToTargetTCPServer' is up to date because no structural, parameter
### Successful completion of build procedure for model: TargetToTargetTCPServer
### Created MLDATX ..\TargetToTargetTCPServer.mldatx
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

```

- Close the model.

```

if (mdlOpen)
    bdclose('TargetToTargetTCPServer');
end

```

Open, build, and download the client model

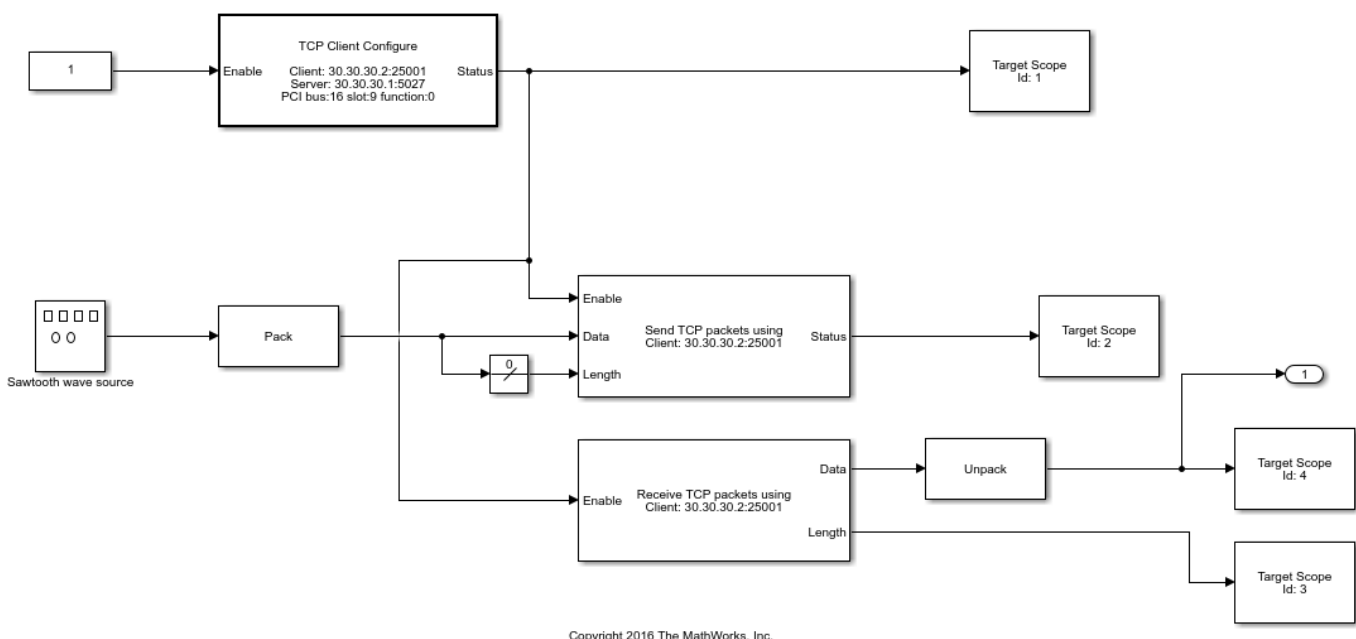
Click [here](#) to open model 2: TargetToTargetTCPClient.

Open the model.

```

mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if ~any(strcmp('TargetToTargetTCPClient', systems))
    mdlOpen = 1;
    open_system(fullfile(matlabroot,'toolbox','rtw','targets','xpc','xpcdemos','TargetToTargetTCPClient'));
end

```



Build the model and download to the target computer.

- Configure client TCP settings.
- Build and download application.

```
set_param('TargetToTargetTCPClient/TCP Client Configure','PCIBus','8');
set_param('TargetToTargetTCPClient/TCP Client Configure','PCISlot','10');
set_param('TargetToTargetTCPClient/TCP Client Configure','PCIFunction','0');
rtwbuild('TargetToTargetTCPClient');
tg2 = slrt('TargetPC2');
load(tg2,'TargetToTargetTCPClient');
```

```
### Starting Simulink Real-Time build procedure for model: TargetToTargetTCPClient
### Generated code for 'TargetToTargetTCPClient' is up to date because no structural, parameter
### Successful completion of build procedure for model: TargetToTargetTCPClient
### Created MLDATX ..\TargetToTargetTCPClient.mldatx
### Looking for target: TargetPC2
### Download model onto target: TargetPC2
```

- Close the model.

```
if (mdlOpen)
    bdclose('TargetToTargetTCPClient');
end
```

Run both models

Using the Simulink Real-Time object variables `tg1` and `tg2`, start the models.

- Start the TargetPC1 model.
- Start the TargetPC2 model.

```
start(tg1);
start(tg2);
pause(4);
```

Stop both models

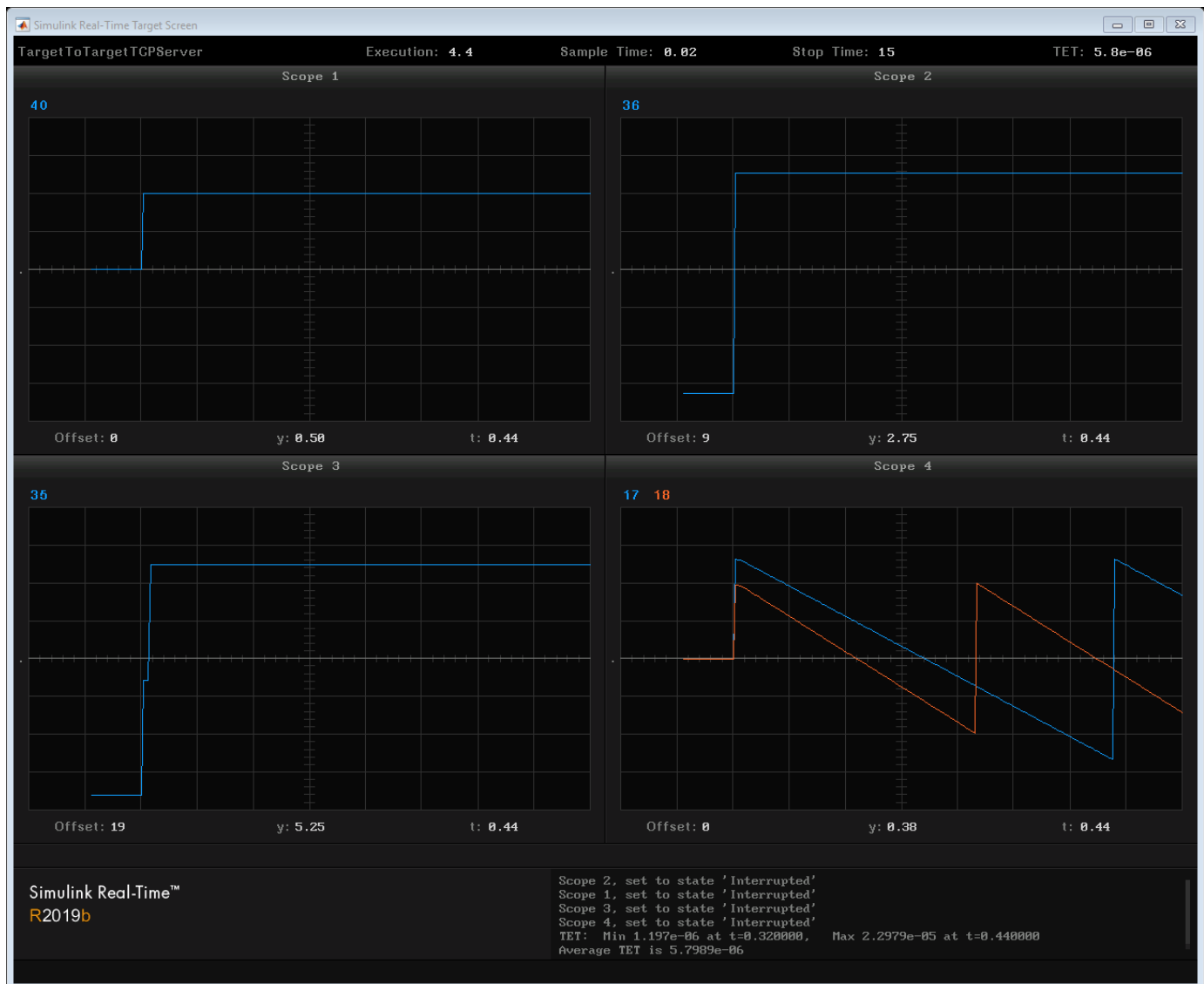
Using the Simulink Real-Time object variables `tg1` and `tg2`, stop the models.

- Stop the TargetPC1 model.
- Stop the TargetPC2 model.

```
stop(tg1);
stop(tg2);
```

Generate Server (TargetPC1) Plot

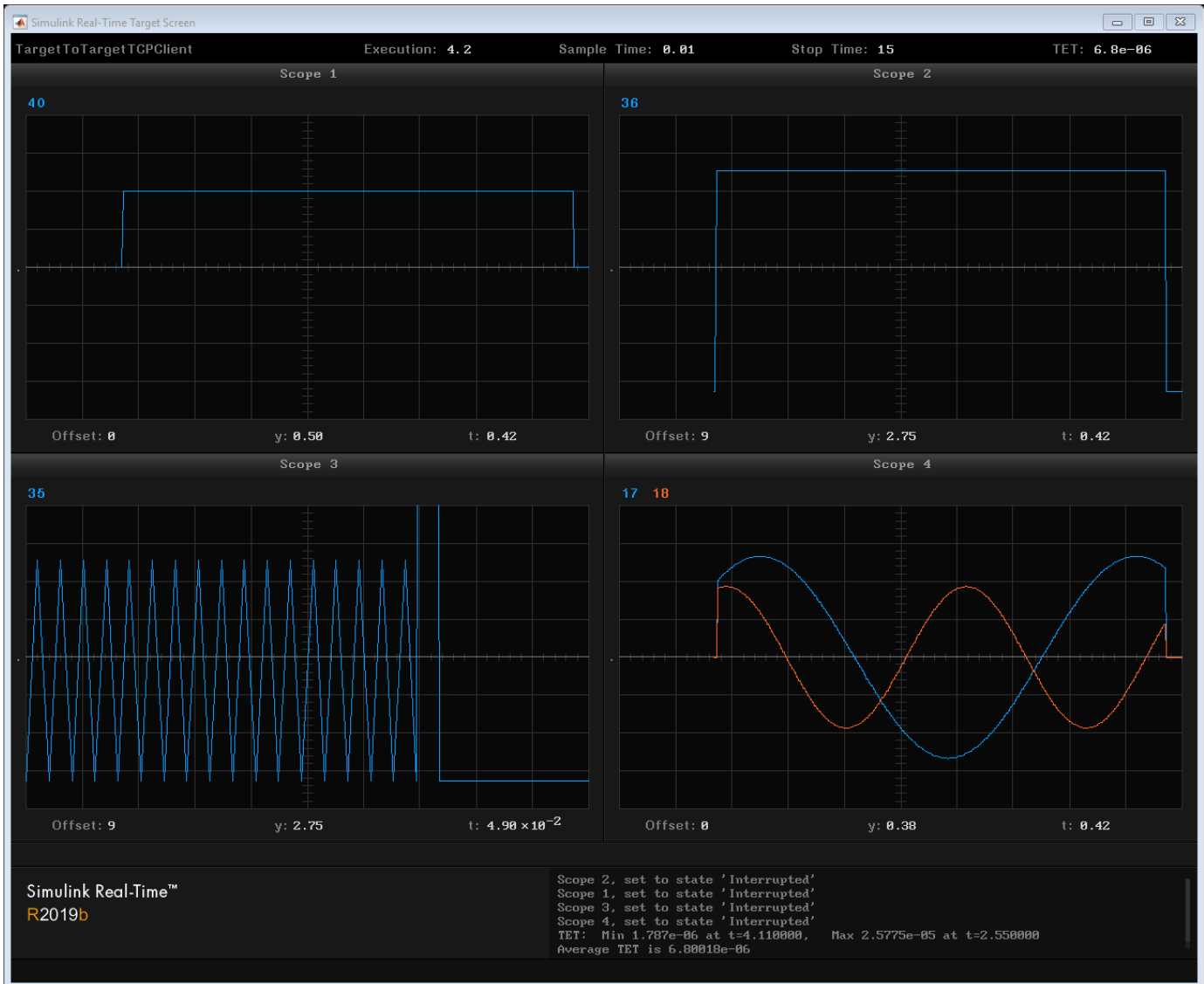
```
tg1.viewTargetScreen();
```



- Scope 1 is the 'Status' output of the 'TCP Server Configure' block. The value goes to 1 when a connection is established.
- Scope 2 is the 'Status' output of the 'TCP Send' block. After the connection is established the value goes to 16 to indicate that 16 bytes are sent every sample time.
- Scope 3 is the 'Length' output of the 'TCP Receive' block in the server model. The value in the scope goes from 0 to 16 and then stays at 32. This is due to the fact that the server is running half as fast as the client. As a result for every sample time on the server, the client sends it 2 packets of 16 bytes each. At every sample time, the server blocks have 32 bytes available to read. In this model on 16 bytes are read out so the other 16 bytes are missed out by the model. This will not be a TCP error as the packets are effectively transmitted and received - but the model is configured in a way that causes the packets to be missed.
- Scope 4 plots the received data.

Generate Client (TargetPC2) Plot

```
tg2.viewTargetScreen();
```



- Scope 1 is the 'Status' output of the 'TCP Client Configure' block. The value goes to 1 when a connection is established.
- Scope 2 is the 'Status' output of the 'TCP Send' block. After the connection is established the value goes to 16 to indicate that 16 bytes are sent every sample time.
- Scope 3 is the 'Length' output of the 'TCP Receive' block in the client model. This switches between 0 and 16 since the client runs twice as fast as the server, so for every alternate sample it receives no data.
- Scope 4 plots the received data.

Effect of the difference in sample times

The following plot compares the sine wave sent by the server and received by the client.

Server sine wave

```
tx = tg1.TimeLog;  
x = tg1.OutputLog;
```

Client sine wave (received)

```
ty = tg2.TimeLog;  
y = tg2.OutputLog;
```

Plot a short section

NOTE: This portion only works when you have a second target machine that supports TCP.

```
plot(tx(10:30), x(10:30), 'b--o', ty(10:60), y(10:60), 'g--o');  
legend({'Server (SampleTime: .02)', 'Client (SampleTime: .01)'})  
xlabel('Time (seconds)')  
ylabel('Amplitude')  
title('Compare Tx and Rx sine waveform')
```

- The client runs twice as fast as the server, so there are double the number of green points for the same time duration.
- Since the points in between have no new data sent, the previous value is held.
- The delay between the sine waves depends on various factors like the lack of synchronization between the targets and network caused delays.

Target to Host Transmission using UDP

This example shows how to use UDP blocks to send data from a target computer to a development computer. Signal data are sent by the transmit model running on the target computer, TargetToHostUDPTx, to the receiver model running in Simulink® on the development computer, TargetToHostUDPRx.

Note: When considering UDP as a protocol for communicating data to or from the Simulink Real-Time™ environment it is important to be aware of the following:

- The Simulink model on the development computer is running as fast as it can and is therefore not synchronized to a real-time clock.
- UDP is a *connectionless* protocol that does not check to confirm that packets were transmitted. Data packets can be lost or dropped.
- On the target computer, UDP blocks run in a background task that executes each time step after the real-time task completes. If the block cannot run or complete the background task before the next time step, data may not be communicated.
- UDP data packets are transmitted over the Ethernet link between the development and target computers and must therefore share bandwidth with the Ethernet link.
- For more information on using UDP with Simulink Real-Time, see the documentation on UDP I/O support.

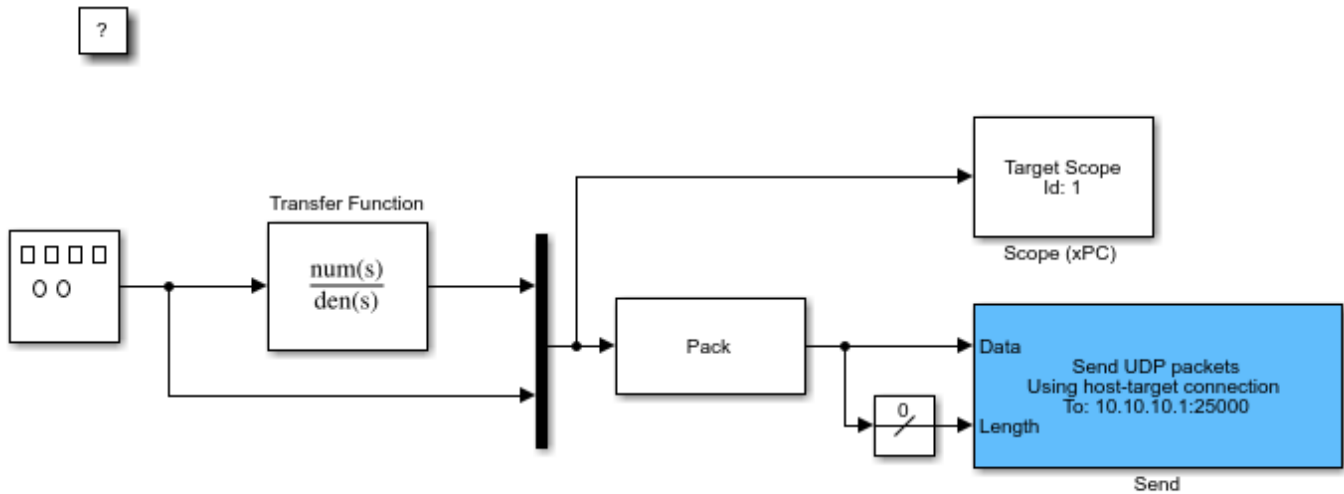
[Click here to open this example: TargetToHostUDP.](#)

Open, Build, and Download the Target Computer Model

[Click here to open the Tx model: TargetToHostUDPTx.](#) This model drives a first order transfer function with a square wave signal and sends the transfer function input and output signals to the development computer using UDP.

Open the model.

```
mdlOpened = 0;
systems = find_system('type', 'block_diagram');
if ~any(strcmp('TargetToHostUDPTx', systems))
    mdlOpened = 1;
    open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'TargetToHostUDPTx')
end
```



Copyright 2016 The MathWorks, Inc.

Build the model and download to the target computer.

- Configure for a non-Verbose build.
- Configure UDP Send block for development computer address.
- Mark the Signal Generator and Transfer Function output for data logging.
- Build and download application.
- Open the Simulation Data Inspector.

This code shows how to mark signals programmatically for data logging. You can also mark signals for data logging in the Simulink Editor. You can view the logged data in in the Simulation Data Inspector.

```
set_param('TargetToHostUDPTx', 'RTWVerbose', 'off');
set_param('TargetToHostUDPTx/Send', 'toAddress', '10.10.10.128');
hSigGen = get_param('TargetToHostUDPTx/Signal Generator', 'PortHandles');
SigGen = hSigGen.Outport(1);
Simulink.sdi.markSignalForStreaming(SigGen, 'on');
hTranFun = get_param('TargetToHostUDPTx/Transfer Function', 'PortHandles');
TranFun = hTranFun.Outport(1);
Simulink.sdi.markSignalForStreaming(TranFun, 'on');
rtwbuild('TargetToHostUDPTx');
tg = slrt('TargetPC1');
load(tg, 'TargetToHostUDPTx');
Simulink.sdi.view;
```

```
### Starting Simulink Real-Time build procedure for model: TargetToHostUDPTx
Warning: This model contains blocks that do not handle sample time
changes at runtime. To avoid incorrect results, only change
the sample time in the original model, then rebuild the model.
### Successful completion of build procedure for model: TargetToHostUDPTx
### Created MLDATX ..\TargetToHostUDPTx.mldatx
```

```
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
```

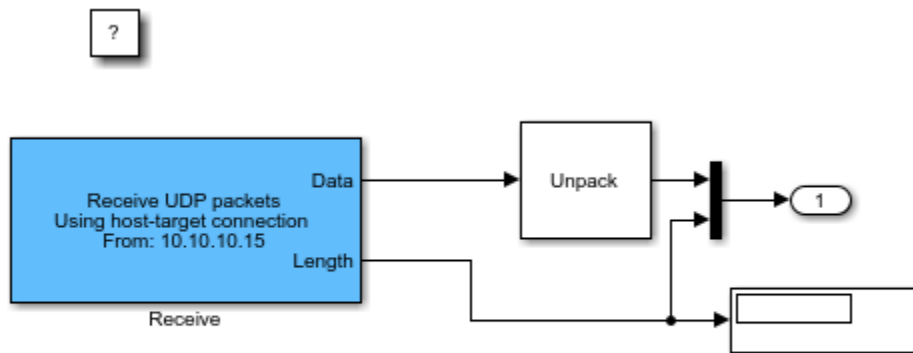
Close the model if we opened it.

```
if (mdlOpened)
    bdclose('TargetToHostUDPTx');
end
```

Open the development computer Model

Click [here](#) to open the Rx model: TargetToHostUDPRx. This model receives data sent by TargetToHostUDPTx.

```
mdlOpened = 0;
systems = find_system('type', 'block_diagram');
if ~any(strcmp('TargetToHostUDPRx', systems))
    mdlOpened = 1;
    open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'TargetToHostUDPRx'));
end
```



Copyright 2016 The MathWorks, Inc.

Before running this model:

- Configure UDP Send block for target computer address.
- Mark the Receive block Data and Length outputs for data logging.

```
set_param('TargetToHostUDPRx/Receive', 'fmAddress', '10.10.10.15');
hUdpReceiveData = get_param('TargetToHostUDPRx/Receive', 'PortHandles');
UdpReceiveData = hUdpReceiveData.Outputport(1);
Simulink.sdi.markSignalForStreaming(UdpReceiveData, 'on');
hUdpReceiveLength = get_param('TargetToHostUDPRx/Receive', 'PortHandles');
UdpReceiveLength = hUdpReceiveLength.Outputport(2);
Simulink.sdi.markSignalForStreaming(UdpReceiveLength, 'on');
```

Run Both Models

Start model on target computer followed by model on development computer.

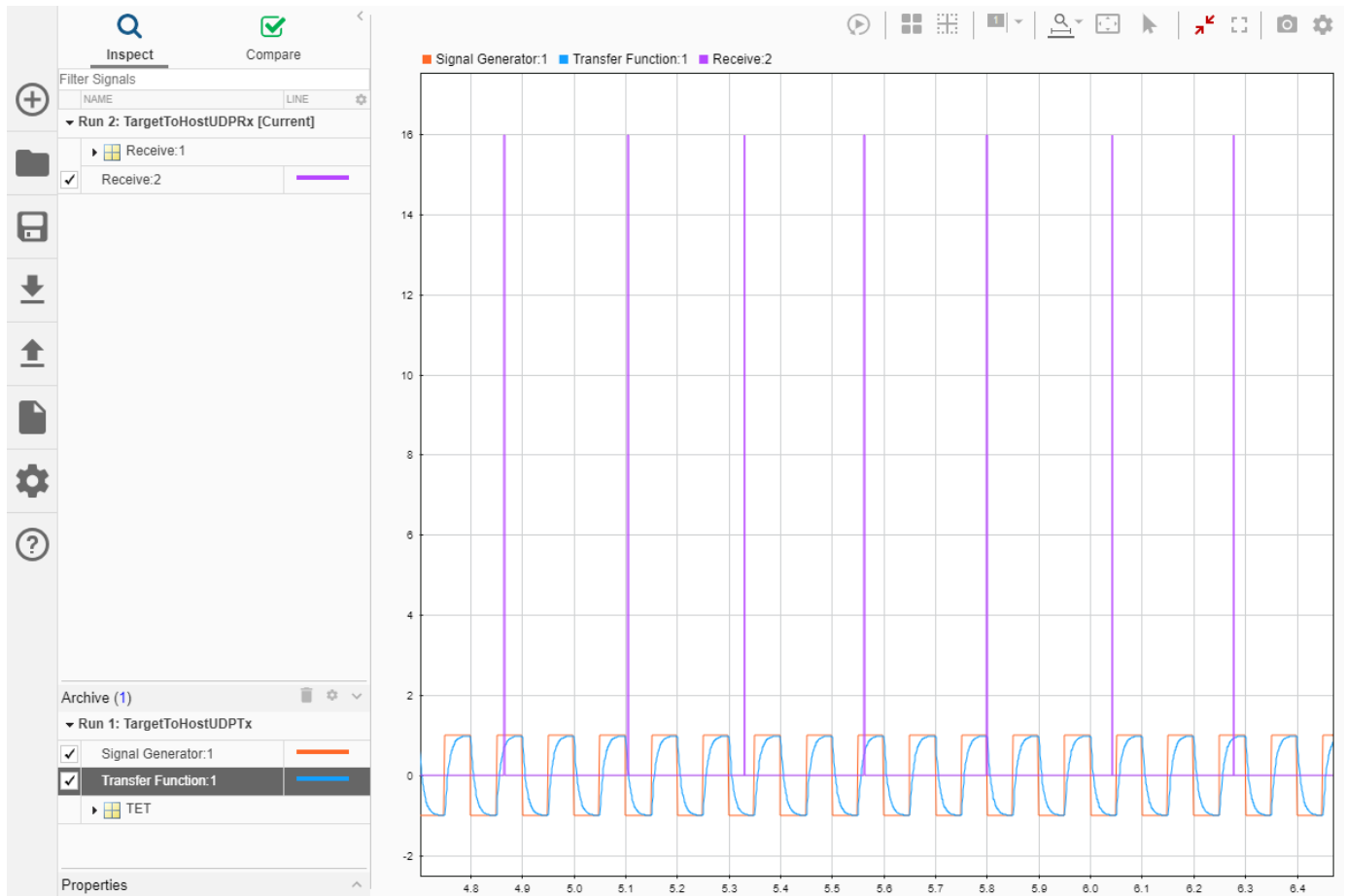
- Start the target computer Tx model.

- Wait for 1 sec.
- Start the development computer Rx model.
- Wait for 5 sec.
- Wait another 1 sec.
- Open the Simulation Data Inspector

```
start(tg);
pause(1);
set_param(bdroot, 'SimulationCommand', 'start');
pause(5);
while ~strcmpi(get_param(bdroot, 'SimulationStatus'), 'stopped')
    pause(1);
end
Simulink.sdi.view;
stop(tg);
```

Compare Signals in the Simulation Data Inspector

The signal that the development computer receives `Receive:2` does not look exactly like the signal that the target computer sent `Transfer Function:1`. On the development computer, the model does not run in real-time. The model on the development computer actually runs faster than real-time. Additionally, it does not run at constant intervals, and the number of steps processed per second varies depending on computer load. Therefore, the `Data` output values are sometimes held from the previous packet that the block received by the development computer model. You can use the second output `Receive:2` of the UDP Receive Binary block to detect the presence of a new packet. The development computer plot shows that whenever there is a new packet, the second output `Receive:2` goes to a non-zero value indicating the number of bytes received. When it is 0, the `Data` output remains the value from the previous packet that the block received.



Target to Target Transmission using UDP

This example shows how to use UDP blocks to send data between two target computers. The model TargetToTargetRealtimeUDP1 runs on TargetPC1. The model TargetToTargetRealtimeUDP2 runs on TargetPC2. Both models send and receive signal data packets.

Note: When considering UDP as a protocol for communicating data to/from the Simulink® Real-Time™ environment, it is important to be aware of the following:

- UDP is a *connectionless* protocol that does not check to confirm that packets were transmitted or received. Data packets can be lost or dropped.
- For more information on using UDP with Simulink Real-Time, see the documentation on UDP I/O support.

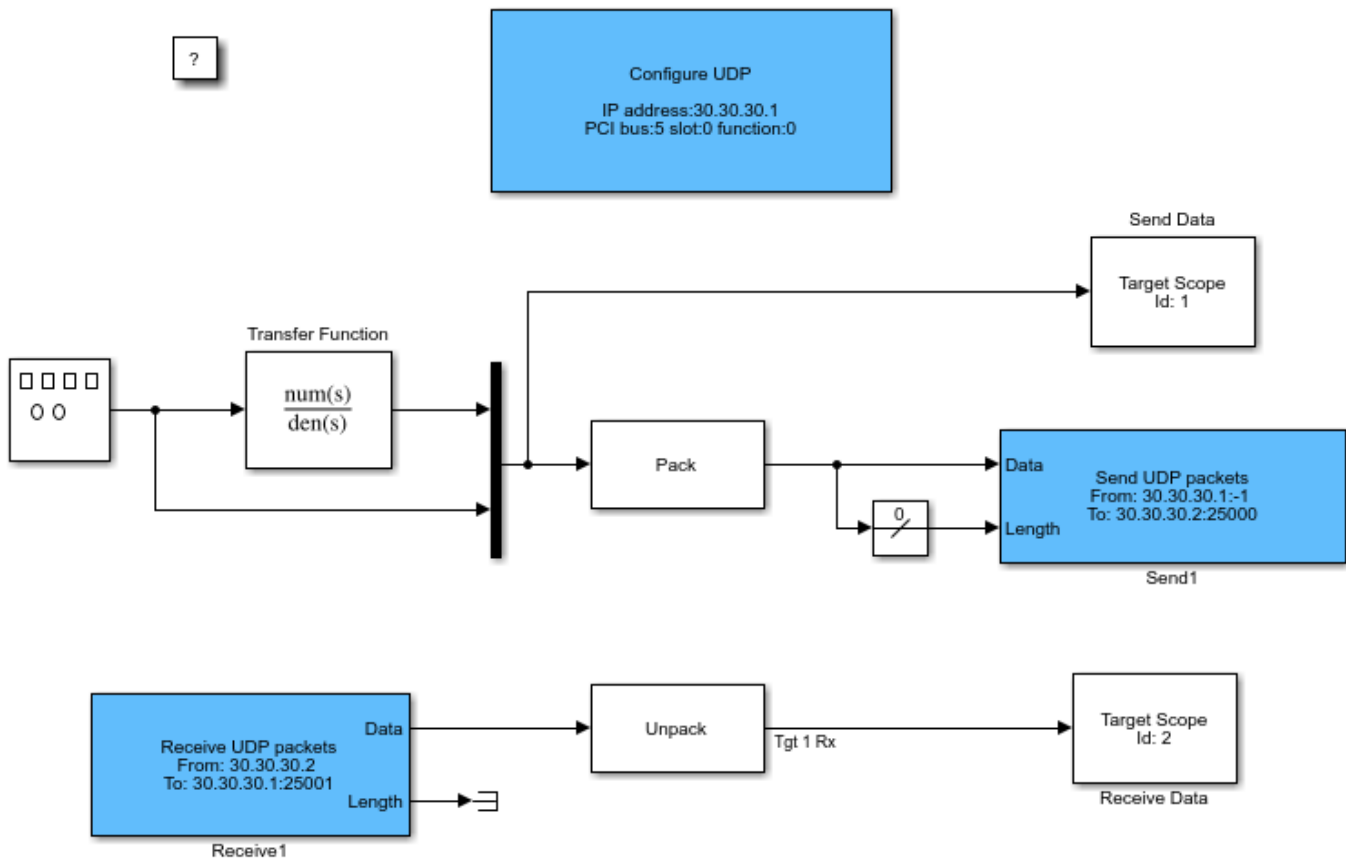
[Click here to open this example: TargetToTargetUDP.](#)

Open, Build, and Download the TargetPC1 Model

[Click here to open model 1: TargetToTargetRealtimeUDP1.](#)

Open the model.

```
mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if ~any(strcmp('TargetToTargetRealtimeUDP1', systems))
    mdlOpen = 1;
    open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'TargetToTargetReal
end
```



Copyright 2016 The MathWorks, Inc.

Build the model and download to the target computer.

- Configure for a non-Verbose build.
- Configure UDP parameters for TargetPC1.
- Build and download application.

```
set_param('TargetToTargetRealtimeUDP1','RTWVerbose','off');
set_param('TargetToTargetRealtimeUDP1/UDP Configure','PCIBus','5')
set_param('TargetToTargetRealtimeUDP1/UDP Configure','PCISlot','0')
set_param('TargetToTargetRealtimeUDP1/UDP Configure','PCIFunction','0')
rtwbuild('TargetToTargetRealtimeUDP1');
tg = slrt('TargetPC1');
load(tg,'TargetToTargetRealtimeUDP1');
```

```
### Starting Simulink Real-Time build procedure for model: TargetToTargetRealtimeUDP1
Warning: This model contains blocks that do not handle sample time
changes at runtime. To avoid incorrect results, only change
the sample time in the original model, then rebuild the model.
### Successful completion of build procedure for model: TargetToTargetRealtimeUDP1
### Created MLDATX ..\TargetToTargetRealtimeUDP1.mldatx
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
```

Close the model if we opened it.

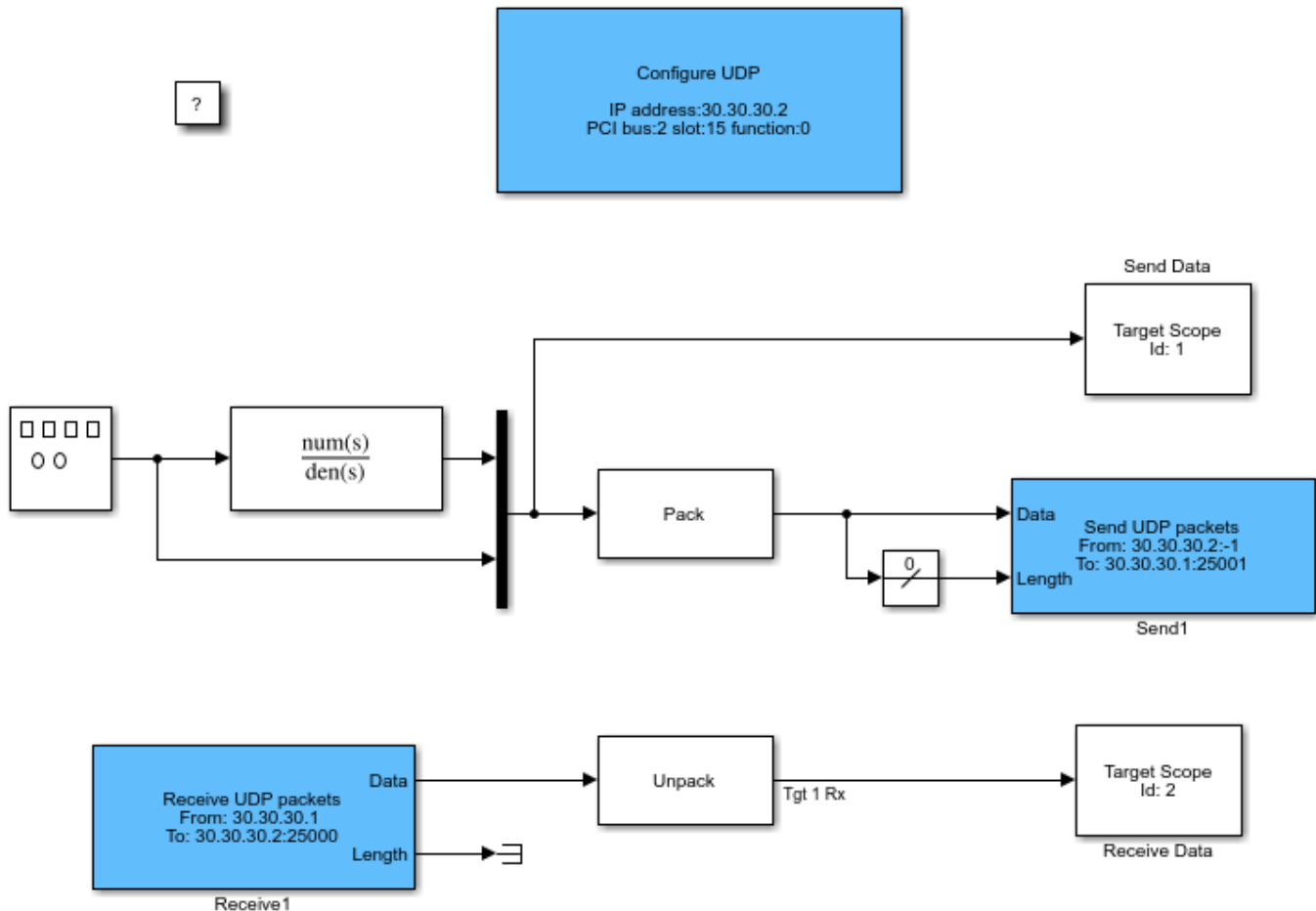
```
if (mdlOpen)
    bdclose('TargetToTargetRealtimeUDP1');
end
```

Open, Build, and Download the TargetPC2 Model

Click here to open model 2: TargetToTargetRealtimeUDP2.

Open the model.

```
mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if ~any(strcmp('TargetToTargetRealtimeUDP2', systems))
    mdlOpen = 1;
    open_system(fullfile(matlabroot, 'toolbox', 'rtw', 'targets', 'xpc', 'xpcdemos', 'TargetToTargetRealtimeUDP2'));
end
```



Copyright 2016 The MathWorks, Inc.

Build the model and download to the target computer.

- Configure for a non-Verbose build.
- Configure UDP parameters for TargetPC2.
- Build and download application.

```
set_param('TargetToTargetRealtimeUDP2','RTWVerbose','off');
set_param('TargetToTargetRealtimeUDP2/UDP Configure','PCIBus','8')
set_param('TargetToTargetRealtimeUDP2/UDP Configure','PCISlot','10')
set_param('TargetToTargetRealtimeUDP2/UDP Configure','PCIFunction','0')
rtwbuild('TargetToTargetRealtimeUDP2');
tg2 = slrt('TargetPC2');
load(tg2,'TargetToTargetRealtimeUDP2');
```

```
### Starting Simulink Real-Time build procedure for model: TargetToTargetRealtimeUDP2
Warning: This model contains blocks that do not handle sample time
changes at runtime. To avoid incorrect results, only change
the sample time in the original model, then rebuild the model.
### Successful completion of build procedure for model: TargetToTargetRealtimeUDP2
### Created MLDATX ..\TargetToTargetRealtimeUDP2.mldatx
### Looking for target: TargetPC2
### Download model onto target: TargetPC2
```

Close the model if we opened it.

```
if (mdlOpen)
    bdclose('TargetToTargetRealtimeUDP2');
end
```

Run both Models

Using the Simulink Real-Time object variables `tg1` and `tg2`, start the models.

- Start the TargetPC1 model. Simulink object is `tg`.
- Start the TargetPC2 model. Simulink object is `tg2`.
- Run for 5 seconds.

```
start(tg);
start(tg2);
pause(5);
```

Stop both Models

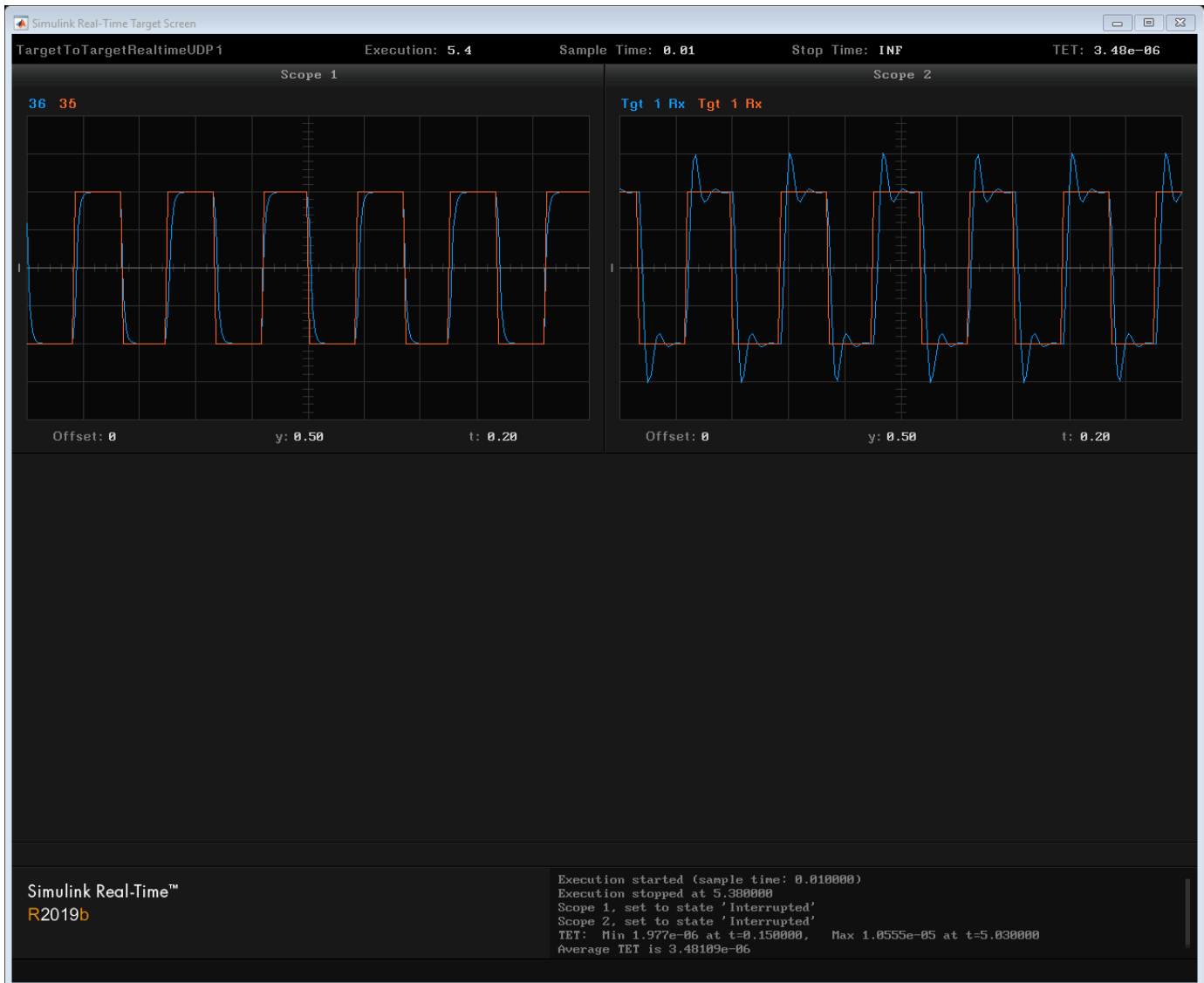
Using the Simulink Real-Time object variables `tg1` and `tg2`, stop the models.

- Stop the TargetPC1 model. Simulink object is `tg`.
- Stop the TargetPC2 model. Simulink object is `tg2`.

```
stop(tg);
stop(tg2);
```

Generate TargetPC1 Plot

```
tg.viewTargetScreen();
```



Generate TargetPC2 Plot

```
tg2.viewTargetScreen();
```



Apply Simulink Real-Time Model Template to Create Real-Time Application

This example shows how to use the Simulink Real-Time template to create a Simulink model. Starting from the model template for Simulink Real-Time provides a new model that has configuration parameters set up for building a real-time application.

To see the Simulink Real-Time commands for each operation in this example, view the example code.

Create a Simulink Model from Template

To create this model from the Simulink start page, in the MATLAB Command Window, type:

```
simulink
```

Select the Simulink Real-Time template from the start page and create the `exampleSlrtApp` model. Or, in the Command Window, use the `Simulink.createFromTemplate` command.

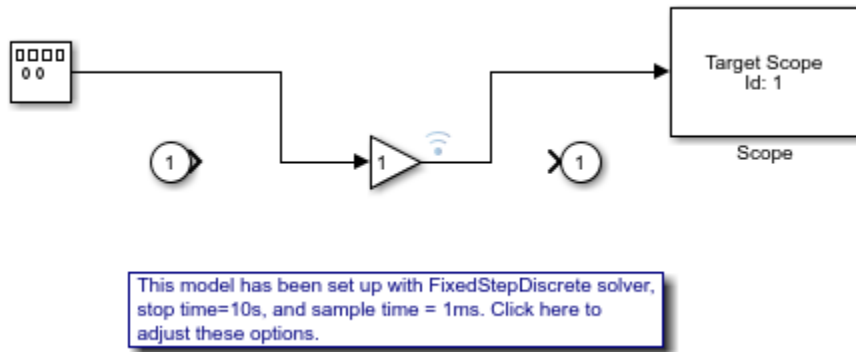


This model has been set up with FixedStepDiscrete solver, stop time=10s, and sample time = 1ms. [Click here to adjust these options.](#)

Add Blocks, Connections, and Data Logging to the Model

The Simulink Real-Time model template contains a Gain block that connects an inport to an output. To make this model produce more interesting data to view on a target scope and to view with the Simulation Data Inspector (SDI), add these blocks and connections. You can use the UI for these steps or use the script commands shown:

- Add a signal generator by using the `add_block` command. Use the `set_param` command to set its Amplitude parameter value to 4 and set its Frequency parameter value to 400.
- Add a target scope by using the `add_block` command.
- Remove the connections between the Gain block, inport, and output by using the `delete_line` command.
- Connect the signal generator to the gain block input by using the `add_line` command.
- Connect the gain block output to the target scope by using the `add_line` command.
- Mark the Gain block output for data logging by using the `set_param` command.



Build the Real-Time Application and View Logged Data

You are ready to build the real-time application, run it on the target computer, and view the logged data with these steps:

- Make sure that the development computer has a connection to the target computer.
- Build the model and download the real-time application to the target computer. On the **Real-Time** tab, click **Run on Target**. Or, use the `rtwbuild` command and the `load` command.
- Run the real-time application and log data by using the `start` command.
- Open the Simulation Data Inspector by double-clicking the Simulation Data Inspector icon on the Gain block output signal or by using the `Simulink.sdi.view` command.

More Information

- "Create and Run Real-Time Application from Simulink Model"
- "Configure and Control a Real-Time Application"
- Simulation Data Inspector

Data Logging With Simulation Data Inspector (SDI)

This example shows how to use Simulink® Real-Time™ with an SDI log of signal data. Signals are logged during model execution. At the end of the run, the SDI interface displays the signal. This example show how to get the signals from SDI interface by using the command line.

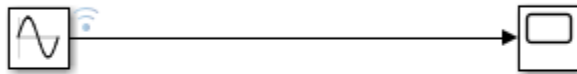
Open, Build, and Download the Model

Open the model `slrtex_file_logging`. This model generates 20 sinusoids, each having a different amplitude.

Note: Scopes of type 'target' are limited to 10 signals.

Open the model.

```
mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if all(~strcmp('slrtex_file_logging', systems))
    mdlOpen = 1;
    open_system(fullfile(matlabroot, 'toolbox', 'slrt', 'slrtexamples', 'slrtex_file_logging'));
end
```



Copyright 2008 - 2018 The MathWorks, Inc.

Build the model and download to the target computer

- Configure for a non-Verbose build.
- Build and download application.

```
set_param('slrtex_file_logging', 'RTWVerbose', 'off');
rtwbuild('slrtex_file_logging');
```

```
### Starting Simulink Real-Time build procedure for model: slrtex_file_logging
Warning: This model contains blocks that do not handle sample time
changes at runtime. To avoid incorrect results, only change
the sample time in the original model, then rebuild the model.
```

```
### Successful completion of build procedure for model: slrtex_file_logging  
### Created MLDATX ..\slrtex_file_logging.mldatx
```

- Close the model if we opened it.

```
if (mdlOpen)  
    bdclose('slrtex_file_logging');  
end
```

Run the Model

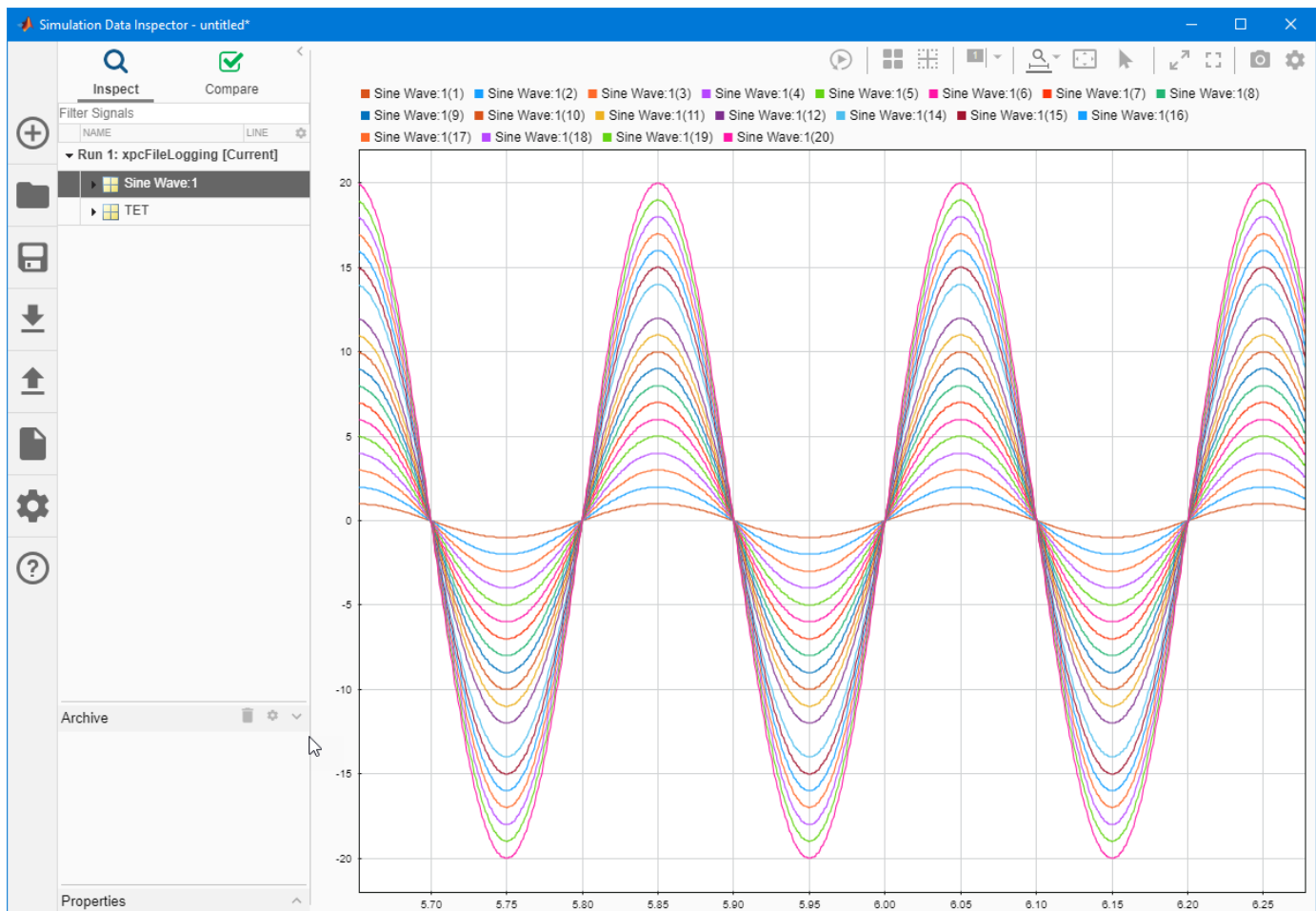
Using the Simulink Real-Time object variable, `tg`, start the model.

```
tg = slrt('TargetPC1');  
load(tg, 'slrtex_file_logging');  
start(tg);  
pause(5);  
while strcmp(tg.Status, 'running')  
    pause(0.05);  
end
```

Display the signals in the Simulation Data Inspector

To view the plotted signal data, open the Simulation Data Inspector.

```
Simulink.sdi.view
```



Retrieve and plot signal data from the Simulation Data Inspector

You can also retrieve the signal data from the SDI and plot the data by using these commands.

- Get all the runs
- Get the run information

```
runIds = Simulink.sdi.getAllRunIDs();
run = Simulink.sdi.getRun(runIds(end));
signals=run.SignalCount;
array=1;
```

- Get the signal.
- Get the signal objects.
- Take only Sine values.

```
for signalIndex=1:signals
    signalID = run.getSignalIDByIndex(signalIndex);
    signalObj = Simulink.sdi.getSignal(signalID);
    if(~isempty(strfind(signalObj.Name,'Sine')))
        signalArray(:,array)=signalObj.Values(:,1).Data;
        timeValues=signalObj.Values(:,1).Time;
        array=array+1;
```



```
    end  
end
```

- Plot the signals.

```
plot(timeValues,signalArray);  
grid on;  
xlabel('Time (sec)'); ylabel('Amplitude');
```

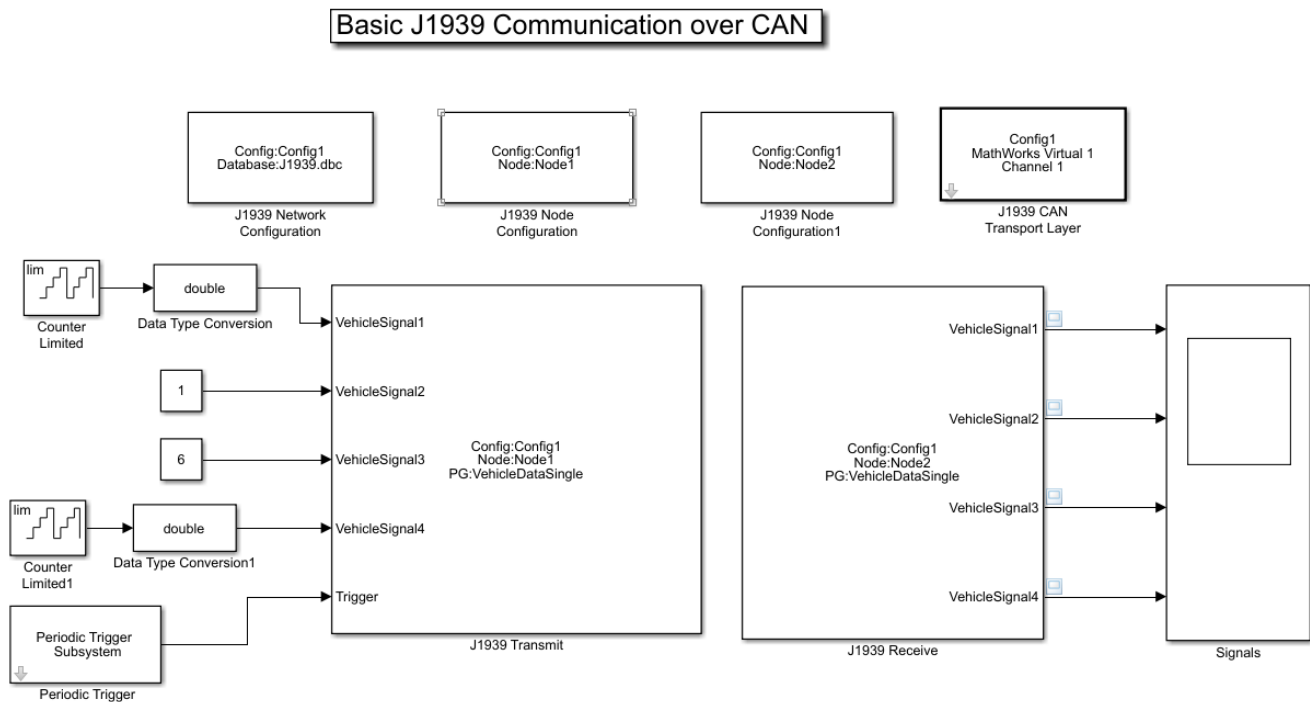
More Information

- “Generate a Simulation Data Inspector Report Programmatically” (Simulink)

Basic J1939 Communication over CAN

This example shows you how to use J1939 blocks to directly send and receive Parameter Group (PG) messages in Simulink®.

Vehicle Network Toolbox™ provides J1939 Simulink blocks for receiving and transmitting Parameter Groups via Simulink models over Controller Area Networks (CAN). This example performs data transfer over a CAN bus using the J1939 Network Configuration, J1939 Node Configuration, J1939 CAN Transport Layer, J1939 Receive and J1939 Transmit blocks. It also uses MathWorks Virtual CAN channels connected in a loopback configuration.



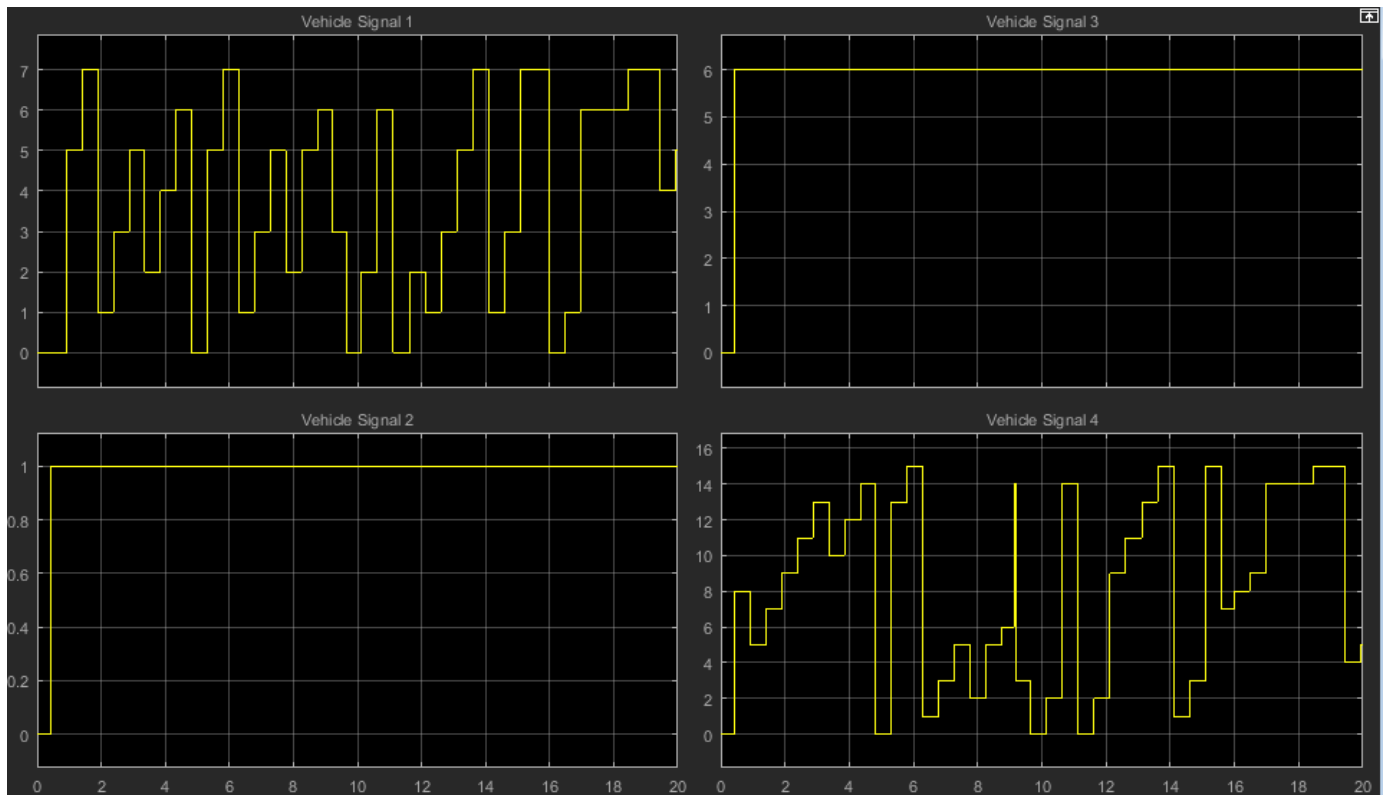
Set Up J1939 Block Parameters

Create a model to set up J1939 receive and transmit over the network. The model is configured to perform single frame transmission between two nodes defined in the J1939 database file.

- Use a J1939 Network Configuration block and select the CAN Database, J1939.dbc. This J1939 database file consists of two nodes and a couple of single-frame and multiframe messages.
- Use a J1939 CAN Transport Layer block and set the Device to MathWorks Virtual Channel 1. The transport layer is configured to transfer J1939 messages over CAN via the specified virtual channel.
- Use basic Simulink source blocks to connect to a J1939 Transmit block. The J1939 Transmit block is set to queue data for transmit at each timestep when the Trigger port is enabled. For this example, a periodic trigger subsystem sends a high pulse every 50 milliseconds.
- Use the J1939 Receive block to receive the messages transmitted over the network.

Visualize Signals Received on the Network

Plot the results to see the vehicle signal values received over the network. The X-axis corresponds to the simulation timestep.



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:

MathWorks Help Center website

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 17-2
- “Test 2: Ping Target Computer with slrtpingtarget” on page 17-4
- “Test 3: Software Restart Target Computer” on page 17-5
- “Test 4: Build and Download slrttestmdl” on page 17-6
- “Test 5: Check Communication with Target Computer” on page 17-7
- “Test 6: Download Prebuilt Real-Time Application” on page 17-8
- “Test 7: Execute Real-Time Application” on page 17-9
- “Test 8: Upload Logged Data and Compare Results” on page 17-10

- 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 `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: MathWorks Help Center website
- 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 26-2.

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:

MathWorks Help Center website.

- “Test 1: Ping Target Computer with System Ping” on page 17-2
- “Test 2: Ping Target Computer with slrtpingtarget” on page 17-4
- “Test 3: Software Restart Target Computer” on page 17-5
- “Test 4: Build and Download slrttestmdl” on page 17-6
- “Test 5: Check Communication with Target Computer” on page 17-7
- “Test 6: Download Prebuilt Real-Time Application” on page 17-8
- “Test 7: Execute Real-Time Application” on page 17-9
- “Test 8: Upload Logged Data and Compare Results” on page 17-10

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 16-2. If you still cannot solve your issue, see “Find Simulink Real-Time Support” on page 26-2.

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 16-2. If you still cannot solve your issue, see “Find Simulink Real-Time Support” on page 26-2.

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”.

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 16-2. If you still cannot solve your issue, see “Find Simulink Real-Time Support” on page 26-2.

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. In the Command Window, type:

```
slrtsetCC ('VisualC', '')
mex -setup
```

If this procedure does not solve your issue, continue with the tests in “Troubleshoot with Confidence Test” on page 16-2. If you still cannot solve your issue, see “Find Simulink Real-Time Support” on page 26-2.

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 16-2. If you still cannot solve the issue, see “Find Simulink Real-Time Support” on page 26-2.

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 by Using Run on Target”).

If this procedure does not solve your issue, continue with the tests in “Troubleshoot with Confidence Test” on page 16-2. If you still cannot solve your issue, see “Find Simulink Real-Time Support” on page 26-2.

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 16-2. If you still cannot solve your issue, see “Find Simulink Real-Time Support” on page 26-2.

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 Output 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 26-3.

If this procedure does not solve your issue, continue with the tests in “Troubleshoot with Confidence Test” on page 16-2. 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 26-2.

Development Computer Configuration

Troubleshoot Halted Boot Drive Creation

The MATLAB interface on my development computer stops while creating a Simulink Real-Time boot disk or network boot image.

What This Issue 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 Workaround

To identify the issue that has stopped boot drive creation, try these workarounds.

Use Another Development Computer Drive

Use another development computer drive to recreate the Simulink Real-Time boot drive or network boot image.

Login as Administrator

On the development computer, login as administrator to recreate the Simulink Real-Time boot drive or network boot image.

Check 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

More About

- “Command-Line Network Boot Method” on page 5-13
- “Command-Line Standalone Boot Method” on page 5-15

External Websites

- <https://www.speedgoat.com/support>

Target Computer Configuration

- “Troubleshoot Target Computer Stack Size” on page 19-2
- “Troubleshoot Target Computer Ethernet and MAC Address Information” on page 19-3

Troubleshoot Target Computer Stack Size

I want to find the optimum amount of target computer memory to use for the stack.

What This Issue 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 that you can adjust the stack size for the application.

Try This Workaround

To discover and adjust the stack size used by the real-time threads on the target computer:

1 Add these 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 want to find Ethernet address information and MAC address information from the target computer for configuring blocks in models.

What This Issue 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 Workaround

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 | Com m | 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 run 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 19-3.

To perform these steps, 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 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.

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 | Com m | 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 82541GILF | Y | Y | 16 | 4 | 0 | N | N | 90:E2:BA:17:5D:15 |

See Also

Simulink Real-Time Explorer | `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 20-2
- “Troubleshoot Communication Timeout with Target Computers” on page 20-4
- “Troubleshoot Communication Timeout with Target Computers and Multiple Ethernet Cards” on page 20-6
- “Troubleshoot Communication Failure Through Firewall” on page 20-7
- “Troubleshoot Network Boot Failure Through Firewall” on page 20-8

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 Issue Means

The development computer communicates with the target computer by using an Ethernet communications link. Various software and hardware configuration issues can interfere with this link.

Try This Workaround

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 20-6.

See Also

More About

- “Troubleshoot with Confidence Test” on page 16-2

External Websites

- <https://www.speedgoat.com/support>

Troubleshoot Communication Timeout with Target Computers


Some issue is causing communications between the development computer and target computer to time out.

What This Issue Means

If the communication link between the development and target computers is broken or misconfigured, the link times out after about 5 seconds. Before you continue troubleshooting, check that you have followed the instructions in “System Configuration”.

Try This Workaround

To identify timeout issues, use 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 then 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 then 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 a timeout.

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 these 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 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.

See Also

More About

- “System Configuration”

External Websites

- <https://www.speedgoat.com/support>

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 time out.

What This Issue 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 Workaround

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 5-8.

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.

See Also

More About

- “System Configuration”
- “Ethernet Card Selection by Index” on page 5-8

External Websites

- <https://www.speedgoat.com/support>

Troubleshoot Communication Failure Through Firewall

Some issue with Windows Defender Firewall of the development computer causes a communications failure with the target computer.

What This Issue Means

This failure occurs when the firewall settings in Windows Defender Security Center block communications with the target computer. The firewall configuration must not block the IP addresses that the development and target computers use to communicate.

Try This Workaround

Configure the firewall settings in Windows Defender Security Center to allow communications between the development and target computers.

- 1 Confirm that the firewall on the development computer is Windows Defender. In the Command Window type:

```
[~,antivirus]=system('WMIC /Node:localhost /Namespace:\\root\SecurityCenter2 Path AntiVirusPr
```

The antivirus software displays as Windows Defender.

- 2 Find Windows Defender Firewall with Advanced Security with Windows search.
- 3 Select **Inbound Rules** and **New Rule**.
- 4 For the **Rule Type**, select **Custom** and **Next**.
- 5 For the **Program**, select **All programs** and **Next**.
- 6 For the **Protocol and Ports**, select **Any** and **Next**.
- 7 For the **Scope**, add the IP address of the development computer in **Which local IP addresses does this rule apply to?** and add the target computer IP address in **Which remote IP addresses does this rule apply to?**
- 8 For the **Action**, select **Allow the connection** and **Next**.
- 9 For the **Profile**, select the **Domain**, **Private**, and **Public** check boxes and **Next**.
- 10 For **Name**, provide a **Name** for this inbound rule (for example, Simulink Real-Time inbound) and **Finish**.
- 11 Select **Outbound Rules** and **New Rule**.
- 12 Repeat steps 4 through 10 for the custom outbound rule.

See Also

More About

- “Troubleshoot Communication Failure with Target Computers” on page 20-2

Troubleshoot Network Boot Failure Through Firewall

Some issue with Windows Defender Firewall of the development computer causes a network boot failure with the target computer. The network boot fails with this error on the target computer.

```
PXE-E51: No DHCP or proxyDHCP offers were received.  
PXE-M0F: Exiting Intel Boot Agent.
```

What This Issue Means

This error message occurs when the firewall settings in Windows block communications for the network boot server application. The firewall configuration must not block operation of the boot server.

Try This Workaround

The net boot server uses the BOOTP and TFTP protocols. These protocols use UDP communications on ports 67 through 70. Consult with your IT department to allow communication across the Windows firewall.

Both TCP and UDP protocol use that port range. Typically, a firewall rule needs to be created for the Ethernet card or IP range that you use to communicate between host and target computer.

See Also

More About

- “Check Communications for Network Boot” on page 20-6

Target Computer Boot Process

Model Compilation

- “Troubleshoot Model Links to DLLs” on page 22-2
- “Troubleshoot Build Error for Accelerator Mode” on page 22-3
- “Troubleshoot Referenced Model with Global Data Stores” on page 22-4

Troubleshoot Model Links to DLLs

Some model build issues are caused by linking to dynamic link libraries (.dll).

What This Issue 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 Workaround

When you build your models, check that you link to only static link libraries. When you compile with Simulink Real-Time S-functions, linking to static libraries avoids the dependency issues that occur in dynamic libraries. Each static library must be self contained. The static library must not be 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 Issue 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 `xpcEnetDemolRx` model, change the Simulink mode to rapid accelerator, and run the model, Simulink displays this error:

```
### Build procedure for model: 'xpcEnetDemolRx' aborted due to an error.  
Unable to build a standalone executable to simulate the model  
'xpcEnetDemolRx' 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 the Simulink Real-Time target computer. Any blocks that access hardware report a build error if you compile them by using accelerator mode or rapid accelerator mode.

Try This Workaround

Change the simulation mode to normal mode or external mode.

See Also

More About

- “How Acceleration Modes Work” (Simulink)
- “Simulink Real-Time Options Pane”

Troubleshoot Referenced Model with Global Data Stores

Some Simulink Real-Time model build issues are caused by using global data stores in referenced models. The diagnostic viewer displays an error:

```
Error: Simulink Real-time does not support global data stores across model reference blocks.
```

What This Issue Means

Simulink Real-Time model builds do not support global data stores for referenced models.

Try This Workaround

Restructure the model to eliminate global data stores.

See Also

Data Store Memory

More About

- “Local and Global Data Stores” (Simulink)

Real-Time Application Execution

- “Troubleshoot Missing or Unreadable Crash Information” on page 23-2
- “Troubleshoot Unexpected Measured Sample Time Value” on page 23-4
- “Troubleshoot Changed Sample Time at Run Time That Affects Results” on page 23-6
- “Troubleshoot Unexpected Measured Stop Time Value” on page 23-7

Troubleshoot Missing or Unreadable Crash Information

I get missing or unreadable development computer crash information and errors for the `SimulinkRealTime.crashInfo` function.

What This Issue 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 Workaround

If you see one of the messages in the examples, look for one of these causes and try the related workaround.

Wait 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.

Check 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.

Check 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.

Check 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.

Check 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 Issue 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:

$$119 \text{ ticks} / (1.193 \times 10^6 \text{ ticks/s}) = 99.75 \times 10^{-6} \text{ s} \\ (99.75 \text{ microseconds})$$

Compared to the requested original sample time of 100 microseconds, this value is 0.25% faster.

Try This Workaround

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:

$$1/10200 \text{ s} \times 1.193 \times 10^6 \text{ ticks/s} = 116.96 \text{ ticks}$$

Round this number to the nearest whole number of 117 ticks. The resulting frequency is then:

$$(116.96 \text{ ticks}/117)(50) = 49.983 \text{ Hz}$$

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 10-10

Troubleshoot Changed Sample Time at Run Time That Affects Results

Some blocks produce incorrect results when their sample time is changed at run time.

What This Issue 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 the model build.

Try This Workaround

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 10-10

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 Issue 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.

Example

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 Workaround

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 24-2
- “Troubleshoot Missing Mux Block Output on Scope” on page 24-3

Troubleshoot Invalid File IDs on Target Computer

I get invalid file ID errors on the target computer.

What This Issue 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 Workaround

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:

MathWorks Help Center website.

See Also

Gain

External Websites

- MathWorks Help Center website

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 Issue 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 Workaround

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

- “Types of Composite Signals” (Simulink)

Real-Time Application Performance

- “Troubleshoot Unsatisfactory Real-Time Performance” on page 25-2
- “Troubleshoot Overloaded CPU from Executing Real-Time Application” on page 25-5
- “Troubleshoot Task Execution Time” on page 25-7
- “Troubleshoot Failed Read of Profiling Data” on page 25-8
- “Troubleshoot Timeout During File System Access” on page 25-9

Troubleshoot Unsatisfactory Real-Time Performance

I want some recommended methods to improve unsatisfactory real-time application performance.

What This Issue 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 Workaround

You can improve run-time performance and reduce the task execution time (TET) of a model with these methods.

Run Performance Tools

Use these performance tools:

- Run Performance Advisor. On the **Debug** tab, click **Performance Advisor** 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 10-22.
- 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 10-15.

Use a 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.

Minimize the 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.

Contact Technical Support

For additional guidance, refer to these sources:

- MathWorks Tech Support: MathWorks Help Center website
- 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 26-2

See Also

More About

- “Sample Time and Throughput in Real-Time Applications” on page 10-22
- “Improve Performance of Multirate Model” on page 10-2
- “Sample Time and Throughput in Real-Time Applications” on page 10-22
- “Execution Profiling for Real-Time Applications” on page 10-15
- “Partition Your Model Using Explicit Partitioning” (Simulink)
- “Polling Mode” on page 7-3
- “Find Simulink Real-Time Support” on page 26-2
- “Multicore Programming with Simulink” (Simulink)

External Websites

- www.speedgoat.com/products
- <https://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 Issue 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 occur:

- The Simulink Real-Time kernel halts model execution.
- If you allow the overload, model execution continues until a predefined event occurs (see “Permit CPU Overloads for Diagnosis” on page 25-5). 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 10-10).
- 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 Workaround

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.

Permit Long Initialization Time

For some real-time applications, normal initialization can extend beyond the first sample time. Use the `TLCOptions` property `xPCStartupFlag` with the smallest effective value, up to approximately 5.

Permit 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 10-11
- “TLC Command-Line Options”

Troubleshoot Task Execution Time

I want to find the task execution time (TET) for one sample step.

What This Issue Means

Task execution time (TET) measures how long it takes the kernel to run for one base-rate time step.

Try This Workaround

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 `getProfilerData` function.

What This Issue Means

A call of the `getProfilerData` function produces an error because:

- The file does not exist in the expected location.
- The attempt to read the file causes an error.

Try This Workaround

To address this issue, try this procedure:

- 1 Check that you have set the profiling options in **Code Generation > Verification**.
- 2 Build and download the real-time application.
- 3 In the Command Window, enter:

```
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 10-15
- “Troubleshoot Timeout During File System Access” on page 25-9

Troubleshoot Timeout During File System Access

Some issue is causing a timeout during access to the target computer file system.

What This Issue 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 Workaround

Increase the communication timeout value. If using file scopes, try using the Simulation Data Inspector 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 20-4
- “Troubleshoot Failed Read of Profiling Data” on page 25-8

Simulink Real-Time Support

- “Find Simulink Real-Time Support” on page 26-2
- “Install Simulink Real-Time Software Updates” on page 26-3

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:
MathWorks Help Center website
- 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.

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 `slrteplr`.
- 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”).
- 5 Select **Boot configuration** and click **Create boot disk**.
- 6 Restart the target computer.
- 7 Build each model that you want to execute. In Simulink Editor, on the **Real-Time** tab, click **Run on Target**.

See Also

More About

- “PCI Bus Ethernet Setup”

External Websites

- <https://www.mathworks.com/downloads>
- <https://www.speedgoat.com/support>

