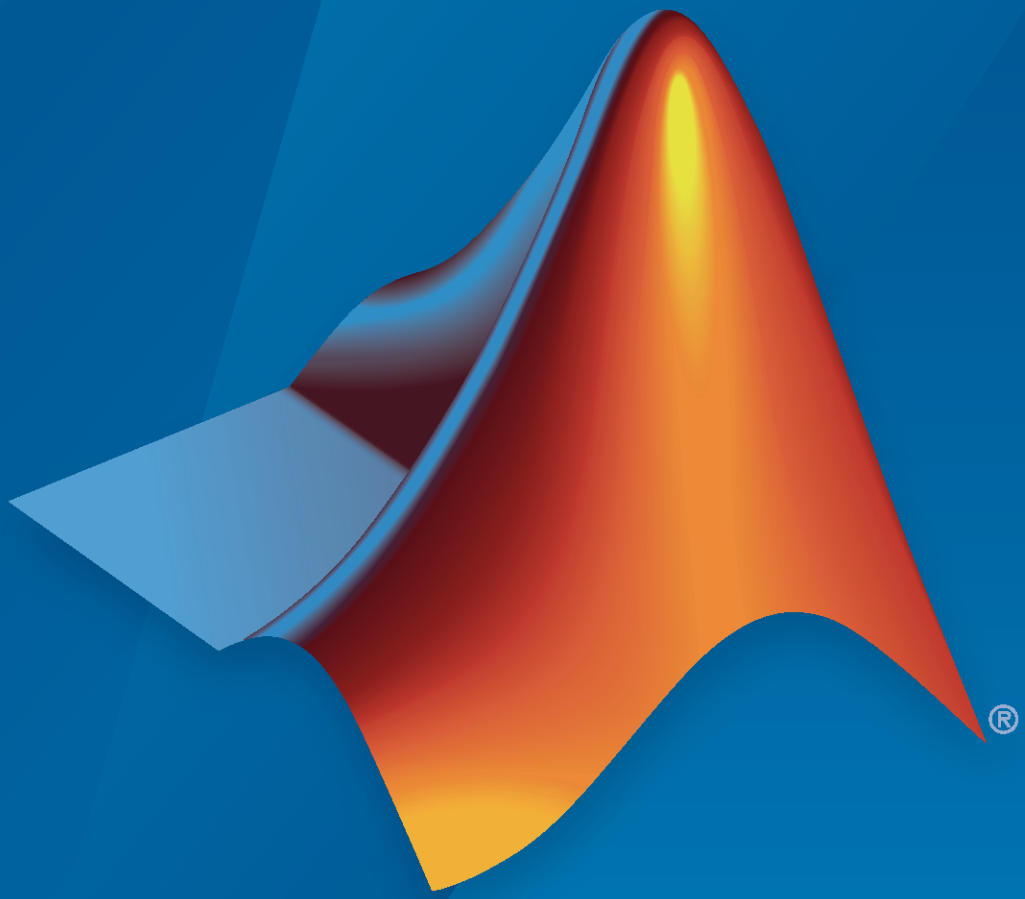
Simulink® Real-Time™

User's Guide



MATLAB® & SIMULINK®

R2020b

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

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

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

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

Phone: 508-647-7000

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

Simulink® Real-Time™ User's Guide

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)
September 2020	Online only	Revised for Version 7.0 (Release 2020b)

Contents

Real-Time Application Setup

5 Real-Time Application Environment

6 Signals and Parameters

9 Tuning Performance

Execution with MATLAB Scripts

10 Real-Time Application Objects and Options in the MATLAB Interface

11 Simulink Real-Time Instrument Object

12 Automated Test with Simulink Test

Examples

13 Simulink Real-Time Examples

16 Model Compilation

17 Real-Time Application Performance

18 Simulink Real-Time Support

Introduction

Simulink Real-Time Product Description

Build, run, and test real-time applications

Simulink Real-Time lets you create real-time applications from Simulink models and run them on Speedgoat target computer hardware connected to your physical system. It is designed for real-time simulation and testing tasks, including rapid control prototyping (RCP), DSP and vision system prototyping, and hardware-in-the-loop (HIL) simulation.

With Simulink Real-Time you can extend your Simulink models with Speedgoat I/O driver blocks and automatically build real-time applications. Tests can be automated or run interactively on a Speedgoat target computer equipped with a real-time kernel, multicore CPUs, I/O and protocol interfaces, and FPGAs.

Simulink Real-Time and Speedgoat target computer hardware are designed to work together to create real-time systems for desktop, lab, and field environments. The software and hardware solution supports the latest versions of MATLAB® and Simulink.

Speedgoat Target Computers and I/O Hardware

Speedgoat target computers are real-time computers fitted with a set of I/O hardware, Simulink programmable FPGAs, and communication protocol support. Speedgoat target computers are optimized for use with Simulink Real-Time and fully support the HDL Coder™ workflow.

Connect a development computer to a Speedgoat target computer that meets your requirements: form factor, performance, I/O interface, and protocol interface. Speedgoat target computer systems come with:

- I/O and protocol interfaces, an Intel® CPU, and optional FPGA hardware, configured and ready to use
- I/O cables, terminal boards, Simulink driver blocks, documentation, and a loopback wiring harness that facilitates acceptance testing for each I/O module
- The Simulink Real-Time kernel preinstalled on the target computer

Hardware-In-the-Loop (HIL) Simulators and Rugged Units for Controls (RCP), DSP, and Vision Prototyping

Model Architectures

FPGA Models

Speedgoat FPGA Support with HDL Workflow Advisor

Use Simulink Real-Time and HDL Coder to implement Simulink algorithms and configure I/O functionality on Speedgoat Simulink-Programmable I/O modules. For an example that shows the development workflow for FPGA I/O modules, see “FPGA Programming and Configuration on Speedgoat Simulink-Programmable I/O Modules” (HDL Coder).

When you open the HDL Workflow Advisor in HDL Coder and run the `Simulink Real-Time FPGA I/O` workflow, you generate a Simulink Real-Time interface subsystem. 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 Simulink-Programmable I/O Module Support

Speedgoat Simulink-Programmable I/O modules are part of Speedgoat target computer systems. To run the `Simulink Real-Time FPGA I/O` workflow, install the Speedgoat Library and the Speedgoat HDL Coder Integration Packages. You can then choose the **Target platform** and run the workflow to generate a Simulink Real-Time interface subsystem. To see the documentation for the integration packages, enter this command at the MATLAB command prompt.

```
speedgoat.hdlc.doc
```

To learn about	See links
The integration packages and how you can install them.	See Speedgoat - HDL Coder Integration Packages.
Speedgoat I/O modules that are supported with the HDL Workflow Advisor.	See Speedgoat Real-Time FPGA Application Support from HDL Coder.
Applications and use cases	See Common Use Cases and Applications.
Supported interfaces for various types of I/O connectivity and protocols as well as fundamental functionality such as PCIe read/write and DMA.	See Supported Interfaces.
Provided examples for all supported I/O modules and functionality	See Speedgoat I/O Examples.

Prepare for FPGA Workflow

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

- HDL Coder and Simulink Real-Time.
- Xilinx® design tools with specific tool and version listed in “HDL Language Support and Supported Third-Party Tools and Hardware” (HDL Coder). You must also set up the path to the tool by using the `hdlsetuptoolpath` function.
- Speedgoat Library and the Speedgoat HDL Coder Integration Packages.
- Speedgoat FPGA I/O module in the Speedgoat target machine.

You can use the workflow in HDL Coder to generate HDL code for your FPGA target device.

See Also

Related Examples

- “FPGA Programming and Configuration on Speedgoat Simulink-Programmable I/O Modules” (HDL Coder)

More About

- “HDL Language Support and Supported Third-Party Tools and Hardware” (HDL Coder)
- “Tool Setup” (HDL Coder)

External Websites

- www.speedgoat.com

Interrupt Configuration

Simulink Real-Time software schedules the real-time application by 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. You can:

- 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. For more information, see “Speedgoat Target Computers and I/O Hardware” on page 1-3.

See Also

Functional Mockup Units and Simulink Real-Time

Apply Functional Mockup Units by Using 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. These 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 numeric-valued and string-valued parameters. Simulink Real-Time generates code for only numeric- 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 `slrealtime.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.
- Generate a shared object SO file by using the QNX® Neutrino® tools for the FMU. For more information, see “Create the FMU File” on page 3-3.

You can then build and download the model to a target computer and run the real-time application. This process loads the required FMU binary files on the target computer. For more information about creating the FMU files, see “Compile Source Code for Functional Mockup Units” on page 3-3.

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

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

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

See Also

FMU

More About

- “Import FMUs”

External Websites

- https://fmi-standard.org/

Compile Source Code for Functional Mockup Units

When you build a model that includes FMU blocks, you must compile the FMU source code by using the QNX Neutrino compiler `qcc` or `q++`. This compiler creates shared object SO files that you include in the FMU. This process makes sure that the FMU contains the code to run on a Simulink Real-Time target computer. For more information, see “Apply Functional Mockup Units by Using Simulink Real-Time” on page 3-2.

Configure Compiler Environment Variables

The support package for the target computer includes the QNX Neutrino C/C++ compiler `qcc` or `q++`. Before using the compiler to generate the FMU file, configure the compiler environment variables.

1. Open a Windows `cmd` window and change folders to the root folder for the Simulink Real-Time Target Support Package. If you have changed the default folder for MATLAB support packages, adjust this command to match your custom location. At the Windows command prompt, type:

   ```
   C:
   cd C:\ProgramData\MATLAB\SupportPackages\<release>\toolbox\slrealtime\target\supportpackage\q
   ```

2. To set the Windows environment variables that are required to use the QNX Neutrino compiler from the command line, run the batch file `qnxsdp-env.bat`. At the Windows command prompt, type:

   ```
   qnxsdp-env.bat
   ```

3. Ensure that the `qcc` compiler is ready to use. At the Windows prompt, type:

   ```
   which qcc
   ```

   The command returns:

   ```
   C:/ProgramData/MATLAB/SupportPackages/<release>/toolbox/slrealtime/target/supportpackage/qnx7
   ```

Create the FMU File

The FMU file contains a hierarchy of files and folders. A set of these is provided in this example. The example shows how to create the shared object file for a single source file that is linked in the real-time application that runs on the target computer.

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

```
cd(fullfile(matlabroot, 'toolbox', 'slrealtime', ...
'examples', 'slrt_ex_fmu_work'))
```

To open the model for this example, in the MATLAB Command Window, type:

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

1. The example FMU file `vanDerPol_slrt.fmu` contains this set of folders:

   ```
   C:\work\my_fmu_work\
   C:\work\my_fmu_work\binaries\
   C:\work\my_fmu_work\binaries\slrt_x64
   C:\work\my_fmu_work\binaries\win64
   C:\work\my_fmu_work\documentation
   ```

```
C:\work\my_fmu_work\resources
C:\work\my_fmu_work\sources\
```

2. The example has FMU source files in the sources folder. This example uses files: `fmuTemplate.c`, `fmuTemplate.h`, `vanDerPol.c`
3. The example has a dynamically linked library file from the source files that was created by using a Window-based compiler. This file is in the win64 folder. This example uses file: `vanDerPol.dll`
4. To compile the sources in the example by using `qcc`, copy the files for the FMU from folder `matlab/slrealtime/examples/slrt_ex_fmu_work` to folder `C:\work\my_fmu_work`.
5. In a Windows `cmd` window, change the current folder to be the location of your source files. At the Windows command prompt, type:

   ```
   cd C:\work\my_fmu_work\sources
   ```
6. To generate the QNX Neutrino shared object SO file, the example compiles the sources by using the `-Vgcc_ntox86_64` flag. At the Windows command prompt, type:

   ```
   qcc -Vgcc_ntox86_64 -shared -o ../binaries/slrt_x64/vanDerPol.so -fPIC -DFMI_COSIMULATION -IC
   ```

   Note The QNX Neutrino compiler does not support Windows long file names (for example, paths with space characters) for the library search switch `-I`. To work around this limitation, you can install MATLAB in a path without spaces (for example, `C:\MATLAB`) or you can use the DOS 8.3 path in the compiler command. To get the 8.3 folder names, you can use the `dir /x` command in a Windows `cmd` window.
7. To generate the functional mockup unit FMU file that contains the QNX Neutrino shared object file, the example archives the folders and files. Use the FMU extension for the archive file. At the Windows command prompt, type:

   ```
   cd ..
   zip -r vanDerPol.fmu *
   ```

Implement the FMU Block in Model

To implement the `vanDerPol` block in the Simulink model by using the FMU, specify the FMU name for the block. Open the model `slrt_ex_vanderpol`, double-click the FMU block `vanDerPol`, and select the `vanDerPol.fmu` file for the FMU name block parameter.

Build the model, load the real-time application on the target computer, and run the real-time application.

See Also

FMU

More About

- “Import FMUs”

External Websites

- https://fmi-standard.org/

Third-Party Calibration Support

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”.
- Use the build process to generate `model.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.a2l` file for the real-time application and separate `refmodel.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”
- “Prepare ASAP2 Data Description File” on page 4-3
- “Calibrate Parameters with Vector CANape” on page 4-10
- “Calibrate Parameters with ETAS Inca” on page 4-14
- “XCP Master Mode”

External Websites

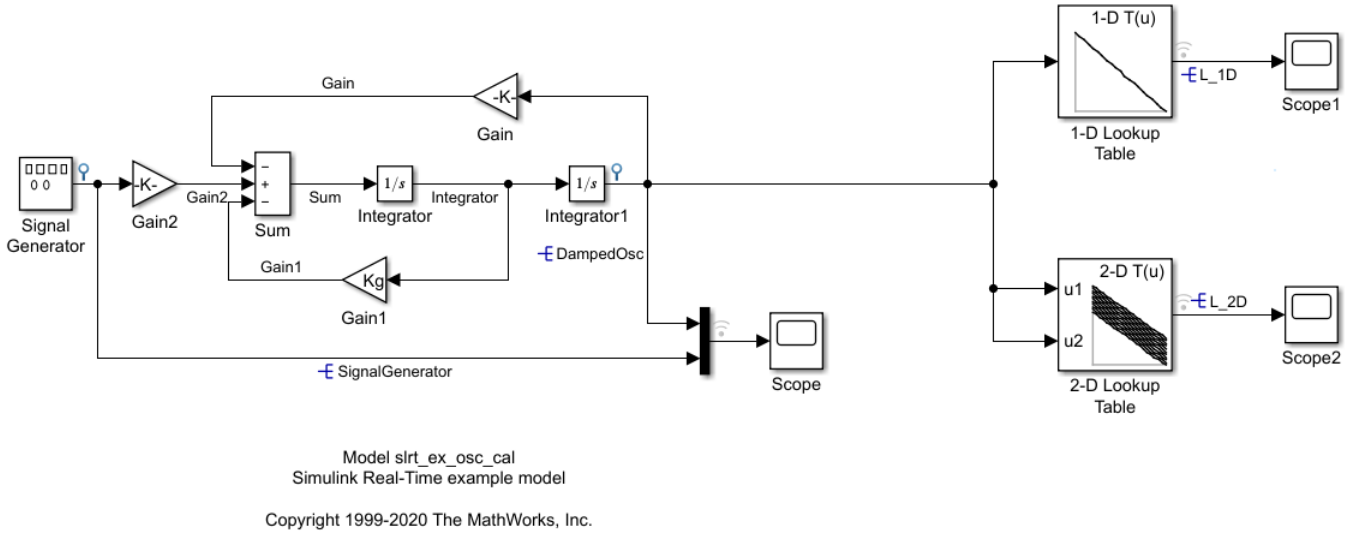
- www.vector.com
- www.etas.com

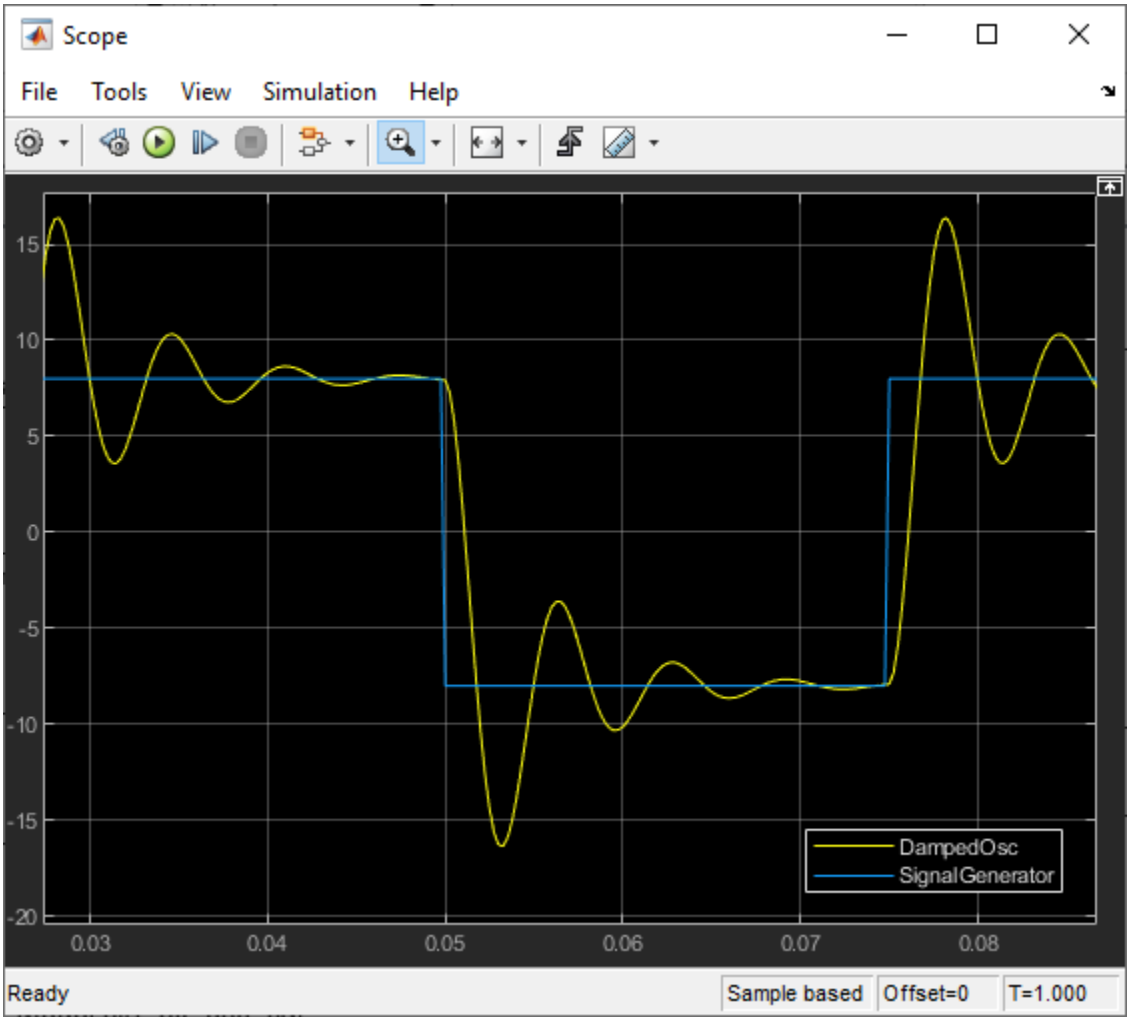
Prepare ASAP2 Data Description File

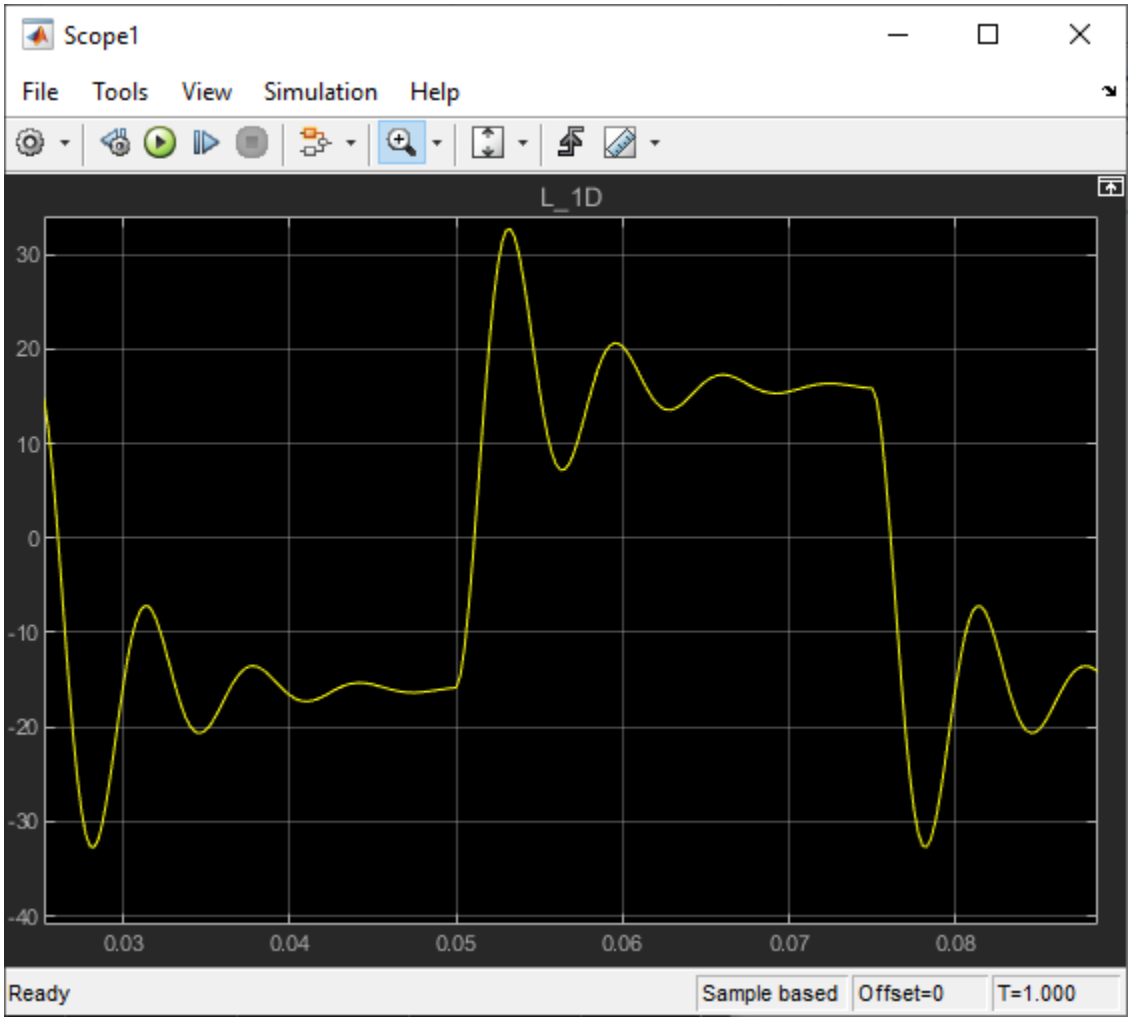
This example shows how to configure a Simulink Real-Time model so that the build generates an ASAP2 (A2L) data description file for the real-time application. The real-time application models a damped oscillator that feeds into 1-D and 2-D lookup tables, which invert and rescale the input waveform.

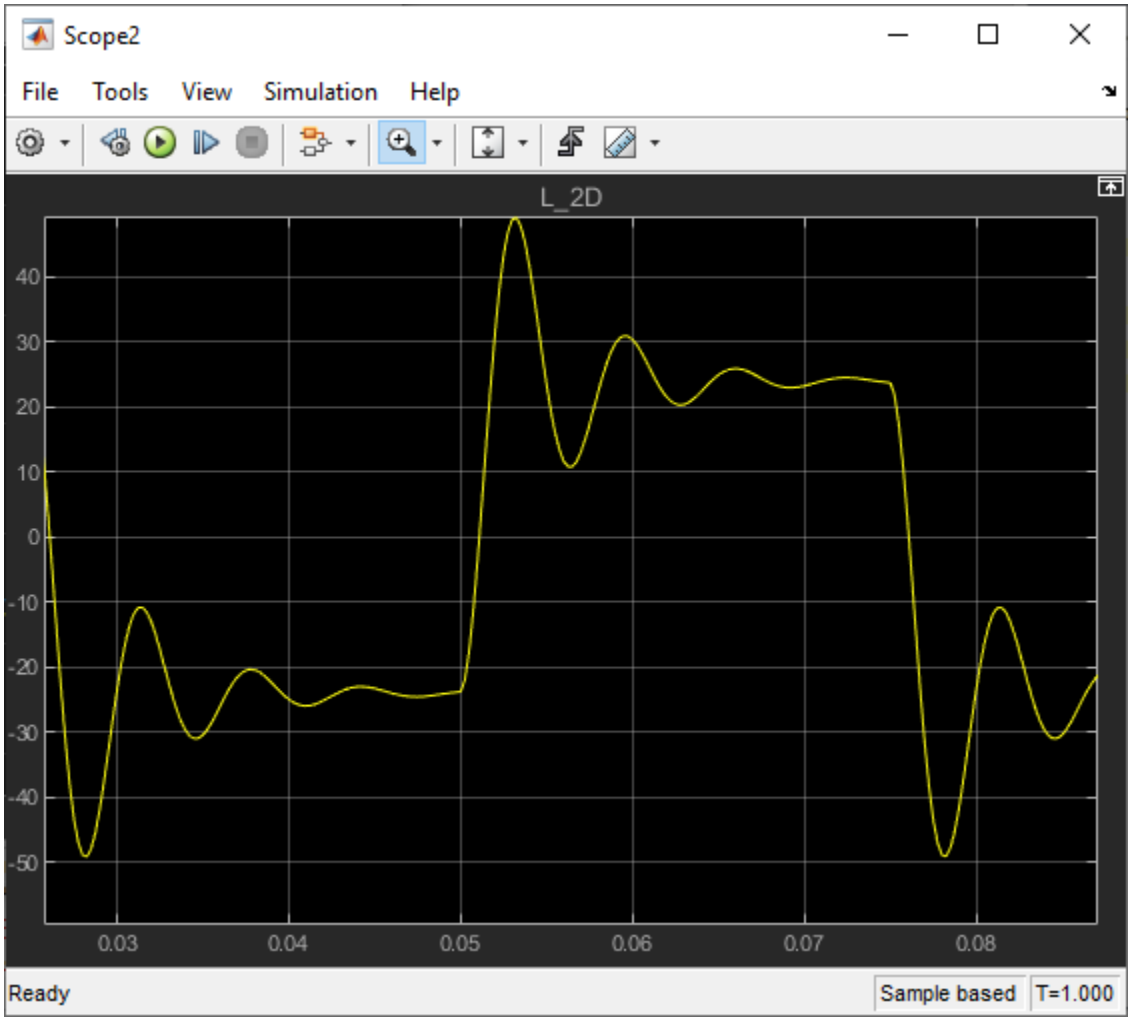
This example uses model `slrt_ex_osc_cal`. To open the model, in the MATLAB Command Window, type:

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




Scope
File
Tools
View
Simulation
Help
15
10
5
0
-5
-10
-15
-20
0.03
0.04
0.05
0.06
0.07
0.08
DampedOsc
SignalGenerator
Ready
Sample based
Offset=0
T=1.000


Scope1
File
Tools
View
Simulation
Help
L_1D
30
20
10
0
-10
-20
-30
-40
0.03
0.04
0.05
0.06
0.07
0.08
Ready
Sample based
Offset=0
T=1.000



Calibration of parameters reduces the ringing in signals `DampedOsc`, `L_1D`, and `L_2D`.

Initial Setup

Open the model and check for model data.

1. Open `slrt_ex_osc_cal`

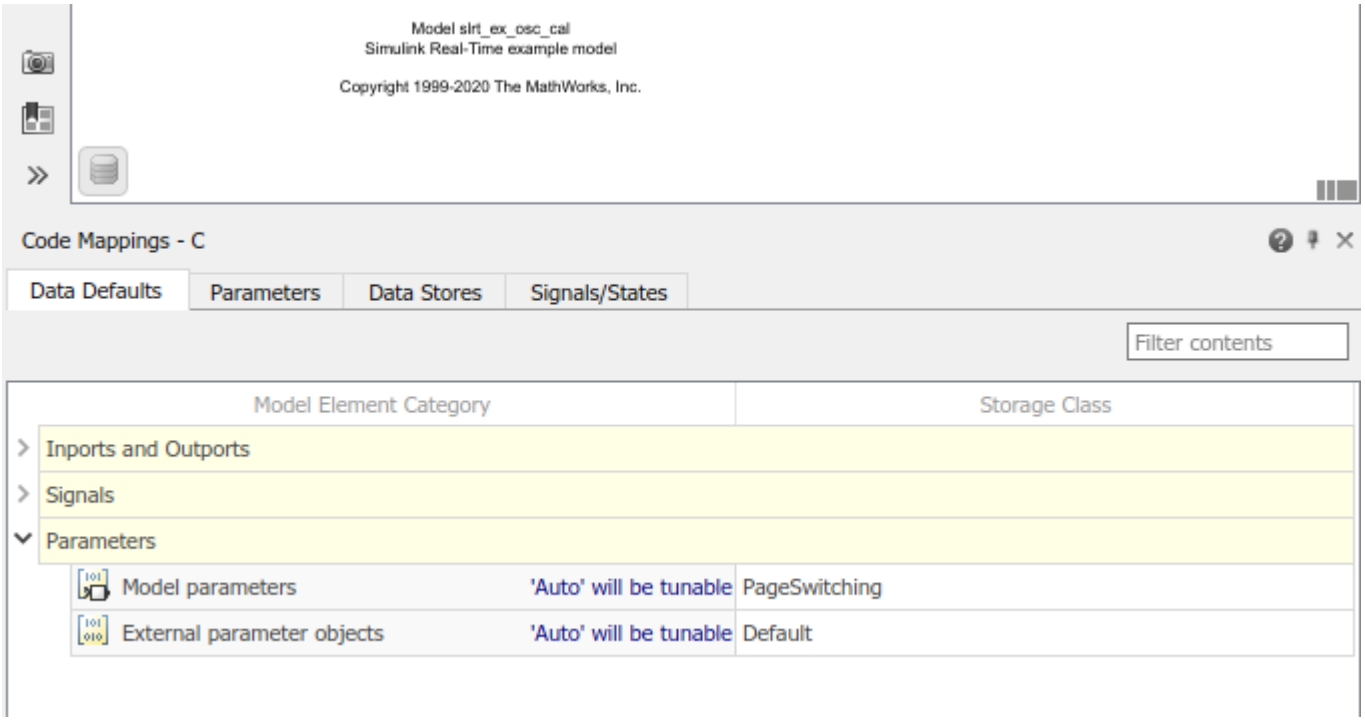
   ```
   open_system(fullfile(matlabroot,'toolbox','slrealtime', ...
   'examples', 'slrt_ex_osc_cal'))
   ```

   The **Model Workspace** variables contain these functions:

   - `Kg` — Parameter object for the `Gain1` block
   - `DampedOsc`, `SignalGenerator`, `L_1D`, `L_2D` — Signal objects for output signals
   - `LUT_1D_obj`, `LUT_2D_obj` — 1-D and 2-D lookup tables data respectively
   - `SignalGenerator` — Test input data

2. Set the **Default parameter behavior** configuration parameter to `Tunable`.

3 In the Code Mappings Editor (Embedded Coder) in **Data Defaults**, specify the storage class as `PageSwitching` for **Model parameters** under **Parameters**.



Note The model default setting for parameters sets the storage class as `PageSwitching`.

Set Up Parameters

Set up parameter tuning by using Simulink parameter objects.

1 In `slrt_ex_osc_cal`, on the **Modeling** tab, click **Design** > **Model Explorer**.
2 Select **Model Workspace** in the **Model Hierarchy** pane.
3 Make sure that the `Kg` parameter object exists and has these properties:

- **Value** — `400`
- **Data type** — `double`

4 If the parameter object does not exist, add it. On the toolbar, click the **Add Simulink Parameter** button.
5 Open `slrt_ex_osc_cal/Gain1`.
6 Make sure 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 `slrt_ex_osc_cal`, on the **Modeling** tab, click **Design** > **Model Explorer**.

2. Select **Model Workspace** in the **Model Hierarchy** pane.
3. Make sure that the `DampedOsc` signal object exists and has these properties:
   - **Minimum** — `-10`
   - **Maximum** — `10`
   - **Data type** — `double`
4. Make sure that the `SignalGenerator` signal object exists and has these properties:
   - **Minimum** — `-10`
   - **Maximum** — `10`
   - **Data type** — `double`
5. Make sure that the `L_1D` signal object exists and has these properties:
   - **Minimum** — `-15`
   - **Maximum** — `15`
   - **Data type** — `double`
6. Make sure that the `L_2D` signal object exists and has these properties:
   - **Minimum** — `-15`
   - **Maximum** — `15`
   - **Data type** — `double`
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. Make sure that you selected the **Signal name must resolve to Simulink signal object** and the **Test point** check boxes.

Set Up Lookup Tables

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

1. Open the block parameters for the 1-D Lookup Table block.
2. In the **Table and Breakpoints** pane, verify these settings:
   - **Number of table dimensions** — `1`
   - **Data specification** — `Lookup table object`
   - **Name** — `LUT_1D_obj`
3. Open the block parameters for the 2-D Lookup Table block.
4. In the **Table and Breakpoints** pane, check these settings:
   - **Number of table dimensions** — `2`
   - **Data specification** — `Lookup table object`
   - **Name** — `LUT_2D_obj`

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

Generate Data Description File

1. On the **REAL-TIME** tab, select **RUN ON TARGET > Build Application**. The build produces a file named `slrt_ex_osc_cal_slrt.mldatx` in the working folder containing A2L file.
2. To retrieve the A2L file and update target IP address in the A2L file, use `extractASAP2` command.
3. Connect to the target by using a third-party calibration tool.

See Also

n-D Lookup Table

More About

- “Calibrate Parameters with Vector CANape” on page 4-10
- “Calibrate Parameters with ETAS Inca” on page 4-14

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 4-3.

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 the real-time application `slrt_ex_osc_cal`.
2. Start the real-time application by selecting **REAL-TIME > RUN ON TARGET > Start Application**.
3. Disconnect the connection from MATLAB:

   ```
   tg = slrealtime
   disconnect(tg)
   ```

   You can now connect to third-party calibration tools.
4. Open Vector CANape.
5. Create a Vector CANape project with project name `slrt_ex_osc_cal`.

   Accept the default folder.

Prepare Device

1. From the extracted `slrt_ex_osc_cal.a2l`, create an XCP device named `slrt_ex_osc_cal_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 `slrt_ex_osc_cal_slrt`, and then open `slrt_ex_osc_cal.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`.

Measure Signals and Calibrate Parameters

1. Start the Vector CANape measurement.
2. To shorten the ring time on `DampedOsc`, `L_1D`, and `L_2D`, set parameter `Kg` to `800`.
3. As required, toggle between calibration RAM active and inactive.

When using the **Run on Target** button on the **Simulink Editor Real-Time** tab to run the simulation, there is a time lag of a couple of seconds between the start of the real-time application on the target computer and the connect model operation on the development computer. If you are examining signals at or very close to application start, consider using the step-by-step approach to connect model and then using an SSH connection (for example, using PuTTY) start the real-time application. For more information, see “Execute Real-Time Application in Simulink External Mode by Using Step-by-Step Commands” and “Execute Target Computer RTOS Commands at Target Computer Command Line” on page 8-3.

See Also

More About

- “Prepare ASAP2 Data Description File” on page 4-3
- “Vector CANape Limitations” on page 4-12
- “Troubleshoot Vector CANape Operation” on page 4-13

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 4-3. In addition to the limitations listed in “Vector CANape Limitations” on page 4-12, there are various issues that can prevent the 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 4-3
- “Vector CANape Limitations” on page 4-12

External Websites

- MathWorks Help Center website
- 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 4-3.

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 `slrt_ex_osc_cal`.
2. Start the real-time application by selecting **REAL-TIME > RUN ON TARGET > Start Application**.
3. Disconnect the connection from MATLAB:

   ```
   tg = slrealtime
   disconnect(tg)
   ```

   You can then connect to third-party calibration tools.
4. Open ETAS Inca.
5. Add an ETAS Inca database by using the folder named `SLRTDatabase`.
6. Add subfolders named `Experiment`, `Project`, and `Workspace`.

Prepare Project

1. Under folder `Project`, add an ECU project.
2. When prompted, select A2L file `slrt_ex_osc_cal.a2l`, which is extracted from the project file. 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 `slrt_ex_osc_cal_wksp`.
2. Add project `slrt_ex_osc_cal_slrt` to workspace `slrt_ex_osc_cal_wksp`.
3. When prompted, add an Ethernet system XCP device to the workspace.
4. Configure the XCP device and initialize it. Autoconfigure 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 `slrt_ex_osc_cal_exp`.
2. Add experiment `slrt_ex_osc_cal_exp` to workspace `slrt_ex_osc_cal_wksp`.

Configure Signals and Parameters

1. Start experiment `slrt_ex_osc_cal_exp`.
2. To create graphic controls for the variables, add variables `Kg`, `DampedOsc`, `SignalGenerator`, `L_1D`, and `L_2D`.
3. Add YT oscilloscopes for `DampedOsc`, `SignalGenerator`, `L_1D`, and `L_2D`.
4. For each signal, set the sample time to the base sample time of the real-time application (`250` µs).

Measure Signals and Calibrate Parameters

1. Start the ETAS Inca measurement.
2. To shorten the ring time on `DampedOsc`, `L_1D`, and `L_2D`, set parameter `Kg` to `800`.
3. As required, toggle between the reference page and the working page.

When using the **Run on Target** button on the **Simulink Editor Real-Time** tab to run the simulation, there is a time lag of a couple of seconds between the start of the real-time application on the target computer and the connect model operation on the development computer. If you are examining signals at or very close to application start, consider using the step-by-step approach to connect model and then using an SSH connection (for example, using PuTTY) start the real-time application. For more information, see “Execute Real-Time Application in Simulink External Mode by Using Step-by-Step Commands” and “Execute Target Computer RTOS Commands at Target Computer Command Line” on page 8-3.

See Also

More About

- “Prepare ASAP2 Data Description File” on page 4-3
- “ETAS Inca Limitations” on page 4-16
- “Troubleshoot ETAS Inca Operation” on page 4-17

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.
- Tunability of parameters with `ExportedGlobal` storage class when the model has other parameters with `PageSwitching` storage class. As a work around you can:
  - Place all the parameters you want to tune in model workspace. Or
  - Change the default mapping for storage class from `PageSwitching` to `default`. The `PageSwitching` storage class is not used, and the page switching functionality is not available.

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 4-3. In addition to the limitations listed in “ETAS Inca Limitations” on page 4-16, there are various issues that can prevent the 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`.

Transport Layer Failure

When a transport layer failure occurs, ETAS Inca can display this message:

```
ERROR: Transport Layer Failure, Inconsistent MsgCounter
```

This error appears in ETAS Inca when the incorrect setting is used for `'Counter Consistency Mode'`. Make sure that the `'Counter Consistency Mode'` is set to `'one counter for all CTOs+DTOs'` in the hardware settings for your experiment.

See Also

More About

- “Prepare ASAP2 Data Description File” on page 4-3
- “ETAS Inca Limitations” on page 4-16
- “Troubleshoot ETAS Inca Operation” on page 4-17

External Websites

- MathWorks Help Center website
- www.etas.com

Real-Time Application Setup

Real-Time Application Environment

Select Default Target Computer

When you start Simulink Real-Time Explorer for the first time, it opens a default target computer 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. To set a target computer node as the default:

1. Select a nondefault target computer node from the **Targets Tree** panel in Simulink Real-Time Explorer.
2. In the **Target Configuration** tab, select the **Default** checkbox.

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 you cannot delete it.

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

The `Targets` object manages collective and individual target computer environments. For more information, see “Set Up Target Computer Ethernet Connection” on page 5-3.

When you call the `Targets` object `getTargetSettings` function without arguments, the constructor gets the real-time environment settings for the default target computer.

```
my_tgs = slrealtime.Targets();
my_tgs_settings = getTargetSettings(my_tgs);
```

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

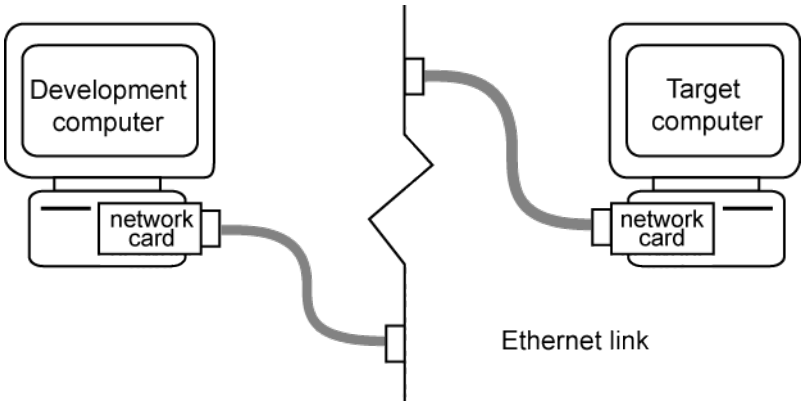
```
tg = slrealtime;
```

See Also

Simulink Real-Time Explorer | `Target` | `Targets` | `getDefaultTargetName` | `setDefaultTargetName` | `slrealtime`

Set Up Target Computer Ethernet Connection

To install PCI bus Ethernet protocol interface hardware in your Speedgoat target computer, see the Speedgoat website at www.speedgoat.com.



To configure the target computer Ethernet hardware:

1. If the target computer already contains one or more Ethernet cards, to get a list of these Ethernet cards, see your Speedgoat target machine documentation.
2. Assign a static IP address to the target computer Ethernet card by using Simulink Real-Time Explorer.

   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 by using an unshielded twisted-pair (UTP) cable.

   You can directly connect your computers by 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.

To build and download a real-time application by using the installed Ethernet card, 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

This procedure sets up Ethernet protocol for the default target computer `TargetPC1`:

1. Open Simulink Real-Time Explorer.
2. In the **Targets Tree** panel, select target computer `TargetPC1` and then click the **Change IP Address** button.
3. Configure the **New IP Address** and **New Netmask** fields in the Configure Target Computer IP Address dialog box. Click **OK**.

4 Click the **Disconnected** label, toggling it to **Connected**.

You can also configure the target computer Ethernet protocol by using MATLAB commands. For more information, see the `Targets` object functions and examples.

Target Computer Update, Reboot, and Startup Application

With Simulink Real-Time Explorer, you can update the target computer RTOS software, reboot the target computer, and configure a startup application that runs each time you start the target computer.

To update the target computer software:

1. Open Simulink Real-Time Explorer.
2. In the **Targets Tree** panel, select target computer `TargetPC1`.
3. To update the target computer RTOS software, click the **Update Software** button.
4. Click the **Disconnected** label, toggling it to **Connected**.

To reboot the target computer:

1. Open Simulink Real-Time Explorer.
2. In the **Targets Tree** panel, select target computer `TargetPC1`.
3. To reboot the target computer, click the **Reboot** button.
4. Click the **Disconnected** label, toggling it to **Connected**.

To configure a startup real-time application:

1. Open Simulink Real-Time Explorer.
2. In the **Targets Tree** panel, select target computer `TargetPC1`.
3. To load a real-time application on the target computer, click the **Load Application** button.
4. After you load the application, select the application from the **Applications on target computer** list and select the **Startup App** check.box. The next time the target computer starts or reboots, the application runs on startup.

See Also

`reboot` | `setStartupApp` | `update`

Signals and Parameters

Important prototyping tasks include:

- Changing parameters in your real-time application while it is running
- Viewing the resulting signal data
- Checking the results

The Simulink Real-Time software includes command-line and graphical user interfaces to complete these tasks.

Signal Monitoring Basics

Signal monitoring acquires real-time signal data without time information during real-time application execution. There is a minimal additional load on the real-time tasks.

You can monitor signals by using:

- Simulink Real-Time Explorer and the Simulation Data Inspector
- MATLAB language and the `Instrument` object
- Simulink external mode and a Scope block

See Also

`Instrument` | Scope

More About

- “Display and Filter Hierarchical Signals and Parameters” on page 6-57
- “View Data in the Simulation Data Inspector”
- “Troubleshoot Signals Not Accessible by Name” on page 6-61

Monitor Signals by Using Simulink Real-Time Explorer

This procedure uses the model `slrt_ex_osc`. You must have already completed this setup:

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

To monitor a signal in the real-time application:

1. In Simulink Real-Time Explorer, click **Load Application**. Select the `slrt_ex_osc` application from the **Applications on target computer** list and click **Load**.
2. Click the **Parameters** tab.
3. Select the signals to monitor from the list, and then click **Add to signals in instrument**  .
4. To start the real-time application, click **Start**.
5. To view the signals in the Simulation Data Inspector, click **Data Inspector**.
6. To stop execution, click **Stop**.

See Also

More About

- “Export and Import Signals in Instrument by Using Simulink Real-Time Explorer” on page 6-8
- “Display and Filter Hierarchical Signals and Parameters” on page 6-57
- “Troubleshoot Signals Not Accessible by Name” on page 6-61

Instrument a Stateflow Subsystem

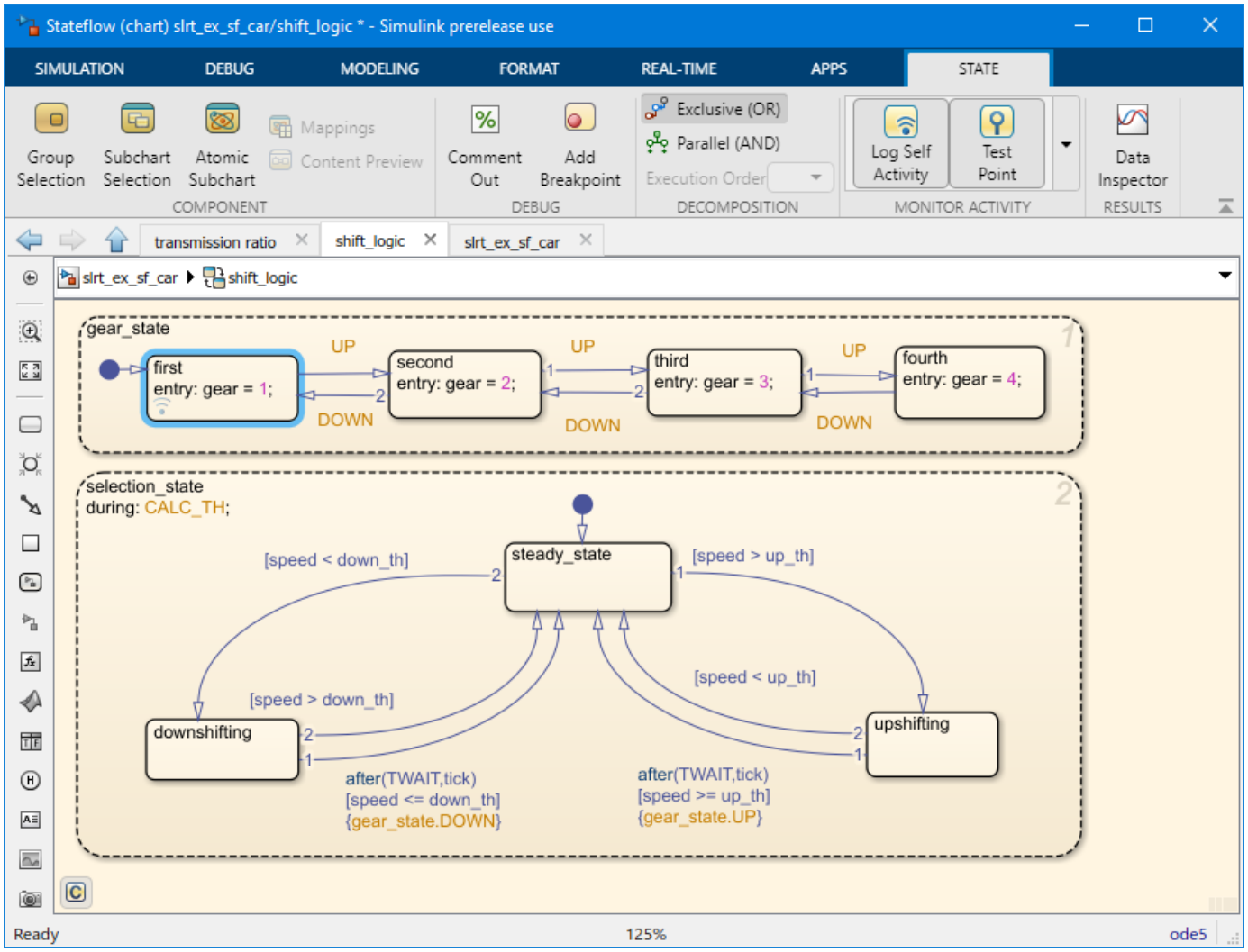
A Simulink Real-Time model that uses Stateflow blocks can provide visual confirmation that your chart behaves as expected.

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

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

To make Stateflow states available in the Simulation Data Inspector, select them and mark them for **Log Self Activity**:

1. Open the `slrt_ex_sf_car` model.
2. Double-click the `shift_logic` chart.



3. In the **gear_state** chart, select the **first** state
4. Click the **Log Self Activity** button and the **Test Point** button.

5 Repeat steps 3–4 for **gear_state** values **second**, **third**, and **fourth**.
6 Build and download the real-time application to the target computer. On the **Real-Time** tab, click **Run on Target**.
7 Monitor Stateflow states by using the Simulation Data Inspector. For more information, see "View Data in the Simulation Data Inspector" and "View State Activity by Using the Simulation Data Inspector" (Stateflow).

See Also

More About

- "View Data in the Simulation Data Inspector"
- "View State Activity by Using the Simulation Data Inspector" (Stateflow)
- "Animate Stateflow Charts with Simulink External Mode" on page 6-6

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 configure the Stateflow states for animation in the model. If you have not, see “Animate Stateflow Charts” (Stateflow). This example uses model `slrt_ex_sf_car`. To open the model, in the MATLAB Command Window, type:

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

1. 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:
   a. To direct the trigger to re-arm after the trigger event completes, set **Mode** to `normal`.
   b. To select the number of base rate steps for which external mode uploads data after a trigger event, in the **Duration** box, enter `5`.
   c. 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”.
5. Connect to the target computer. On the **Real-Time** tab, toggle the **Disconnected** indicator to **Connected**.
6. Build and download the model to the target computer. On the Real-Time tab, click **Run on Target**.
7. The simulation begins to run. You can observe the animation by opening the Stateflow Editor for your model.
8. To stop the simulation, on the **Real-Time** tab, click **Stop**.

See Also

More About

- “Animate Stateflow Charts” (Stateflow)
- “Configure Host Monitoring of Target Application Signal Data”
- “Simulink External Mode Interface”

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 by using the Simulation Data Inspector. You can upload the data from a File Log block to the development computer and display it using the Simulation Data Inspector.

You trace signals by marking the signals for logging or connecting the signals to File Log blocks. View the signals by using Simulink Real-Time Explorer, Simulink external mode, and the Simulation Data Inspector.

See Also

More About

- “Display and Filter Hierarchical Signals and Parameters” on page 6-57
- “View Data in the Simulation Data Inspector”
- “Troubleshoot Signals Not Accessible by Name” on page 6-61

Export and Import Signals in Instrument by Using Simulink Real-Time Explorer

When testing a complex model with many signals, you frequently must select signals for tracing or monitoring from multiple parts and levels of the model hierarchy. You can make this task easier by using Simulink Real-Time Explorer to select the signals in instrument and save the list of signals to disk.

This procedure uses the model `slrt_ex_osc`. You must have already completed this setup:

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

To add signals to the signals in instrument, export the list of signals, and import the list of signals:

1. In Simulink Real-Time Explorer, click **Load Application**. Select the `slrt_ex_osc` application from the **Applications on target computer** list and click **Load**.
2. Click the **Parameters** tab.
3. Select the signals to monitor from the list and then click **Add to signals in instrument**  .
4. To export the list, click **Export instrument to file**  . Name the files and click **Save**.
5. To remove signals from the signals in the instrument, select the signals in the list, and then click **Remove signals from instrument**  .
6. To import the list, click **Import instrument from file**  . Select the file and click **Open**.

When developing an App Designer application or an m-script that connects to a real-time application, it is helpful to have MATLAB code for the signals in the instrument. This code provides access to signals in an Instrument object (or instrumented signals), which are signals that are configured for streaming signal data from a real-time application.

To generate this code from the **Signals in Instrument**:

1. In Simulink Real-Time Explorer, click **Load Application**. Select the `slrt_ex_osc` application from the **Applications on target computer** list and click **Load**.
2. Click the **Signals** tab.
3. Select the signals to monitor from the list, and then click **Add to signals in instrument**  .
4. To create MATLAB code for the signals in the instrument, click **Generate MATLAB code to create Instrument programmatically**  . An editor window opens in MATLAB and displays the code for the signals in the Instrument.

See Also

`Instrument` | `addSignal` | `connectLine` | `connectScalar` | `validate`

More About

- “Monitor Signals by Using Simulink Real-Time Explorer” on page 6-3
- “Display and Filter Hierarchical Signals and Parameters” on page 6-57

Trace Signals by Using Simulink External Mode

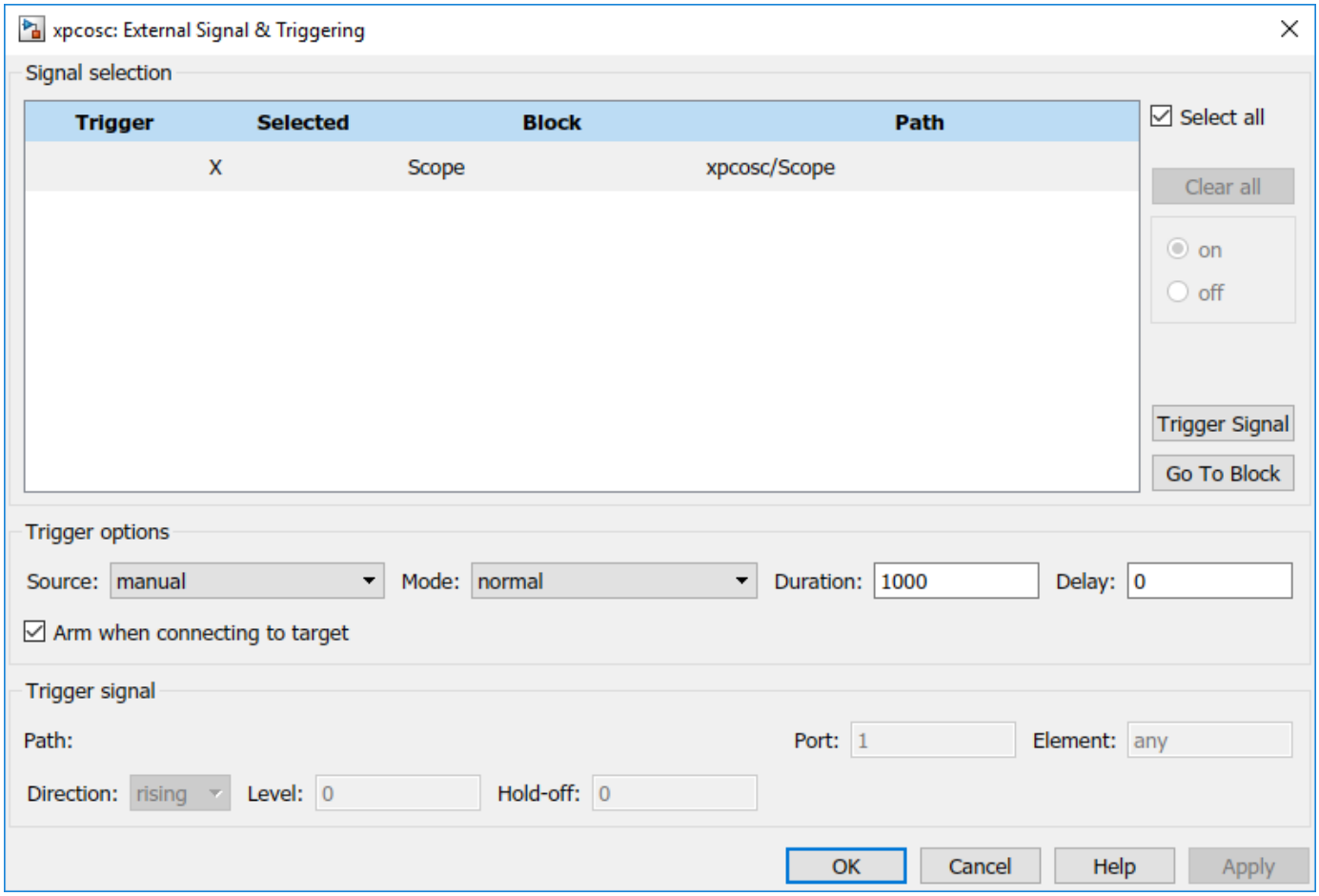
You can use Simulink external mode to establish a communication channel between your Simulink block diagram and your real-time application. The block diagram becomes a user interface to your real-time application. Simulink scopes can display signal data from the real-time application, including from models referenced inside a top model. You can control which signals to upload through the External Signal & Triggering dialog box. See “Select Signals to Upload” and “TCP/IP or Serial External Mode Control Panel”.

Note Do not use Simulink external mode while Simulink Real-Time Explorer is running. Use only one interface to control the real-time application.

This procedure uses model `slrt_ex_osc`. This model contains a Simulink Scope block.

To set up triggering for the external mode simulation:

1. Open model `slrt_ex_osc`.
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 **Signal & Triggering**.
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.

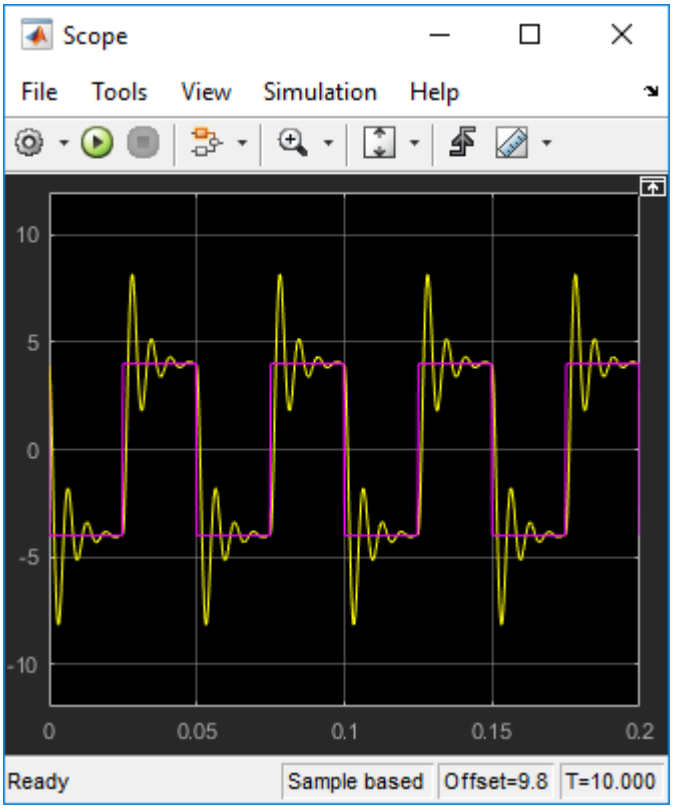


9 Click **Apply**, and then **Close**. In the External Mode Control Panel dialog box, click **OK**.

To set the stop time and run the simulation:

1 In the Simulink toolbar, increase the simulation stop time to, for example, `50`.
2 Save the model as `ex_slrt_ext_osc`. On the **Simulation** tab, from **Save**, click **Save As**.
3 If a scope window is not displayed for the Scope block, double-click the Scope block.
4 Connect to the target computer. On the **Real-Time** tab, toggle the **Disconnected** indicator to **Connected**.
5 Build and download the real-time application to the target computer. Click **Run on Target**.

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



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

See Also

Data Logging with Simulation Data Inspector (SDI)

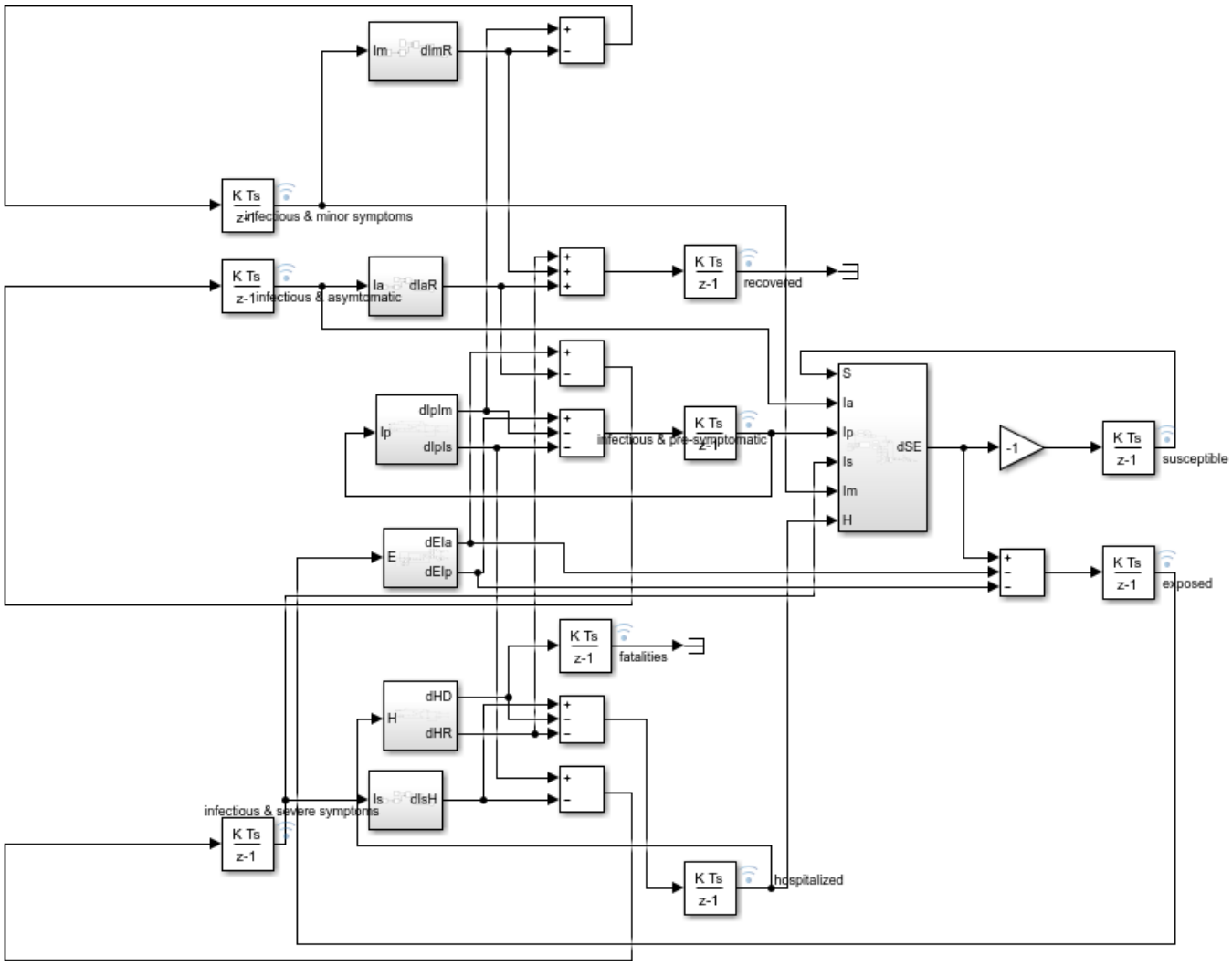
This example shows how to use a Simulink® Real-Time™ log of signal data and the Simulation Data Inspector. Signals are logged during model execution. At the end of the run, the Simulation Data Inspector interface displays the signal. This example show how to get the signals from the Simulation Data Inspectore interface by using the command line.

Open, Build, and Download the Model

Open the model `slrt_ex_soc_dist`. This model calibrates the control efforts through social distancing on an infectious disease outbreak.

Open the model.

```
mdl = 'slrt_ex_soc_dist';
mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if all(~strcmp(mdl, systems))
    mdlOpen = 1;
    open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_soc_dist.slx'));
end
```



Model slrt_ex_soc_dist
Simulink Real-Time example model

Build the model and download to the target computer:

- Configure for a non-Verbose build.
- Build and download application.

```
set_param(mdl,'RTWVerbose','off');
rtwbuild(mdl);
```

```
### Successful completion of build procedure for: slrt_ex_soc_dist
### Created MLDATX ..\slrt_ex_soc_dist.mldatx

Build Summary
```

```
Top model targets built:

Model             Action                        Rebuild Reason
=========================================================================================================
slrt_ex_soc_dist  Code generated and compiled  Code generation information file does not exist.

1 of 1 models built (0 models already up to date)
Build duration: 0h 1m 3.422s
```

- Close the model it is open.

```
if (mdlOpen)
    bdclose(mdl);
end
```

Run the Model to Evaluate Effects of No Social Distancing During the Outbreak

Using the Simulink Real-Time object variable, `tg`, load and start the model, and modify model parameters.

```
tg = slrealtime;
tg.load(mdl);
tg.setparam('','soc_dist_level',1);
tg.setparam('','thresh_int_level',1);
tg.start;
while ~strcmp(tg.status,'stopped')
    pause(5);
end
tg.stop;
```

Run the Model to evaluate the effect of social distancing during the outbreak.

Using the Simulink Real-Time object variable, `tg`, load and start the model, and modify model parameters

```
tg = slrealtime;
tg.load(mdl);
tg.setparam('','soc_dist_level',0.2);
tg.setparam('','thresh_int_level',0.2);
tg.start;
while ~strcmp(tg.status,'stopped')
    pause(5);
end
tg.stop;
```

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

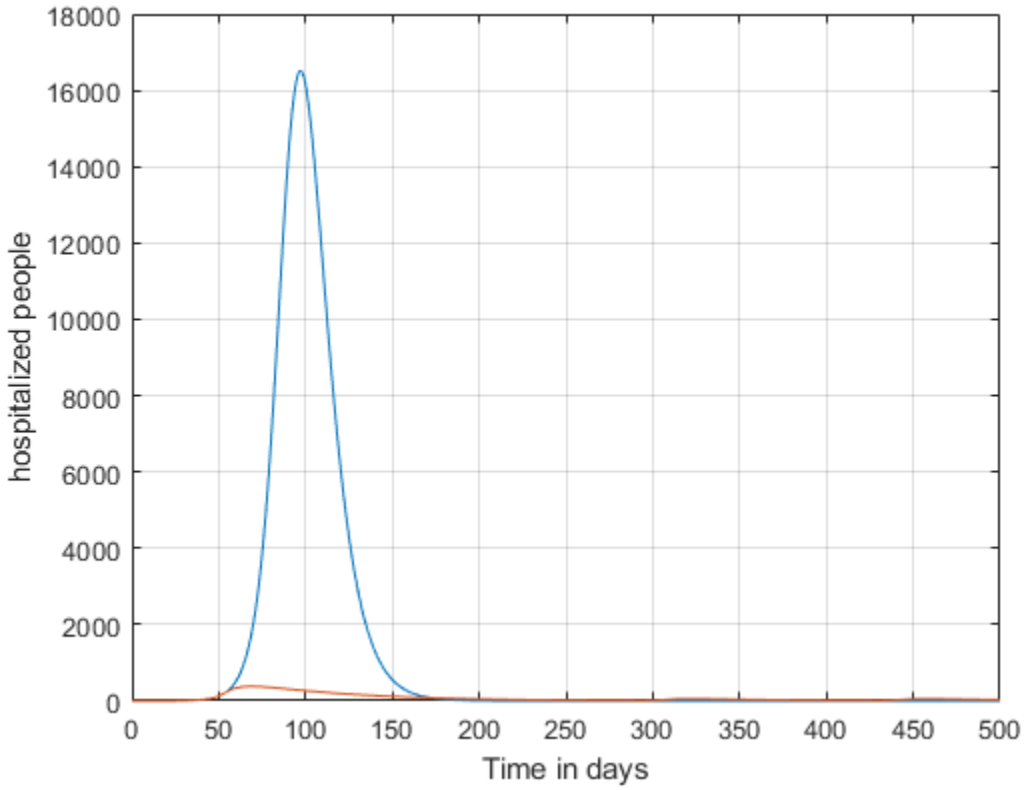
You can also retrieve the signal data from the SDI and plot the data by using these commands.

- Get all the runs
- Get the run information

- Get the signal.
- Get the signal objects.
- Take only infectious group values.
- Plot the signals.

The result shows that social distancing can reduce the number of hospitalized people

```
runIds = Simulink.sdi.getAllRunIDs();

for i = 1:length(runIds)
    run = Simulink.sdi.getRun(runIds(i));
    signalID = run.getSignalIDsByName('hospitalized');
    if ~isempty(signalID)
        signalObj = Simulink.sdi.getSignal(signalID);
        signalArray(:,i) = signalObj.Values(:,1).Data;
        timeValues = 100*(signalObj.Values(:,1).Time);
        plot(timeValues,signalArray);
        drawnow;
    end
end

grid on;
xlabel('Time in days'); ylabel('hospitalized people');
```



See Also

SLRT Overload Options | `slrtTETMonitor`

More About

- “Trace or Log Data with the Simulation Data Inspector” on page 6-22
- Simulation Data Inspector

Parameter Tuning and Data Logging

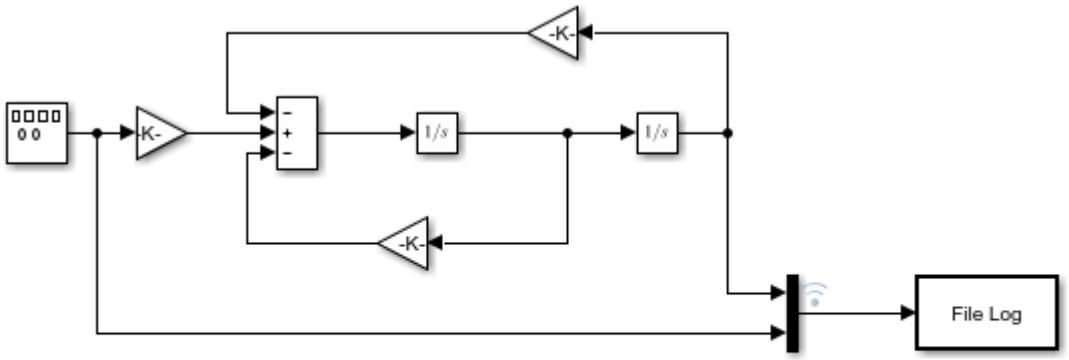
This example shows how to use real-time parameter tuning and data logging with Simulink® Real-Time™. After the example builds the model and downloads the real-time application, `slrt_ex_param_tuning`, to the target computer, the example executes multiple runs with the gain 'Gain1/Gain' changed (tuned) before each run. The gain sweeps from 0.1 to 0.7 in steps of 0.05.

The example uses the data logging capabilities of Simulink Real-Time to capture signals of interest during each run. The logged signals are uploaded to the development computer and plotted. A 3-D plot of the oscillator output versus time versus gain is displayed.

Open, Build, and Download Model to the Target Computer

Open the model, `slrt_ex_param_tuning`. The model configuration parameters select the `slrealtime.tlc` system target file as the code generation target. Building the model creates a real-time application, `slrt_ex_param_tuning.mldatx`, that runs on the target computer.

```
model = 'slrt_ex_param_tuning';
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples',model));
```



Simulink Real-Time example model

Build the model and download the real-time application, `slrt_ex_param_tuning.mldatx`, to the target computer.

- Configure for a non-Verbose build.
- Build and download application.

```
set_param(model,'RTWVerbose','off');
set_param(model,'StopTime','0.2');
rtwbuild(model);
tg = slrealtime;
load(tg,model);
```

```
### Successful completion of build procedure for: slrt_ex_param_tuning
### Created MLDATX ..\slrt_ex_param_tuning.mldatx

Build Summary
```

```
Top model targets built:

Model                 Action                       Rebuild Reason
=============================================================================================
slrt_ex_param_tuning  Code generated and compiled  Code generation information file does not exis

1 of 1 models built (0 models already up to date)
Build duration: 0h 0m 28.55s
```

Run Model, Sweep 'Gain' Parameter, Plot Logged Data

This code accomplishes several tasks.

Task 1: Create Target Object

Create the MATLAB® variable, `tg`, that contains the Simulink Real-Time target object. This object lets you communicate with and control the target computer.

- Create a Simulink Real-Time target object.
- 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 for each run.

- If no plot figure exist, create the figure.
- If the plot figure exist, make it the current figure.

Task 3: Loop over damping factor z

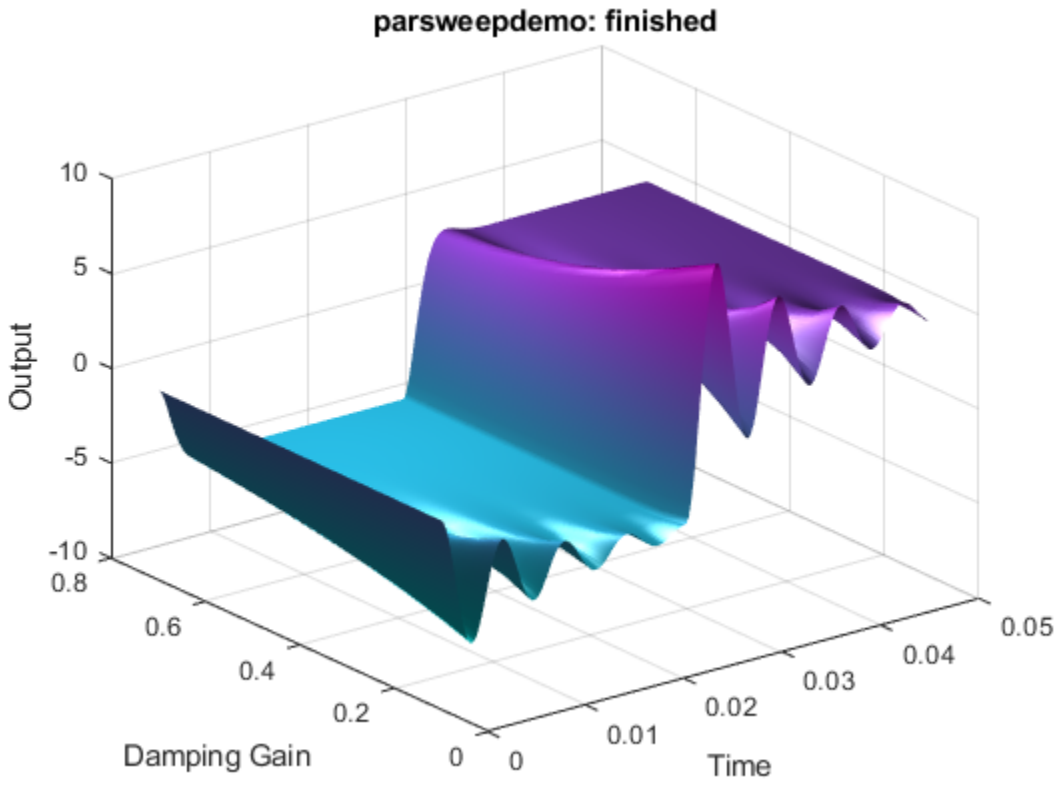
- Set damping factor (Gain1/Gain).
- Start run of the real-time application.
- Store output data in `outp`, `y`, and `t` variables.
- Plot data for current run.

Task 4: Create 3-D Plot (Oscillator Output vs. Time vs. Gain)

- Loop over damping factor.
- Create a plot of oscillator output versus time versus gain.
- Create 3-D plot.

```
figh = findobj('Name', 'parsweepdemo');
if isempty(figh)
  figh = figure;
  set(figh, 'Name', 'parsweepdemo', 'NumberTitle', 'off');
else
  figure(figh);
end
y = []; flag = 0;
for z = 0.1 : 0.05 : 0.7
  if isempty(find(get(0, 'Children') == figh, 1))
      flag = 1;
      break;
```

```
    end
    load(tg,model);
    tg.setparam([model '/Gain1'],'Gain',2 * 1000 * z);
    tg.start('AutoImportFileLog',true, 'ExportToBaseWorkspace', true);
    pause(0.4);
    outp = logsOut.FileLogSignals{1}.Values;
    y    = [y,outp.Data(:,1)];
    t    = outp.Time;
    plot(t,y);
    set(gca, 'XLim', [t(1), t(end)], 'YLim', [-10, 10]);
    title(['parsweepdemo: Damping Gain = ', num2str(z)]);
    xlabel('Time'); ylabel('Output');
    drawnow;
  end
  if ~flag
    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(model,0);
```

See Also

`slrtTETMonitor`

More About

- “Trace or Log Data with the Simulation Data Inspector” on page 6-22
- 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, you connect signals to File Log blocks in the model. In the real-time application, these blocks store data for the buffered signals on the target computer. At the end of execution, the real-time application transmits the data to the development computer for display in the Simulation Data Inspector. 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, see **Simulation Data Inspector**.

To set up the model for logging signal data:

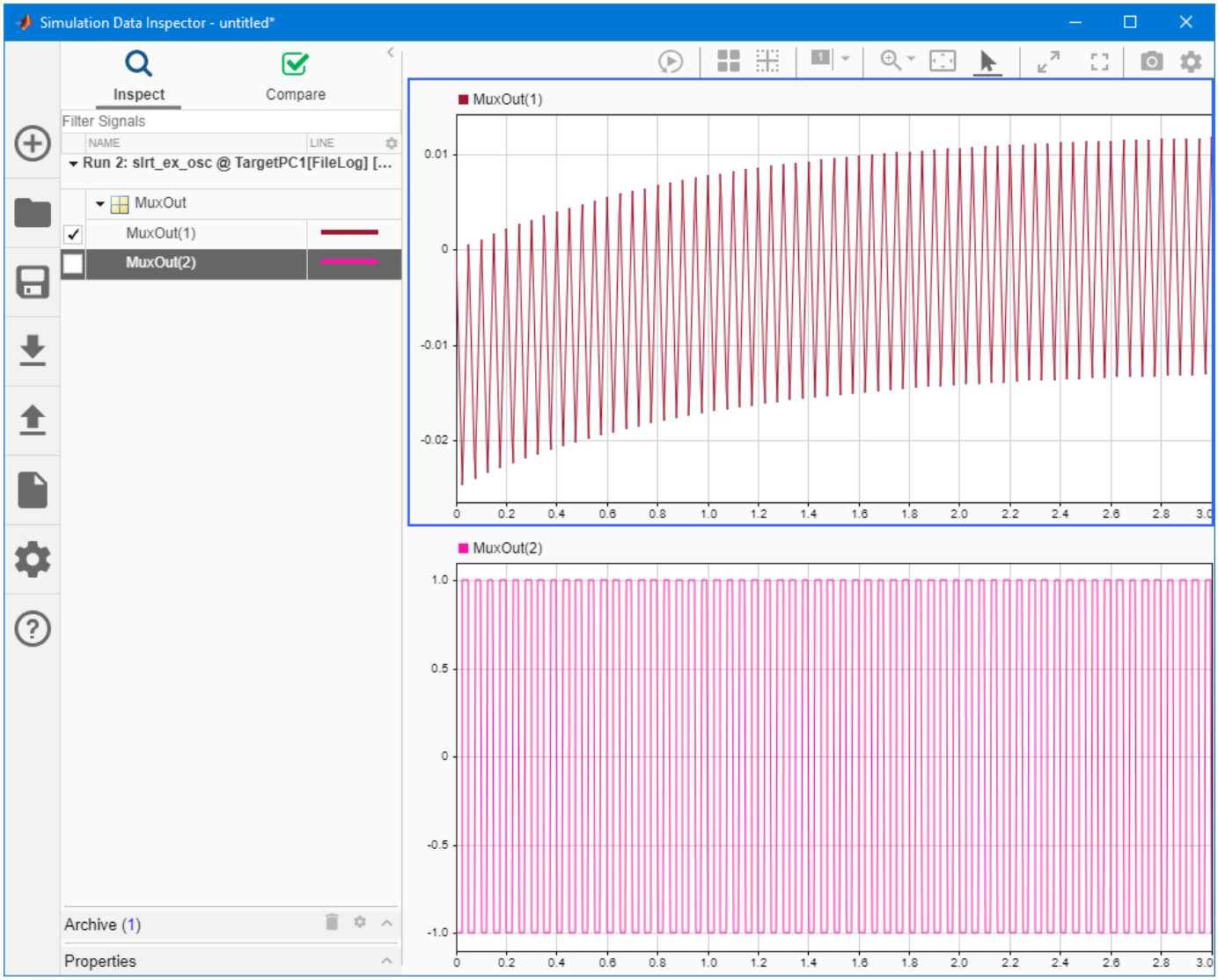
1. Open `slrt_ex_osc`.
2. Select the `MuxOut` output signal, place your cursor over the signal, and select **Enable Data Logging**.
3. Double-click the File Log block. The **Decimation** value is 1.

To set up the Simulation Data Inspector:

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

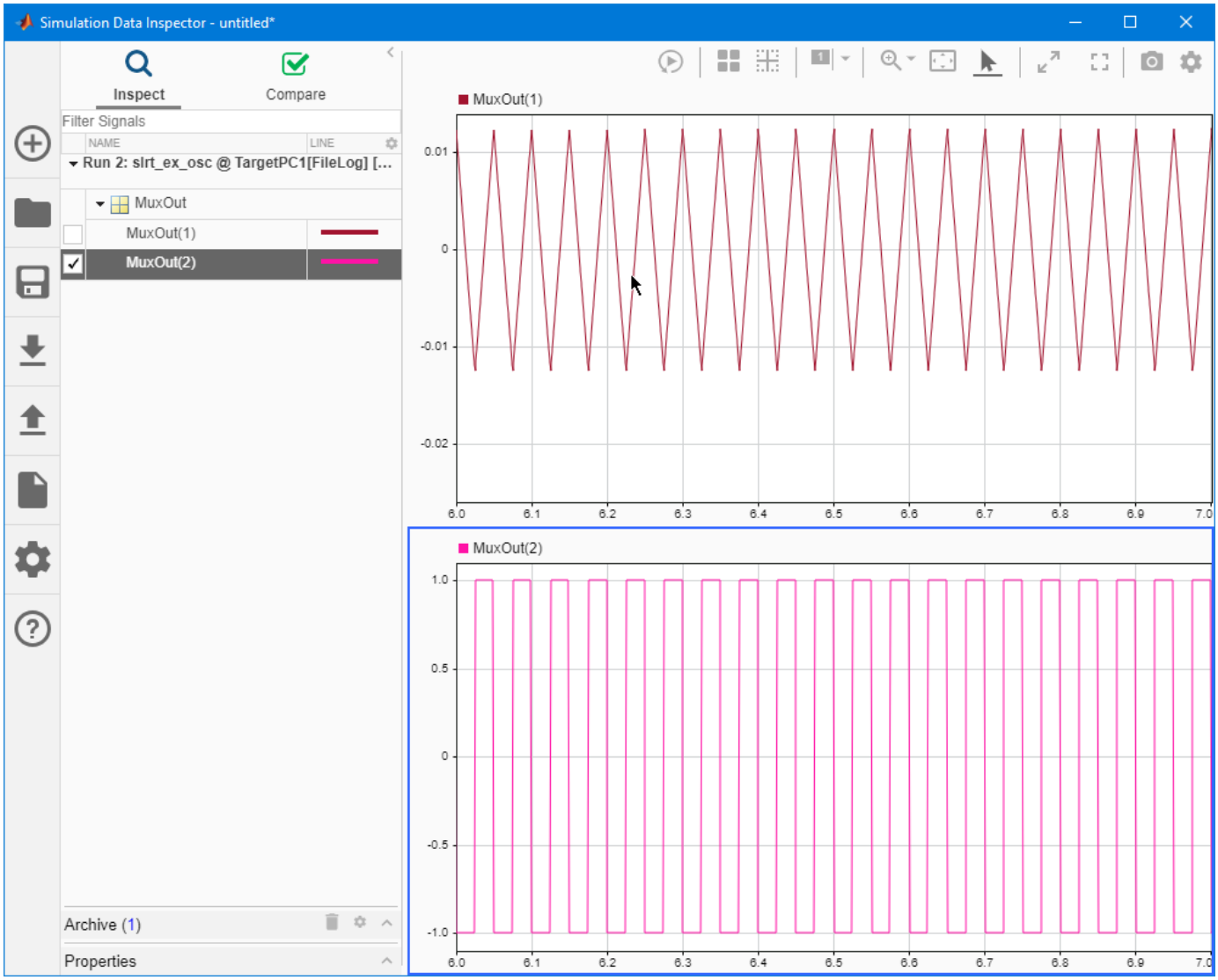
To view the simulation data:

1. Build and download `slrt_ex_osc`.
2. Start real-time execution.
3. When the Simulation Data Inspector button glows  , click the top display and select the `Sum` output signal.
4. Click in the bottom display and select the `Mux` output signals.

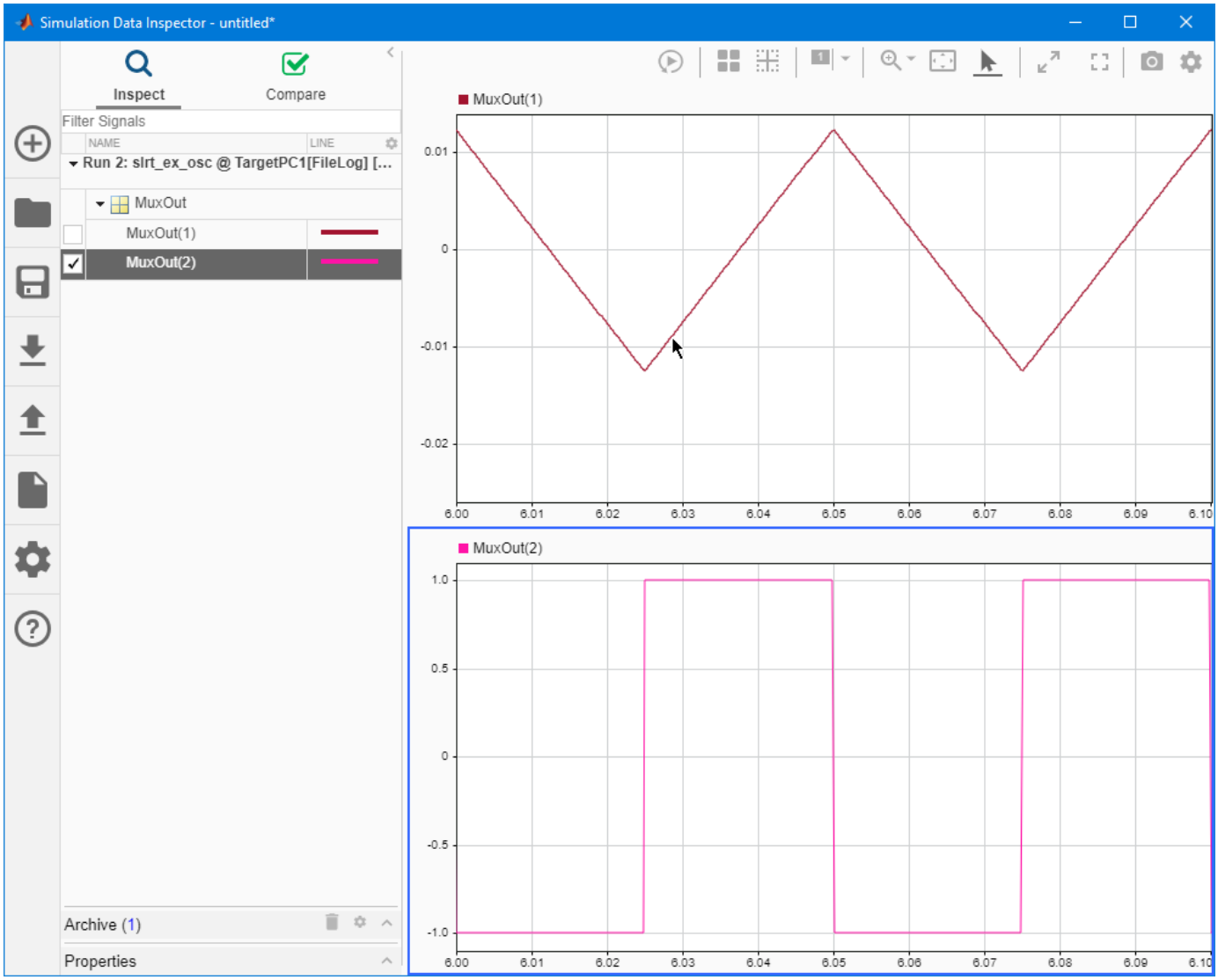


5 Stop real-time execution.

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



6 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.



7 To save the Simulation Data Inspector session as an MLDATX file, click **Save**.

See Also

More About

- “Data Logging with Simulation Data Inspector (SDI)” on page 6-13
- 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.

See Also

More About

- “Trace Signals by Using Simulink External Mode” on page 6-10
- “Trace or Log Data with the Simulation Data Inspector” on page 6-22
- “Simulink External Mode Interface”

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 the target computer to the 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.

- Simulink Real-Time Explorer works with multidimensional signals in column-major format.
- Some signals are not observable.

You can log signals by:

- Mark signals for immediate logging to the Simulation Data Inspector.
- Connect signals to File Log blocks for buffered logging to the Simulation Data Inspector.

See Also

More About

- “Troubleshoot Signals Not Accessible by Name” on page 6-61
- “Display and Filter Hierarchical Signals and Parameters” on page 6-57

Tune Parameters by Using Simulink Real-Time Explorer

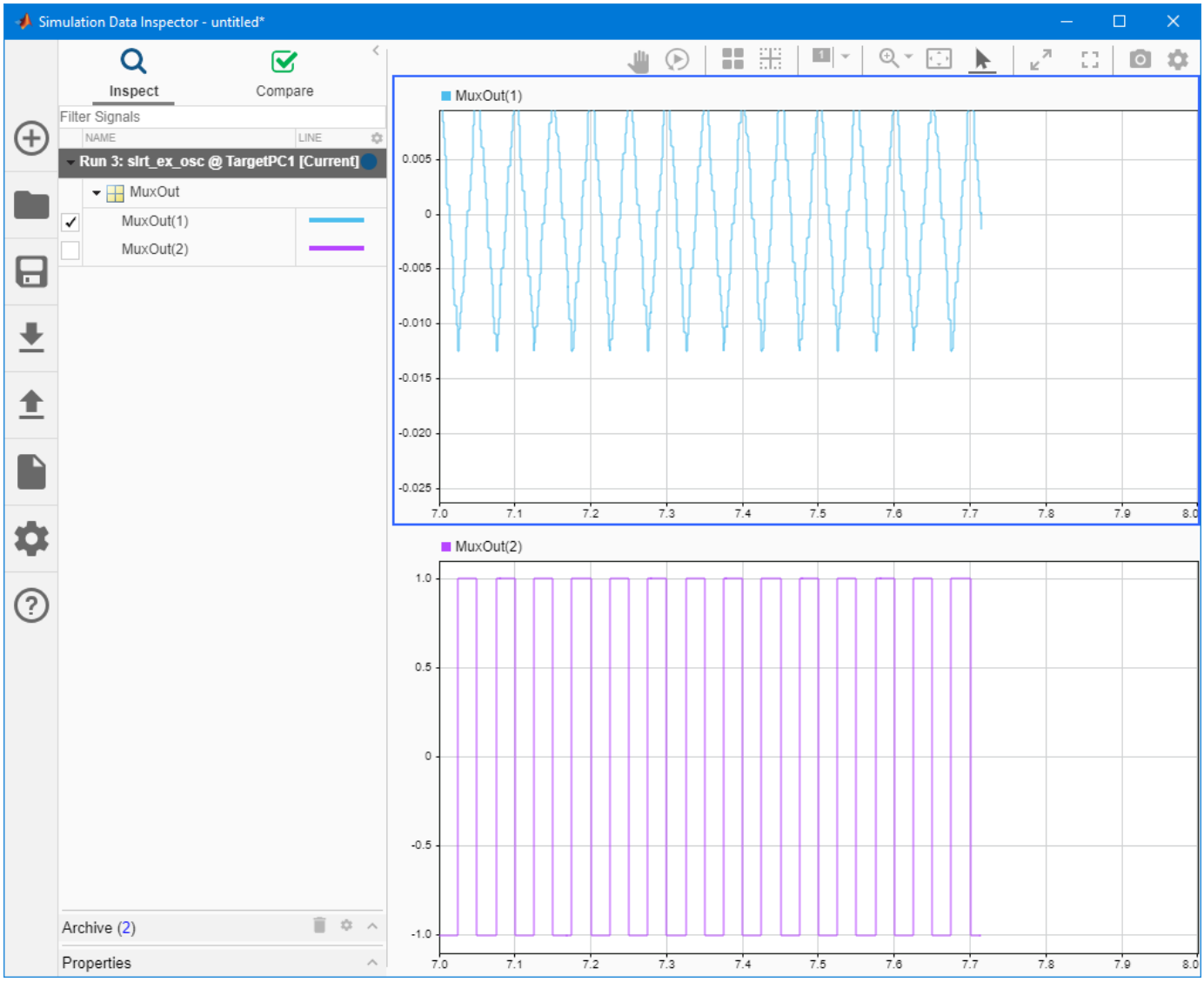
You can use Simulink Real-Time Explorer to change parameters in your real-time application while it is running or between runs. You do not need to rebuild the Simulink model, set the Simulink interface to external mode, or connect the Simulink interface with the real-time application.

This procedure uses the model `slrt_ex_osc`.

Set Up the Simulation Data Inspector

Before tuning parameter values, set up the Simulation Data Inspector:

1. Open the Simulation Data Inspector ().
2. Click **Layout** ().
3. Select two horizontal displays.
4. Open model `slrt_ex_osc`. Set property **Stop time** to `inf`. In the Simulink Editor, on the **Real-Time** tab, select **Run on Target** > **Stop Time** and set **Stop Time** to `inf`.
5. Connect to the target computer. Toggle the **Disconnected** indicator to **Connected**.
6. Build and download the real-time application to the target computer. Click **Run on Target**.
7. In the Simulation Data Inspector, drag the **MuxOut(1)** signal to the top display and drag the **MuxOut(2)** signal to the bottom display.



View Initial Parameter Values

To view the initial parameter values:

1. Open Simulink Real-Time Explorer. On the **Real-Time** tab, click **Prepare** > **SLRT Explorer**.
2. Select the **Parameters** tab. The tab lists parameters `Amplitude`, `Frequency`, `A`, and `C` with their values.

Modify Parameter Values

To update a parameter value:

1. Select the parameter value for the `Amplitude` parameter and change the value to `0.5`.
2. Select the parameter value for the `Frequency` parameter and change the value to `15`.

After each change, the signal display in the Simulation Data Inspector alters to match the effect of the parameter change. You can change multiple parameters at the same time by using batch mode. For more information, see “Tune Parameters by Using Batch Mode and Update All” on page 6-33.

See Also

More About

- “Simulink Real-Time Operation Modes”
- “Display and Filter Hierarchical Signals and Parameters” on page 6-57
- “Tune Parameters by Using Batch Mode and Update All” on page 6-33
- “Troubleshoot Parameters Not Accessible by Name” on page 6-62

Tune Parameters by Using MATLAB Language

To change block parameters, you can use the MATLAB functions. With these functions, you do not need to set the Simulink interface to external mode or 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 by using Simulink Real-Time functions.

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

This procedure uses the Simulink model `slrt_ex_osc`. You must have already created and downloaded the real-time application to the default target computer.

1. To create the target object and application object, in the MATLAB Command Window, type:

   ```
   tg = slrealtime('TargetPC1');
   app = slrealtime.Application('slrt_ex_osc');
   ```
2. The Parameters property of the Application object is a structure that includes a BlockPath and BlockParameterName for each parameter. To display the parameter name of the first of parameter in the real-time application, in the MATLAB Command Window, type:

   ```
   app.Parameters(1).BlockParameterName
   ```
3. To change the gain for 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 this 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

- “Simulink Real-Time Operation Modes”
- “Troubleshoot Parameters Not Accessible by Name” on page 6-62

Tune Parameters by Using Simulink External Mode

To connect your Simulink model to your real-time application, you use Simulink external mode simulation. The model becomes a user interface to your real-time application. 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 `slrt_ex_osc`. You must have already built and downloaded the real-time application for that model.

1. Open model `slrt_ex_osc`.
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. Click **Run on Target**.

   The real-time application begins running on the target computer.
4. From the Simulation block diagram, double-click the block labeled **Gain1**
5. In the Block Parameters: Gain1 parameter dialog box, in 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, click **Stop**.
7. Disconnect to the target computer. 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.

Tune Parameters by Using Batch Mode and Update All

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 `slrt_ex_osc`. To open this model, in the MATLAB Command Window, type:.

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

1. Open model `slrt_ex_osc`.
2. In the Simulink Editor, on the **Real-Time** tab, click **Run on Target**.
3. Click **Prepare** > **Batch Mode**. The editor remains in batch mode until you click **Batch Mode** again.

   To set parameter values, you can set values either by clicking each block or by using the Model Data Editor in the base workspace.
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 in the base workspace.
6 Click **Prepare > Update All Parameters**.
7 To stop the simulation before it ends, click **Stop**.

See Also

More About

- “Simulink Real-Time Operation Modes”
- “Troubleshoot Parameters Not Accessible by Name” on page 6-62

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 the 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 field in the block dialog box, and tune the MATLAB variable or object.

For more information about inlined parameters, see “Default parameter behavior”.

Tune Global Parameters by Using 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. For more information, see “Tune Parameters by Using Simulink External Mode” on page 6-33.

You can 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. Do one of the following:

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

Tune Global Parameters by Using 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. For more information, see “Tune Parameters by Using Simulink Real-Time Explorer” on page 6-28.

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

Tune Global Parameters by Using MATLAB Language

To change the values of tunable block parameters and tunable global parameters during execution, use the Simulink Real-Time command `setparam`. For more information, see “Tune Parameters by Using MATLAB Language” on page 6-31.

These code examples use the model `slrt_ex_osc`. 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:

```
rtwbuild(slrt_ex_osc);
tg = slrealtime('TargetPC1');
load(tg,'slrt_ex_osc')
start(tg);
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`:

```
rtwbuild(slrt_ex_osc);
tg = slrealtime('TargetPC1');
```

```
load(tg,'slrt_ex_osc')
start(tg);
setparam(tg, 'A.Ampl', 4.57)
```

See Also

`getparam` | `setparam`

More About

- “Tune Inlined Parameters by Using Simulink Real-Time Explorer” on page 6-38
- “Default parameter behavior”
- “Specify Source for Data in Model Workspace”
- “Troubleshoot Parameters Not Accessible by Name” on page 6-62
- “Tune and Experiment with Block Parameter Values”
- “Share and Reuse Block Parameter Values by Creating Variables”
- “How Generated Code Stores Internal Signal, State, and Parameter Data”
- “Preserve Variables in Generated Code”

Tune Inlined Parameters by Using Simulink Real-Time Explorer

This procedure describes how you can tune inlined parameters through the Simulink Real-Time Explorer.

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

The procedure starts with the Simulink model `slrt_ex_osc_inlined`. To open the model, in the MATLAB Command Window, type:

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

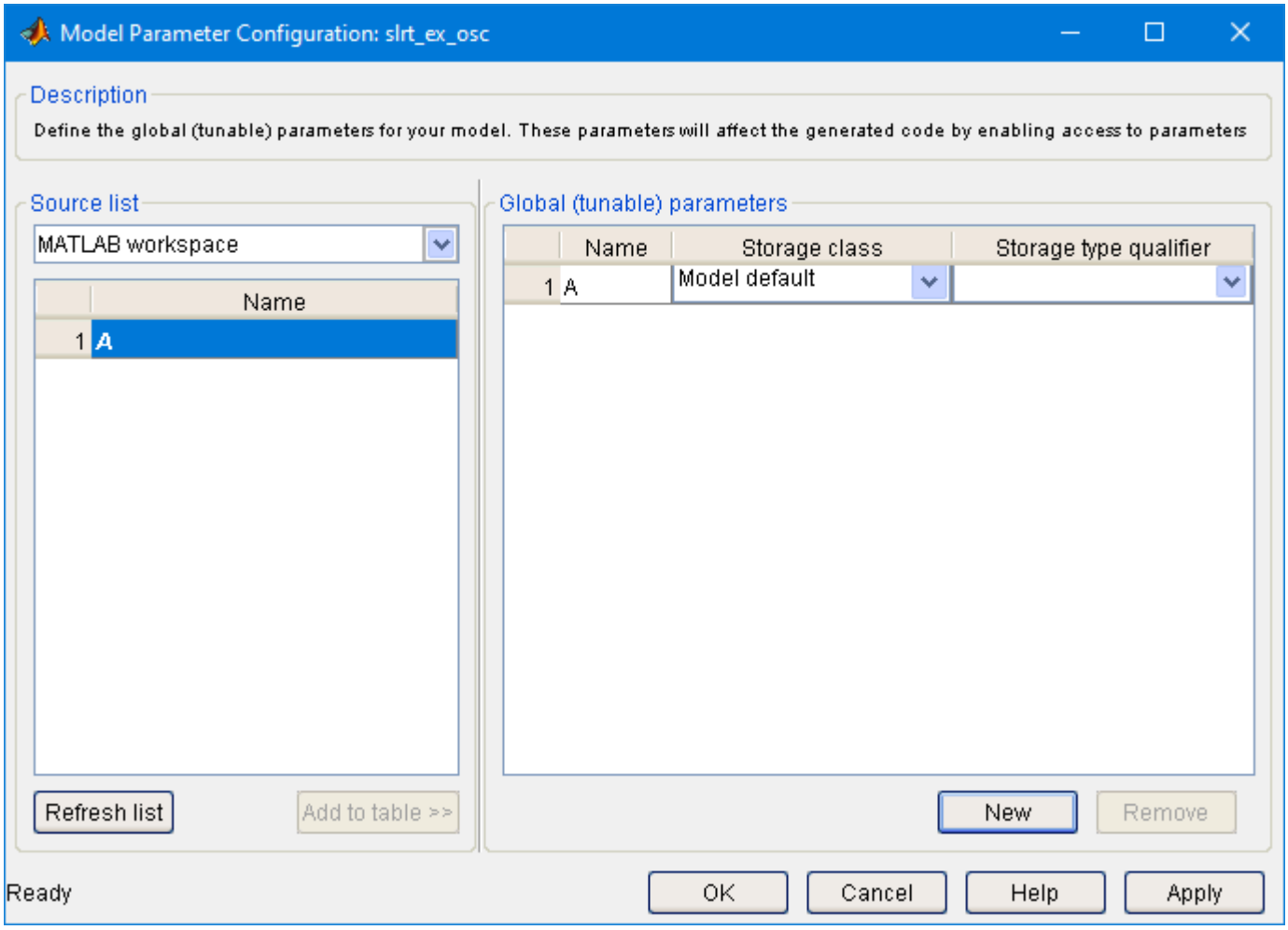
Configure Model to Tune Inlined Parameters

1. Open model `slrt_ex_osc_inlined`.
2. In the Simulink Editor, select the input to the Scope block and mark it for data logging by using 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 MATLAB Command Window, type:

      ```
      A = 4
      ```

      The value is displayed in the MATLAB workspace.
4. Open the Configuration Parameters dialog box. 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**.

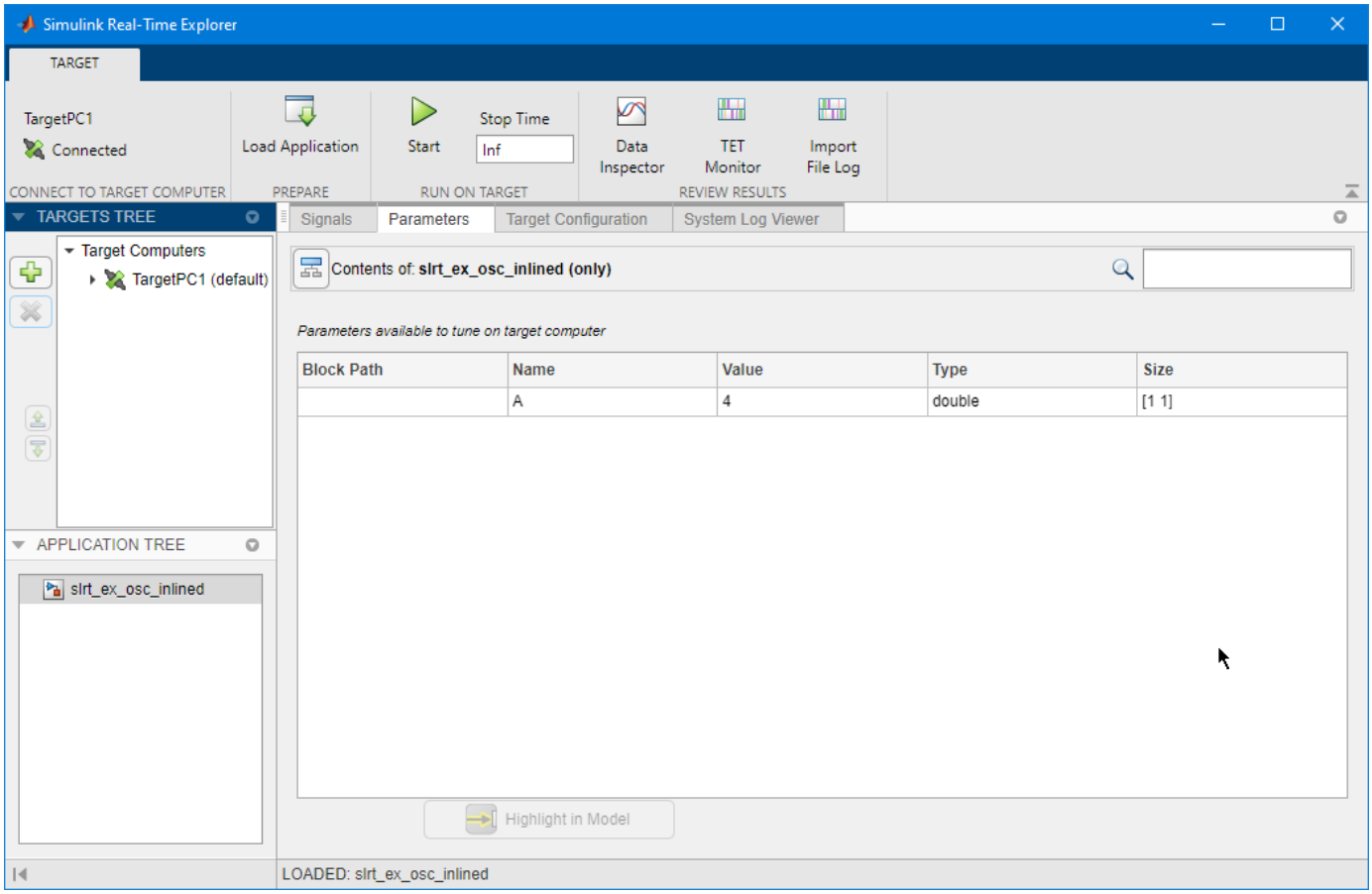


8 Click **Apply**, and then click **OK**.
9 In the Configuration Parameters dialog box, click **Apply**, and then **OK**.
10 Save the model as `slrt_ex_osc_inlined`. On the **Simulation** tab, from **Save**, click **Save As**. For example, save it as `slrt_ex_osc_inlined`.
11 Build and download the model to your target computer. On the **Real-Time** tab, click **Run on Target**.

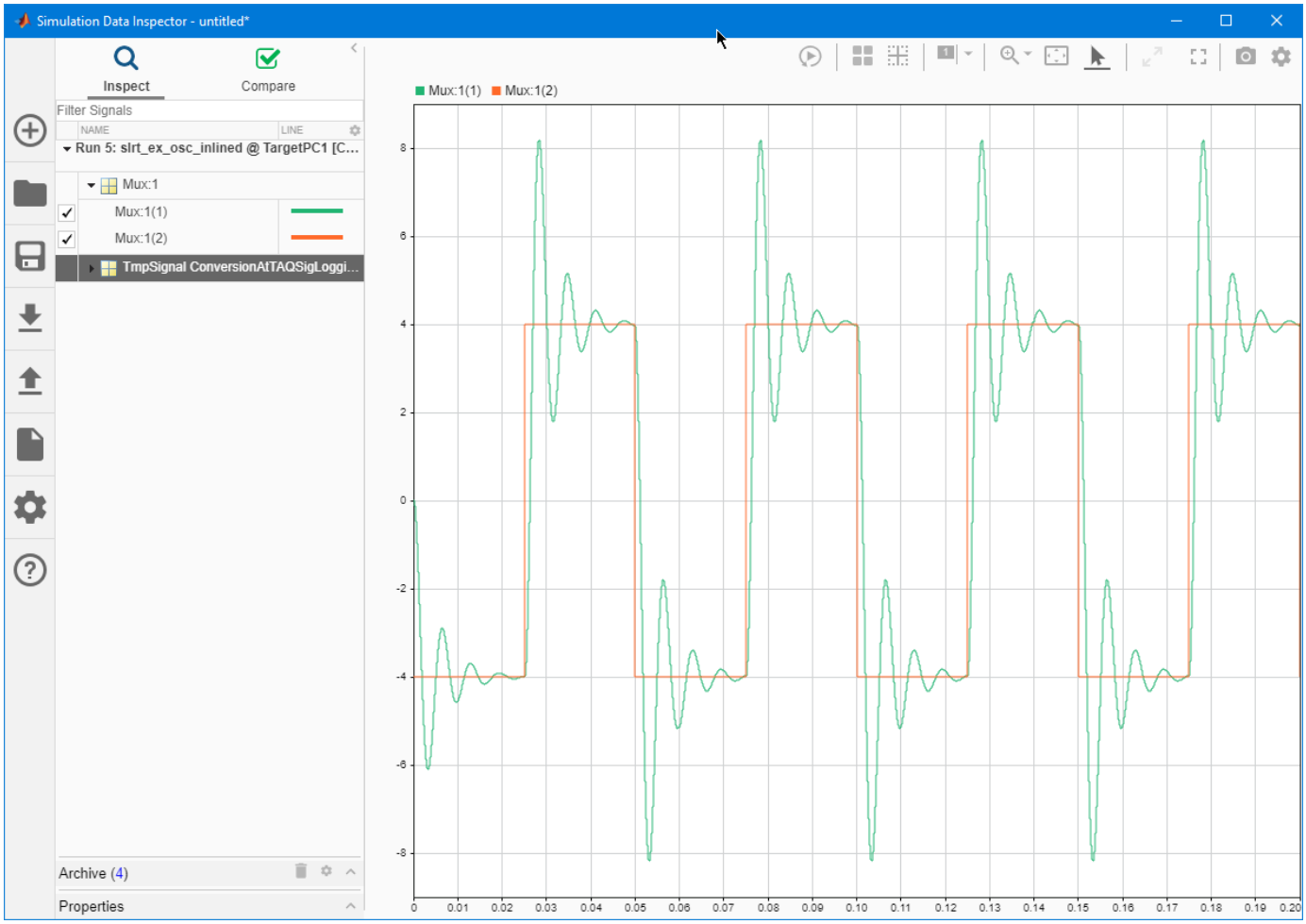
Initial Value

This procedure assumes that you have completed the steps in “Configure Model to Tune Inlined Parameters” on page 6-38.

1 Open Simulink Real-Time Explorer. On the **Real-Time** tab, click **Prepare > SLRT Explorer**.
2 Load the `slrt_ex_osc_inlined` real-time application. Click **Load Application**, select the application, and click **Load**.
3 Set the application stop time to **inf**.
4 To start execution, click **Start**.
5 In the **Applications** pane, expand both the real-time application node and the **Model Hierarchy** node.
6 Select the **Parameters** tab.



7 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 6-39.

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**.
2. The Simulation Data Inspector display changes to show the new signal amplitude.
3. To stop execution, click **Stop**.

See Also

More About

- “Tune Inlined Parameters by Using MATLAB Language” on page 6-42
- “Display and Filter Hierarchical Signals and Parameters” on page 6-57
- “Troubleshoot Parameters Not Accessible by Name” on page 6-62

Tune Inlined Parameters by Using MATLAB Language

You can tune inlined parameters through the MATLAB interface.

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

You must have already built and downloaded the model `slrt_ex_osc_inlined`. To open this model, in the MATLAB Command Window, type:

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

With the real-application `slrt_ex_osc_inlined` already running, you can tune inlined parameter `A` by using the `setparam` function.

1. Save the following code in a MATLAB file. For example, `change_inlineA`.

   ```
   A = 4;
   open_system(fullfile(matlabroot, 'toolbox', 'slrealtime', ...
   'examples', 'slrt_ex_osc_inlined'));
   rtwbuild('slrt_ex_osc_inlined');
   tg = slrealtime;
   load(tg,'slrt_ex_osc_inlined');
   setparam(tg,'','A',2);
   ```
2. Execute that MATLAB file. Type:

   ```
   change_inlineA
   ```
3. To see the new parameter value, type:

   ```
   getparam(tg,'','A')
   ```

See Also

More About

- “Troubleshoot Parameters Not Accessible by Name” on page 6-62

Tune Parameter Structures by Using Simulink Real-Time Explorer

In this section...

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”.

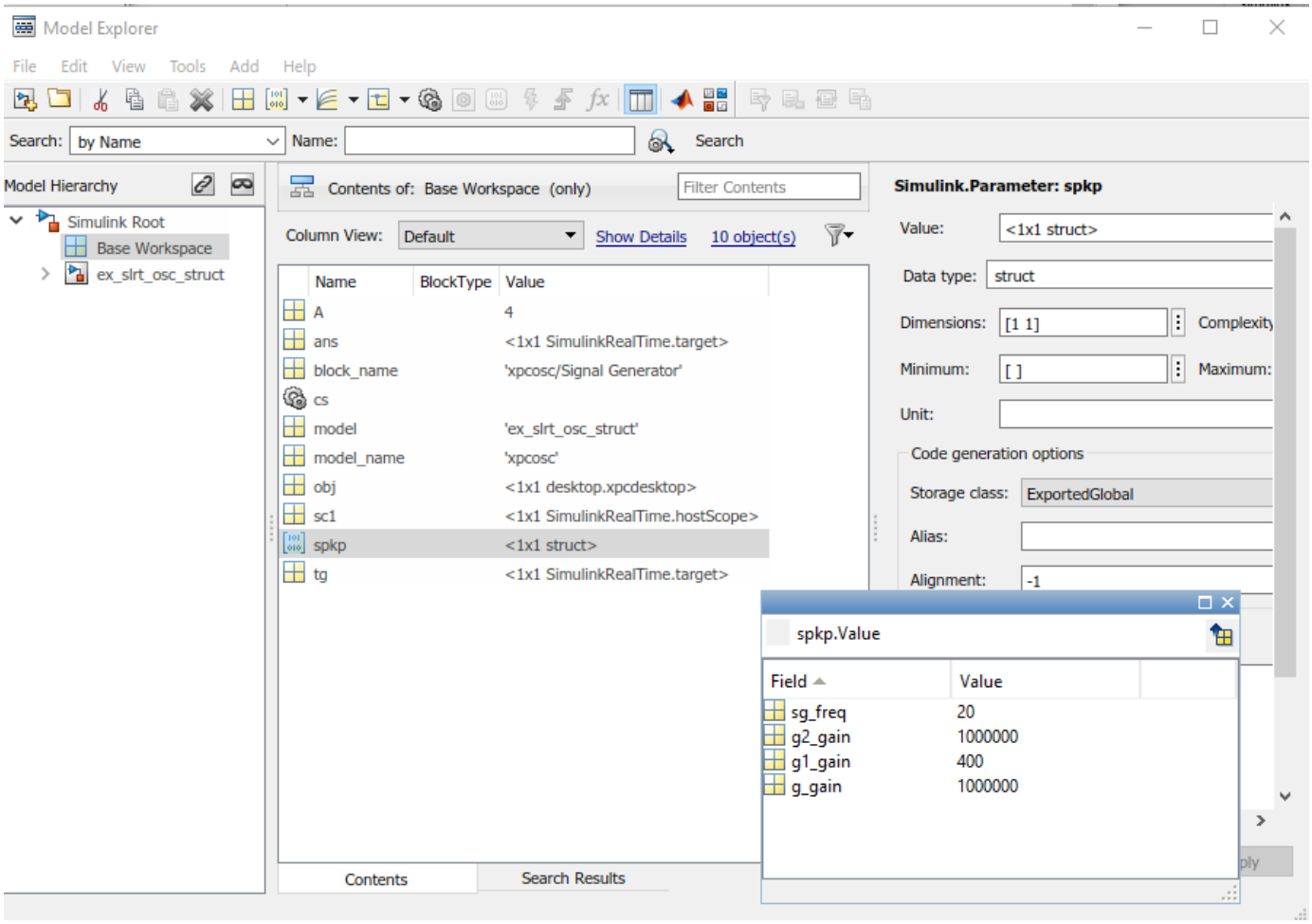
In this example, the initial model `slrt_ex_osc` has four parameters that determine the shape of the output waveform.

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

Create Parameter Structure

1. Open model `slrt_ex_osc`, 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 **Add Simulink Parameter**  .
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)
   ```
7. 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 **Edit**  .



8 Click **Apply**.
9 Save the model as `slrt_ex_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`.

Save and Load Parameter Structure

1 In Model Explorer, right-click row `spkp`.
2 Click **Export selected** and save the variable as `slrt_ex_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”.

Tune Parameters in a Parameter Structure

If you have not completed the steps in “Create Parameter Structure” on page 6-43, “Replace Block Parameters with Parameter Structure Fields” on page 6-44, and “Save and Load Parameter Structure” on page 6-44, you can start by using the completed model. To open the model, in the MATLAB Command Window, type:

```
open_system(fullfile(matlabroot, 'toolbox', 'slrealtime', ...
'examples', 'slrt_ex_osc_struct'));
load(fullfile(matlabroot, 'toolbox', 'slrealtime', ...
'examples', 'slrt_ex_osc_struct.mat'));
```

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. Set the real-time application **Stop Time** to `Inf`.
4. Click the **Parameters** tab.
5. Start the real-time application.
6. Open the Simulation Data Inspector and view the signals from the real-time application.
7. In the **Values** text box for `spkp(1).g1_gain`, change the value to `800` and press **Enter**.
8. Observe the change to the signals in the Simulation Data Inspector.
9. Stop the real-time application.

See Also

More About

- “Organize Related Block Parameter Definitions in Structures”
- “Display and Filter Hierarchical Signals and Parameters” on page 6-57
- “Model Callbacks”

Tune Parameter Structures by Using MATLAB Language

In this section...

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”.

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

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

Create Parameter Structure

1. Open model `slrt_ex_osc` 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
```

Save and Load Parameter Structure

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

```
save 'slrt_ex_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”.

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

If you have not completed the steps in “Create Parameter Structure” on page 6-46, “Replace Block Parameters with Parameter Structure Fields” on page 6-47, and “Save and Load Parameter Structure” on page 6-47, you can start by using the completed model. To open the model, in the MATLAB Command Window, type:

```
open_system(fullfile(matlabroot, 'toolbox', 'slrealtime', ...
'examples', 'slrt_ex_osc_struct'));
load(fullfile(matlabroot, 'toolbox', 'slrealtime', ...
'examples', 'slrt_ex_osc_struct.mat'));
```

1. Build and download the model to the target computer.

   ```
   rtwbuild('slrt_ex_osc_struct');
   tg = slrealtime('TargetPC1');
   load(tg,'slrt_ex_osc_struct');
   ```
2. Set stop time to `inf`.

   ```
   setStopTime(tg,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. View the signals in the Simulation Data Inspector.

   ```
   Simulink.sdi.view;
   ```

See Also

More About

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

Define and Update Inport Data

In this section...

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

Required Files

This procedure has these file dependencies:

- `slrt_ex_osc_inport` — Damped oscillator that takes its input data from input port `In1` and sends its multiplexed output to output port `Out1`. To open this model, in the MATLAB Command Window, type:

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

- `slrt_ex_inport_square.mat`— One second of output from a Signal Generator block that is configured to output a square wave. To load this data, in the MATLAB Command Window, type:

  ```
  (load(fullfile(matlabroot, 'toolbox', 'slrealtime', ...
  'examples', 'slrt_ex_inport_square.mat'))
  ```

- `slrt_ex_inport_sawtooth.mat` — One second of output from a Signal Generator block that is configured to output a sawtooth wave. To load this data, in the MATLAB Command Window, type:

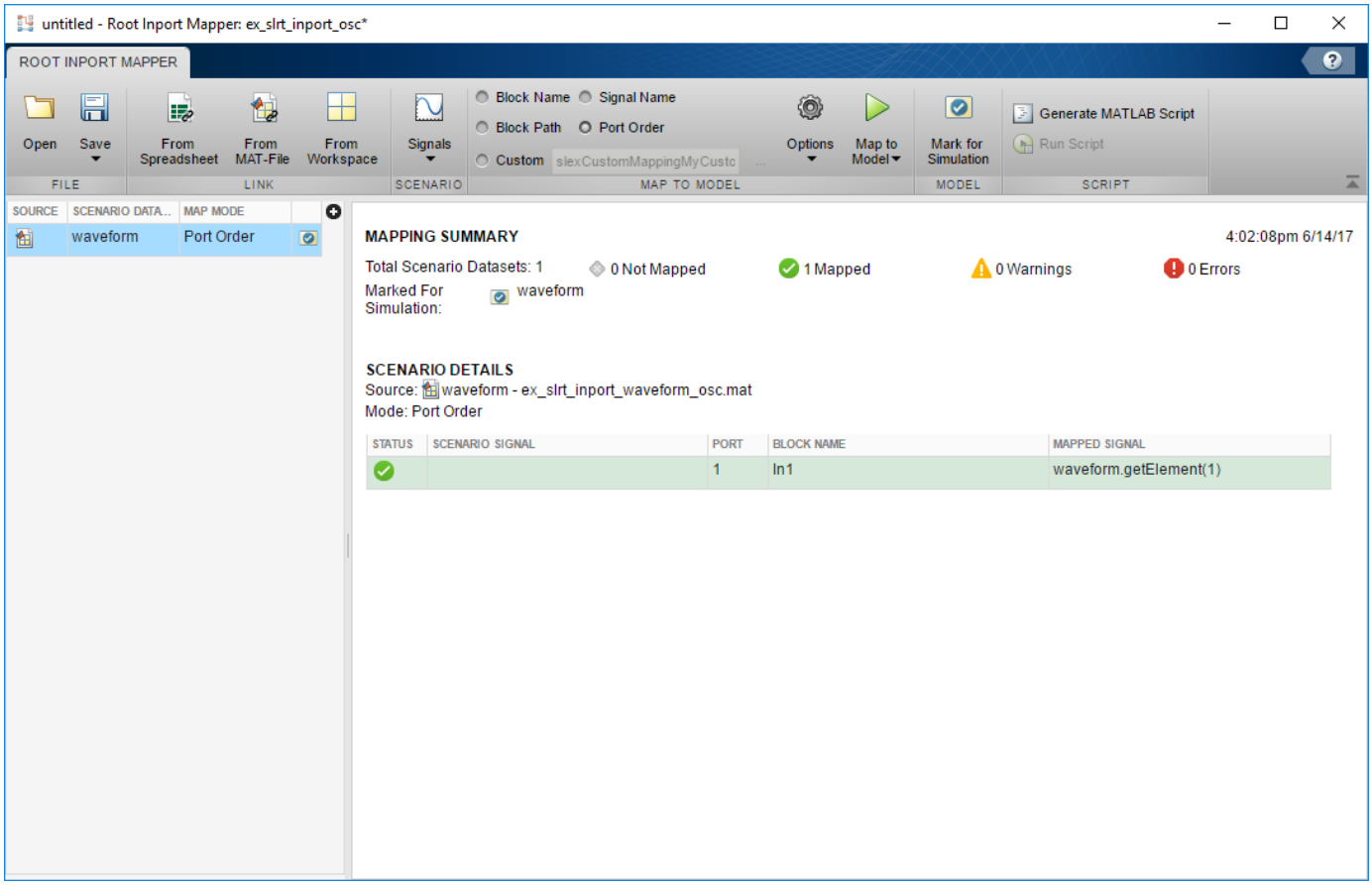
  ```
  (load(fullfile(matlabroot, 'toolbox', 'slrealtime', ...
  'examples', 'slrt_ex_inport_sawtooth.mat'))
  ```

Before starting this procedure, navigate to a working folder.

Map Inport to Use Square Wave

1. Open model `slrt_ex_osc_inport` and save a copy to a working folder.
2. Load `slrt_ex_inport_square.mat` and assign `square` to a temporary workspace variable for use with the Root Inport Mapper.

   ```
   waveform = square;
   ```
3. Double-click input port `In1`.
4. Clear **Interpolate data**, and then click **Connect Input**.
5. In the 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 the map to model option **Port order** and, from 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**.



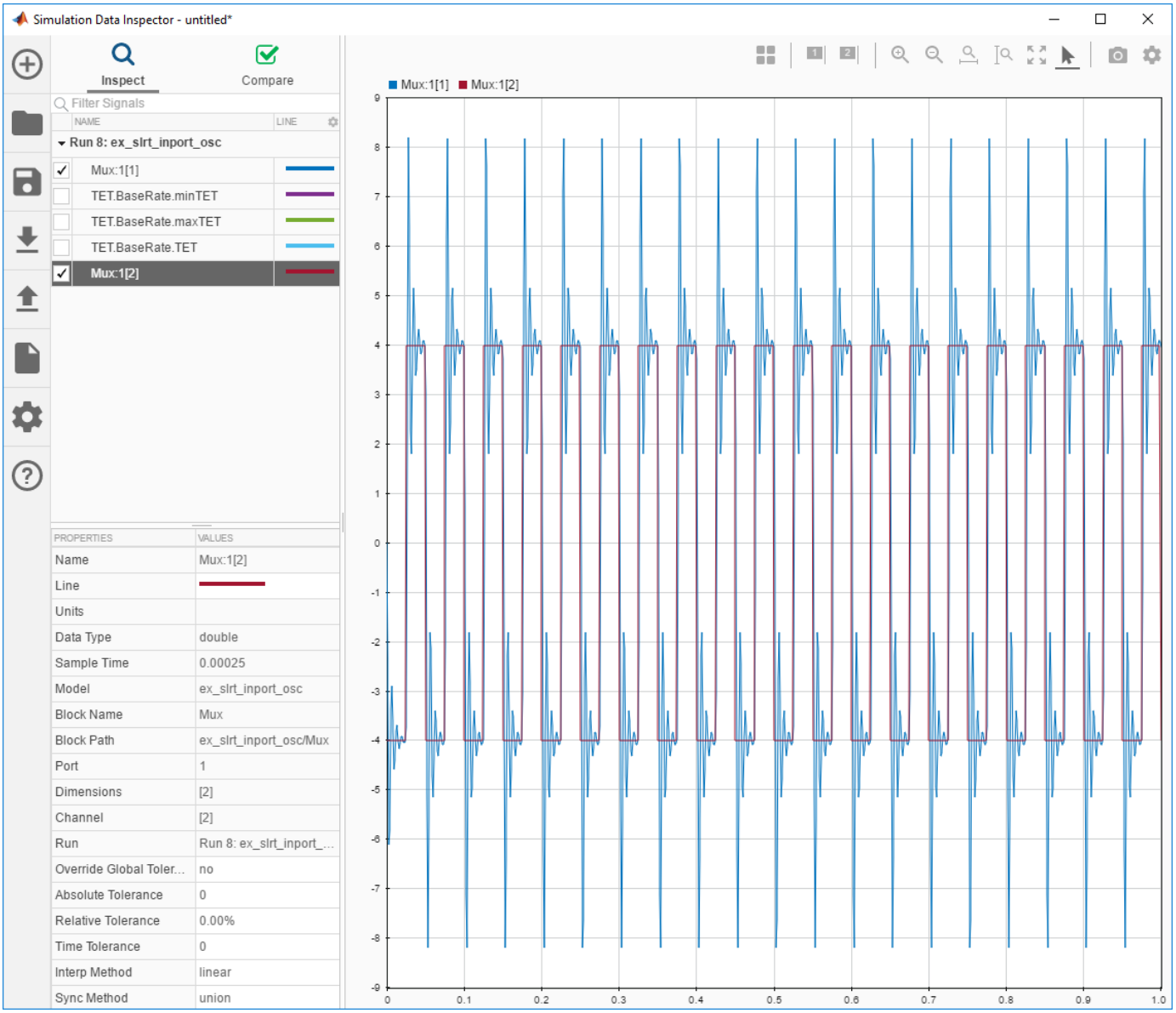
10 Click **Save**.

Save the scenario under a name such as `slrt_ex_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 by using 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 `slrt_ex_osc_inport.mldatx` file must be in the working folder.

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

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

2. Create an application object.

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

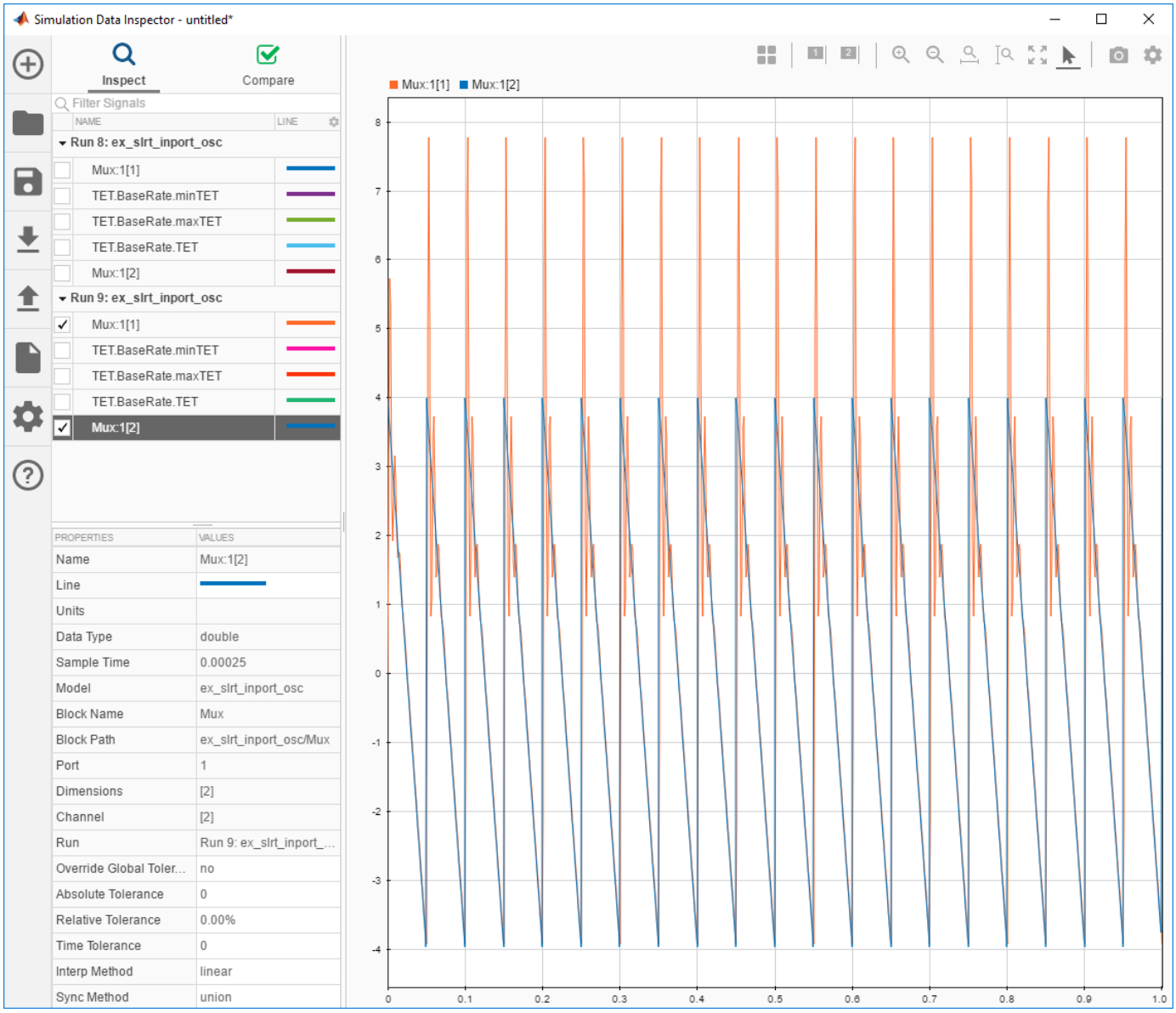
3 Update the application object.

```
updateRootLevelInportData(app_object);
```

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

```
tg = slrealtime;
load(tg, 'slrt_ex_osc_inport');
start(tg);
```

5 Display the output by using the Simulation Data Inspector.



See Also

More About

- “Define and Update Inport Data by Using MATLAB Language” on page 6-54
- “Load Data to Root-Level Input Ports”
- “Inport Data Mapping Limitations” on page 6-56
- “Data Logging with Simulation Data Inspector (SDI)” on page 6-13

Define and Update Inport Data by Using MATLAB Language

In this section...

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.

Required Files

This procedure has these file dependencies:

- `slrt_ex_osc_inport` — Damped oscillator that takes its input data from input port `In1` and sends its multiplexed output to output port `Out1`. To open this model, in the MATLAB Command Window, type:

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

- `slrt_ex_inport_square.mat`— One second of output from a Signal Generator block that is configured to output a square wave. To load this data, in the MATLAB Command Window, type:

  ```
  (load(fullfile(matlabroot, 'toolbox', 'slrealtime', ...
  'examples', 'slrt_ex_inport_square.mat'))
  ```

- `slrt_ex_inport_sawtooth.mat` — One second of output from a Signal Generator block that is configured to output a sawtooth wave. To load this data, in the MATLAB Command Window, type:

  ```
  (load(fullfile(matlabroot, 'toolbox', 'slrealtime', ...
  'examples', 'slrt_ex_inport_sawtooth.mat'))
  ```

Before starting this procedure, navigate to a working folder.

Map Inport to Use Square Wave

1. Open `slrt_ex_osc_inport`.

   ```
   model = fullfile(matlabroot, 'toolbox', 'slrealtime', ...
   'examples', 'slrt_ex_osc_inport'));
   open_system(model);
   save_system(model,'H:\workdir\slrt_ex_osc_inport.slx');
   ```

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

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

3. Open `slrt_ex_osc_inport/In1`

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

4. Turn off inport data interpolation.

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

5 Set the 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 = slrealtime('TargetPC1');
load(tg,model);
start(tg);
```

8 View the signals in the Simulation Data Inspector.

```
Simulink.sdi.view;
```

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 `slrt_ex_osc_inport.mldatx` file must be in the working folder.

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

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

2 Create an application object.

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

3 Update the application object.

```
updateRootLevelInportData(app_object);
```

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

```
tg = slrealtime;
load(tg, 'slrt_ex_osc_inport');
start(tg);
```

5 View the signals in the Simulation Data Inspector.

```
Simulink.sdi.view;
```

See Also

More About

- “Define and Update Inport Data” on page 6-49
- “Load Data to Root-Level Input Ports”
- “Inport Data Mapping Limitations” on page 6-56
- “Data Logging with Simulation Data Inspector (SDI)” on page 6-13

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 6-49
- “Load Data to Root-Level Input Ports”

Display and Filter Hierarchical Signals and Parameters

In this section...

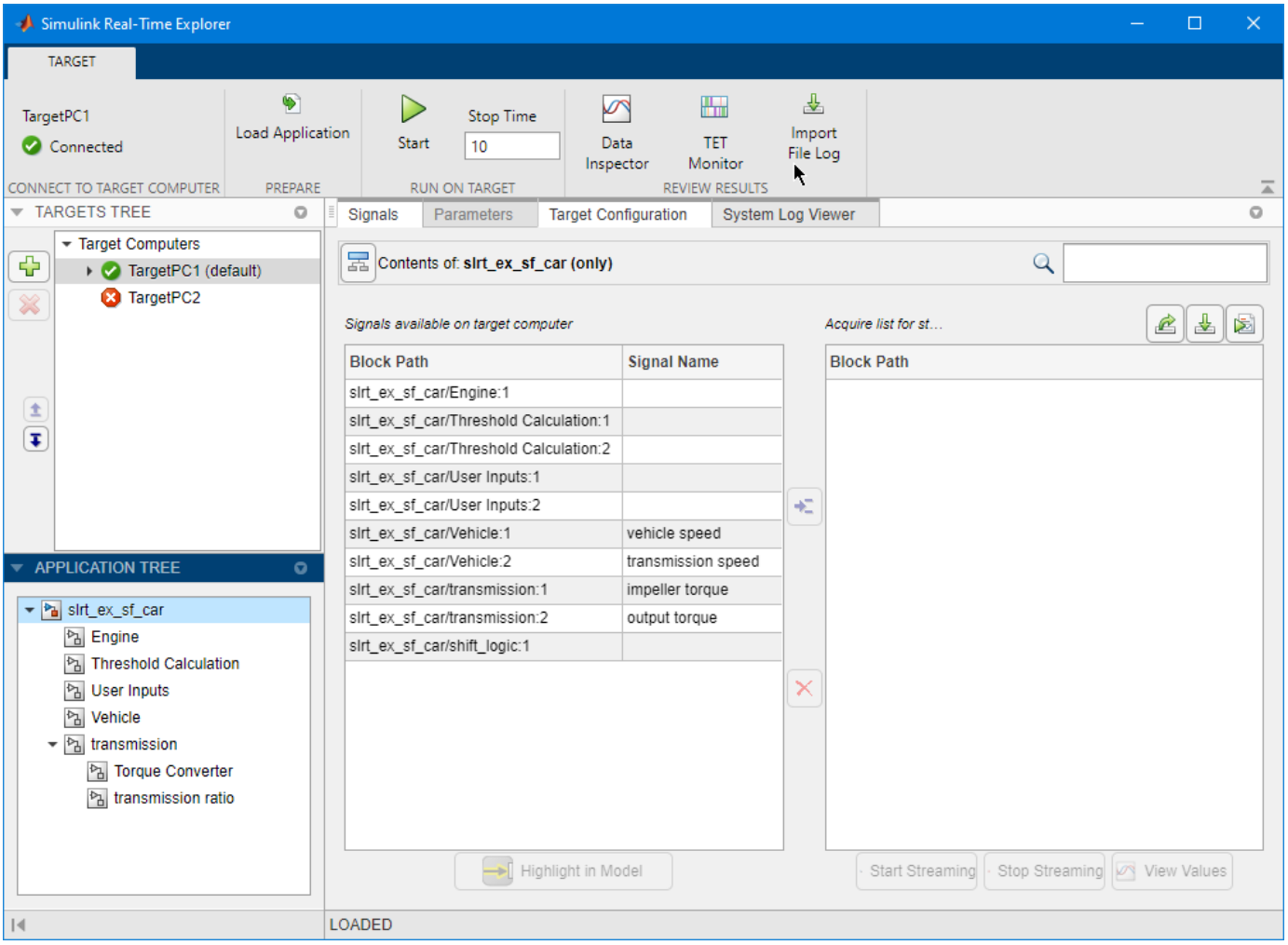
In Simulink Real-Time Explorer, the default view of the signal and parameter lists shows the signals and parameters only at the hierarchy level that you selected. You can display signals and parameters for the current level and below and filter the display to show only the items that you are interested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

The figure shows the contents of the top level of the `slrt_ex_sf_car` real-time application. To open this model, in the MATLAB Command Window, type:

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



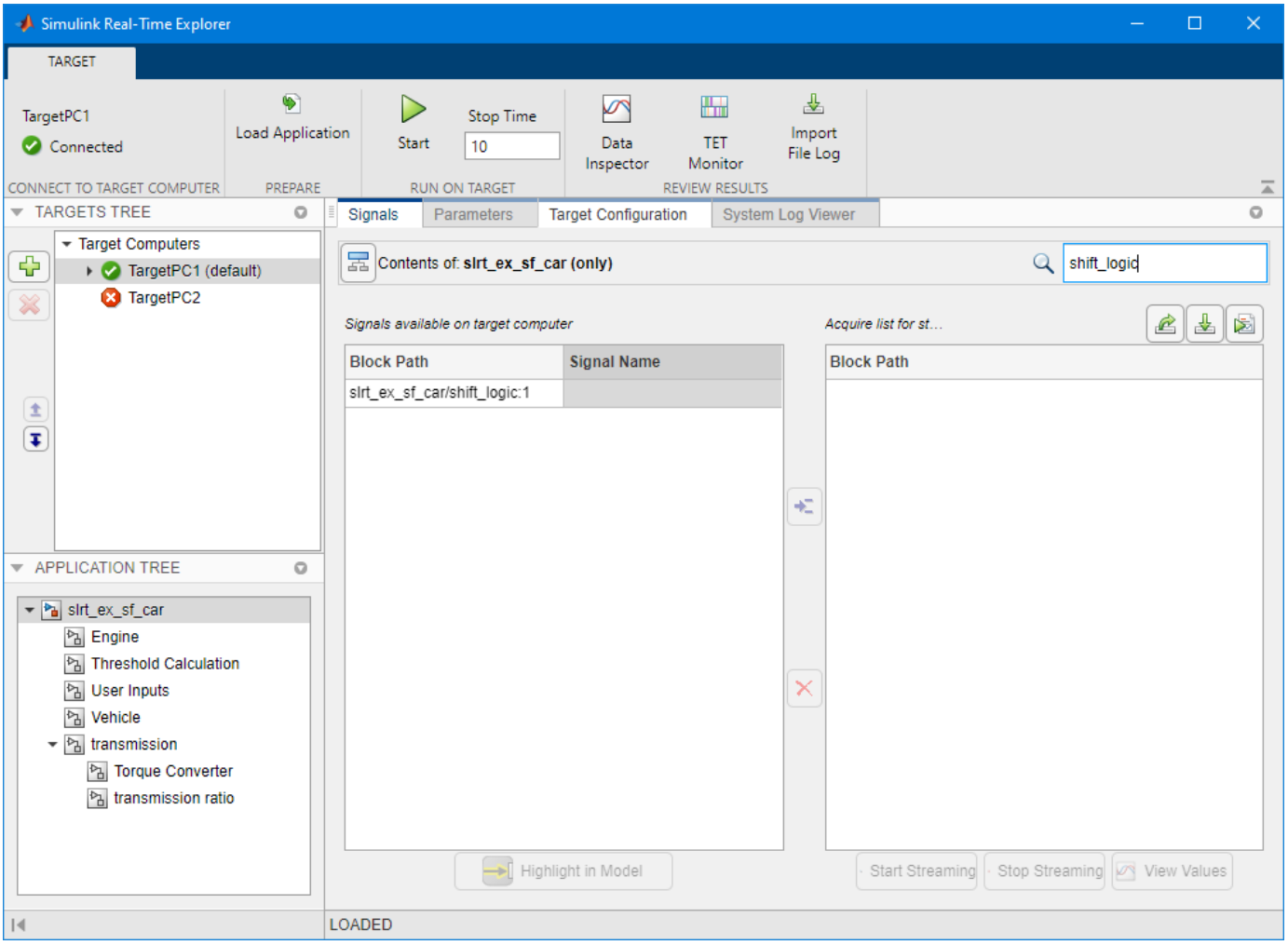
Filtered Display

To restrict the display to signals or parameters with a particular characteristic, use the **Filter** text box. You can restrict the scope of the filtered display by selecting a level of the application in the **Application Tree** panel.

Simulink Real-Time Explorer supports filtering by values in these columns:

- Signals — **Block Path** and **Signal Name**
- Parameters — **Block Path** and **Name**

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.



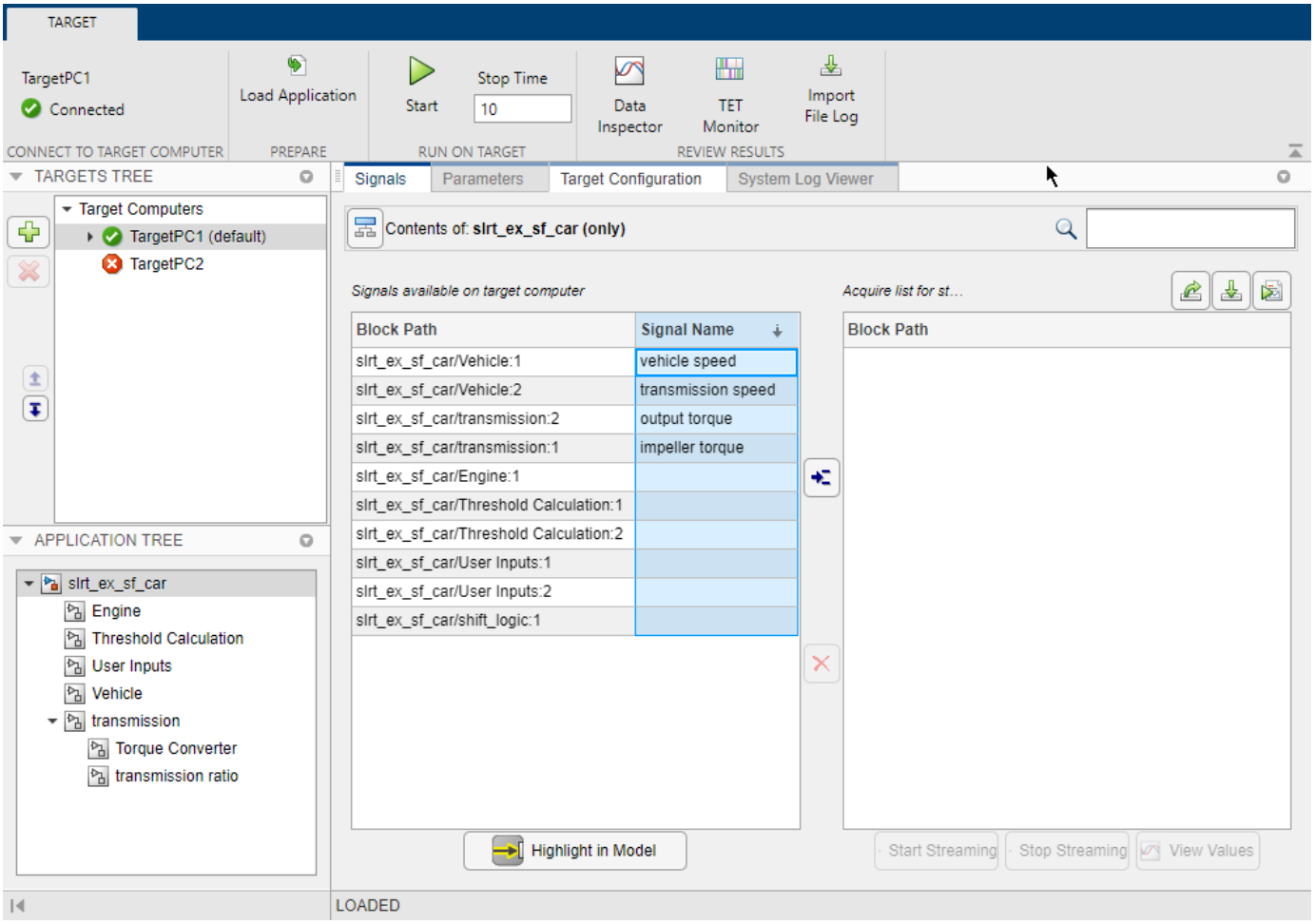
Sorted Display

To group signals and parameters by columns, select the column head, hover the cursor near the right border of the column head (displays the **Sort by** icon), and click the **Sort by** icon.

Explorer supports grouping by the following columns:

- Signals — **Block Path** and **Signal Name**
- Parameters — **Block Path**, **Name**, **Value**, **Type**, and **Size**

For example, to sort signals by name, right-click the **Signal Name** column and select the **Sort by** icon.


TARGET
TargetPC1
Connected
CONNECT TO TARGET COMPUTER
Load Application
PREPARE
Start
Stop Time
10
RUN ON TARGET
Data Inspector
TET Monitor
Import File Log
REVIEW RESULTS
TARGETS TREE
Target Computers
TargetPC1 (default)
TargetPC2
APPLICATION TREE
slrt_ex_sf_car
Engine
Threshold Calculation
User Inputs
Vehicle
transmission
Torque Converter
transmission ratio
Signals
Parameters
Target Configuration
System Log Viewer
Contents of: slrt_ex_sf_car (only)
Signals available on target computer
Block Path
Signal Name
slrt_ex_sf_car/Vehicle:1
vehicle speed
slrt_ex_sf_car/Vehicle:2
transmission speed
slrt_ex_sf_car/transmission:2
output torque
slrt_ex_sf_car/transmission:1
impeller torque
slrt_ex_sf_car/Engine:1
slrt_ex_sf_car/Threshold Calculation:1
slrt_ex_sf_car/Threshold Calculation:2
slrt_ex_sf_car/User Inputs:1
slrt_ex_sf_car/User Inputs:2
slrt_ex_sf_car/shift_logic:1
Acquire list for st...
Block Path
Highlight in Model
Start Streaming
Stop Streaming
View Values
LOADED

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 Simulink 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.
- 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 6-61.

See Also

Gain

More About

- “Nonvirtual and Virtual Blocks”
- “Types of Composite Signals”
- “Signal storage reuse”
- “Block reduction”
- “Troubleshoot Parameters Not Accessible by Name” on page 6-62
- “Internationalization Issues” on page 6-63

External Websites

- MathWorks Help Center website

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 the 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 6-62.

See Also

More About

- “Troubleshoot Signals Not Accessible by Name” on page 6-61
- “Internationalization Issues” on page 6-63

External Websites

- MathWorks Help Center website

Internationalization Issues

Simulink Real-Time inherits the internationalization support of the products that it works with: 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 the Simulation Data Inspector to observe signals, the non-ASCII signal names are displayed as expected. 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.

Third-party code (for example, parsers for vendor configuration files) sometimes does not support cross-locale, cross-platform internationalization. For such code, you must give files and folders 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.

See Also

More About

- “Troubleshoot Signals Not Accessible by Name” on page 6-61
- “Troubleshoot Parameters Not Accessible by Name” on page 6-62

Execution Modes

Execution Modes

The Simulink Real-Time kernel has two mutually exclusive execution modes.

- Interrupt mode — The scheduler implements real-time single-tasking and multitasking execution of single-rate or multirate systems, including asynchronous events (interrupts). You can interact with the target computer while the real-time application is executing at high sample rates. To use this real-time mode:
  - Leave the **Force polling mode** configuration parameter disabled (default).
  - Leave the `pollingThreshold` application option at the default value.
- Polling 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 enables you to achieve real-time application sample times that you cannot achieve by 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. To use this real-time mode, either:
  - Enable the **Force polling mode** configuration parameter.
  - Set the `pollingThreshold` application option sample time value to a rate below the base rate of the model.

For more information, see **Force polling mode** and `Application`.

See Also

Thread Trigger | “TLC Command-Line Options”

Related Examples

- “Concurrent Execution on Simulink® Real-Time™” on page 13-6

More About

- “Set Configuration Parameters”
- “Performance Optimization”
- “About RTOS Tasks and Priorities”
- “Troubleshoot Overloaded CPU from Executing Real-Time Application” on page 17-4

Real-Time Application Execution

Working with 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 on the target computer. You can load, start, stop, and check the status of the real-time application.

These commands let you interact with real-time applications on standalone target computers that are not connected to Simulink Real-Time software on a development computer.

To enter commands, type the commands by using a keyboard attached to the target computer or by using an SSH utility (such as PuTTY) to send commands to the target computer from a development computer.

The target computer commands are case-sensitive. For more information, see “Target Computer Command-Line Interface”.

To read the target computer console log, open the **Simulink Real-Time Explorer** and click the **System Log Viewer** tab. You can also export the system log by using the `SystemLog` function.

See Also

Simulink Real-Time Explorer | `SystemLog` | `slrtExplorer`

Related Examples

- “Target Object Commands”
- “Target Computer RTOS System Commands”

Execute Target Computer RTOS Commands at Target Computer Command Line

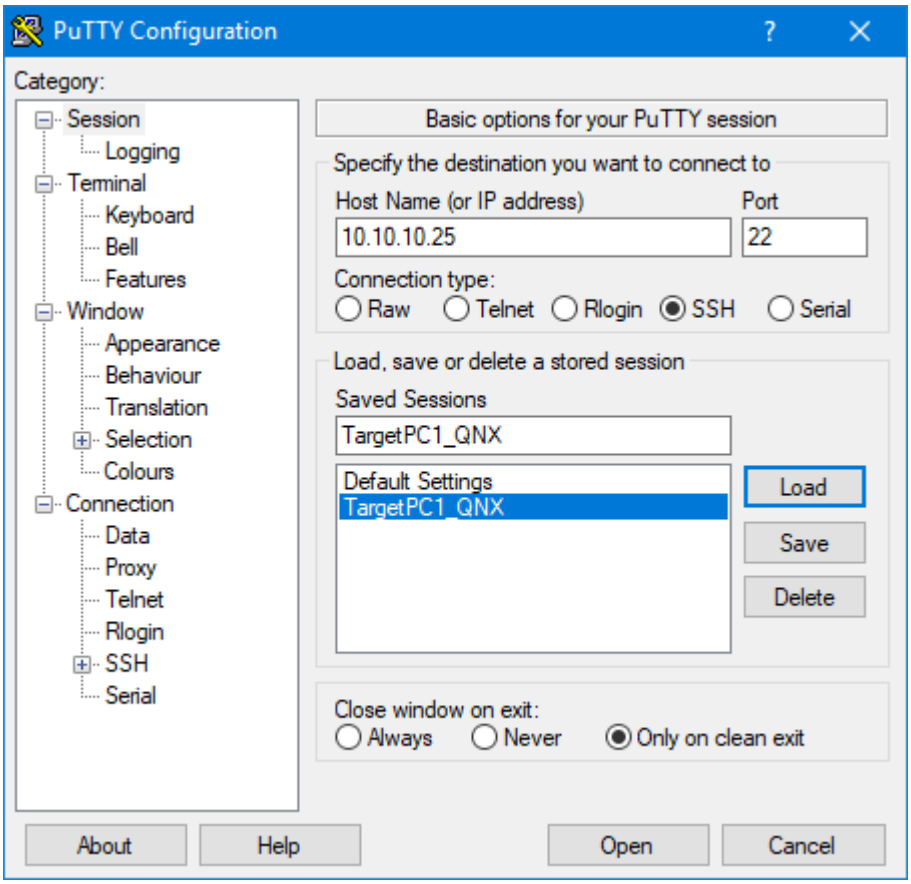
To enter target computer RTOS commands, type the commands by using a keyboard attached to the target computer or by using an SSH utility (such as PuTTY) to send commands to the target computer from a development computer.

The target computer commands are case-sensitive. For more information, see “Target Computer Command-Line Interface”.

The command examples use the PuTTy SSH utility. You can download and install this utility from www.putty.org.

1. Boot the target computer.
2. Connect the development computer and target computer. In the MATLAB Command Window, type:

   ```
   tg = slrealtime;
   connect(tg);
   ```
3. Start the SSH utility. This example uses PuTTY.
4. Load the PuTTY session for the target computer and click **Open**.



5. To configure the target computer date, log in to the PuTTY session as user `root` with password `root`.

6. Set the time zone. This example sets the time zone to Eastern Standard Time.

```
# env TZ=EST5EDT
# export TZ=EST5EDT
# setconf _CS_TIMEZONE EST5EDT
```

7. Set the date and time. This example sets the date and time to September 10, 2019 at 11:25 AM.

```
# date 091011252019
Tue Sep 10 11:25:15 EDT 2019
```

8. Set the hardware clock from the system date and time.

```
# rtc -s hw
```

See Also

Related Examples

- "Target Object Commands"
- "Target Computer RTOS System Commands"

External Websites

- QNX Momentics IDE 7.0 User's Guide
- QNX Momentics IDE 7.0 User's Guide, Utilities Reference

Tuning Performance

CPU Overload

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 is triggered while the previous step is running.

See Also

SLRT Overload Options

Related Examples

- “Monitor CPU Overload Rate” on page 9-3
- “Concurrent Execution on Simulink® Real-Time™” on page 13-6

More About

- “Troubleshoot Overloaded CPU from Executing Real-Time Application” on page 17-4

Monitor CPU Overload Rate

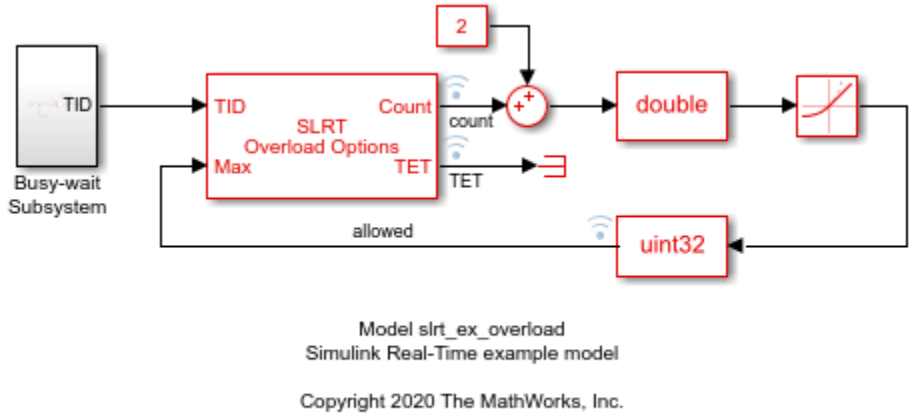
The SLRT Overload Options block outputs the current CPU overload count for the identified sample rate.

This example shows how to design a model that uses the SLRT Overload Options block to monitor the rate at which CPU overloads occur. The rate of CPU overloads information can be useful when tuning performance of a model for which a low CPU overload rate is acceptable.

Open, Build, and Run the Model

In the MATLAB Command Window, type:

```
open_system(fullfile(matlabroot, 'toolbox','slrealtime','examples','slrt_ex_overload'));
```



Name the signal coming out from the outport of rate limiter block as Rate Limiter and log it in the Simulation Data Inspector.

```
p = get_param('slrt_ex_overload/Rate Limiter','PortHandles');
l = get_param(p.Outport,'Line');
set_param(l,'Name','Rate Limiter');
Simulink.sdi.markSignalForStreaming('slrt_ex_overload/Rate Limiter',1,'on');
```

Build the model.

```
set_param('slrt_ex_overload', 'RTWVerbose', 'off');
rtwbuild('slrt_ex_overload');
```

```
### Successful completion of build procedure for: slrt_ex_overload
### Created MLDATX ..\slrt_ex_overload.mldatx

Build Summary

Top model targets built:

Model             Action                        Rebuild Reason
================================================================================
slrt_ex_overload  Code generated and compiled  Generated code was out of date.
```

```
1 of 1 models built (0 models already up to date)
Build duration: 0h 0m 30.403s
```

Download the application and run it on the target computer.

```
tg = slrealtime;
connect(tg);
load(tg,'slrt_ex_overload');
start(tg);
pause(20);
stop(tg);
```

Open Simulation Data Inspector

To view the rate at which CPU overloads occur, open the Simulation Data Inspector.

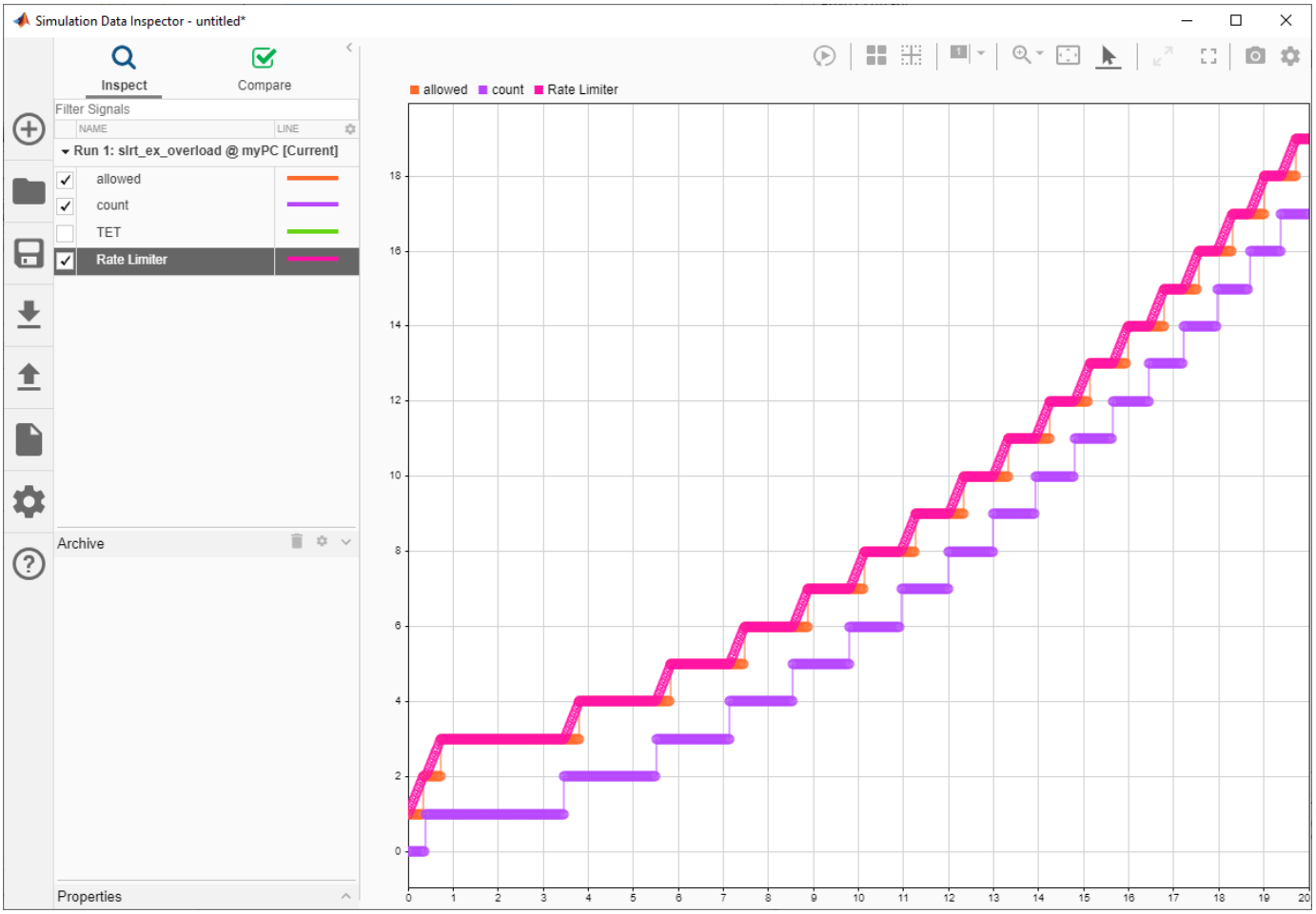
In the MATLAB Command Window, type:

```
Simulink.sdi.view;
```

Examine CPU Overload Rate Data

In the Simulation Data Inspector, the graph shows:

- Bottom rising stair step signal -- This signal indicates the number of CPU overloads that occurred.
- Top rising stair step signal -- This signal indicates the number of CPU overloads that are allowed, which is (occurred + 2).
- Rising slew rate -- This signal indicates the rate at which CPU overloads occur. When the rising slew rate becomes greater than the top rising stair step signal, the rate of CPU overloads is greater than are allowed.



Modify Rate of CPU Overloads

To modify the rate at which CPU overloads occur in the model, modify the `Constant2` parameter value.

Modify Allowed Rate of CPU Overloads

To modify the rate of CPU overloads that are acceptable in the model, modify the `RisingSlowLimit` parameter value.

Build and Run Model with Changed Overload Rates

In the MATLAB Command Window, type:

```
load(tg,'slrt_ex_overload');
```

To modify the rate of CPU overloads that are acceptable in the model

```
tg.setparam('slrt_ex_overload/Rate Limiter','RisingSlewLimit',0.004);
```

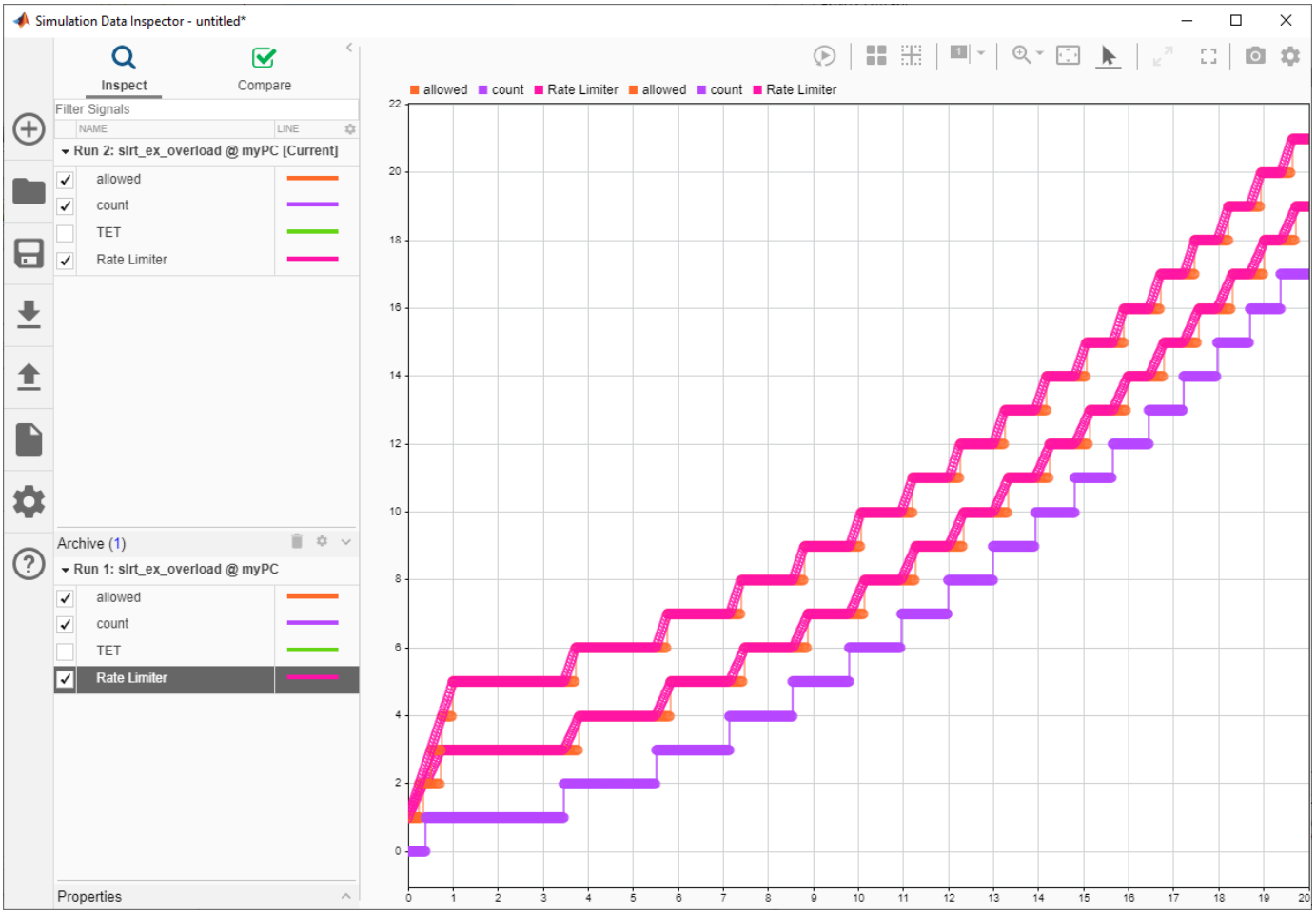
To modify the rate at which CPU overloads occur in the model

```
tg.setparam('slrt_ex_overload/Constant2','Value',4);
```

run the modified application on the target computer

```
start(tg);
pause(20);
stop(tg);
```

In the Simulation Data Inspector, compare the signal data from the simulation runs and observe the change to the CPU overload rate.




```
bdclose('all');
```

See Also

SLRT Overload Options

Related Examples

- “Concurrent Execution on Simulink® Real-Time™” on page 13-6

More About

- “CPU Overload” on page 9-2

- “Troubleshoot Overloaded CPU from Executing Real-Time Application” on page 17-4

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 `slrt_ex_mds_and_tasks`. To open this model, open the subsystem models first:

- slrt_ex_mds_subsystem1
- slrt_ex_mds_subsystem2
- slrt_ex_mds_and_tasks

1. Open model `slrt_ex_mds_and_tasks`.

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('slrt_ex_mds_and_tasks','CodeProfilingInstrumentation','Coarse');
```

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

Generate Real-Time Application Execution Profile

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

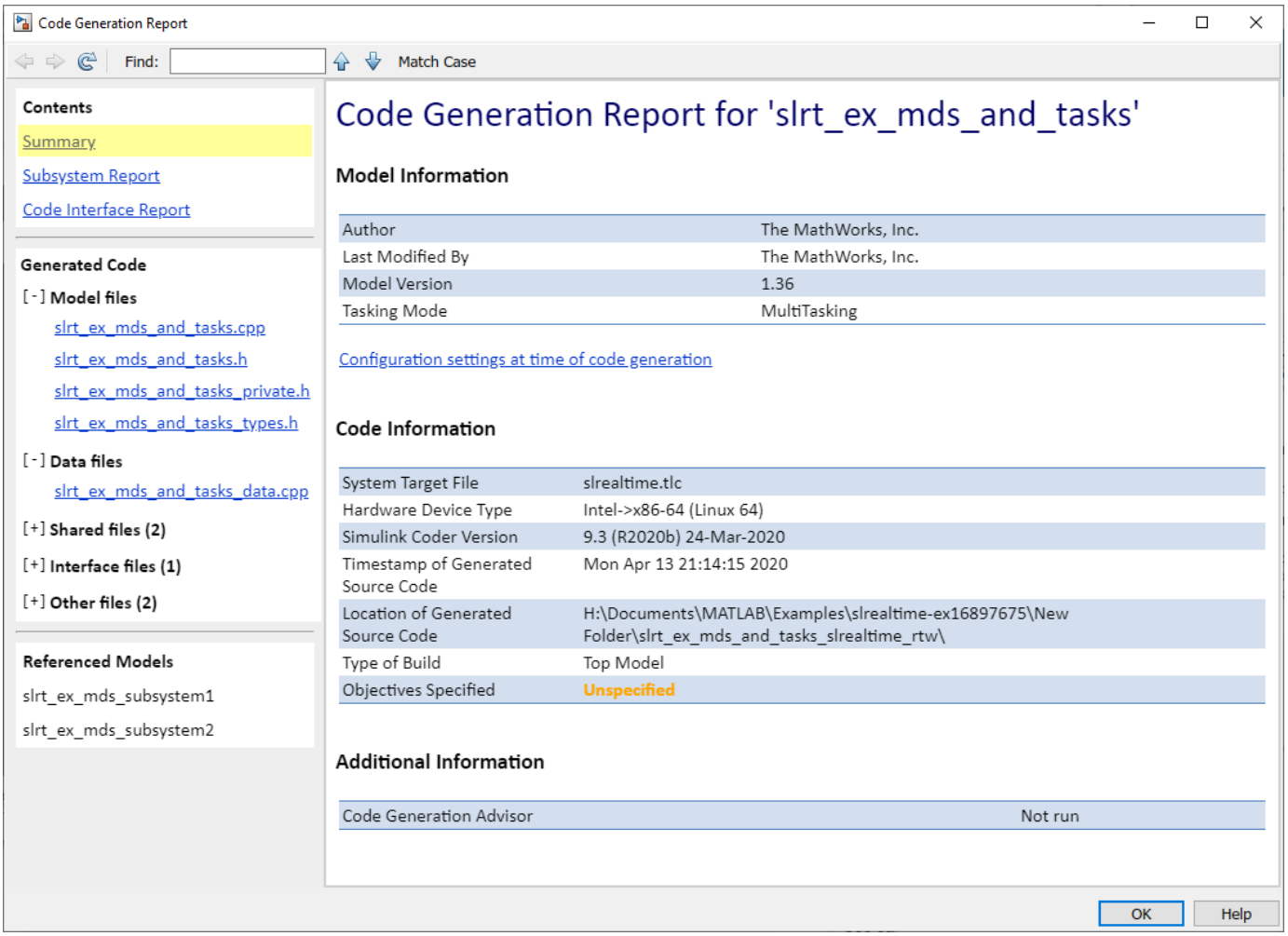
This procedure assumes that you have configured the target computer to take advantage of multiple cores. It also assumes that you previously configured the model for task and function execution profiling.

1. Open, build, and download the model.

```
mdl = 'slrt_ex_mds_and_tasks';
open_system(mdl);
rtwbuild(mdl);
```

```
tg = slrealtime;
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.



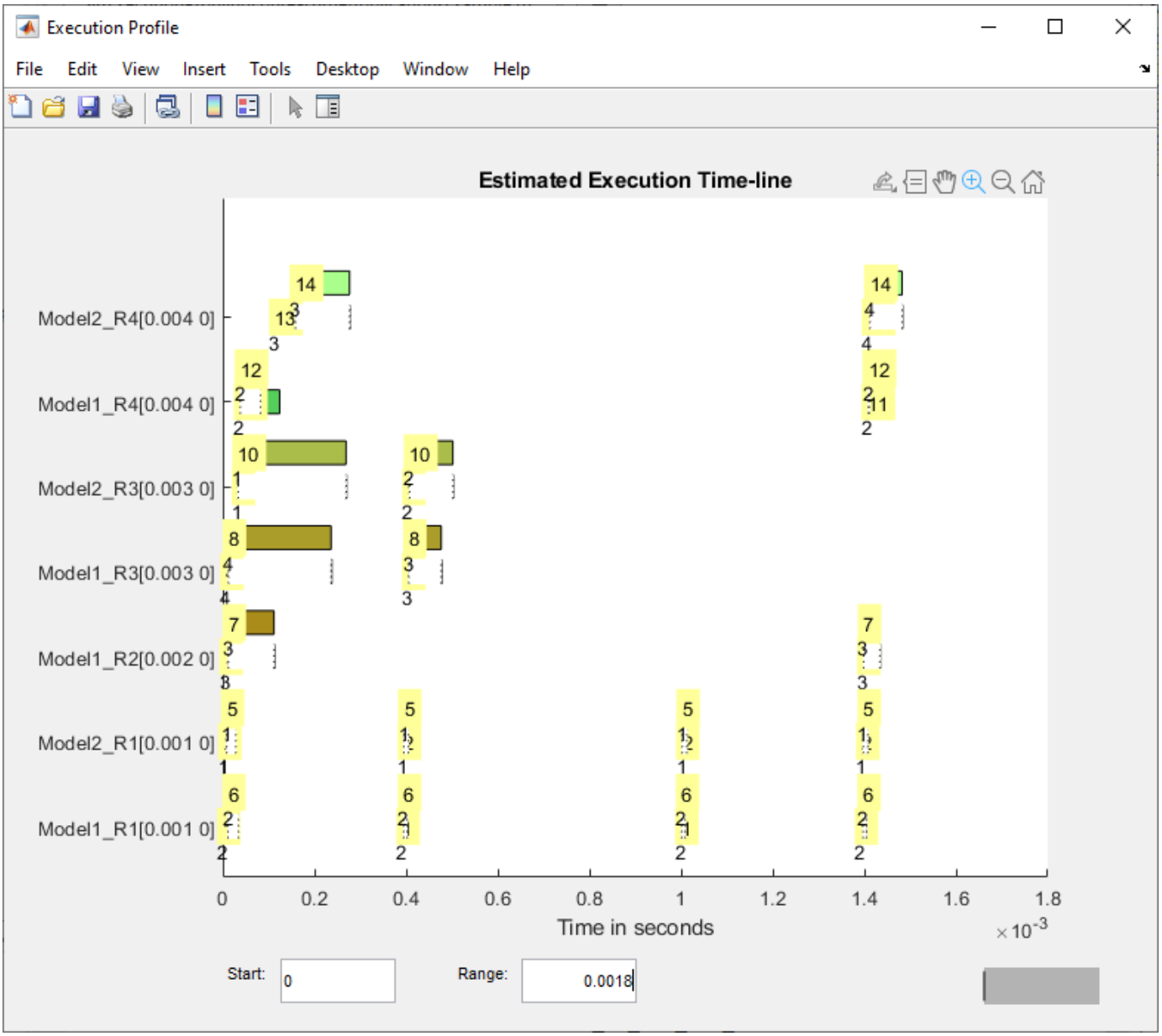
2. Start the profiler and then execute the real-time application.

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

3. Display the profiler data.

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

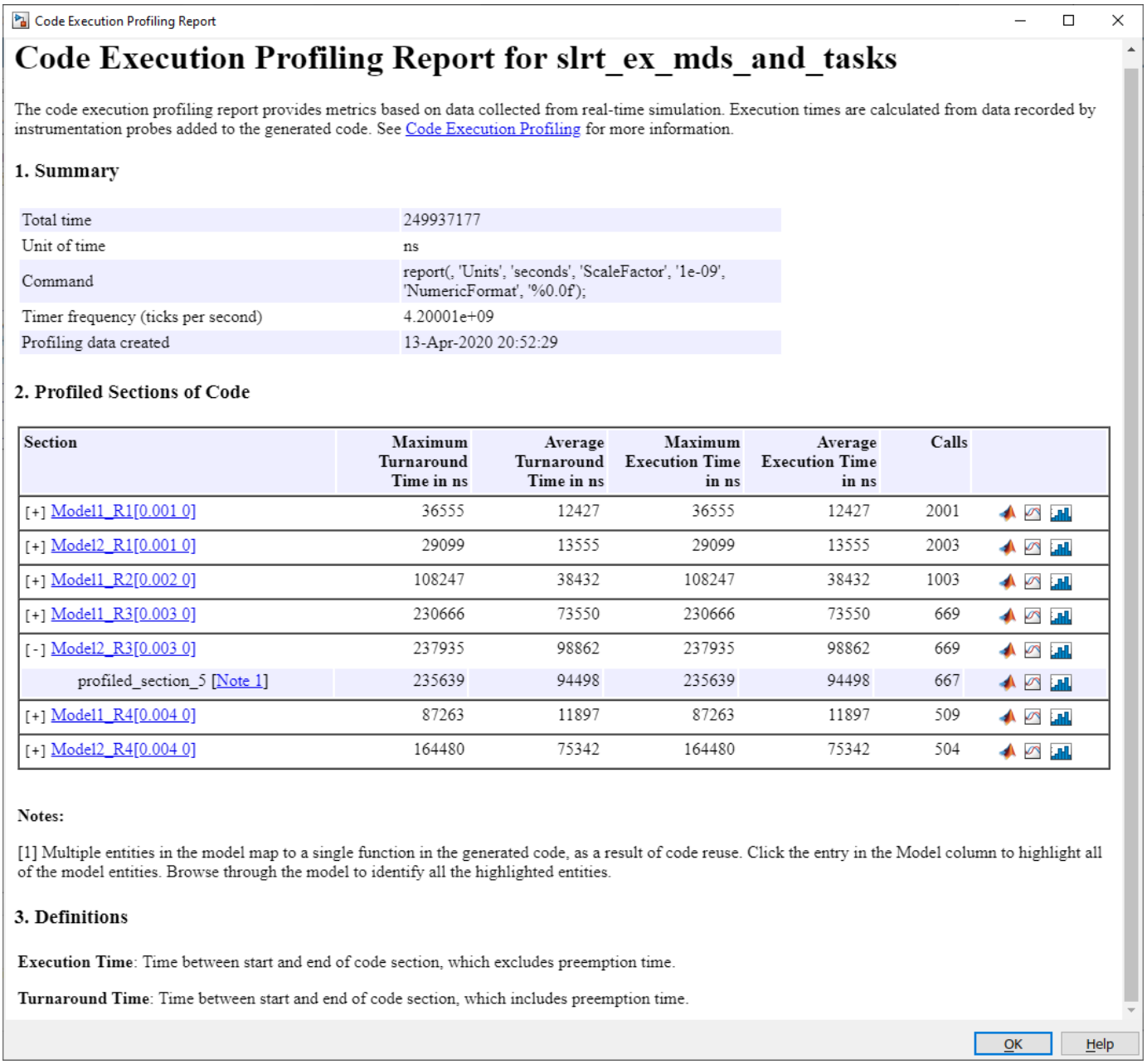
The Execution Profile plot shows the allocation of execution cycles across the four processors, indicated by the colored horizontal bars. The model sections are listed in the Code Execution Profiling Report. The cores are indicated by the numbers underneath the bars.



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.



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, you can distribute code generation and compilation for each referenced model 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.

You can 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.

For this procedure, you must 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. On each cluster computer, install:
      - MATLAB
      - Parallel Computing Toolbox
      - MATLAB Parallel Server
      - Simulink Real-Time
      - Simulink Real-Time Target Support Package
   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.

   In this configuration, the development computer and the remote workers have installed a supported version of a C++ compiler that is compatible with the code generation target. For the current list of supported compilers, see Supported and Compatible Compilers.
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” option. This selection enables the parameter “MATLAB worker initialization for builds”. For more information, see “Reduce Build Time for Referenced Models by Using Parallel Builds”.
8. Build and download your model.

See Also

`parpool` | `pctRunOnAll`

More About

- “Reduce Build Time for Referenced Models by Using Parallel Builds”

Execution with MATLAB Scripts

10

Real-Time Application Objects and Options in the MATLAB Interface

Target and Application Objects

The Simulink Real-Time software uses a `Target` object to represent a target computer and an `Application` object to represent a real-time application. To run and control real-time applications on the target computer, use the object functions.

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

A `Target` object on the development computer represents the interface to a real-time application and the RTOS on the target computer. To run and control the real-time application, use `Target` objects.

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

To create a `Target` object for the default target computer, in the MATLAB Command Window, type:

```
tg = slrealtime
```

A `Target` object has properties and functions specific to that object. The real-time application object functions enables 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 Command 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 slrealtime/function_name
```

For example, to get help on the `load` function, type:

```
doc slrealtime/load
```

To get a list of all the functions for the `Target` object, use the `methods` function. For example, to get the functions for `Target` object `tg`, type:

```
methods(tg)
```

If you want to control the real-time application from the target computer, use target computer commands (see “Control Real-Time Application at Target Computer Command Line” on page 8-2).

Control Real-Time Application by Using Objects

To create a real-time application and control it by using `Target` and `Application` objects:

1. Open a model and build a real-time application. This example uses the `slrt_ex_osc` model.

   ```
   open_system(fullfile(matlabroot, 'toolbox', 'slrealtime', ...
   'examples', 'slrt_ex_osc'));
   rtwbuild('slrt_ex_osc');
   ```
2. Create `Target` and `Application` objects to represent the target computer and the real-time application.

   ```
   tg = slrealtime('TargetPC1');
   app = slrealtime.Application('slrt_ex_osc');
   ```
3. Load the real-time application on the target computer by using the `Target` object.

   ```
   load(tg,'slrt_ex_osc');
   ```

4. Set the `Target` object `stoptime` property for the real-time application.

```
setStopTime(tg,inf);
```

5. Get the `Application` object options property values from the real-time application.

```
app.Options.get("stoptime")
```

```
ans =

    Inf
```

6. Start the real-time application by using the `Target` object .

```
start(tg);
```

7. Stop the real-time application by using the `Target` object .

```
stop(tg);
```

Use Real-Time Application Object Functions

To run `Target` object and `Application` functions, 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 start a real-time application on target computer `tg`, in the MATLAB Command Window, type:

```
tg = slrealtime;
start(tg);
```

See Also

`Application` | `Target`

More About

- “Control Real-Time Application at Target Computer Command Line” on page 8-2

Simulink Real-Time Instrument Object

Instrumentation Apps for Real-Time Applications

To visualize the behavior of a real-time application running on a target computer, you can create instrument panel apps. An instrument panel app is an user-interface application into which you can insert one or more instruments. To create an instrument panel app, use App Designer or an m-script.

- When you create an instrument panel app in the App Designer **Design View**, you add instrument components from the App Designer **Component Library** to the app. You configure each instrument by using fields in the **Component Browser**. In the App Designer **Code View**, you add callback code to handle component events, such as new streaming data or interaction with the app. For more information, see “App Building Components” and “Manage Code in App Designer Code View”.
- When you create an instrument panel app by using an m-script, you use a programmatic approach to add each instrument to the panel as UI component. For more information, see “Write Callbacks for Apps Created Programmatically”.

To stream signal and parameter data to the instrument panel app from the real-time application, you use the `Instrument` object. After you create an instrument object for a real-time application, you can use instrument object functions to connect signals and parameters from the real-time application to instrument panel app callbacks.

When identifying parameters and output signals to stream signal to the instrument panel app from the real-time application, it can be helpful to use the hierarchical display of signals and parameters. See **Simulink Real-Time Explorer**. For more information, see “Display and Filter Hierarchical Signals and Parameters” on page 6-57.

See Also

`Instrument` | **Simulink Real-Time Explorer**

Related Examples

- “Add App Designer App to Inverted Pendulum Model” on page 13-13

More About

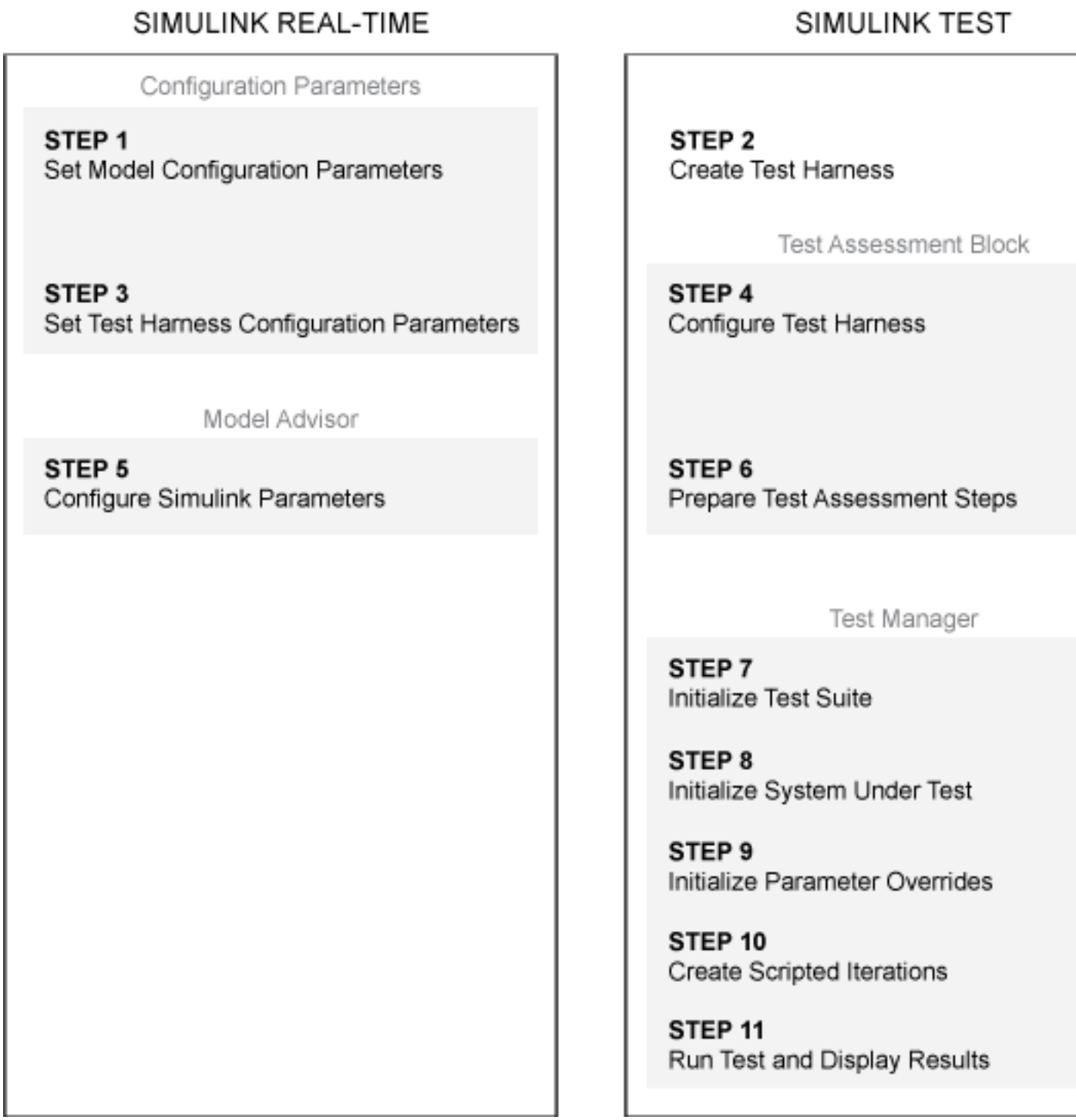
- “App Building Components”
- “Manage Code in App Designer Code View”
- “Display and Filter Hierarchical Signals and Parameters” on page 6-57

Automated Test with Simulink Test

Test Real-Time Application

This example shows how to perform a frequency-response test of the model `slrt_ex_osc_sltest`.

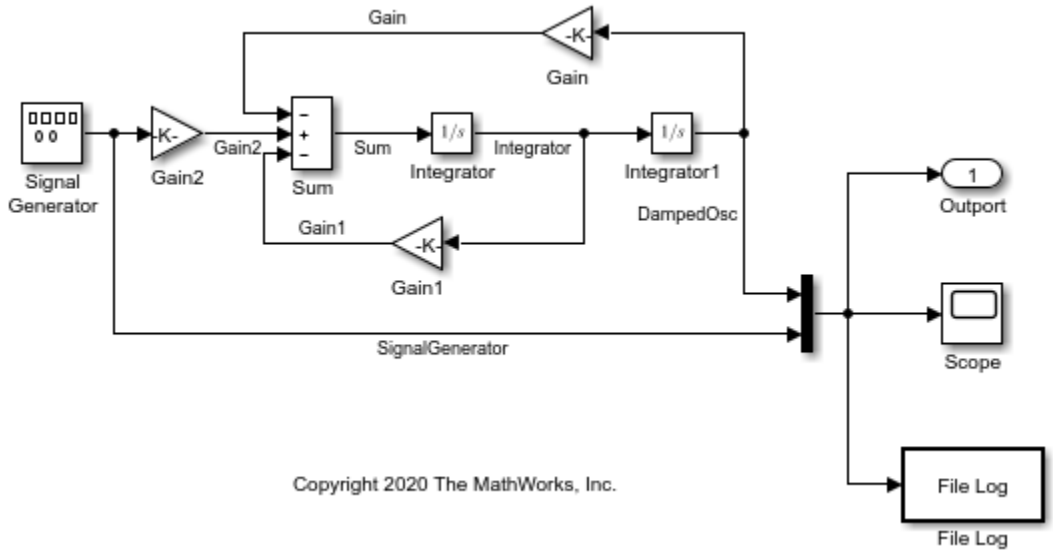
Using this information, in the design phase, you can modify the internal parameters of the model to meet your frequency requirements. In the production phase, you can bin manufactured parts based on frequency response.



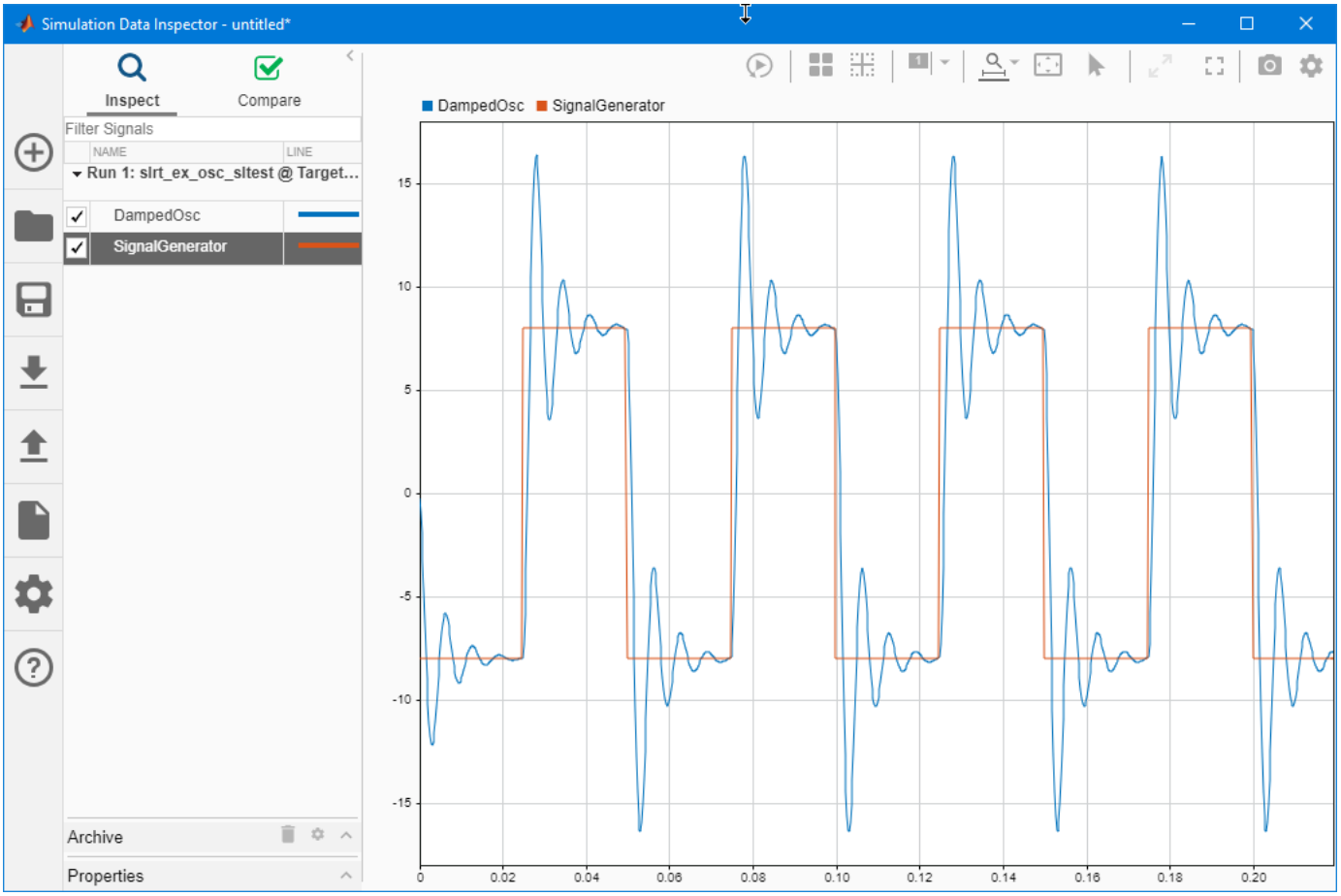
Open the Model

To open the model, in the MATLAB Command Window, type:

```
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_osc_sltest'));
```



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.



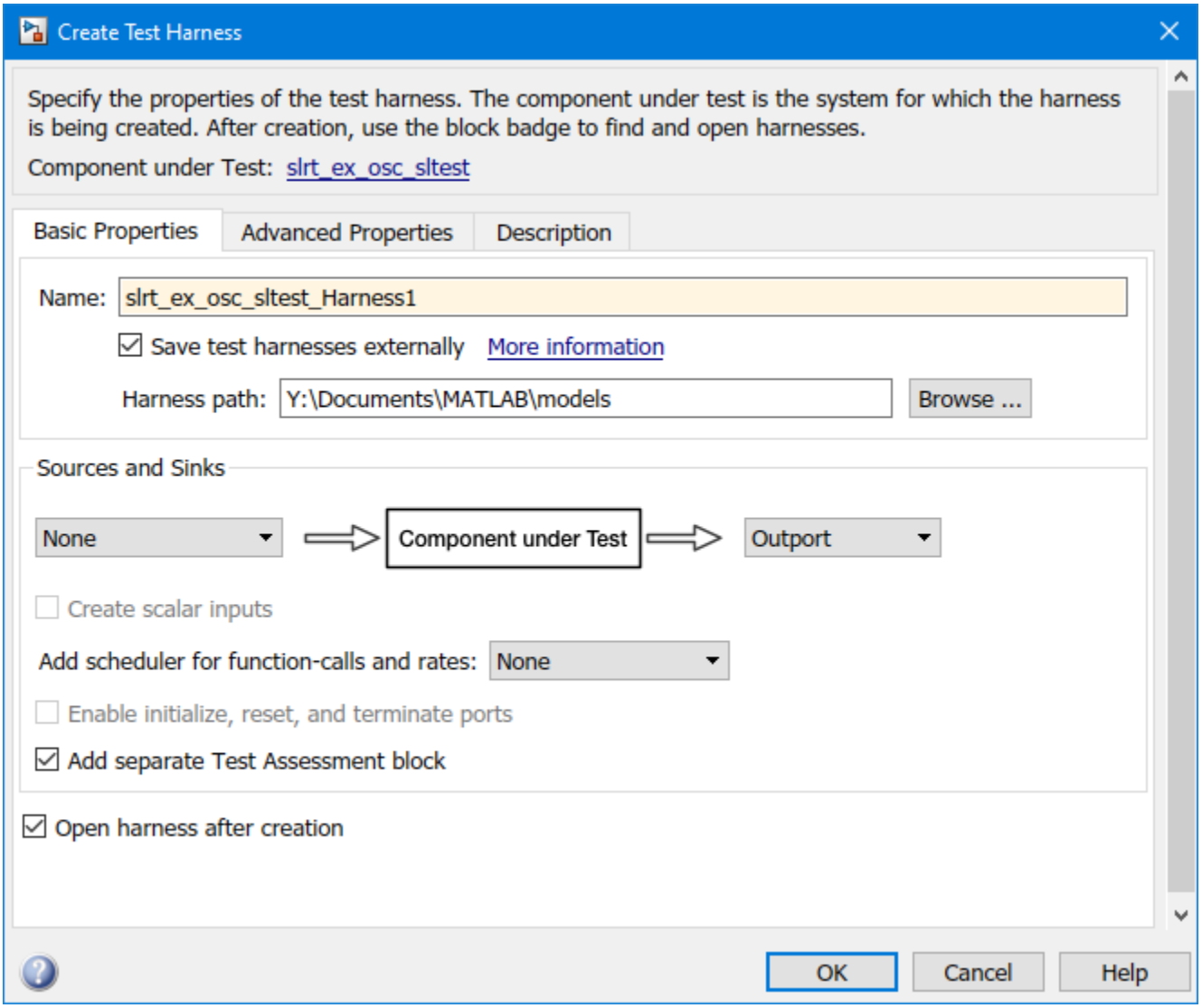
The test determines the highest frequency at which the output values of the Integrator and Signal Generator blocks are within a specified criterion of each other. The test uses the model itself as a signal source and uses a test harness to compare the outputs of the Integrator and Signal Generator blocks.

Step 1. Set Model Configuration Parameters

1. Open model `slrt_ex_osc_sltest` 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 `slrt_ex_osc_sltest_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 `Outport`.
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.



9. Click **OK**.

The example model `slrt_ex_osc_sltest` 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 `slrt_ex_osc_sltest_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

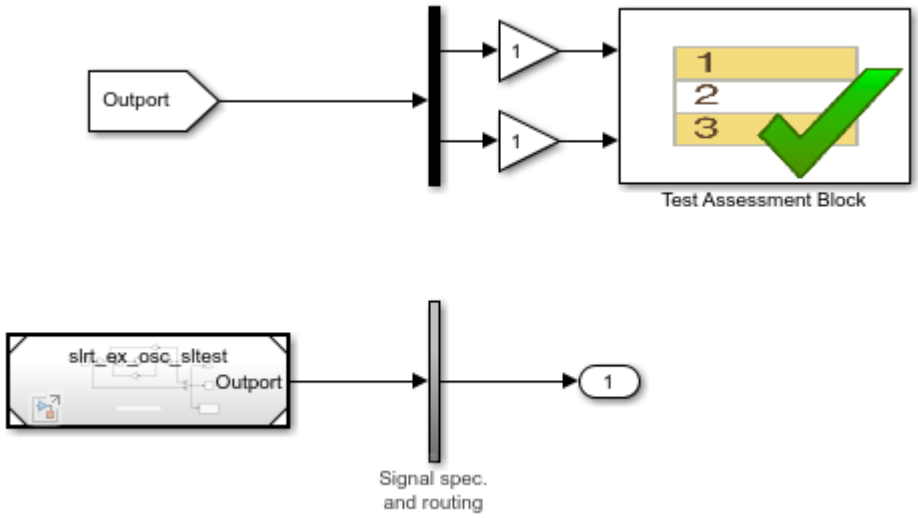
1. Open test harness `slrt_ex_osc_sltest_Harness1`.
2. Open the Configuration Parameters. On the **Real-Time** tab, click **Hardware Settings**.
3. Select **Model Referencing > Total number of instances allowed per top model > One**.
4. Select **Data Import/Export > Format > Structure with time**.
5. 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 `Outport` with inputs `Int1` and `SigGen`.
3. In `slrt_ex_osc_sltest_Harness1`, connect a Demux block to `slrt_ex_osc_sltest/Outport`.
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`.

The final test harness looks like `slrt_ex_osc_sltest_harness1`.

```
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_osc_sltest_harness1'))
```

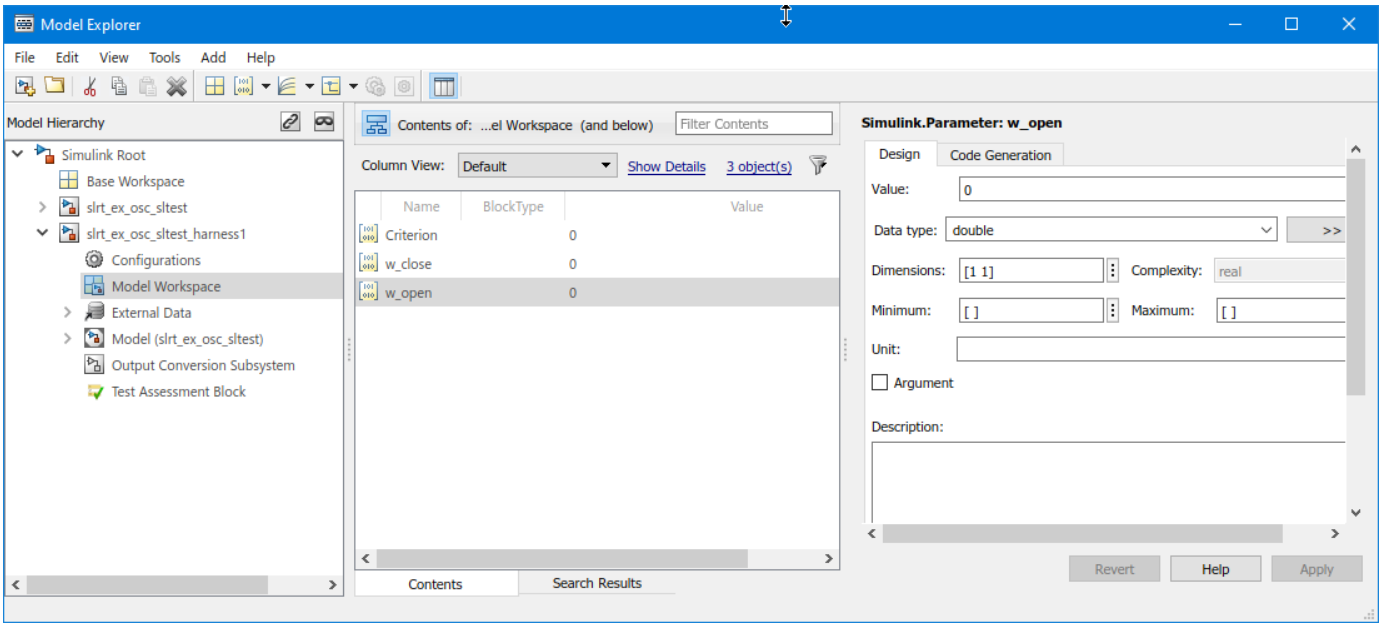


Step 5. Configure Simulink Parameters

1. Open the Model Explorer. On the **Modeling** tab, pull down the **Design** section and click **Model Explorer**.
2. Click node **slrt_ex_osc_sltest_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 `slrt_ex_osc_sltest_Harness1` model workspace as model parameters, you access them by name directly, without model hierarchy.



6. Save the model.

Step 6. Prepare Test Assessment Steps

1. Open the Test Assessment block

2. Add these parameters to the Parameter symbol list:

- Criterion
- w_open
- w_close

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

`CheckSetting`

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

5. To add a substep to `CheckSetting`, click **Add sub-step**, and type:

```
Hi when (SigGen > 0)
```

The when expression selects one half of the waveform.

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

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

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

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

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

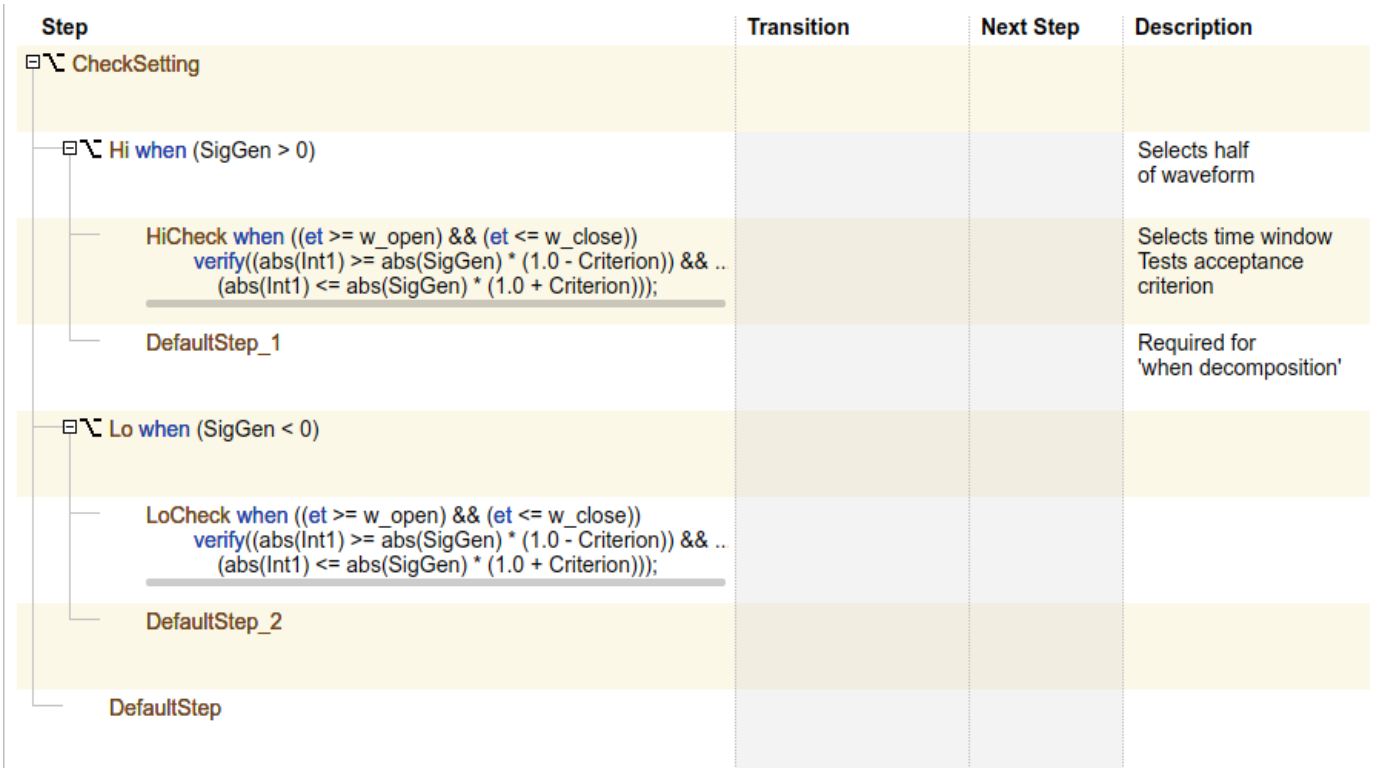
```
Lo when (SigGen < 0)
```

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

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

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

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



Step 7. Initialize Test Suite

1. Click on the `slrt_ex_osc_sltest` 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 `slrt_ex_osc_sltest`.
5. Set **Target Computer** to `TargetPC1`.
6. In tab **Test Harness**, set **Harness** to `slrt_ex_osc_sltest_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 `slrt_ex_osc_sltest_Harness1` and `slrt_ex_osc_sltest`.
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', ...
       'slrt_ex_osc_sltest/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 `slrt_ex_osc_sltest` 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 (slrt_ex_osc_sltest)**.

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.
5. Click the **Zoom in Time** button and select the range `4.00-4.1`.

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

See Also

Test Assessment | Test Sequence

More About

- “Test Models in Real Time” (Simulink Test)
- “Reuse Desktop Test Cases for Real-Time Testing” (Simulink Test)

Examples

Simulink Real-Time Examples

Parameter Tuning and Data Logging

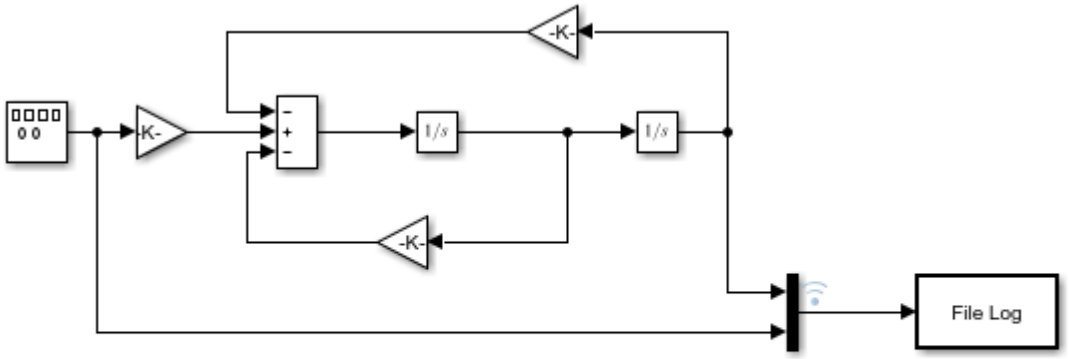
This example shows how to use real-time parameter tuning and data logging with Simulink® Real-Time™. After the example builds the model and downloads the real-time application, `slrt_ex_param_tuning`, to the target computer, the example executes multiple runs with the gain 'Gain1/Gain' changed (tuned) before each run. The gain sweeps from 0.1 to 0.7 in steps of 0.05.

The example uses the data logging capabilities of Simulink Real-Time to capture signals of interest during each run. The logged signals are uploaded to the development computer and plotted. A 3-D plot of the oscillator output versus time versus gain is displayed.

Open, Build, and Download Model to the Target Computer

Open the model, `slrt_ex_param_tuning`. The model configuration parameters select the `slrealtime.tlc` system target file as the code generation target. Building the model creates a real-time application, `slrt_ex_param_tuning.mldatx`, that runs on the target computer.

```
model = 'slrt_ex_param_tuning';
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples',model));
```



Simulink Real-Time example model

Build the model and download the real-time application, `slrt_ex_param_tuning.mldatx`, to the target computer.

- Configure for a non-Verbose build.
- Build and download application.

```
set_param(model,'RTWVerbose','off');
set_param(model,'StopTime','0.2');
rtwbuild(model);
tg = slrealtime;
load(tg,model);
```

```
### Successful completion of build procedure for: slrt_ex_param_tuning
### Created MLDATX ..\slrt_ex_param_tuning.mldatx

Build Summary
```

```
Top model targets built:

Model                 Action                       Rebuild Reason
=============================================================================================
slrt_ex_param_tuning  Code generated and compiled  Code generation information file does not exis

1 of 1 models built (0 models already up to date)
Build duration: 0h 0m 28.55s
```

Run Model, Sweep 'Gain' Parameter, Plot Logged Data

This code accomplishes several tasks.

Task 1: Create Target Object

Create the MATLAB® variable, `tg`, that contains the Simulink Real-Time target object. This object lets you communicate with and control the target computer.

- Create a Simulink Real-Time target object.
- 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 for each run.

- If no plot figure exist, create the figure.
- If the plot figure exist, make it the current figure.

Task 3: Loop over damping factor z

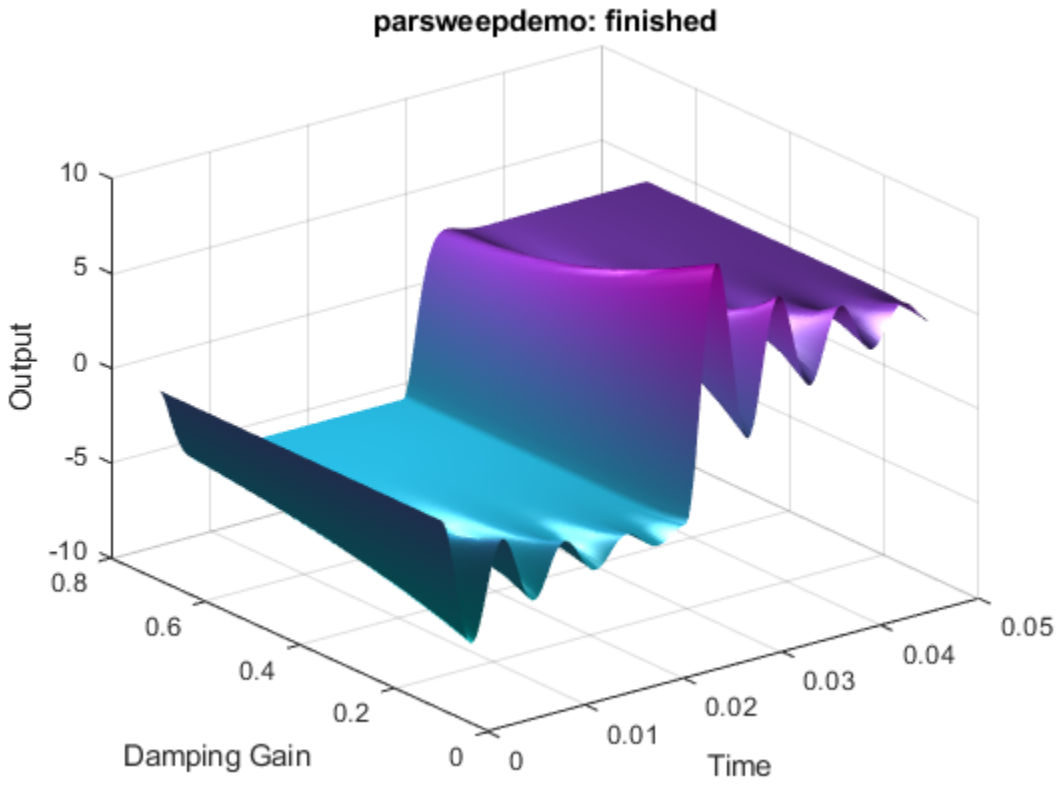
- Set damping factor (Gain1/Gain).
- Start run of the real-time application.
- Store output data in `outp`, `y`, and `t` variables.
- Plot data for current run.

Task 4: Create 3-D Plot (Oscillator Output vs. Time vs. Gain)

- Loop over damping factor.
- Create a plot of oscillator output versus time versus gain.
- Create 3-D plot.

```
figh = findobj('Name', 'parsweepdemo');
if isempty(figh)
  figh = figure;
  set(figh, 'Name', 'parsweepdemo', 'NumberTitle', 'off');
else
  figure(figh);
end
y = []; flag = 0;
for z = 0.1 : 0.05 : 0.7
  if isempty(find(get(0, 'Children') == figh, 1))
      flag = 1;
      break;
```

```
    end
    load(tg,model);
    tg.setparam([model '/Gain1'],'Gain',2 * 1000 * z);
    tg.start('AutoImportFileLog',true, 'ExportToBaseWorkspace', true);
    pause(0.4);
    outp = logsOut.FileLogSignals{1}.Values;
    y    = [y,outp.Data(:,1)];
    t    = outp.Time;
    plot(t,y);
    set(gca, 'XLim', [t(1), t(end)], 'YLim', [-10, 10]);
    title(['parsweepdemo: Damping Gain = ', num2str(z)]);
    xlabel('Time'); ylabel('Output');
    drawnow;
  end
  if ~flag
    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(model,0);
```

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. 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 `slrt_ex_mds_and_tasks` runs at sample rate of 0.001 second.

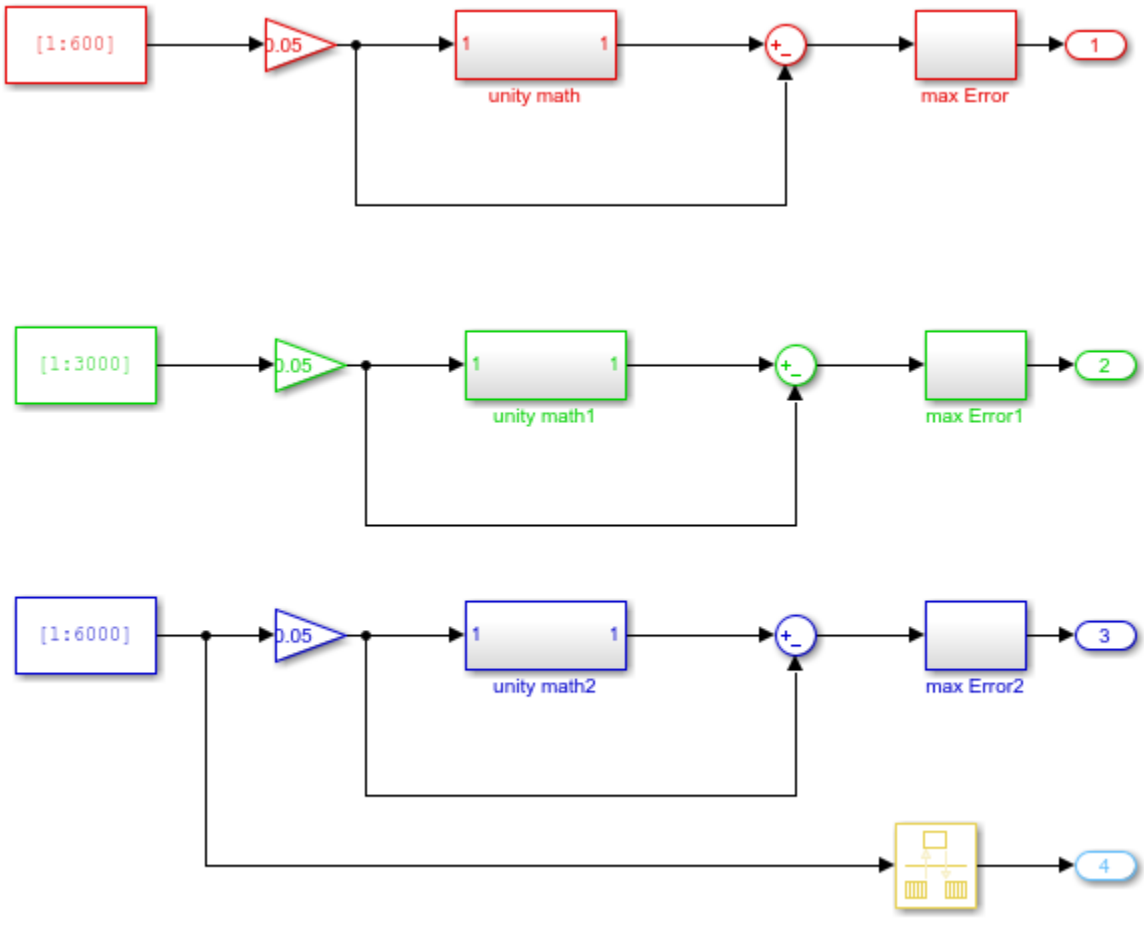
To run the model with adjusted sample rate of 0.01 second, change the sample rated before running the example. In the MATLAB Command Window, type:

Ts = 0.01;

Open, Build, and Download the Model

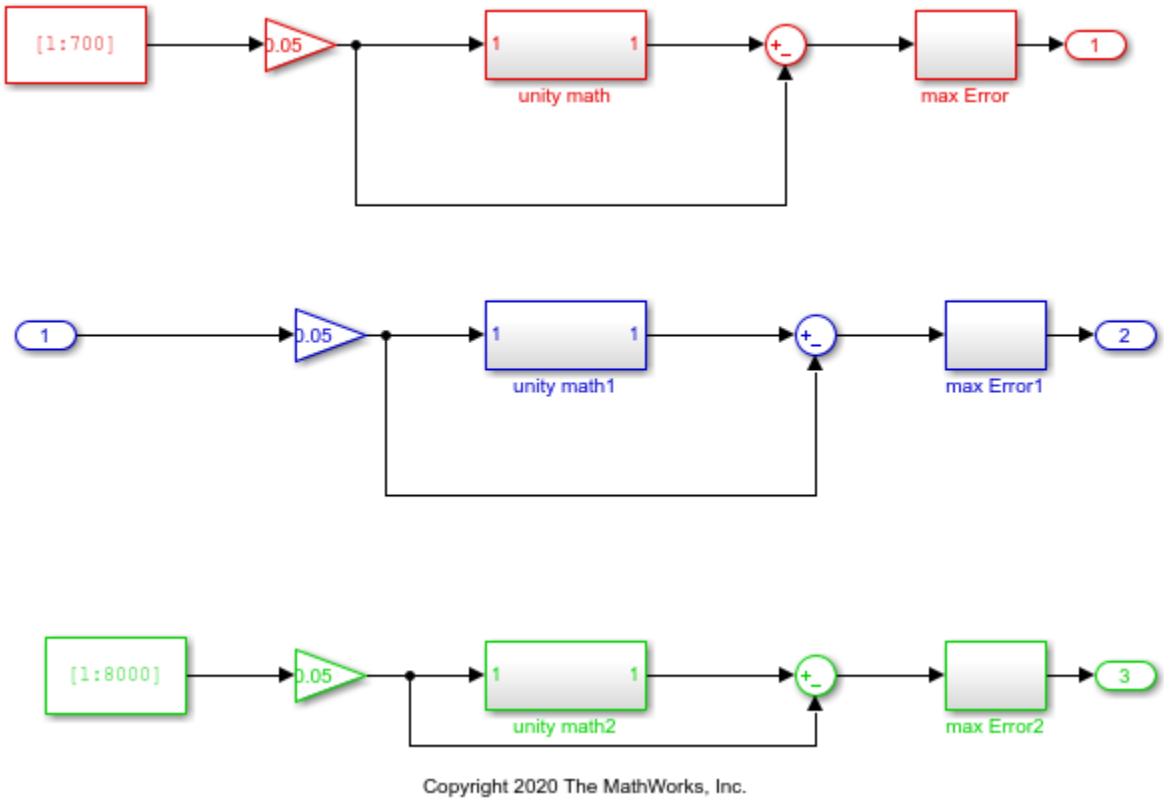
The explicit partitioning in the top-level model occurs in subsystem1.

```
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_mds_subsystem1'));
```



The explicit partitioning in the top-level model occurs in subsystem2.

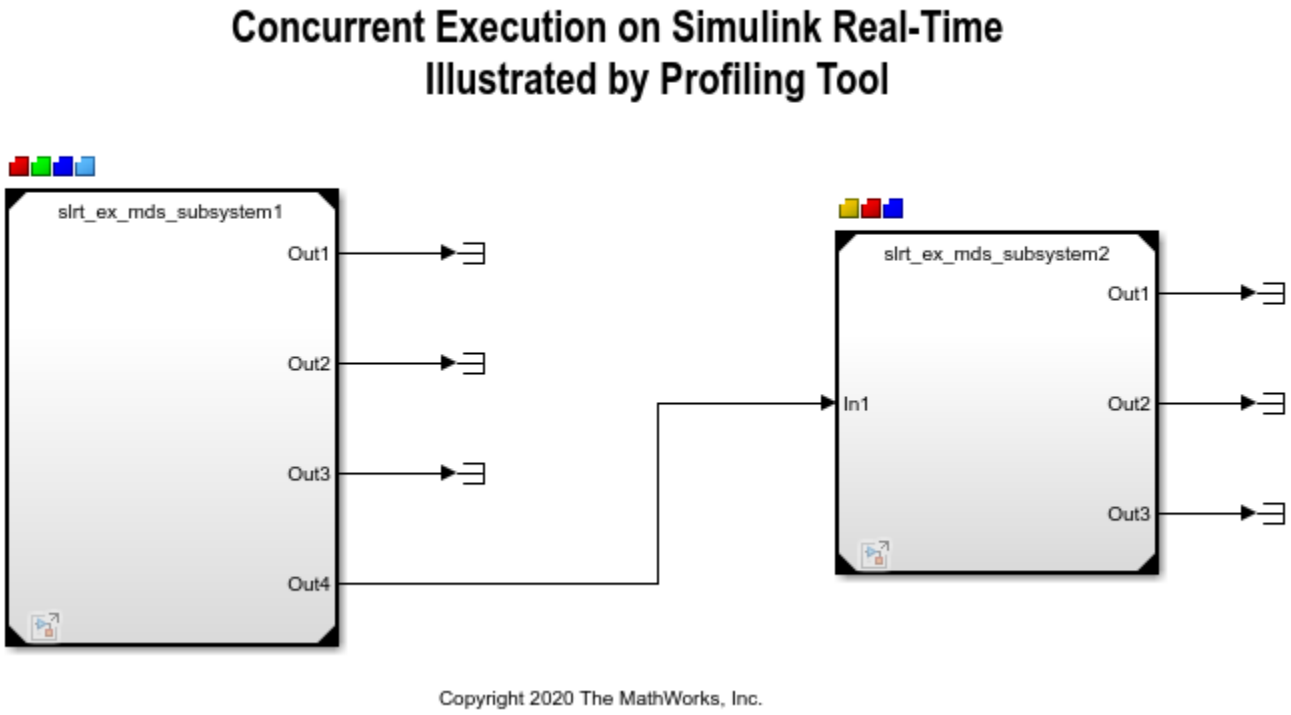
```
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_mds_subsystem2'));
%
```



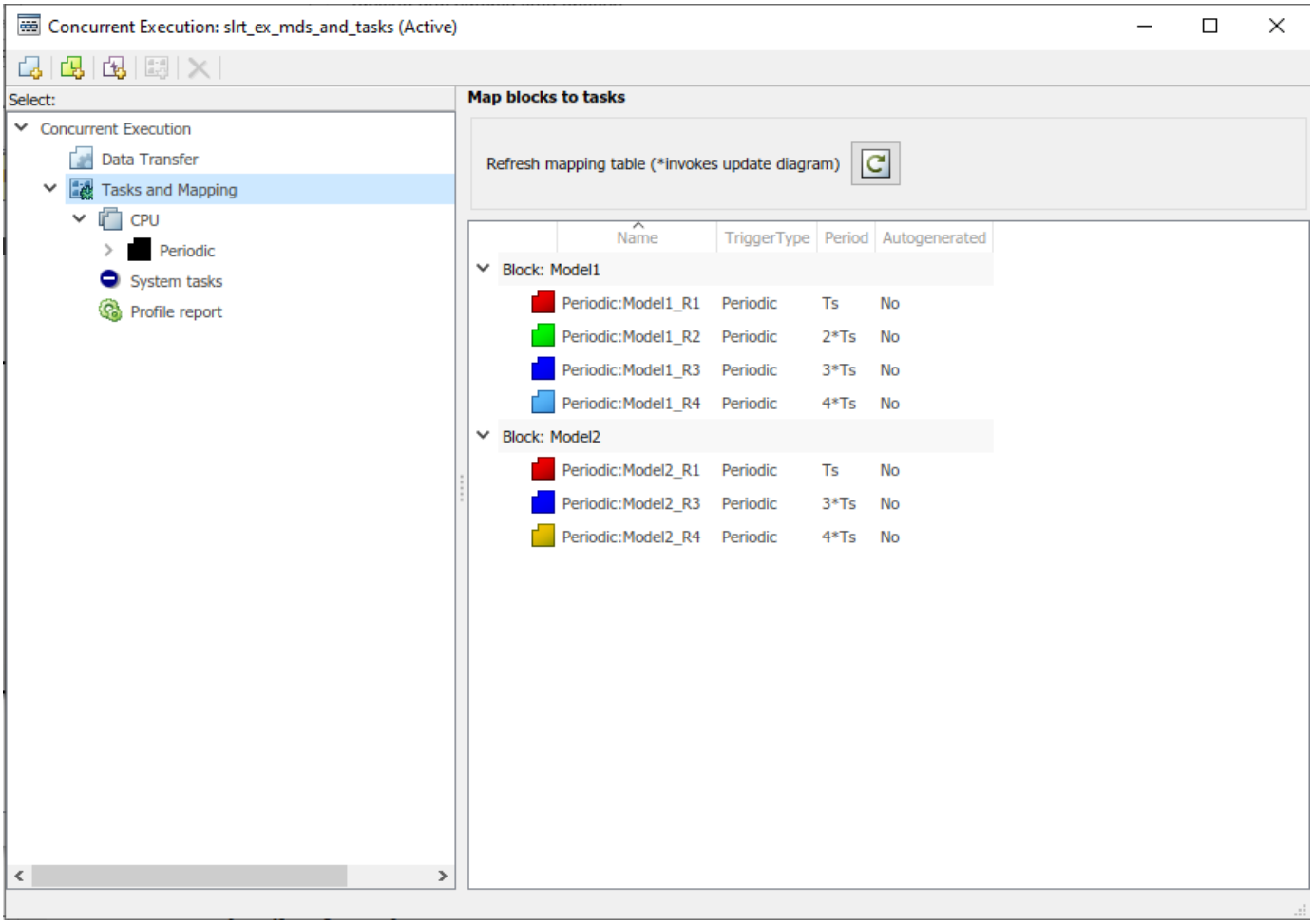
Open the model `slrt_ex_mds_and_tasks`. The model is mapped to seven threads: `Model1_R1`, `Model1_R2`, `Model1_R3`, `Model1_R4`, `Model2_R1`, `Model2_R3`, and `Model2_R4`.

These threads run at sample rates of `Ts`, `2*Ts`, `3*Ts`, `4*Ts`, `Ts`, `3*Ts`, and `4*Ts`.

```
mdl='slrt_ex_mds_and_tasks';
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples',mdl));
```



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 = slrealtime;
load(tg,mdl);
% Open TET Monitor
slrtTETMonitor;
% Start profiler on the target computer
startProfiler(tg);
start(tg);
pause(2);
stop(tg);

### Starting serial model reference simulation build
### Starting serial model reference code generation build
### Successful completion of build procedure for: slrt_ex_mds_subsystem1
### Successful completion of build procedure for: slrt_ex_mds_subsystem2
### Successful completion of build procedure for: slrt_ex_mds_and_tasks
Unable to find symbol(s) 'slrt_ex_mds_and_tasks_DW.Model1_InstanceData.rtb.RateTransition_Buf0, s
### Created MLDATX ..\slrt_ex_mds_and_tasks.mldatx

Build Summary
```

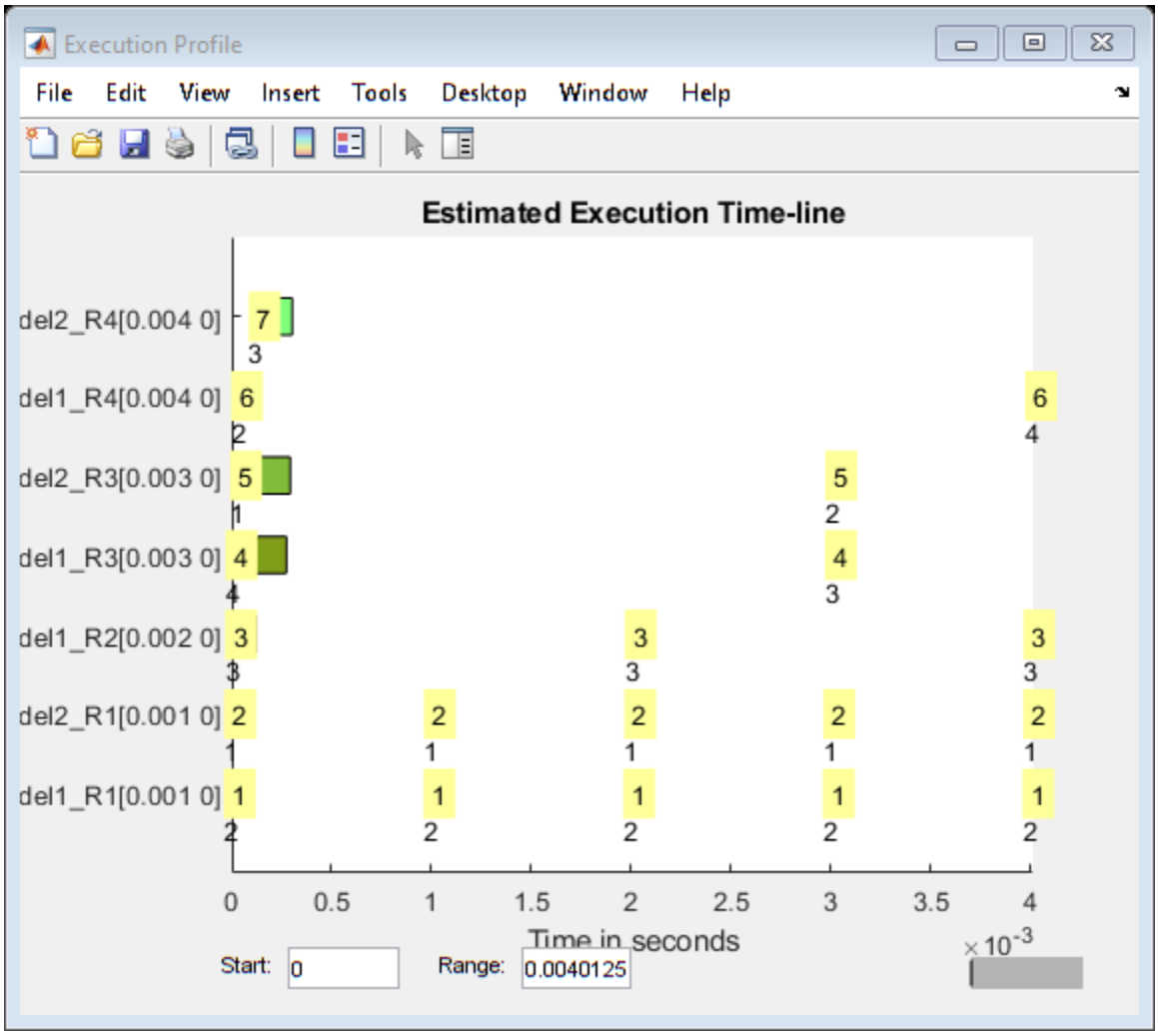
```
Code generation targets built:

Model                    Action                        Rebuild Reason
=========================================================================================================
slrt_ex_mds_subsystem1  Code generated and compiled  slrt_ex_mds_subsystem1.cpp does not exist.
slrt_ex_mds_subsystem2  Code generated and compiled  slrt_ex_mds_subsystem2.cpp does not exist.

Top model targets built:

Model                   Action                       Rebuild Reason
=========================================================================================================
slrt_ex_mds_and_tasks  Code generated and compiled  Code generation information file does not ex

3 of 3 models built (0 models already up to date)
Build duration: 0h 1m 22.867s
```

Display Profiling Data

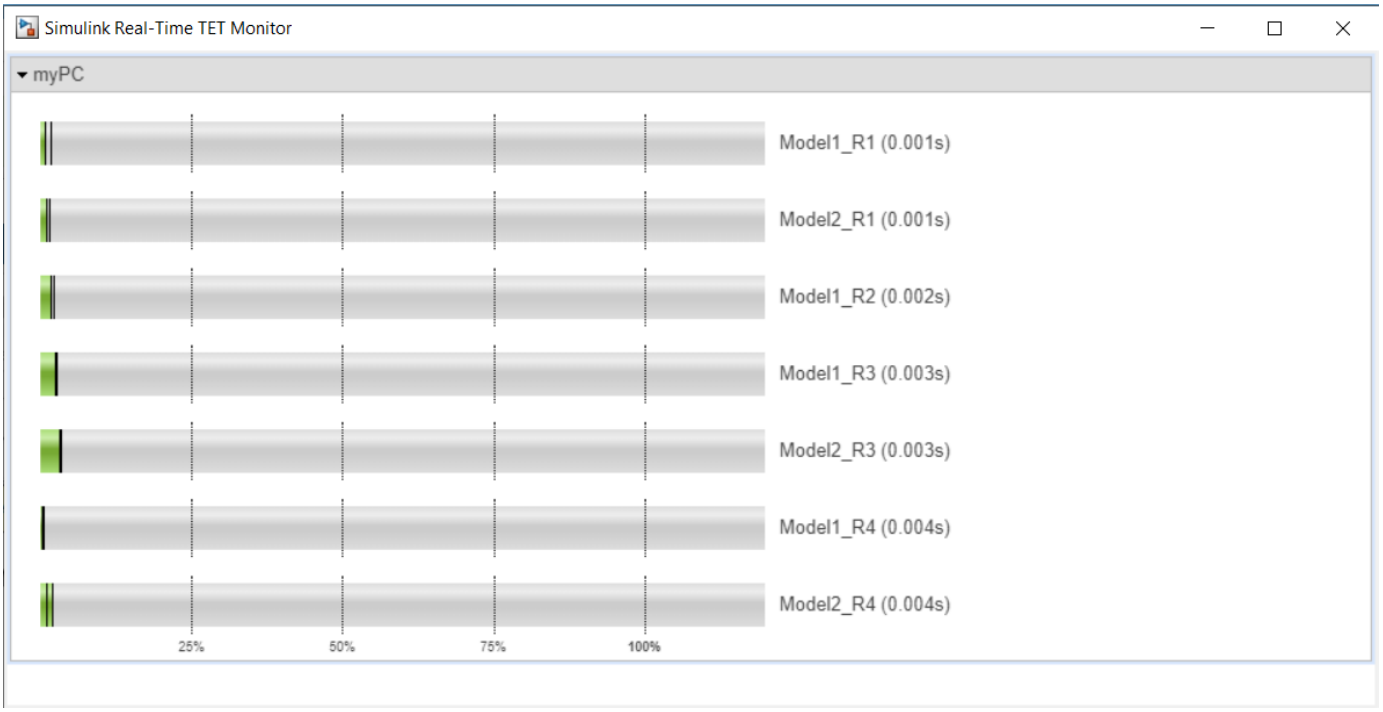
The profiling data shows the execution time of each thread on a multi-core target computer.

```
profData = tg.getProfilerData;
profData.plot;
```

```
Processing data on target computer ...
Transferring data from target computer ...
Processing data on host computer ...
```


Execution Profile
File Edit View Insert Tools Desktop Window Help
Estimated Execution Time-line
del2_R4[0.004 0]
del1_R4[0.004 0]
del2_R3[0.003 0]
del1_R3[0.003 0]
del1_R2[0.002 0]
del2_R1[0.001 0]
del1_R1[0.001 0]
0 0.5 1 1.5 2 2.5 3 3.5 4
Time in seconds
×10-3
Start: 0
Range: 0.0040125

View model's TET information on TET monitor



Close the Model

```
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 `slrt_ex_pendulum_100Hz`. The instrument panel contains these App Designer components:

- Drop down window — To show all the available target computers.
- Connect/disconnect button — To connect or disconnect the target computer chosen in the drop down window.
- Load button — To load the application to the target computer.
- Start/stop button — To start or stop the application on the target computer.
- Stop time edit field — To display and set the stop time of the application loaded on the target computer.
- Axes — To display an animation for the two inverted pendulum and cart system.
- Axes — To display signal output for responses to disrupting the pendulum.
- Nudge button — To apply input (nudge) to the cart that hold the pendulum.
- Reference position spinner — To change the reference position of the pendulum and cart system.
- Reference variation pattern knob — To add a variation pattern to the reference position of the pendulum and cart system.
- Amplitude slider — To adjust the amplitude of the chosen reference variation pattern.
- Frequency slider — To modify the frequency of the chosen reference variation pattern.

To stream signal and parameter data between the real-time application and the instrument panel app, the app uses the instrumentation object.

```
load_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_pendulum_100Hz'));
```

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 `slrt_ex_pendulum_100Hz`.
3. Connect the development computer to the target computer. Build the `slrt_ex_pendulum_100Hz` model.
4. Deploy the real-time application to the target computer.

In the MATLAB Command Window, type:

```
set_param('slrt_ex_pendulum_100Hz', 'RTWVerbose', 'off');
tg = slrealtime;
rtwbuild('slrt_ex_pendulum_100Hz');
load(tg,'slrt_ex_pendulum_100Hz');
```

```
### Generated code for 'slrt_ex_pendulum_100Hz' is up to date because no structural, parameter o
### Successful completion of build procedure for: slrt_ex_pendulum_100Hz
### Created MLDATX ..\slrt_ex_pendulum_100Hz.mldatx

Build Summary
```

```
0 of 1 models built (1 models already up to date)
Build duration: 0h 0m 3.951s
```

Run App Designer Instrument Panel App

The App Designer instrument panel app `slrt_ex_pendulumApp` provides controls to start and interact with the real-time application `slrt_ex_pendulum_100Hz`.

1. Run the app. To start the App Designer app `slrt_ex_pendulumApp.mlapp` and create the handle `app`, in the MATLAB Command Window, type:

```
openExample('SlrtAddAppDesignerAppToInvertedPendulumModelExample');
app = slrt_ex_pendulumApp;
```

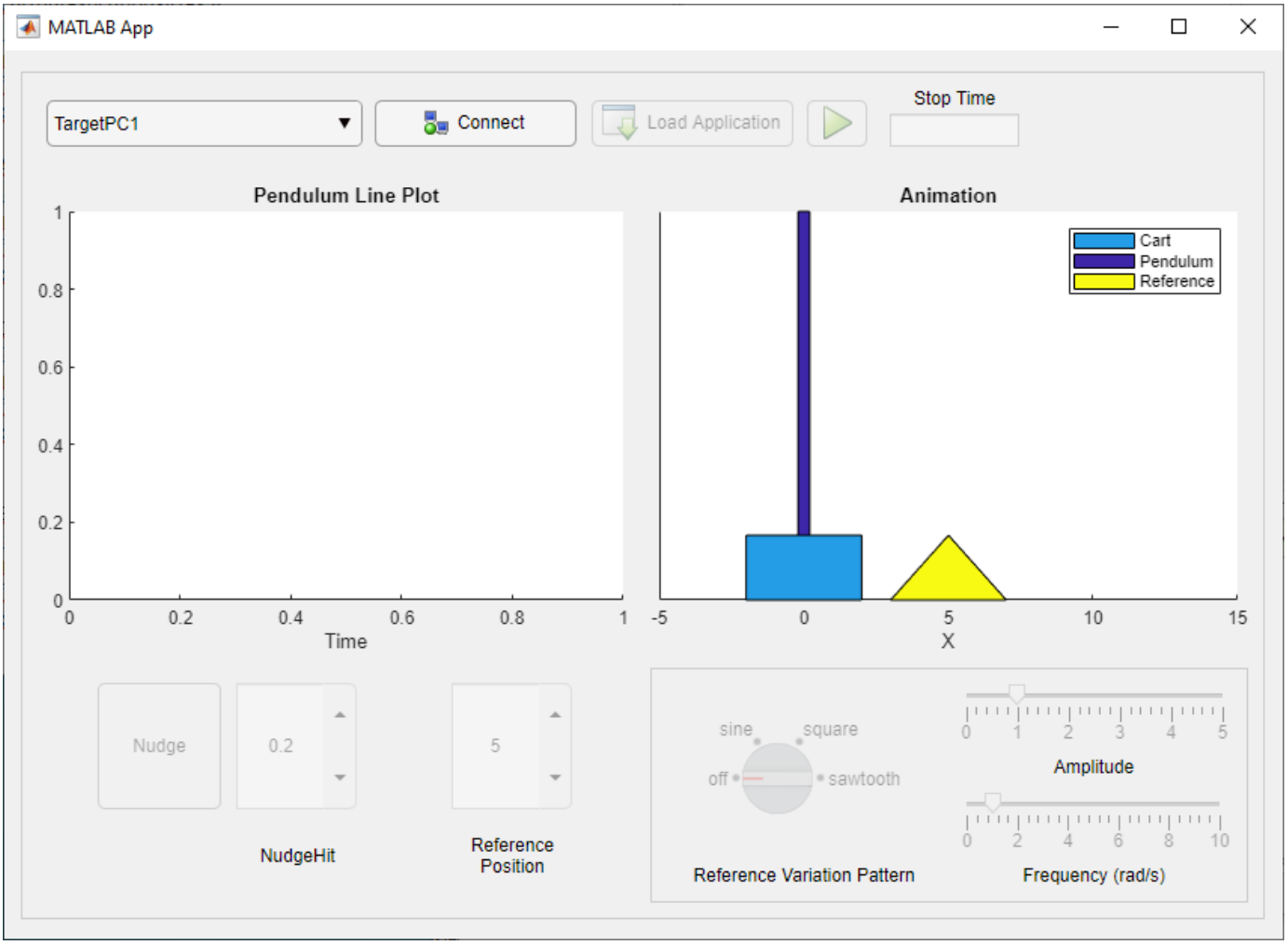
2. To connect with an available target computer, click the **connect** button. The text on the button will switch to 'disconnect' and the **load** button will be enabled.

3. To load the application to the target computer, click the **load** button. After the application is loaded on the target computer, the **start** button and **stop time** edit field will be enabled.

4. To set the stop time of the application, type your preferred stop time in the edit field and hit **enter** button.

5. To start running the application, click the **start** button.

6. To disrupt the equilibrium of the pendulum on each cart, click the **Nudge** button. You can adjust the nudge magnitude by using the value selection next to the button, hange the reference position by adjusting the value of reference position spinner, or choose a variation pattern for the reference position.



App Callback Code

The instrument panel app functionality is provided by callback code.

Comments in the callback code in the instrument panel app `slrt_ex_pendulumApp.mlapp` describe the callback operations and programming suggestions. To view the callback code, open `slrt_ex_pendulumApp.mlapp` in the App Designer, and then click the **Code View** tab. In the Command Window, type:

```
edit slrt_ex_pendulumApp
```

Specify Block Paths for Signals in Referenced Models

To stream data from signals in the model, see the use of `connectLine` functions in the `setupInstrumentation(app)` function in the app.

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

In the callback code, this processing is visible as `AcquireGroupData` signal groups in the `updateAnimationCallback` function.

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:

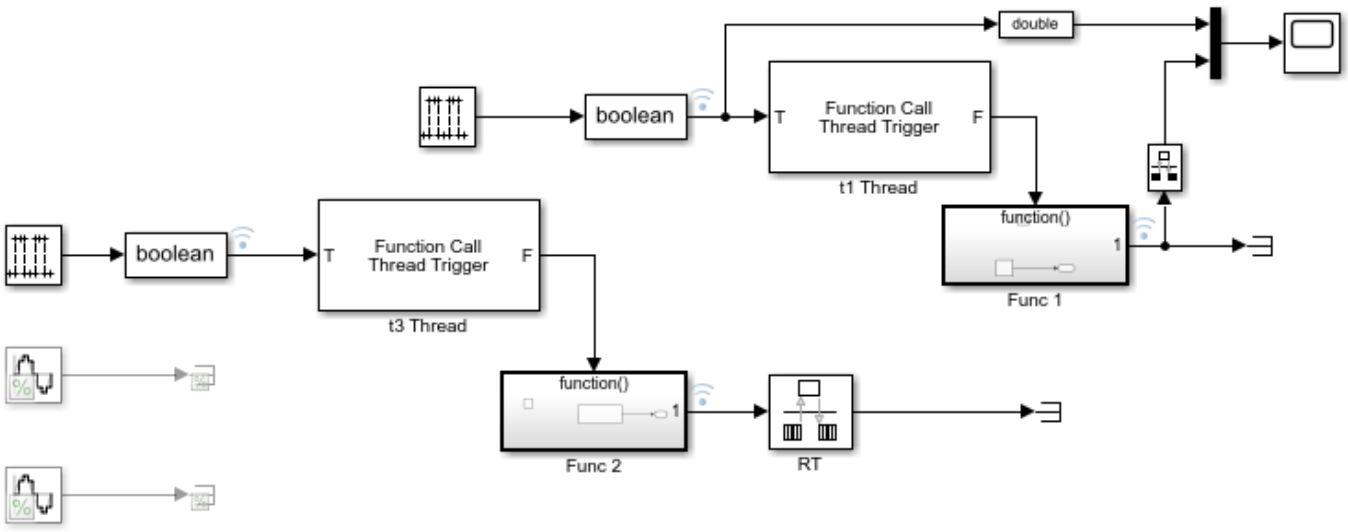
```
bdclose ('all');
```

Connect Triggered Subsystem by Using Thread Trigger

This example shows how to connect the Thread Trigger block and create a triggered subsystem. This not-often-used approach lets you use conditions in the model to trigger tasks instead of by using the much more typical approach of using a hardware interrupt from an I/O device in the target computer to trigger tasks.

To open this model, in the MATLAB Command Window, type:

```
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_thread_trigger_fc_subs
```



See Also

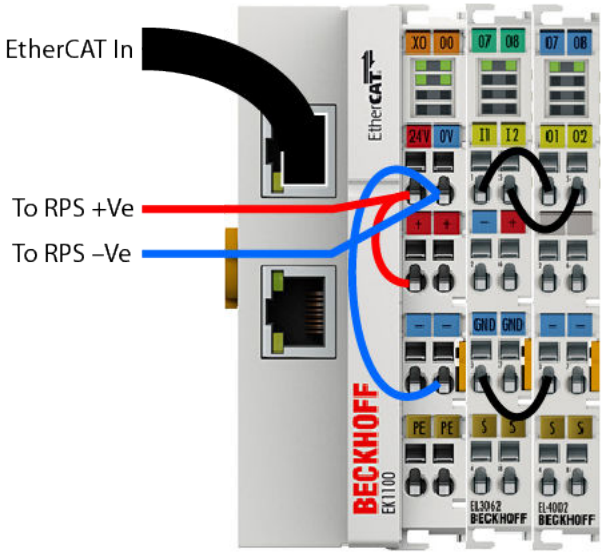
- Thread Trigger
- Function-Call Subsystem
- Triggered Subsystem
- “About RTOS Tasks and Priorities”
- “Execution Modes” on page 7-2

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 port that is reserved for EtherCAT using the Ethernet Configuration tool on the target computer. Use the reserved port for EtherCAT communication. This port is in addition to the port used for the Ethernet link between the development and target computers.



To test this model:

1. Connect the reserved network port in the target computer to the network IN port of the Beckoff® 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 “Modeling EtherCAT Networks”.

Open the Model

This model creates two sine wave signals and sends the signals to the EL4002 terminal. 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. To open the model, in the MATLAB Command Window, type:

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

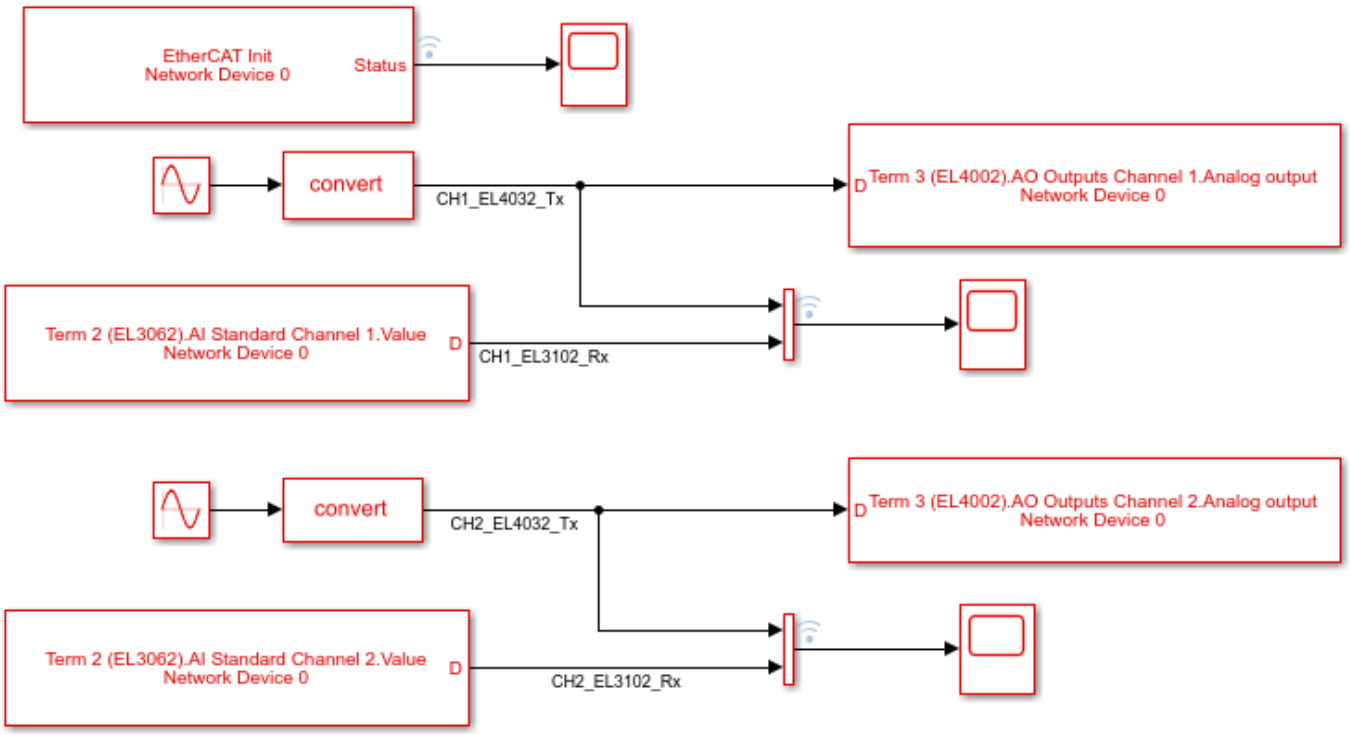


Figure 1: EtherCAT model using Beckhoff® analog I/O slave devices EL3062 and EL4002.

Configure the Model

Open the mask for the **EtherCAT Init** block and observe the pre-configured values. The EtherCAT slave devices that are daisy chained together with Ethernet cable is a Device, also referred to as an EtherCAT network. The Device Index selects one such chained EtherCAT network. The Ethernet Port Number identifies which Ethernet port to use to access that Device. The EtherCAT Init block connects these two so that other EtherCAT blocks use the Device Index to communicate with the slave devices on that EtherCAT network.

If you only have one connected network of EtherCAT slaves, and you have only reserved one Ethernet port with the Ethernet configuration tool, use Device Index = 0 and Ethernet Port Number = 1.

Create an ENI file for different A/D D/A slave devices, if needed

If you need to create a new ENI file you need to use a third-party EtherCAT configurator such as TwinCAT 3 from Beckhoff that you install on a development computer. The EtherCAT configuration (ENI) file preconfigured for this model is **BeckhoffAIOconfig.xml**.

The ENI (EtherCAT Network Information) file that is provided with this example has an EK1100 with EL3062 and EL4002 slaves attached, in that order. If you have different analog IO modules, you need to create a new ENI file for that collection.

For an overview of the process for creating an ENI file, see “Configure EtherCAT Network by Using TwinCAT 3”.

Each EtherCAT configuration file (ENI 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 ENI 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 EL3102 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`.

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`.

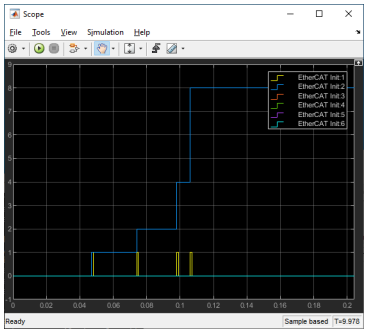
Build, Download, and Run the Model

To build, download, and run the model:

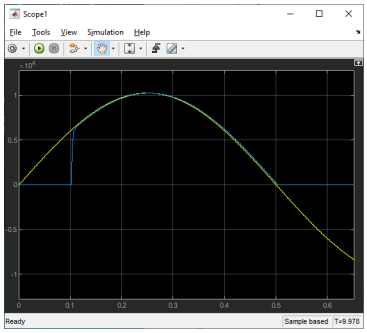
1. In the **Simulink Editor**, from the targets list on the **Real-Time** tab, select the target computer on which to run the real-time application.
2. Click **Run on Target**.

If you open the three host side scopes by double clicking each, data is relayed from the target back to the development computer and displayed there.

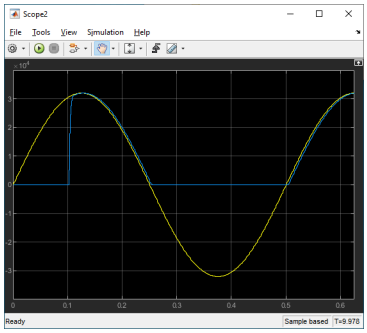
Zooming into the first quarter second of execution for this model, on all three of the scopes shows:



Scope shows the notifications in yellow and the state in blue. The only notificatons have the value of 1 which has meaning that the state has changed. Each of those is aligned with a step in the state output. Because this ENI file does not use distributed clock synchronization, the progression to Op state is very fast, just over 0.1 second. Also, because this ENI file does not use distributed clocks, the last 4 elements of the vector out of the init block are all 0.



Scope1 shows the 1Hz sinewave output in yellow and the value read back by the A/D in blue. Notice that there is no input until the EtherCAT state has progressed to Op state just after .1 seconds. If you zoom in tighter, you notice that the A/D signal is delayed by several clock cycles from the D/A output. This is because the A/D is read before the D/A is commanded to a new value and the A/D value is not available until the next sample time. This D/A slave takes a signed int as input, but can only output in the range of [0,+10] volts so the input values only show positive values, even though this A/D can read inputs from [-10,+10].



Scope2 shows the 2Hz sinewave sent to the second D/A channel, with the same delayed start on input and delayed response to a change.

The second way is to build the model (rtwbuild() or ^B), download from the MATLAB command line and run from the command line. In that case, the scope blocks do not display data, but the Simulation Data Inspector can be used.

The model is preconfigured to run for 10 seconds. If you want to run the model longer, use the MODELING tab on the model editor toolstrip to change the Stop Time and rebuild.

Display the Target Computer data

After running the model, you can also use the Simulation Data Inspector to view any signal that has been marked for signal logging. Signals marked for signal logging have a dot with two arcs above it in the model editor.

Stop and Close the Model

When the example completes its run, stop and close the model.

```
close_system('slrt_ex_ethercat_beckhoff_aio');
```

See Also

- “EtherCAT® Communication with Beckhoff® Digital IO Slave Devices EL1004 and EL2004” on page 13-23
- “Modeling EtherCAT Networks”
- “Configure EtherCAT Network by Using TwinCAT 3”

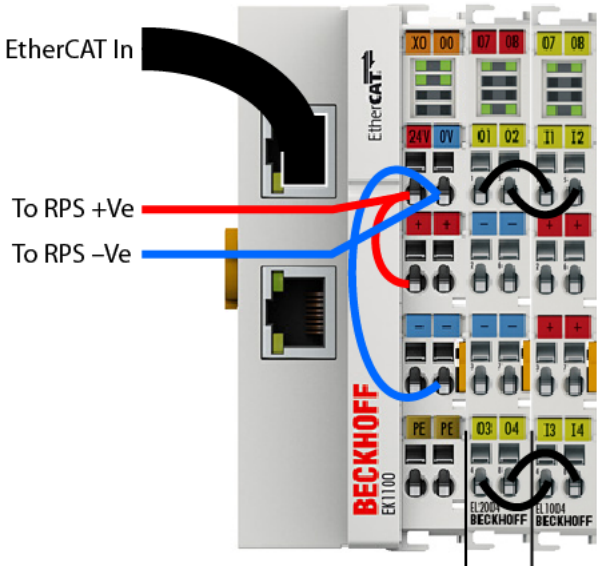
EtherCAT® Communication with Beckhoff® Digital IO Slave Devices EL1004 and EL2004

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 attached to an EK1100 coupler.

EtherCAT in Simulink Real-Time requires a dedicated network port on the target computer that is reserved for EtherCAT use by using the Ethernet configuration tool. Configure the dedicated port for EtherCAT communication, not with an IP address. The dedicated port must be distinct from the port used for the Ethernet link between the development and target computers.



To test this model:

1. Connect the port that is reserved for EtherCAT in the target computer to the network IN port of the Beckoff® EK1100 coupler.
2. Assemble Terminals EL1004 and EL2004 with Coupler EK1100.
3. Loop back the first two I/O ports: Connect ports numbered O1 and O2 of Terminal EL2004 to ports numbered I1 and I2 of Terminal EL1004. Ports O3, O4, I3 and I4 are not used by this example.
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

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 can be configured with either the full path to the ENI file or with a relative path that can be found with the MATLAB `which` command. Copy the example configuration file from the example folder to the current folder. To open the model, in the MATLAB Command Window, type:

```
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_ethercat_beckhoff_dio
```

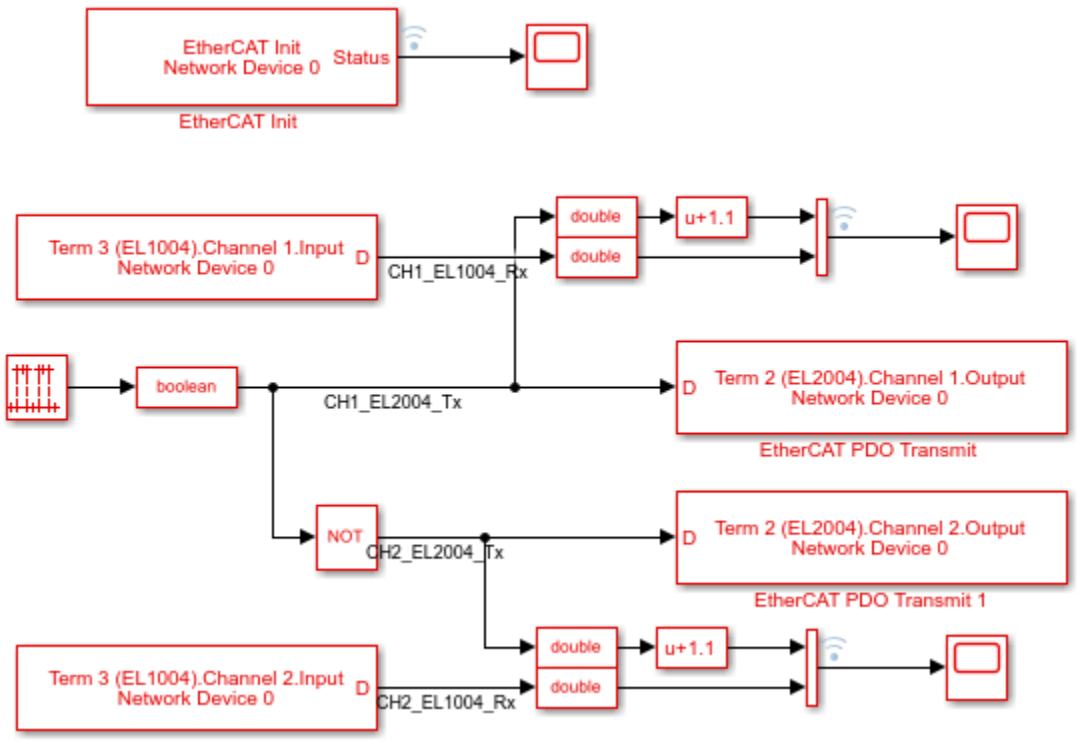


Figure 1: EtherCAT model using Beckhoff® digital I/O terminals EL1004 and EL2004.

Configure the Model

Open the parameter dialog for the **EtherCAT Init** block and observe the pre-configured values. The EtherCAT slave devices that are daisy chained together with Ethernet cable is a Device, also referred to as an EtherCAT network. The Device Index selects one such chained EtherCAT network. The Ethernet Port Number identifies which Ethernet port to use to access that Device. The EtherCAT Init block connects these two so that other EtherCAT blocks use the Device Index to communicate with the slave devices on that EtherCAT network.

If you only have one connected network of EtherCAT slaves, and you have only reserved one Ethernet port with the Ethernet configuration tool, use Device Index = 0 and Ethernet Port Number = 1.

Describe Network with Configurator

Using a third-party EtherCAT configuration program that you install on a development computer, generate an EtherCAT configuration (ENI) file. The ENI file for this example is `BeckhoffDIOconfig.xml`.

The ENI (EtherCAT Network Information) file that is provided with this example has an EK1100 with EL2004 and EL1004 slaves attached, in that order. If you have different digital IO modules, you need to create a new ENI file for that collection.

For an overview of the process for creating an ENI file, see “Configure EtherCAT Network by Using TwinCAT 3”.

Each EtherCAT configuration file (ENI 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 ENI 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`.

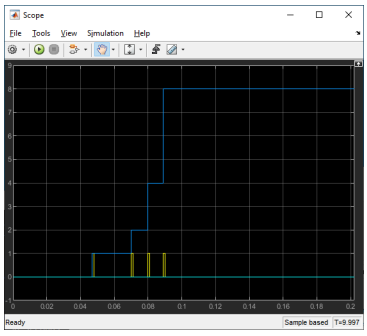
Build, Download, and Run the Model

To build, download, and run the model:

1. In the **Simulink Editor**, from the targets list on the **Real-Time** tab, select the target computer on which to run the real-time application.
2. Click **Run on Target**.

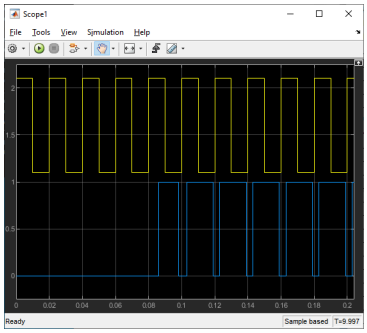
If you open the three host side scopes by double clicking each, data is relayed from the target back to the development computer and displayed.

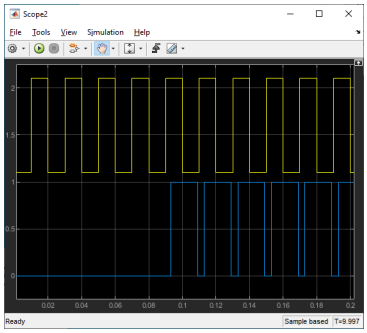
The three scopes are Scope, Scope1 and Scope2.



Both notifications from the EtherCAT state machine and the current state are displayed in Scope. Since there are no errors, the only notifications visible are the value 1 which means a state change at

that execution time step. The current state indicates the state that resulted from that state change. Notice that Op (=8) state is reached very fast since this ENI file does not include distributed clock synchronization. This view is zoomed in to the first 0.2 seconds of execution to show the transition to Op state clearly.





Scope1 and Scope2 show almost the same thing, but for two different channels. The signal is inverted between the two of them as can be seen if you compare the time when there is a rising edge in the yellow trace. The time step when physical IO starts is when the state goes to Op state. Before that, there is no input or output and the blue traces stay at 0. There is a time delay between the signal being sent to the output blocks and the signal that comes back from the input blocks for two reasons.

There is a 2 time step delay due to EtherCAT communication which is followed by an additional delay due to the speed of the hardware IO. The return signal shows a definite asymmetry between the delay after sending a rising edge and the delay after sending a falling edge. If you inspect the actual output signal with an oscilloscope, you see that the output is actually symmetric, but it is the input that has additional hardware delay in it. Other DIO slaves show different delay characteristics.

The model is preconfigured to run for 10 seconds. If you want to run the model longer, pull down the **Run on Target** menu and change the number on the bottom line. Press the green arrow to configure, build, and run.

Display the Target Computer data

After running the model, you can use the Simulation Data Inspector to view any signal that has been marked for signal logging. Signals marked for signal logging have a dot with two arcs above it in the model editor.

Observations to notice

Because data is both received from and sent to the slaves as the final action during execution and received data on one time step is only available during the following time step, you should see a delay

between the data being sent and the return value. In addition with digital IO, writing a new value to an output takes a few microseconds to appear as a change in voltage which is after the input was captured, there is a 2 time step delay from an output edge until the input shows the edge in the data.

Close the Model

When the example completes its run, stop and close the model.

```
close_system('slrt_ex_ethercat_beckhoff_dio');
```

See Also

- “EtherCAT® Communication with Beckhoff® Analog IO Slave Devices EL3062 and EL4002” on page 13-18
- “Modeling EtherCAT Networks”
- “Configure EtherCAT Network by Using TwinCAT 3”

EtherCAT® Communication - Motor Velocity Control with Accelnet™ Drive

This example shows how to control the velocity of a motor by using EtherCAT communication. The example motor drive is from Copley Instruments. This drive uses the CIA-402 (Can In Automation 402) device profile common to many drives. The example can work with other CIA-402 EtherCAT drives if you generate an appropriate ENI file.

Requirements

This example is preconfigured to use 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.

EtherCAT in Simulink Real-Time requires a dedicated network port on the target computer that is reserved for EtherCAT use by using the Ethernet configuration tool. Configure the dedicated port for EtherCAT communication, not with an IP address. The dedicated port must be distinct from the port used for the Ethernet link between the development and target computers.

To test this model:

1. Connect the dedicated network port 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 “EtherCAT® Communication - Sequenced Writing Slave CoE Configuration Variables” on page 13-57.

Open the Model

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. To open the model, in the MATLAB Command Window, type:

```
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_ethercatVelocityContro
```

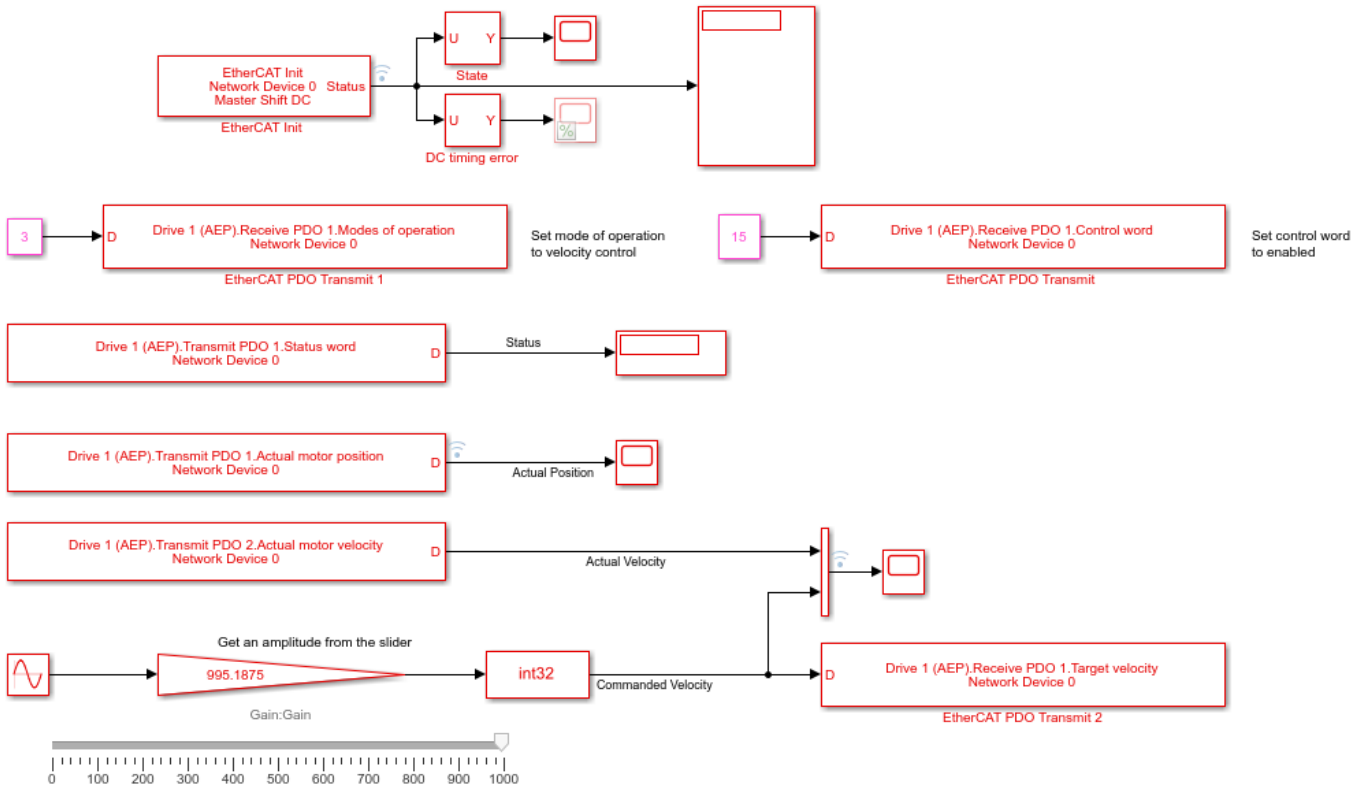


Figure 1: EtherCAT model for motor velocity control.

Configure the Model

Open the parameter dialog for the **EtherCAT Init** block and observe the pre-configured values. The EtherCAT slave devices that are daisy chained together with Ethernet cable is a Device, also referred to as an EtherCAT network. The Device Index selects one such chained EtherCAT network. The Ethernet Port Number identifies which Ethernet port to use to access that Device. The EtherCAT Init block connects these two so that other EtherCAT blocks use the Device Index to communicate with the slave devices on that EtherCAT network.

If you only have one connected network of EtherCAT slaves, and you have only reserved one Ethernet port with the Ethernet configuration tool, use Device Index = 0 and Ethernet Port Number = 1.

Create an ENI file for a Different CIA-402 Drive

If you need to create a new ENI file, you need to use a third-party EtherCAT configurator such as TwinCAT 3 from Beckhoff that you install on a development computer. The EtherCAT configuration (ENI) file preconfigured for this model is `CopleyMotorVelocityConfig.xml`.

Each EtherCAT configuration file (ENI 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 one Accelnet™ drive from Copley Controls. If you have a different EtherCAT drive that uses the CIA-402 command set, this example still works, but you need to create a new ENI file that uses your drive.

For an overview of the process for creating an ENI file, see “Configure EtherCAT Network by Using TwinCAT 3”.

For this example, four 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 Target Velocity. The fourth variable: Profile Target Position is used in example “EtherCAT® Communication - Motor Position Control with an Accelnet™ Drive” on page 13-33.

- 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 EtherCAT User Guide from Copley Controls for details on the bits mapping of this variable. This variable and bit mapping is in the CIA-402 standard set.
- 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`. For details on supported modes of operation, see the Refer to the Copley Controls EtherCAT User Guide. This variable and bit mapping is in the CIA-402 standard set.
- 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 using the slider connected to the gain block parameter.

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. Note that EtherCAT refers to variables that the slave sets as transmit variables which are received by the target model.

- 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 by opening the dialogs and selecting the current variable again.

Build, Download, and Run the Model

To build, download, and run the model:

1. In the **Simulink Editor**, from the targets list on the **Real-Time** tab, select the target computer on which to run the real-time application.
2. Click **Run on Target**.

If you open the host side scopes by double clicking each, data is relayed from the target back to the development computer and displayed.

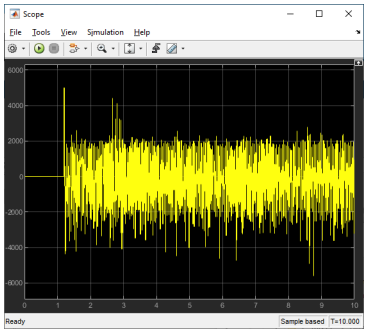
Included in the model is the ability to control the peak amplitude of the velocity. With the Run on Target button, the slider is active and connected to the Amplitude constant block.

The model is preconfigured to run for 10 seconds. If you want to run the model longer, pull down the **Run on Target** menu and change the number on the bottom line. Press the green arrow to configure, build, and run.

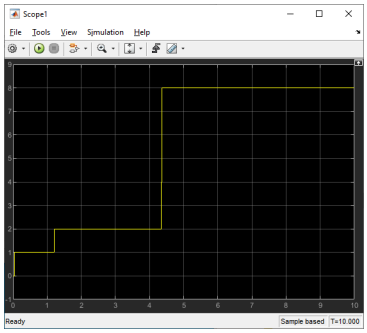
Display the Target Computer Scopes

If you run the model using the **Run on Target** button, external mode is connected and you can double click the scope blocks and see the data on the host. Also, the slider is active in external mode.

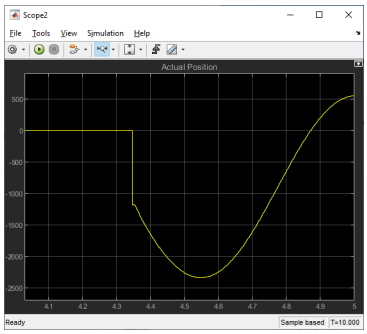
The Scope output images are referred to by the name in the title bar for each image. Discussion follows each image.



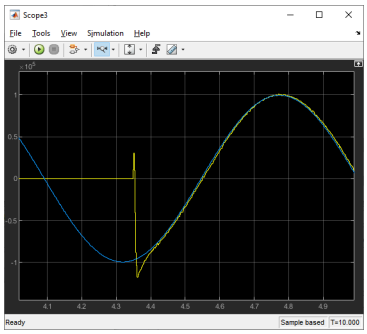
Scope shows the target to slave timing error as synchronization takes place using the bus shift method. The slave is adjusted to match the target timing resulting in a damped wave showing good phase lock around 4.5 to 5 seconds. The hash is a manifestation of the QNX execution scheduler and is what is expected. On this graph, 5000 is in nanoseconds, so this shows synchronization between 0 and -2 microseconds with residual random errors.



Scope1 shows the progression of states as the drive is initialized. Most of the time is taken to achieve time synchronization between target and EtherCAT slaves. The SafeOp (=4) to Op (=8) state transition occurs after a short settling time once the timing error is below the allowed error.



Scope2 shows the position of the motor which is a phase shifted version of the sine wave velocity that is sent to the motor. Note that the motor position does not change until the drive goes to Op state around 4.3 seconds.



Scope3 shows the velocity that is sent to the drive and the velocity read back from the drive. The velocity does not change until the drive goes into Op state.

After running the model, you can also use the Simulation Data Inspector to view any signal that has been marked for signal logging. Signals marked for signal logging have a dot with two arcs above it in the model editor.

Observations to notice

The velocity command for the motor is a low frequency sine wave. The actual velocity read back from the controller is delayed by several sample times and the actual position is out of phase by 90 degrees from the actual velocity, as expected for sinewave variation.

Stop and Close the Model

When the example completes its run, stop and close the model.

```
close_system('slrt_ex_ethercatVelocityControl');
```

See Also

- “EtherCAT® Communication - Motor Position Control with an Accelnet™ Drive” on page 13-33
- “Modeling EtherCAT Networks”
- “Configure EtherCAT Network by Using TwinCAT 3”

EtherCAT® Communication - Motor Position Control with an Accelnet™ Drive

This example shows how to control the position of a motor by using EtherCAT communication. The example motor drive is from Copley Instruments. This drive uses the CIA-402 (Can In Automation 402) device profile common to many drives. The example can work with other CIA-402 EtherCAT drives if you generate an appropriate ENI file.

Requirements

This example is preconfigured to use 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.

EtherCAT in Simulink Real-Time requires a dedicated network port on the target computer that is reserved for EtherCAT use by using the Ethernet configuration tool. Configure the dedicated port for EtherCAT communication, not with an IP address. The dedicated port must be distinct from the port used for the Ethernet link between the development and target computers.

To test this model:

1. Connect the port that is reserved for EtherCAT in the target computer to the EtherCAT IN port of the Accelnet™ drive.
2. Connect a motor to the Accelnet™ Drive.
3. Make sure the Accelnet™ drive is supplied with a 24-volt power source.
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 “Modeling EtherCAT Networks”.

Open the Model

This model creates a sine wave, and modulates it by multiplying by the value of the slider control. 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. Copy the example configuration file from the example folder to the current folder. To open the model, in the MATLAB Command Window, type:

```
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_ethercatPositionContro
```

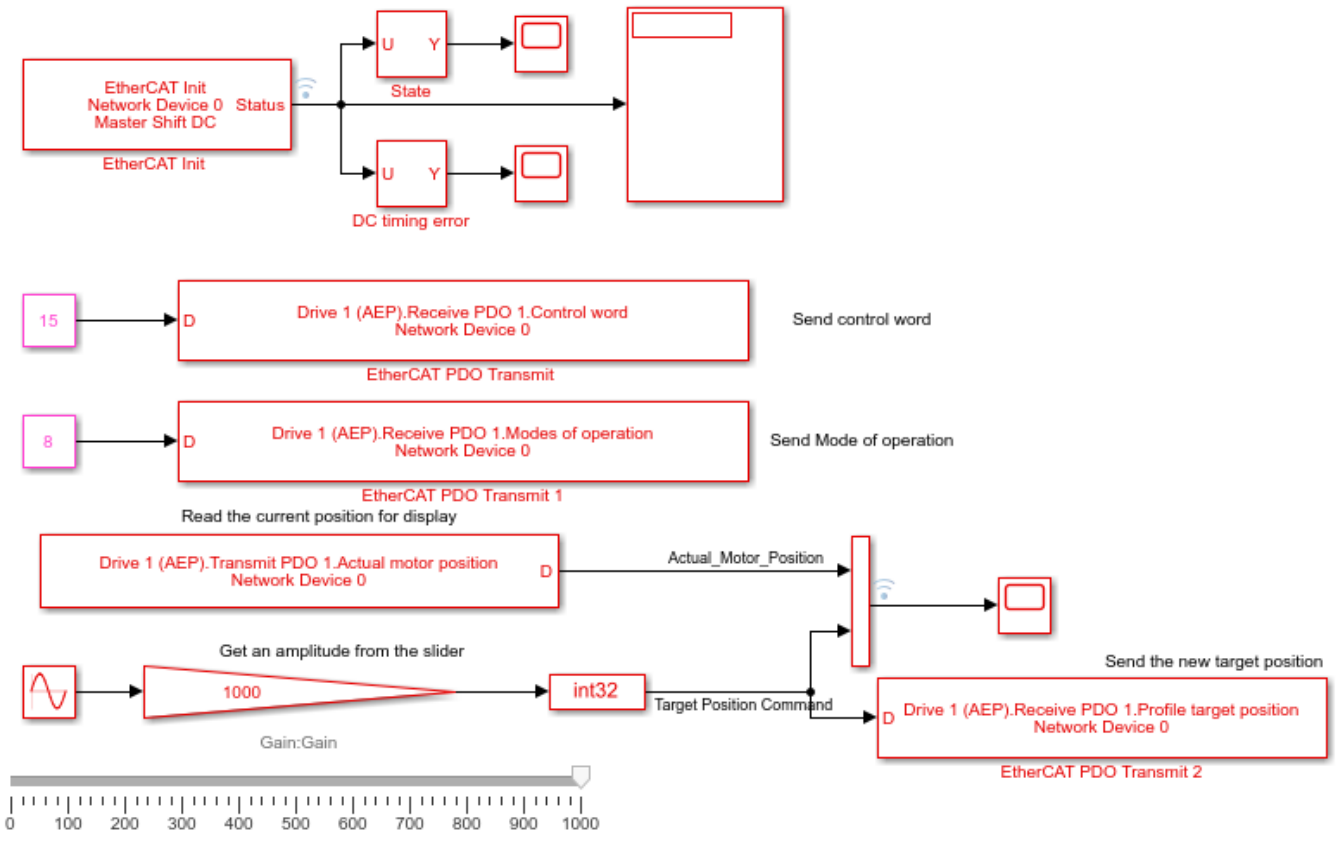


Figure 1: EtherCAT model for controlling the position of a motor.

Configure the Model

Open the parameter dialog for the **EtherCAT Init** block and observe the pre-configured values. The EtherCAT slave devices that are daisy chained together with Ethernet cable is a Device, also referred to as an EtherCAT network. The Device Index selects one such chained EtherCAT network. The Ethernet Port Number identifies which Ethernet port to use to access that Device. The EtherCAT Init block connects these two so that other EtherCAT blocks use the Device Index to communicate with the slave devices on that EtherCAT network.

If you only have one connected network of EtherCAT slaves, and you have only reserved one Ethernet port with the Ethernet configuration tool, use Device Index = 0 and Ethernet Port Number = 1.

Create an ENI File for a Different CIA-402 Drive

If you need to create a new ENI file, you need to use a third-party EtherCAT configurator such as TwinCAT 3 from Beckhoff that you install on a development computer. The EtherCAT configuration (ENI) file preconfigured for this model is `CopleyMotorPositionConfig.xml`.

Each ENI 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 one Accelnet™ drive from Copley Controls. If you have a different EtherCAT drive that uses the CIA-402 CanOpen profile, this example still works, but you need to create a new ENI file that uses your drive. Refer to Can In

Automation web site at www.can-cia.org for details. EtherCAT CoE embeds CanOpen addressing for process variables using EtherCAT as the transport layer instead of CAN.

An overview of the process for creating an ENI file is at “Configure EtherCAT Network by Using TwinCAT 3”

For this example, four 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 fourth variable: Target Velocity is used in example “EtherCAT® Communication - Motor Velocity Control with Accelnet™ Drive” on page 13-28.

- 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. For details on the bit mapping of this variable, refer to the Can In Automation web site. This variable and bit mapping is in the CIA-402 device profile.
- 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`. For detailed documentation, refer to the Can In Automation web site. This variable is in the CIA-402 device profile.
- 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 constant Amplitude value linked to the slider control in the model.

Transmit PDO variables (transmitted by the slave) are also defined in the configuration file and one is used in the **EtherCAT PDO Receive** block: `Actual Motor Position` for the drive. 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. Note that EtherCAT refers to variables that the slave sets as transmit variables which are received by the target model.

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 by opening the dialogs and selecting the current variable again.

Build, Download, and Run the Model

To build, download, and run the model:

1. In the **Simulink Editor**, from the targets list on the **Real-Time** tab, select the target computer on which to run the real-time application.
2. Click **Run on Target**.

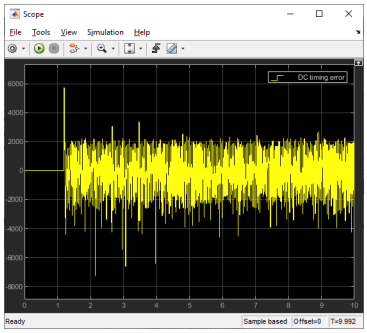
If you open the two host side scopes by double clicking each, data is relayed from the target back to the development computer and displayed.

Included in the model is the ability to control the amplitude of the cycling motion. With the Run on Target button, the slider is active and connected to the Amplitude constant block.

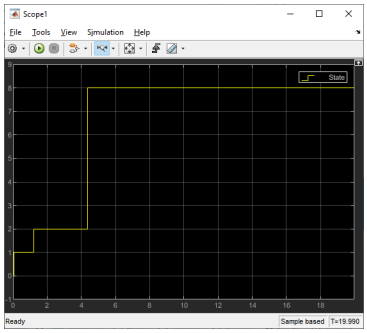
The model is preconfigured to run for 10 seconds. If you want to run the model longer, pull down the **Run on Target** menu and change the number on the bottom line. Press the green arrow to configure, build and run.

Display the Target Computer data

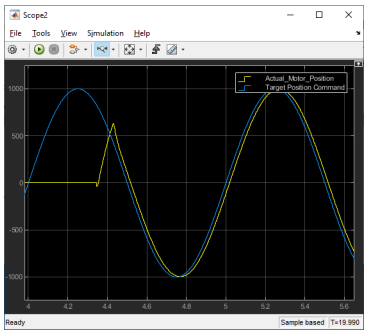
If you run the model using the **Run on Target** button, external mode is connected and you can double click the scope blocks and see the data on the host. Also, the slider is active in external mode.



Scope shows the Distributed Clocks timing difference between the master stack running on the target computer and the timing on the drive. This ENI file is configured to use Master Shift mode for DC. The clock on the target computer is adjusted to match the timing on the EtherCAT reference clock on the first DC enabled slave.



Scope1 shows the state progression from Idle to Init (=1) to PreOp (=2) to SafeOp (=4) for a very short time visible if you zoom in, to Op (=8) at around 4.3 seconds.



Scope2 shows both the sine wave being sent to the drive (blue) and the actual position (yellow). This is zoomed into the few seconds right when the drive went to Op state and external control starts. Since the motor hardware cannot respond instantaneously, and the commanded position is not 0, you see the actual position ramp up and overshoot slightly before settling down to follow the commanded position. The time delay between command and actual is roughly 18 sample time steps with this

drive. The controller inside the drive and motor inertia are responsible for this longer time delay. Other drives may have different delay characteristics.

After running the model, you can use the Simulation Data Inspector to view any signal that has been marked for signal logging. Signals marked for signal logging have a dot with two arcs above it in the model editor.

Observations to notice

This is a simple motor control example. The numerous tunable parameters inside the drive are not adjusted in this model. Adjusting those needs a more advanced model using the CoE/SDO blocks.

Close the Model

When the example completes its run, stop and close the model.

```
close_system('slrt_ex_ethercatPositionControl');
```

See Also

- “EtherCAT® Communication - Motor Velocity Control with Accelnet™ Drive” on page 13-28
- “Modeling EtherCAT Networks”
- “Configure EtherCAT Network by Using TwinCAT 3”

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

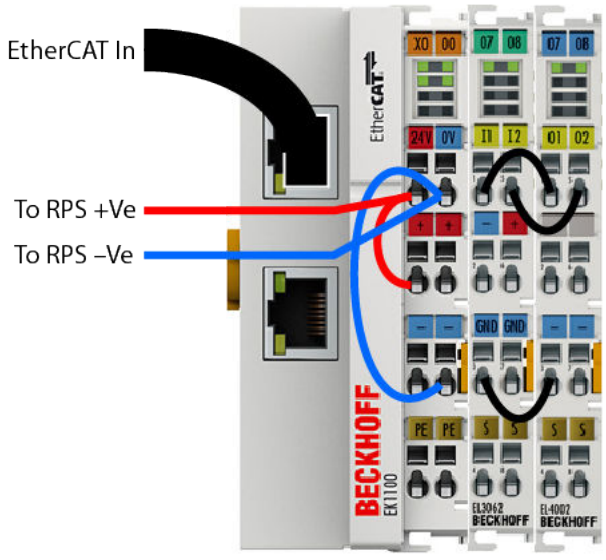
The EK1100 coupler connects EtherCAT with the EtherCAT terminals (ELxxxx). One station consists of an EK1100 coupler, any number of EtherCAT terminals, and a bus end terminal.

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 [-10, 10] V.

The EL4002 analog output terminal generates signals in the range of [0, 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 TwinCAT 3.1 XAE and Run Microsoft Visual Studio® with TwinCAT

The latest version of TwinCAT is the 3.x version and that is the preferred configuration tool.

The XAE sub version does not contain the full run time engine that runs on Windows. This is available free of charge from the Beckhoff web site. For use with Simulink Real-Time, you do not need the run time engine because you are using the run time implementation on the target. The full version with run time engine requires the purchase of a license from Beckhoff.

The TwinCAT 3.1 software requires a supported version of Microsoft Visual Studio to be installed. TwinCAT 3.1 uses the MSVC GUI integration and does not have a GUI by itself. The versions of MSVC with which a given version of TwinCAT works are discussed in the TwinCAT documentation. Installation finds supported MSVC versions on your machine and installs to them.

To install the TwinCAT 3.1 XAE:

1. Go to www.beckhoff.com and select **Download**.
2. Select TwinCAT 3 and download the setup.
3. Install TwinCAT 3.
4. Start Microsoft Visual Studio.
5. From the **TwinCAT** menu, select **Show Realtime Ethernet Compatible Devices**.
6. Select the Ethernet adapter for your EtherCAT device, then select **Install**.

Because TwinCAT installs an Ethernet filter inline with the Ethernet port you have selected, it is good practice to add an extra Ethernet port to use exclusively with EtherCAT to avoid any possible problem the filter can cause when sharing the Simulink Real-Time host-target communication port with TwinCAT.

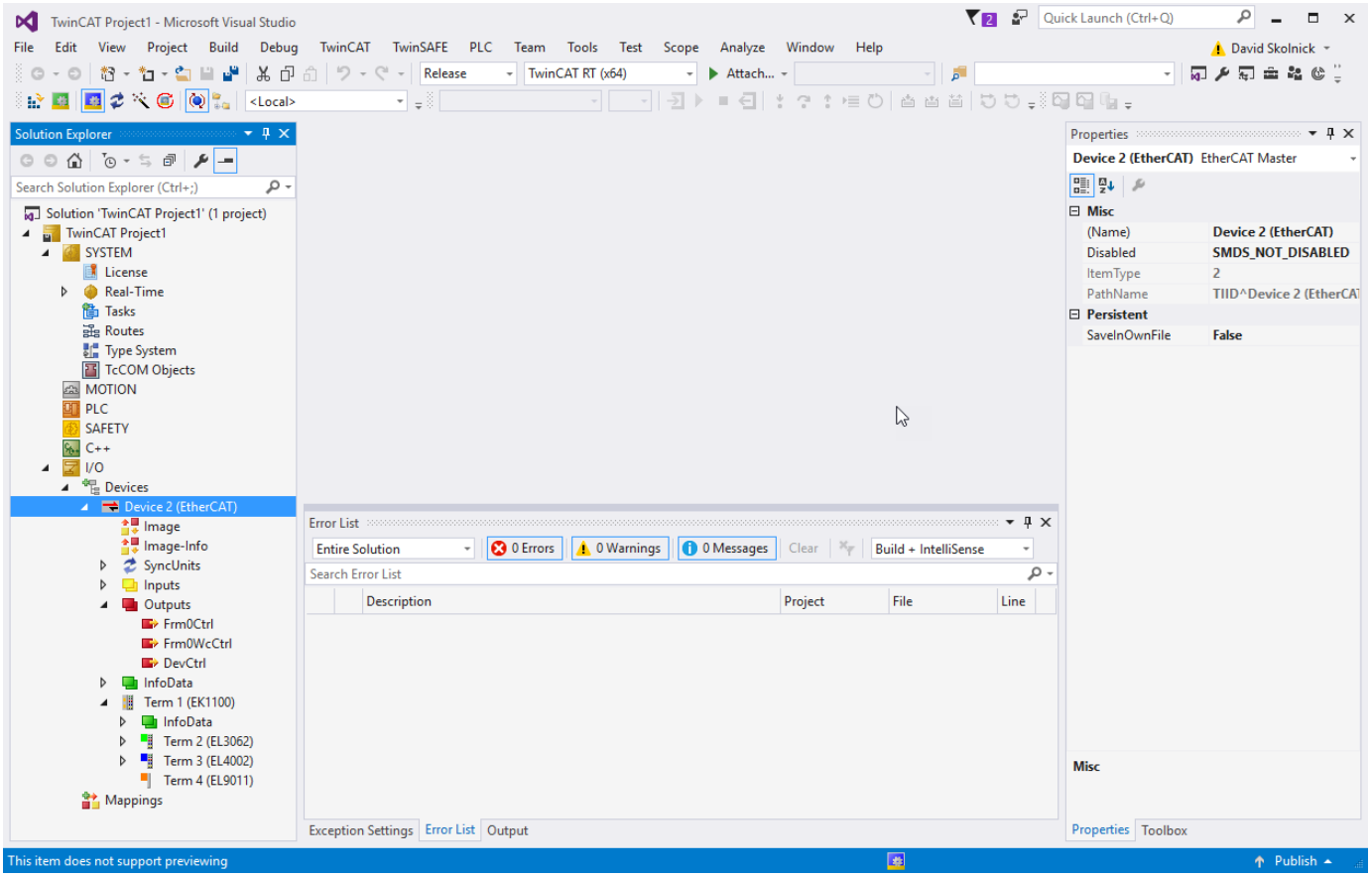
All EtherCAT configuration programs use EtherCAT Slave Information (ESI) files to describe the slaves that are found on the network. These Beckhoff configuration programs come prepopulated with mostly Beckhoff devices. To correctly configure an EtherCAT network with devices from other manufacturers, you may need to get the correct ESI file from the device manufacturer web site. If you do not have an ESI file for a slave on your network, the scan process does not populate the Solution Explorer with the correct name of the device and the read and write variables are not correct.

To create a new TwinCAT project in Visual Studio:

1. Start Visual Studio. Go to **File > New > Project**.
2. Under **Installed**, select **TwinCAT Projects** and click **OK**.
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 and you installed the full version. If you are using TwinCAT in evaluation mode, fill in the Captcha.
5. Observe the **Solution Explorer** pane the left 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.

When you first start TwinCAT the right information panel is not displayed. You need to double-click any item in the tree view the first time. After that the information dialog for any item is displayed by a single click on that item in the **Solution Explorer** tree view.



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 **Tasks** node and select **Add New Item**.

2 In the **Insert Task** dialog box, select **TwinCAT Task With Image**, provide a name for the task, and click **OK**.

3 Select the task that you created. The value **Cycle Ticks** determines the cycle time as a multiple of the **Base Time** determined on the **Real-Time** item. The default task time is set to 10 ms. If you are using Distributed Clock synchronization, a task time of 1-2 ms is the slowest that works with Master Shift DC mode.

4 Create at least one cyclic input/output task. Link this task to at least one input variable and one output variable on each slave device.

5 If you want to run faster than 1ms time, you need to change the base time on the **Real-Time** item above **Tasks**. On the **Settings** tab, you need to change the **Base Time** selection to a faster one.

By using distributed clocks (DC), the EtherCAT 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 configure it accordingly. For example, in the example configuration, the EL4002 supports DC. Most motion controllers (motor drives) support DC and some require it to get to Op state.

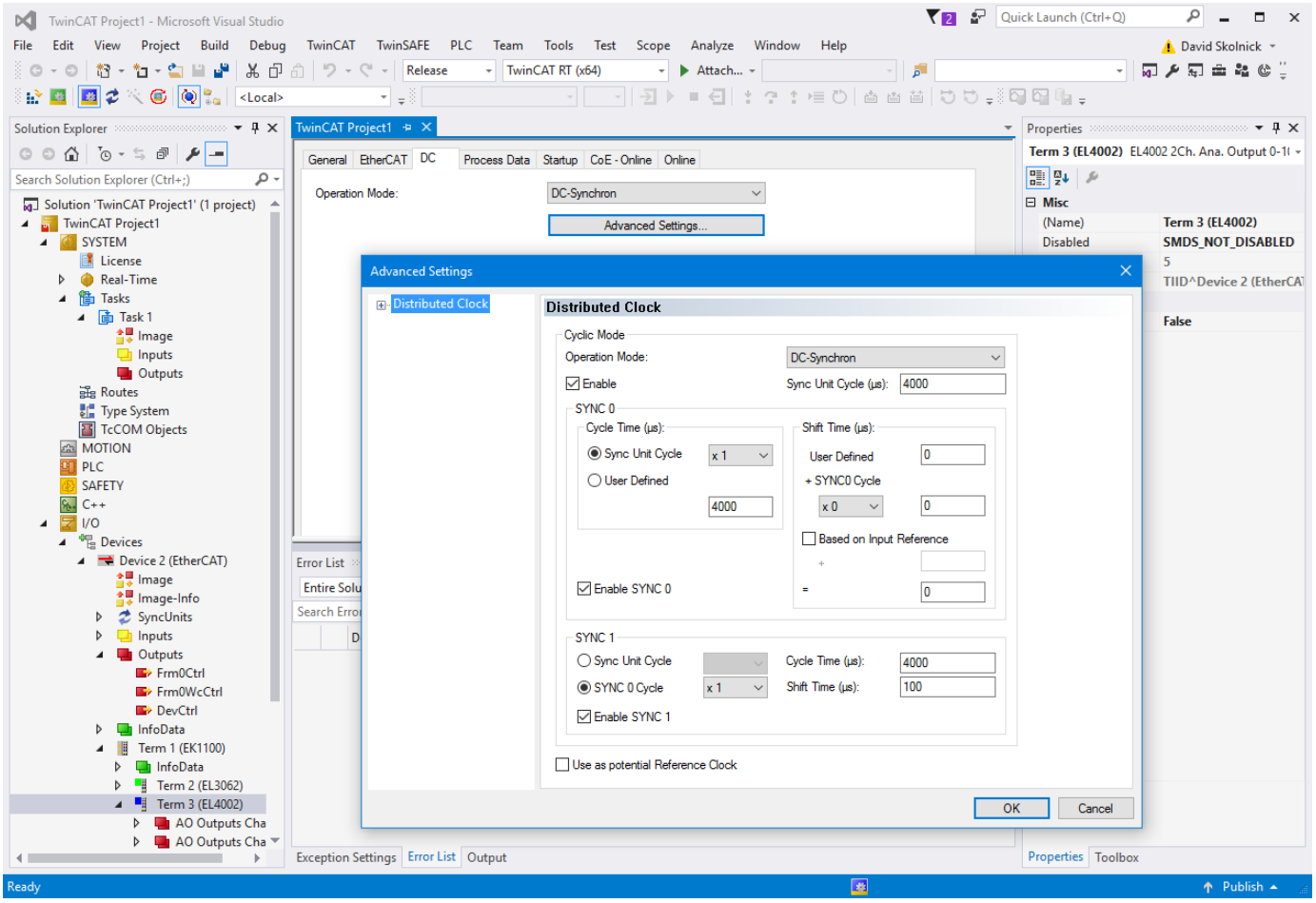
To configure EtherCAT DC:

Enable DC and choose **Bus shift** or **Master shift** DC mode.

1. Click on the **Device n (EtherCAT)** node
2. Select the **EtherCAT** tab in the information panel.
3. Click on **Advanced Settings** which opens a new dialog.
4. In the **Advanced Settings** dialog, select the **Distributed Clocks** page.
5. By default the **Automatic DC Mode Selection** box is checked, which generally gets you Master Shift DC mode. The first DC enabled slave is the reference clock and the Simulink Real-Time execution time is shifted slightly to align execution with the reference clock.
6. Deselect the **Automatic** mode and you have control over which DC mode to use or to turn it off. Next items are with **Automatic** deselected.
7. With **DC in use** deselected, no Distributed clock synchronization takes place. This results in much quicker initialization time to get to Op state, but there is no synchronization between slaves.
8. With **DC in use** selected, you have two different synchronization methods between the Speedgoat target machine and the EtherCAT slaves.
9. **Independent DC Time** uses the first DC enabled slave as the reference clock and the target machine clock is adjusted slightly to phase lock model execution to the first DC enabled slave.
10. **DC Time controlled by TwinCAT Time** should be read as **controlled by target machine time** in Simulink Real-Time. This is bus shift mode where the target machine is the reference clock and the slave execution times are shifted slightly to phase lock to the target machine.
11. Select both **Continuous Run-Time Measuring** and **Sync Window Monitoring**.

Now that you have chosen the DC mode to use, you can visit all of the DC enabled slaves in your network and set them to the correct mode. This example ENI file only supports one DC enabled slave, the EL4002.

1. Click the node **Term 3 (EL4002)** and select the **DC** tab.
2. By default, the **Operation Mode** is set to **SM-Synchron** which does not synchronize output to DC time. Change the **Operation Mode** to **DC-Synchron**. Different slaves have different names for the operating mode.
3. Click **Advanced Settings** and set the **Distributed Clock** options as shown.



To export and save the EtherCAT configuration, generate the ENI file:

1. 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 `simple_adda_eni.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. When you close the TwinCAT project, the editable version of this configuration is saved in the project file. You can modify the configuration by opening this project and by exporting to XML again.

Install and Run EtherCAT Configurator ET9000

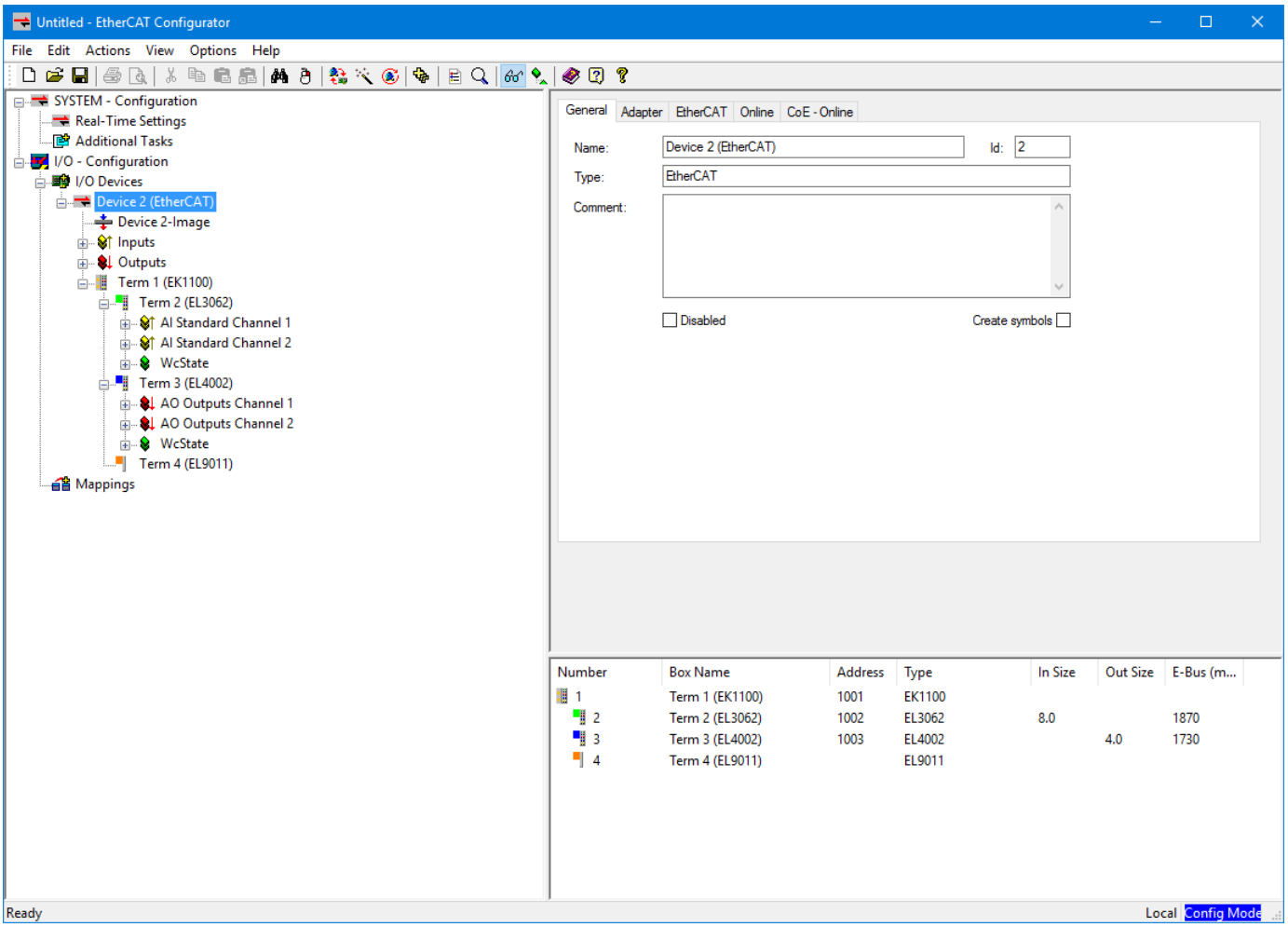
To install the EtherCAT configurator ET9000:

1. Go to 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 tree under **Devices** in the **I/O**. The EL3062 and the EL4002 devices appear under the EK1100 device.



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:

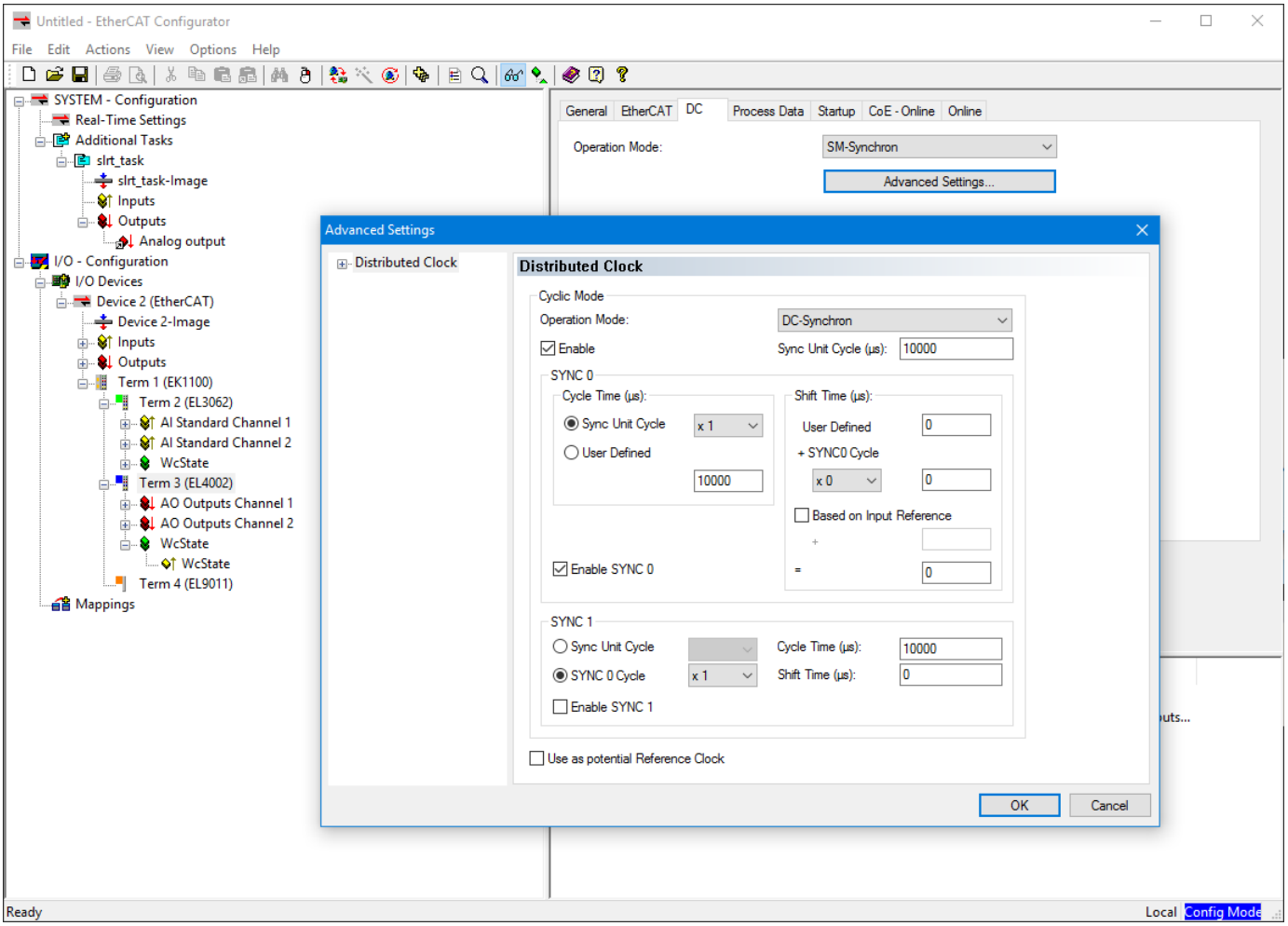
1. Under **SYSTEM - Configuration**, right-click **Additional Tasks > Append Task**.
2. Provide a name for the task and click **OK**. In this example, the name of the task is `slrt_task`.
3. Click the task. The value **Cycle Ticks** determines the cycle time. In the settings, it is set to `10ms`.

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 Run 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 `BeckhoffAIOconfig.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**.

See Also

- “Modeling EtherCAT Networks”
- “Configure EtherCAT Network by Using TwinCAT 3”
- “EtherCAT Configurator Component Mapping”

EtherCAT® Communication - Detect EtherCAT network failure and reset

This example shows how to use the EtherCAT Notifications block to detect a failure in the connected network and to restart the network when the failure is corrected.

Only a disconnected Ethernet cable into the first slave is detected by this example. More complicated failure situations can be detected if you study the pattern of notifications that result and write the embedded MATLAB block to account for those.

Requirements

To run this example as presented, you need a Beckhoff EK1100 with EL1202, EL2202-0100, EL3102 and EL4032 slave modules. The model does not write to any process objects. Replacing the ENI file with one appropriate to your network works as well.

EtherCAT in Simulink Real-Time requires a dedicated network port on the target computer that is reserved for EtherCAT use by using the Ethernet configuration tool. Configure the dedicated port for EtherCAT communication, not with an IP address. The dedicated port must be distinct from the port used for the Ethernet link between the development and target computers.

To test this model:

1. Connect the port that is reserved for EtherCAT in the target computer to the EtherCAT IN port of the EK1100 interface module.
2. Make sure the EK1100 is supplied with a 24-volt power source.
3. 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 “Modeling EtherCAT Networks”.

Open the Model

This model is a beginning of a full implementation to catch network failures and reinitialize the network once the failure is fixed. The simple state machine in the embedded MATLAB block can be replaced with a State Flow implementation, which may be necessary for more complicated failure detection and recovery.

The EtherCAT initialization block requires that the configuration ENI file is present in the current folder or on the MATLAB path because the file name is present without directory information.

If you want to modify this model to experiment with it, copy the example configuration file and the model file from the example folder to the current folder. To open the model, in the MATLAB Command Window, type:

```
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_ethercat_notifyreset'
```

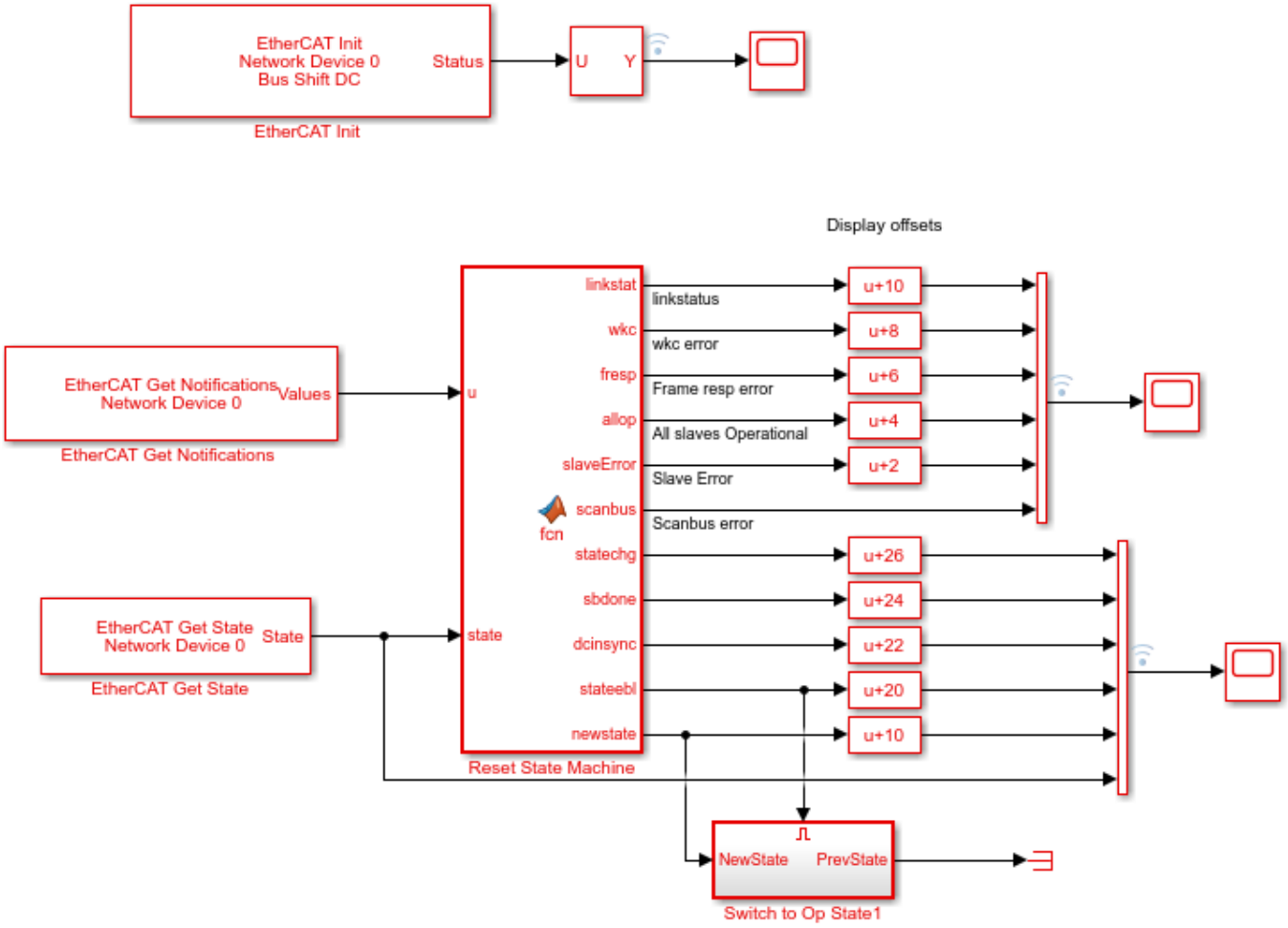


Figure 1: EtherCAT model for detecting a disconnected Ethernet cable at the first slave and reinitializing the network once the cable is reconnected.

Configure the Model

Open the parameter dialog for the **EtherCAT Init** block and observe the pre-configured values. The EtherCAT slave devices that are daisy chained together with Ethernet cable is a Device, also referred to as an EtherCAT network. The Device Index selects one such chained EtherCAT network. The Ethernet Port Number identifies which Ethernet port to use to access that Device. The EtherCAT Init block connects these two so that other EtherCAT blocks use the Device Index to communicate with the slave devices on that EtherCAT network.

If you only have one connected network of EtherCAT slaves, and you have only reserved one Ethernet port with the Ethernet configuration tool, use Device Index = 0 and Ethernet Port Number = 1.

Create an ENI File for a Different Slave Network

If you need to create a new ENI file you need to use a third-party EtherCAT configurator such as TwinCAT 3 from Beckhoff that you install on a development computer. The EtherCAT configuration (ENI) file preconfigured for this model is `Stack4_BS_1ms.xml`.

Each ENI 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 a Beckhoff EK1100 with EL1202, EL2202-0100, EL3102 and EL4032 slave modules. If you have a different EtherCAT drive, this example still works, but you need to create a new ENI file that uses your slave devices.

For an overview of the process for creating an ENI file, see “Configure EtherCAT Network by Using TwinCAT 3”.

Build, Download, and Run the Model

To build, download, and run the model:

1. In the **Simulink Editor**, from the targets list on the **Real-Time** tab, select the target computer on which to run the real-time application.
2. Click **Run on Target**.

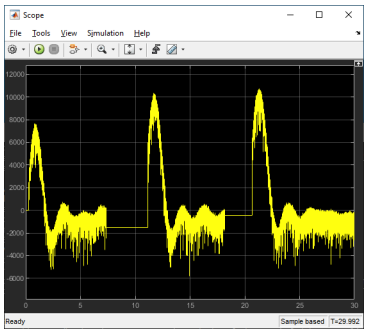
If you open the two scopes by double clicking each, the data is relayed from the target back to the development computer and displayed there.

The model is preconfigured to run for 15 seconds. If you want to run the model longer, pull down the **Run on Target** menu and change the number on the bottom line. Press the green arrow to configure, build, and run.

Display the Target Computer data

If you run the model using the **Run on Target** button, the external mode is connected and you can double click the scope blocks and see the data on the development computer. The **Display** blocks also work.

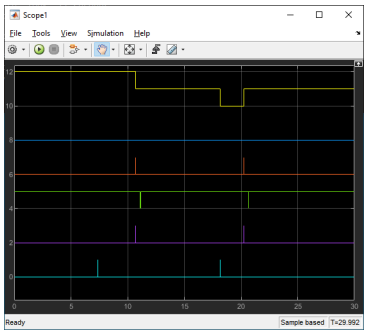
When running this model, to demonstrate the reinitialization stages, you need to disconnect and reconnect the Ethernet cable between the target machine and the EtherCAT slave network. When you reconnect the cable, you see the DC timing perform the same resynchronization that occurs during the initial period.



When using **Run on Target**, **Scope** shows the DC timing error between the master code on the target and the first DC enabled slave. Because the error is returned as nanoseconds, this graph shows that the timing difference settles down to the order of 3-5 microseconds (3000 to 5000 nanoseconds) difference between the DC enabled slaves and the target machine running the code. The residual scatter just reflects task scheduling variability in the target computer RTOS.

In this experimental run, the Ethernet cable was disconnected twice during the 30 second run. Disconnection occurred at about 7 seconds, reconnection at about 12 seconds. This process repeats

at about 18 seconds and 21 seconds. Each time the cable is reconnected, the timing error shows a pulse that shows drift between target and EtherCAT network during the time the cable was disconnected and is the expected resynchronization behavior.

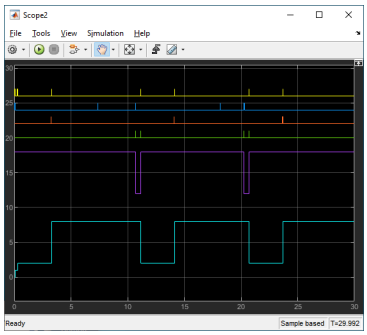


Scope1 shows several logical signals with vertical offsets to show a logic analyzer like display. From the top of the image these are:

1. Link status (yellow)
2. Working count error (blue)
3. Frame response error (red)
4. All slaves Operational (green)
5. Slave Error (purple)
6. Scanbus error (light blue)

Disconnecting the cable caused a scanbus error as seen on the light blue trace. Nothing happens until the cable is reconnected at about 12 seconds. The link status reflects the single time step notifications that indicate the link going away and the link coming back. On the first disconnection, you do not see the link going away notification, but you do see the link coming back. The embedded MATLAB block keeps a persistent variable with the link status with an initial value of 2 and changes it depending on the notifications.

After the link comes back, there is both a slave error and frame response error before All Slaves Operational goes down for a sample time. At that point timing resynchronization starts and you see the damped wave showing the timing errot falling to within a few microseconds of error.



Scope2 shows more status outputs with:

1. statechange (yellow)

2. sbdone (blue)
3. dcinsync (red)
4. statechange request (green)
5. newstate (purple)
6. current state (light blue)

When the link goes down, the stack notices that and performs a scan of devices on the bus. That is the sbdone mark at about 7 seconds that also resulted in the sbscan error shown in Scope1. Next when the link is restored at 12 seconds, another bus scan is performed, shown at 12 seconds in the blue trace. The embedded MATLAB block requests a state change to PreOp (=2) shown in the green and purple traces. Once Preop is reached, you see another state change request to go to Op (=8) state which is the second change in green and purple. That starts resynchronization of the clocks between the development comptuer and the target computer, which takes a few seconds until you see dcinsync at about 14 seconds (red trace) with the transition to Op state right after.

Disconnect the cable again to repeat the whole sequence again starting at about 18 seconds.

While this example needs manual intervention to disconnect and reconnect the Ethernet cable, the same restart can be invoked by just requesting PreOp state follwed by a request for Op state, skipping the interaction with the link status if triggered by some other condition in the model.

If you run the model from the command line, you can use the Simulation Data Inspector to view any signal that is marked for signal logging. Signals marked for logging appear with the dot with two arcs above it in the model editor.

See Also

- “Modeling EtherCAT Networks”
- “EtherCAT® Communication - Sequenced Writing Slave CoE Configuration Variables” on page 13-57
- “EtherCAT® Communication - Sequenced Writing Slave SoE Configuration Variables” on page 13-52

```
close_system( 'slrt_ex_ethercat_notifyreset' );
```

EtherCAT® Communication - Sequenced Writing Slave SoE Configuration Variables

This example shows how to use SoE blocks and a simple state machine to write configuration values to variables that can only be written before going to EtherCAT Op state. For code needed to use the CoE blocks for slaves that understand CoE protocol, see “EtherCAT® Communication - Sequenced Writing Slave CoE Configuration Variables” on page 13-57.

For slaves that understand CoE addressing, restrictions on when a specific object can be written is somewhat rare. For slaves that understand SoE addressing, this restriction is much more common.

Changing configuration objects in slave devices before starting IO to the external connections is useful, even if modifying the values is not restricted.

This example also shows distributed clocks synchronization using the bus shift DC mode where the slaves are shifted in time to match the execution time of the master.

Requirements

To run this example, you need an EtherCAT network that consists of the target computer as EtherCAT Master device and at least one slave that has SoE addressed objects. The supplied ENI file is for a Beckhoff AX5103 drive.

EtherCAT in Simulink Real-Time requires a dedicated network port on the target computer that is reserved for EtherCAT use by using the Ethernet configuration tool. Configure the dedicated port for EtherCAT communication, not with an IP address. The dedicated port must be distinct from the port used for the Ethernet link between the development and target computers.

To test this model:

1. Connect the port that is reserved for EtherCAT in the target computer to the EtherCAT IN port of the AX5103 drive.
2. Make sure the AX5103 is supplied with a 24-volt power source.
3. 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 “Modeling EtherCAT Networks”.

Open the Model

This model illustrates how you can read or write to SoE/SSC objects if they are only writable in EtherCAT PreOp state. You can move the SoE/SSC transfers to EtherCAT SafeOp state by changing the **Initialization end state** in the EtherCAT Init block and by also changing the constant in the **Wait for this state** constant block. These settings direct the state machine to start sending SoE messages when it reaches the initialization end state.

1. Init = 1
2. PreOp = 2
3. SafeOp = 4
4. Op = 8

The EtherCAT initialization block requires that the configuration ENI file is present in the current folder or on the MATLAB path because the file name is present without directory information.

If you want to modify this model to experiment with it, copy the example configuration file and the model file from the example folder to the current folder. To open the model, in the MATLAB Command Window, type:

```
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_ethercat_asyncSoE_SSC_
```

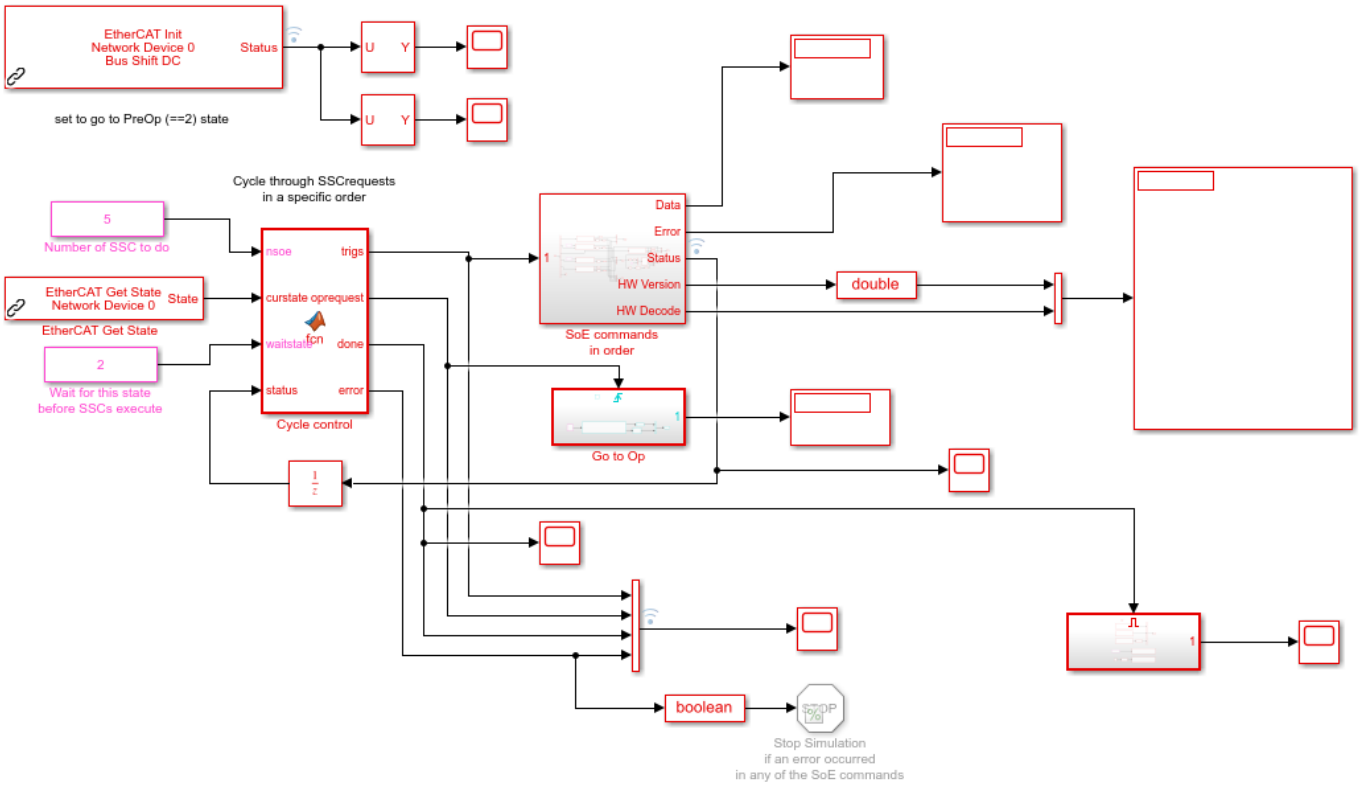


Figure 1: EtherCAT model for sequencing through SoE/SSC commands after pausing initialization at PreOp state

Configure the Model

Open the parameter dialog for the **EtherCAT Init** block and observe the pre-configured values. The EtherCAT slave devices that are daisy chained together with Ethernet cable is a Device, also referred to as an EtherCAT network. The Device Index selects one such chained EtherCAT network. The Ethernet Port Number identifies which Ethernet port to use to access that Device. The EtherCAT Init block connects these two so that other EtherCAT blocks use the Device Index to communicate with the slave devices on that EtherCAT network.

If you only have one connected network of EtherCAT slaves, and you have only reserved one Ethernet port with the Ethernet configuration tool, use Device Index = 0 and Ethernet Port Number = 1.

Create an ENI file for a Different SoE drive

If you need to create a new ENI file you need to use a third-party EtherCAT configurator such as TwinCAT 3 from Beckhoff that you install on a development computer. The EtherCAT configuration (ENI) file preconfigured for this model is `BeckDrive_1ms.xml`.

Each ENI 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 one AX5103 drive. If you have a different EtherCAT drive this example still works, but you need to create a new ENI file that uses your drive.

For an overview of the process for creating an ENI file, see “Configure EtherCAT Network by Using TwinCAT 3”.

If you use a different drive, you need to consult the manual for your devices and find the SoE mapping. Using that mapping, you need to change the SSC commands in the **SoE commands in order** subsystem to use objects on your drive.

Build, Download, and Run the Model

To build, download, and run the model:

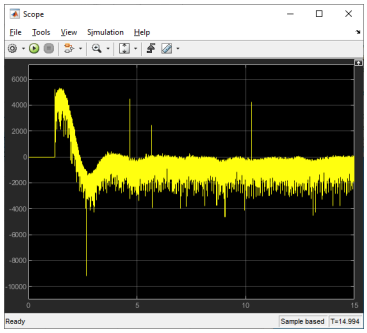
1. In the **Simulink Editor**, from the targets list on the **Real-Time** tab, select the target computer on which to run the real-time application.
2. Click **Run on Target**.

If you open the two scopes by double clicking each, the data is relayed from the target back to the development computer and displayed.

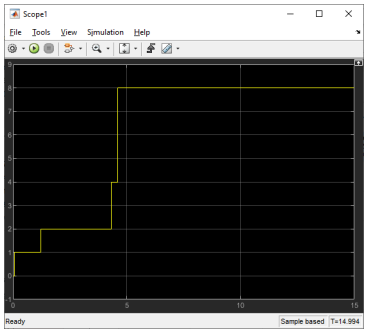
The model is preconfigured to run for 15 seconds. If you want to run the model longer, pull down the **Run on Target** menu and change the number on the bottom line. Press the green arrow to configure, build, and run.

Display the Target Computer Data

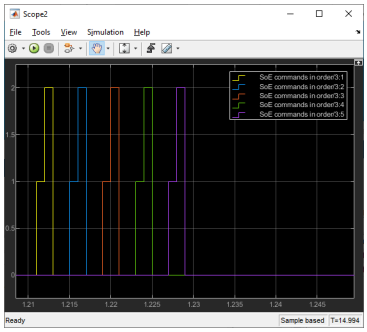
If you run the model using the **Run on Target** button, external mode is connected and you can double click the scope blocks and see the data on the development computer. The **Display** blocks also work.



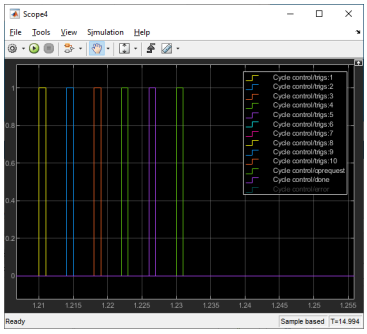
When using **Run on Target**, **Scope** shows the DC timing error between the master code on the target and the first DC enabled slave. Since the error is returned as nanoseconds, this graph shows that the timing difference settles down to the order of 3-5 microseconds (3000 to 5000 nanoseconds) difference between the DC enabled slaves and the target machine running the code. The residual scatter just reflects task scheduling variability in the target computer RTOS.



Scope1 shows the progression of EtherCAT states, from idle to Init to PreOp, SafeOp and finally Op state. SafeOp is only entered briefly and only shows up as the value 4 for a few time steps before the switch to Op state. Since this model uses the distributed clocks mechanism, the switch to Op state only occurs once the timing error settles down.



Scope2 shows the status outputs of the 5 async SDO blocks inside the subsystem. Each SDO block is enabled to write for one time step. The block switches to status = 1 (busy) for a few time steps. On successful completion status = 2 (done), the block switches for one time step. If a block encounters an error, the block switches to status = 3 (error) for one time step. On an error, the **Cycle control** embedded MATLAB code block stops the sequence and sets the error output, which stops the model. In that case, the failing block have output an error code that is displayed on Display1. This display is zoomed into the interval just after state went to PreOp (=2) state.



Scope4 shows several of the outputs from the **Cycle control** block. The first 5 are the enable signals, made true one at a time by **Cycle control**. The **oprequest** output is true for one time step to trigger the request to proceed to Op state. This display is zoomed in to the same interval as in **Scope2**.

When all of the requested SSC commands are complete and the state has progressed to Op state, the **done** signal is set to **true** for the remainder of execution. The rest of your model goes into the **Op State Model** subsystem.

If you need a different number of SSC commands to execute before Op state, you need to edit the **Cycle control** embedded MATLAB code block and modify the persistent array that is currently sized to have length 10, which is larger than the number of SSC commands being requested.

If you run the model from the command line, you can use the Data Inspector, accessible from the toolstrip, to view any signal that has been tagged to log with the **Log Selected Signals** selection found by right clicking on the signal. Those are marked with the dot with two arcs above it in the model editor.

See Also

- “EtherCAT® Communication - Sequenced Writing Slave CoE Configuration Variables” on page 13-57
- “EtherCAT® Communication - Detect EtherCAT network failure and reset” on page 13-47
- “Modeling EtherCAT Networks”
- “Configure EtherCAT Network by Using TwinCAT 3”

EtherCAT® Communication - Sequenced Writing Slave CoE Configuration Variables

This example shows how to use CoE blocks and a simple state machine to write configuration values to variables that can only be written before going to EtherCAT Op state. For code needed to use the SoE blocks for slaves that understand SoE protocol, see “EtherCAT® Communication - Sequenced Writing Slave SoE Configuration Variables” on page 13-52.

For slaves that understand CoE addressing, limited ability to read or write specific objects is somewhat rare. For slaves that understand SoE addressing, this restriction is much more common.

This example also shows distributed clocks synchronization using the bus shift DC mode where the slaves are shifted in time to match the execution time of the master.

Requirements

To run this example, you need an EtherCAT network that consists of the target computer as EtherCAT master device and at least one slave that has CoE addressed objects. The supplied ENI file is for a 5 element slave stack: EK1100+EL1202+EL2202+EL3102+EL4032.

EtherCAT in Simulink Real-Time requires a dedicated network port on the target computer that is reserved for EtherCAT by using the Ethernet configuration tool. Configure the dedicated port for EtherCAT communication, not with an IP address. The dedicated port must be distinct from the port used for the Ethernet link between the development and target computers.

To test this model:

1. Connect the port that is reserved for EtherCAT in the target computer to the EtherCAT IN port of the EL1100 interface.
2. Make sure the EK1100 is supplied with a 24-volt power source.
3. 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 “Modeling EtherCAT Networks”.

Open the Model

This model illustrates how you can read or write to CoE/SDO objects if they are only writable in EtherCAT PreOp state. You can move the CoE/SDO transfers to EtherCAT SafeOp state by changing the **Initialization end state** in the EtherCAT Init block and by also changing the constant in the **Wait for this state** constant block. These settings direct the state machine to start sending CoE messages when it reaches the initialization end state.

1. Init = 1
2. PreOp = 2
3. SafeOp = 4
4. Op = 8

The EtherCAT initialization block requires that the configuration ENI file is present in the current folder.

If you want to modify this model to experiment with it, then copy the example configuration file from the example folder to the current folder. To open the model, in the MATLAB Command Window, type:

```
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_ethercat_asyncCoE_cyc
```

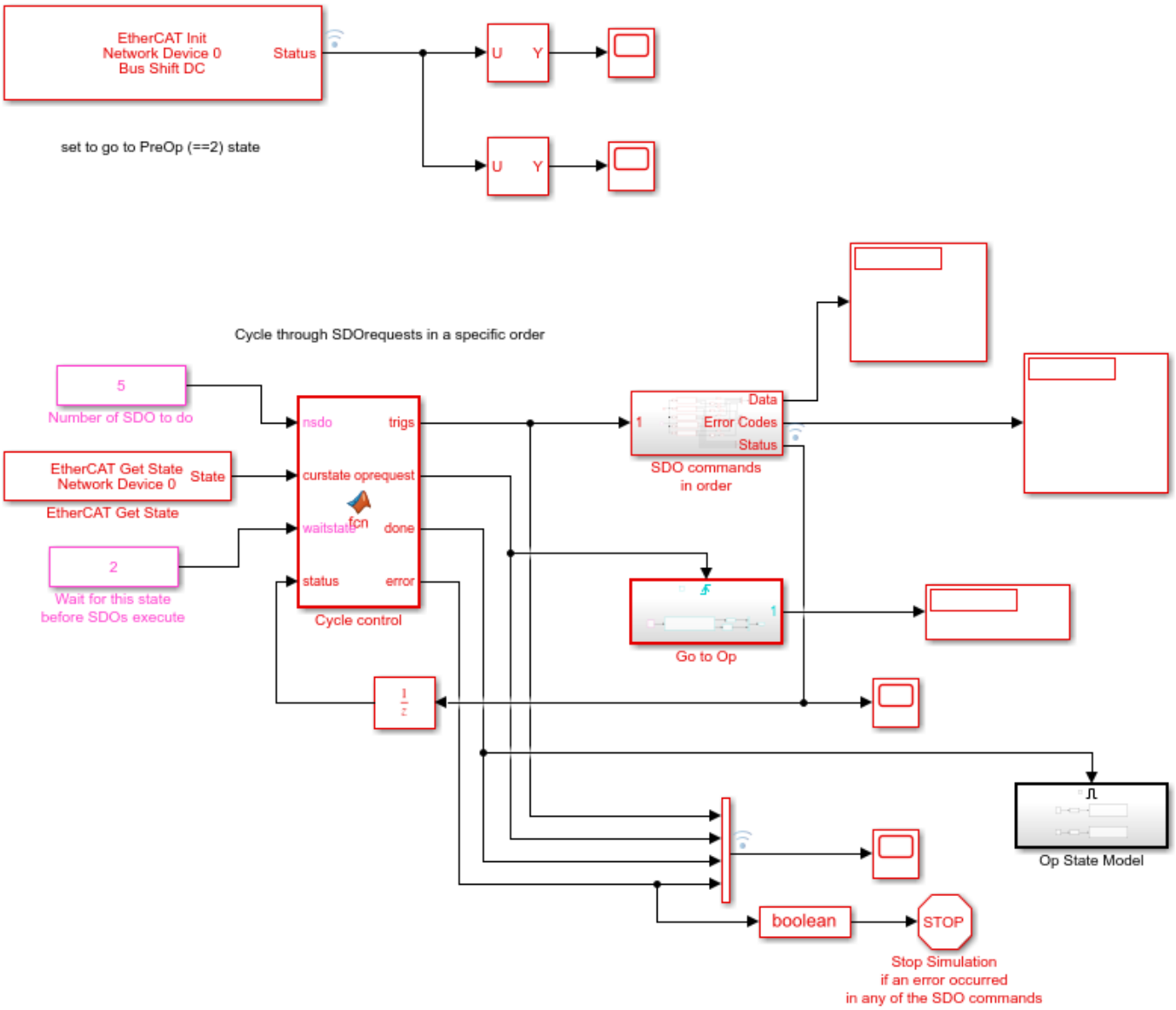


Figure 1: EtherCAT model for sequencing through CoE commands after pausing initialization at PreOp state

Configure the Model

Open the parameter dialog for the **EtherCAT Init** block and observe the pre-configured values. The EtherCAT slave devices that are daisy chained together with Ethernet cable is a Device, also referred to as an EtherCAT network. The Device Index selects one such chained EtherCAT network. The Ethernet Port Number identifies which Ethernet port to use to access that Device. The EtherCAT Init block connects these two so that other EtherCAT blocks use the Device Index to communicate with the slave devices on that EtherCAT network.

If you only have one connected network of EtherCAT slaves, and you have only reserved one Ethernet port with the Ethernet configuration tool, use Device Index = 0 and Ethernet Port Number = 1.

Create an ENI File for a Different Set of Slaves

If you need to create a new ENI file you need to use a third-party EtherCAT configurator such as TwinCAT 3 from Beckhoff that you install on a development computer. The EtherCAT configuration (ENI) file preconfigured for this model is `Stack4_BS_1ms.xml`.

Each ENI 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 an EK1100+EL1202+EL2202+EL3102+EL4032. If you have a different set of EtherCAT slave devices this example works, but you need to create a new ENI file that uses your devices.

For overview of the process for creating an ENI file, see “Configure EtherCAT Network by Using TwinCAT 3”.

If you use different slave devices, you need to consult the manual for your devices and find the CoE mapping. Using that mapping, you need to change the SDO commands in the **SDO commands in order** subsystem to use objects on your devices.

Build, Download, and Run the Model

To build, download, and run the model:

1. In the **Simulink Editor**, from the targets list on the **Real-Time** tab, select the target computer on which to run the real-time application.
2. Click **Run on Target**.

If you open the two host side scopes by double clicking each, data is relayed from the target computer to the development computer and is displayed.

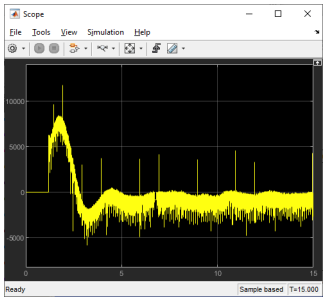
Included in the model is the ability to control the amplitude of the cycling motion. With the **Run on Target** button, the slider is active and connected to the `Amplitude` constant block.

The model is preconfigured to run for 15 seconds. If you want to run the model longer, pull down the **Run on Target** menu and change the number on the bottom line. Press the green arrow to configure, build, and run.

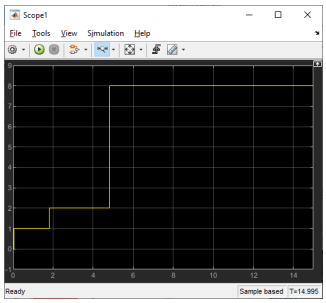
Display the Target Computer Data

If you run the model using the **Run on Target** button, external mode is connected and you can double click the scope blocks and see the data on the development computer. The `Display` blocks also work.

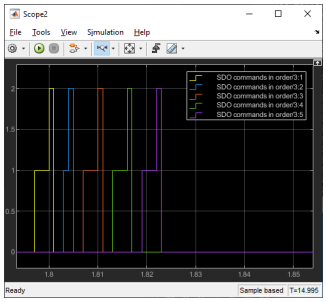
When using **Run on Target**, the **Scope** block shows the DC timing error between the master code on the target computer and the first DC enabled slave. Because the error is returned as nanoseconds, this graph shows that the timing difference settles down to the order of 3-5 microseconds (3000 to 5000 nanoseconds) difference between the DC enabled slaves and the target machine running the code. The residual scatter reflects task scheduling variability in the target computer RTOS.



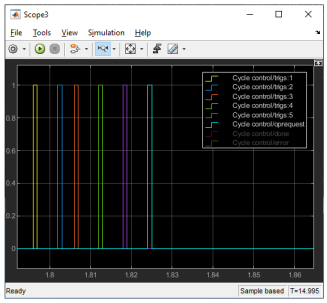
Scope1 shows the progression of EtherCAT states, from idle to Init to PreOp, SafeOp and finally Op state. SafeOp is only entered briefly and only shows up as the value 4 for a few time steps before the switch to Op state. Since this model uses the distributed clocks mechanism, the switch to Op state only occurs once the timing error settles down.



Scope2 shows the status outputs of the 5 async SDO blocks inside the subsystem. Each SDO block is enabled to write for one time step, then switches to status = 1 (busy) for a few time steps, then on successful completion status = 2 (done) for one time step. If a block encounters an error, status = 3 (error) for one time step. On an error, the **Cycle control** embedded Matlab code block stops the sequence and sets the error output which stops the model. In that case, the failing block has output an error code that is displayed on Display1. This display is zoomed into the interval just after state went to PreOp (=2) state.



Scope3 shows several of the outputs from the **Cycle control** block. The first 5 are the enable signals, made true one at a time by **Cycle control**. Then the **oprequest** output is true for one time step to trigger the request to proceed to Op state. This display is zoomed in to the same interval as in **Scope2**.



When the requested SDO commands are complete and the state has progressed to Op state, the **done** signal is set to **true** for the remainder of execution. The rest of your model goes into the **Op State Model** subsystem.

If you need a different number of SDO commands to execute before Op state, you need to edit the **Cycle control** embedded MATLAB code block and modify the persistent array that is currently sized to have length 5, the same as the number of SDO commands being requesting.

If you run the model from the command line, you can use the Simulation Data Inspector to view any signal that has been marked for signal logging. Signals marked for signal logging have a dot with two arcs above it in the model editor.

See Also

- “EtherCAT® Communication - Sequenced Writing Slave SoE Configuration Variables” on page 13-52
- “EtherCAT® Communication - Detect EtherCAT network failure and reset” on page 13-47
- “Modeling EtherCAT Networks”
- “Configure EtherCAT Network by Using TwinCAT 3”

Simple ASCII Encoding/Decoding Loopback Test (With Baseboard Blocks)

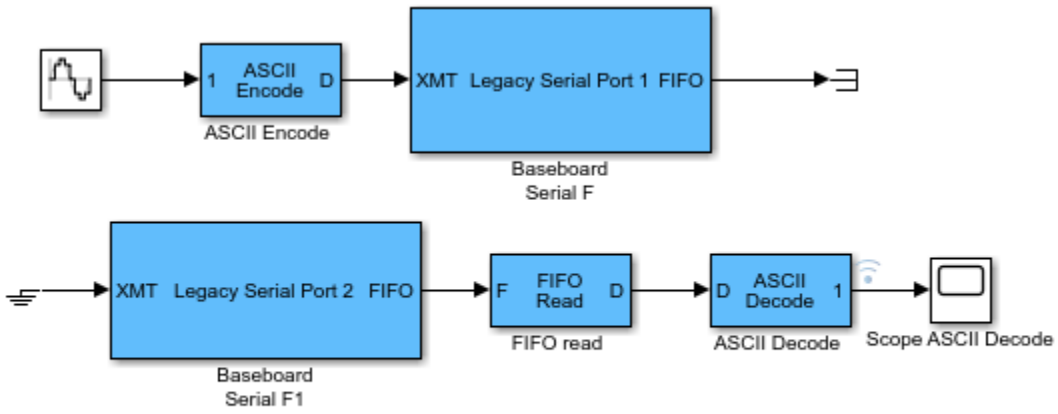
This example model shows how a single floating point number can be converted to ASCII and transmitted over a serial link. The sending serial port and receiving serial port can be in the same system or in different systems.

To test this model:

1. The target computer must have two legacy serial ports.
2. Connect legacy serial port 1 to legacy serial port 2 with a null modem cable.

This example is configured to use baseboard serial ports (legacy serial port 1 and legacy serial port 2). You can also use legacy serial port 3 and legacy serial port 4 by changing the board setup in the Baseboard blocks. Other serial blocks could be used in place of the Baseboard blocks.

```
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_serialbaseboardsimplea
```



See Also

- “RS-232 Serial Communication”
- “RS-232 Legacy Drivers”

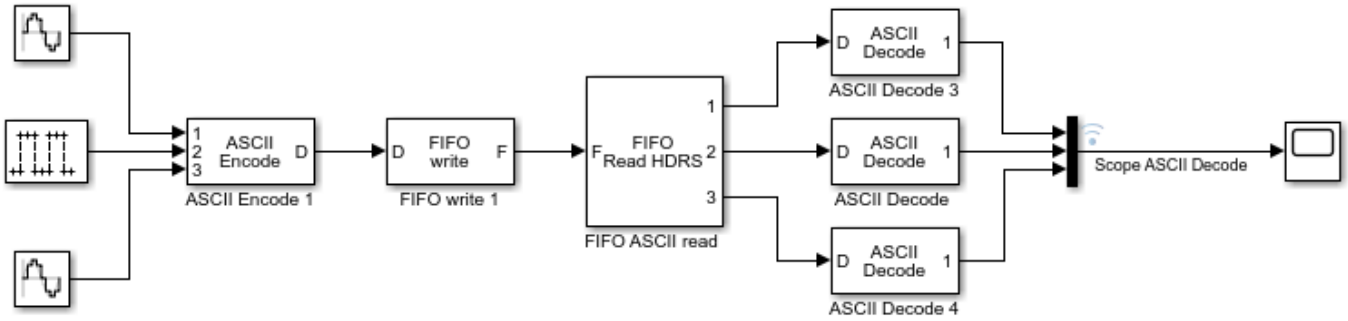
ASCII Encoding/Decoding Loopback Test

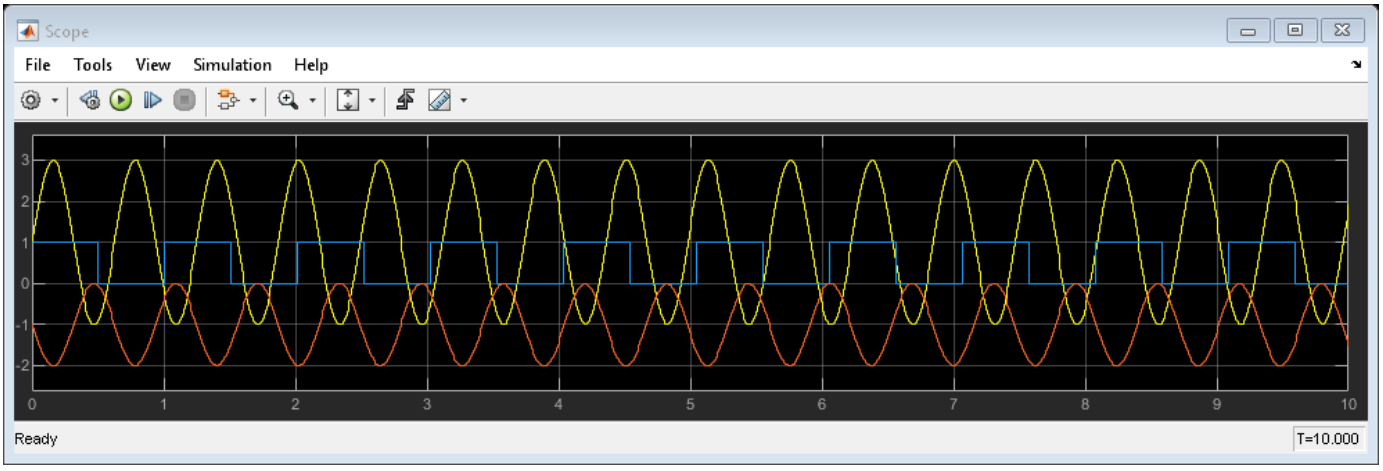
This model shows how to send ASCII data over a serial link.

The ASCII Encode block generates a message with three different sub messages along with some extraneous data 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','slrealtime','examples','slrt_ex_serialasciitest'))
set_param('slrt_ex_serialasciitest','StopTime','30');
sim('slrt_ex_serialasciitest')
```





See Also

- “RS-232 Serial Communication”
- “RS-232 Legacy Drivers”

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 data to show how the FIFO Read HDRS block can remain synchronized to the valid byte stream even in the presence of transmission errors.

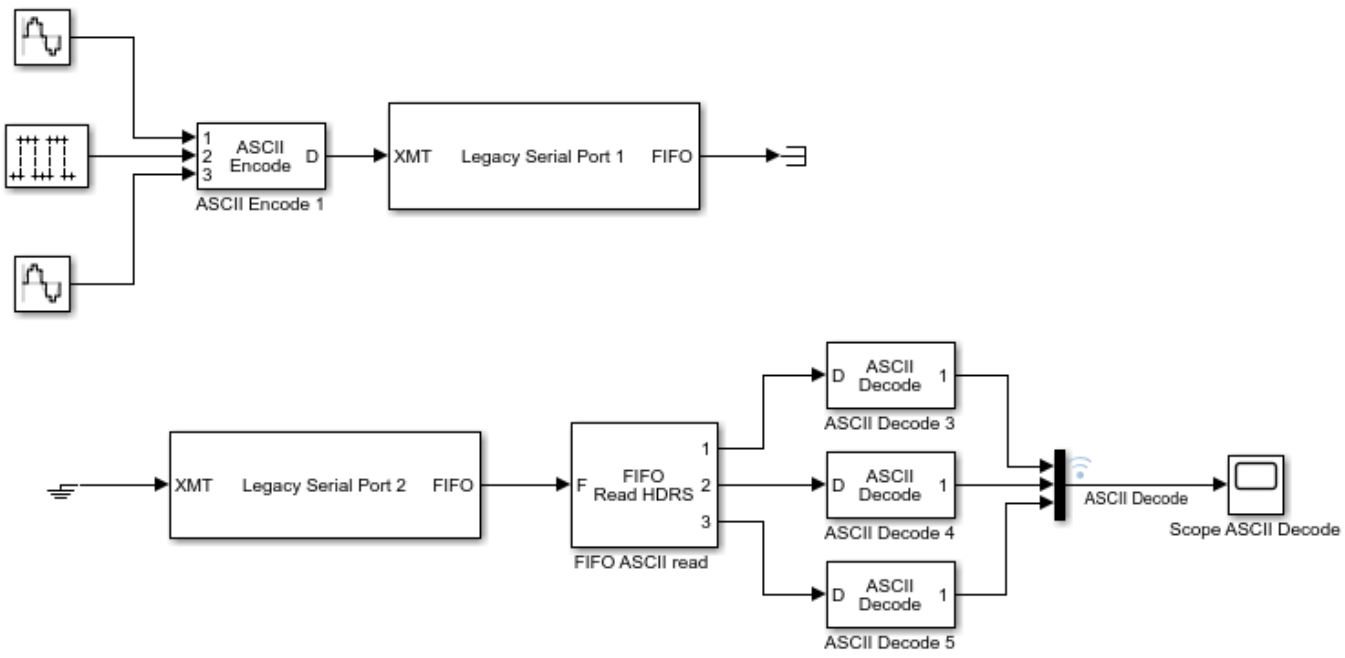
The FIFO Read HDRS block can handle an arbitrary number of headers; just add them as strings to the cell array in the block parameters dialog box. The messages must share the same termination string. In this example, it is a carriage return followed by a line feed: "\r\n".

To test this model:

1. The target computer must have two legacy serial ports.
2. Connect legacy serial port 1 to legacy serial port 2 with a null modem cable.

This example is configured to use baseboard serial ports (legacy serial port and legacy serial port 2). You can also use legacy serial port 3 and legacy serial port 4 by changing the board setup in the Baseboard blocks. Other serial blocks could be used in place of the Baseboard blocks.

```
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_serialbaseboardasciite
```



See Also

- “RS-232 Serial Communication”

- “RS-232 Legacy Drivers”

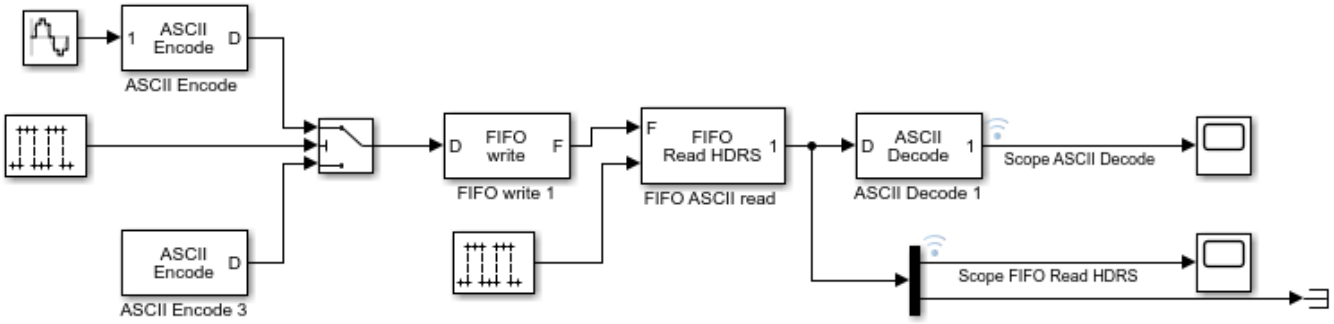
ASCII Encoding/Decoding Resync Loopback Test

This example model shows the ability of the FIFO Read HDRS block to resynchronize after being repeatedly disabled and its 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','slrealtime','examples','slrt_ex_serialasciisplit'))
set_param('slrt_ex_serialasciisplit','StopTime','30');
sim('slrt_ex_serialasciisplit')
```



See Also

- “RS-232 Serial Communication”
- “RS-232 Legacy Drivers”

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.

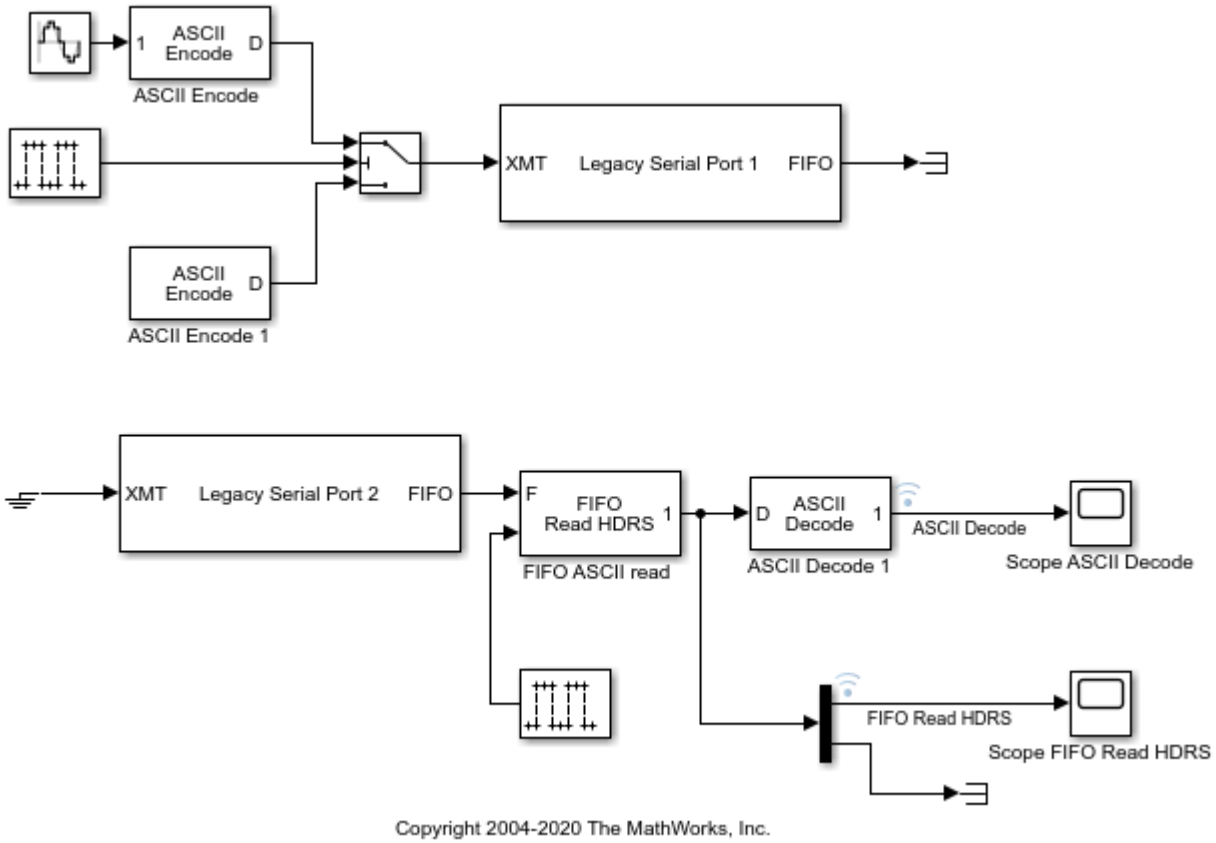
Scope 1 graphs the decoded sine wave data received at each time step. When the Pulse Generator1 block outputs a 0, the count from the FIFO Read HDRS block is 0. When it outputs a 1, the read catches up by throwing away extra data and returns the last complete value found in the FIFO. Scope 2 indicates when new data is present.

To test this model:

1. The target computer must have two legacy serial ports.
2. Connect legacy serial port 1 to legacy serial port 2 with a null modem cable.

This example is configured to use baseboard serial ports (legacy serial port 1 and legacy serial port 2). You can also use legacy serial port 3 and legacy serial port 4 by changing the board setup in the Baseboard blocks. Other serial blocks could be used in place of the Baseboard blocks.

```
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_serialbaseboardasciis
```



See Also

- “RS-232 Serial Communication”
- “RS-232 Legacy Drivers”

Binary Encoding/Decoding Loopback Test

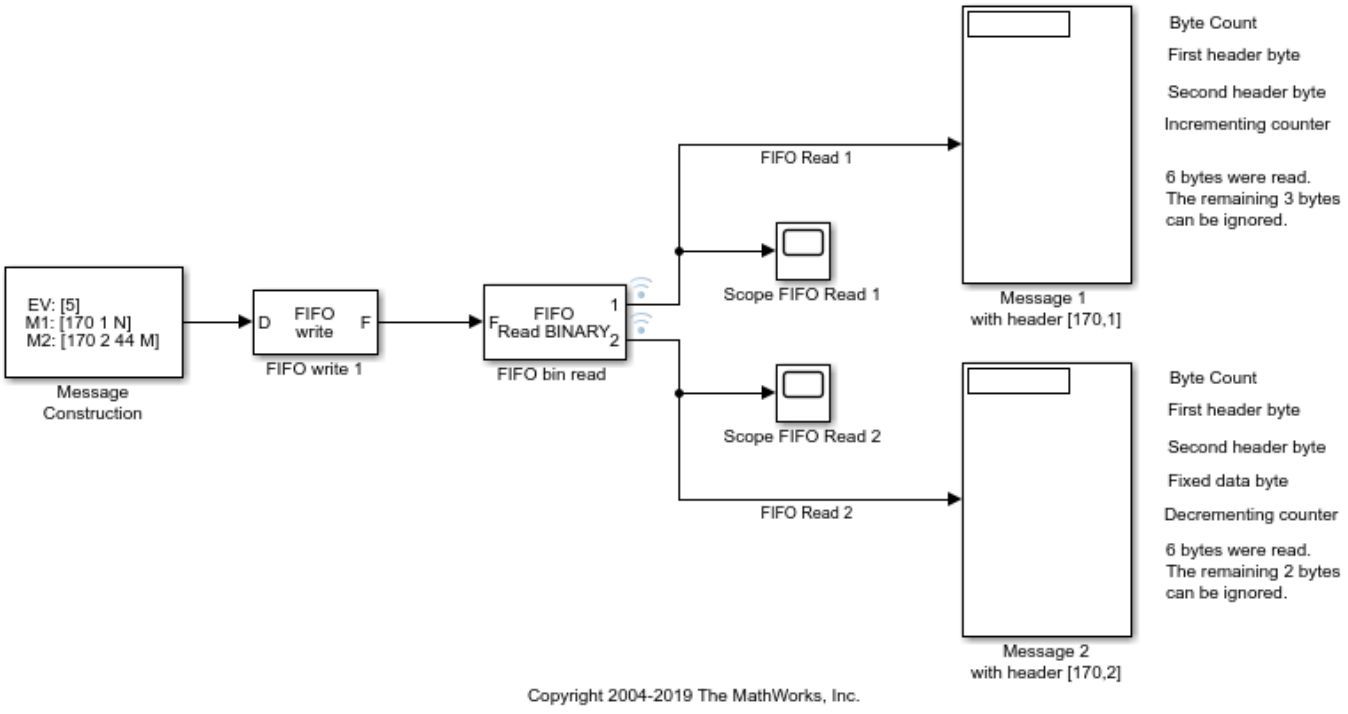
This model shows how to send Binary data over a serial link.

The transmitted data are: `[8,5,170,1,N,170,2,44,M]`. This byte stream contains two messages along with other elements:

- 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','slrealtime','examples','slrt_ex_serialbinarytest'));
```



See Also

- “RS-232 Serial Communication”
- “RS-232 Legacy Drivers”

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.

- 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.

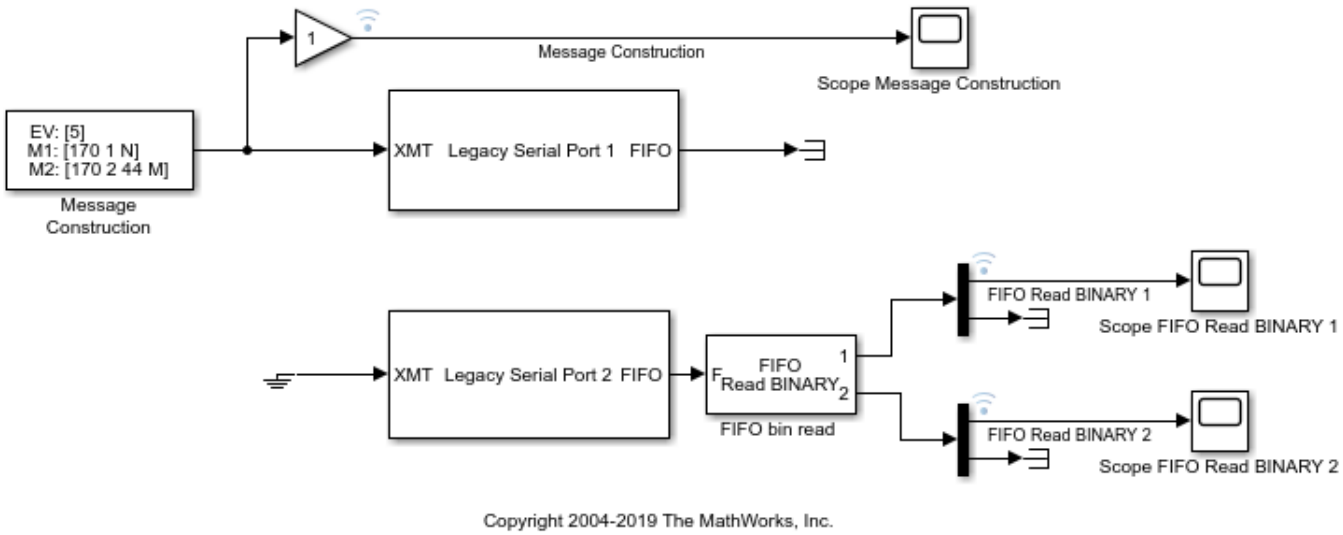
Scope 1 displays the received message 1 data. Scope 2 displays the received message 2 data. Scope 3 shows the transmitted byte stream. The gain block on the signal to Scope 3 makes the elements of the vector non-virtual so the scope can see them.

To test this model:

1. The target computer must have two legacy serial ports.
2. Connect legacy serial port 1 to legacy serial port 2 with a null modem cable.

This example is configured to use baseboard serial ports (legacy serial port 1 and legacy serial port 2). You can also use legacy serial port 3 and legacy serial port 4 by changing the board setup in the Baseboard blocks. Other serial blocks could be used in place of the Baseboard blocks.

```
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_serialbaseboardbinary
```



See Also

- “RS-232 Serial Communication”
- “RS-232 Legacy Drivers”

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.

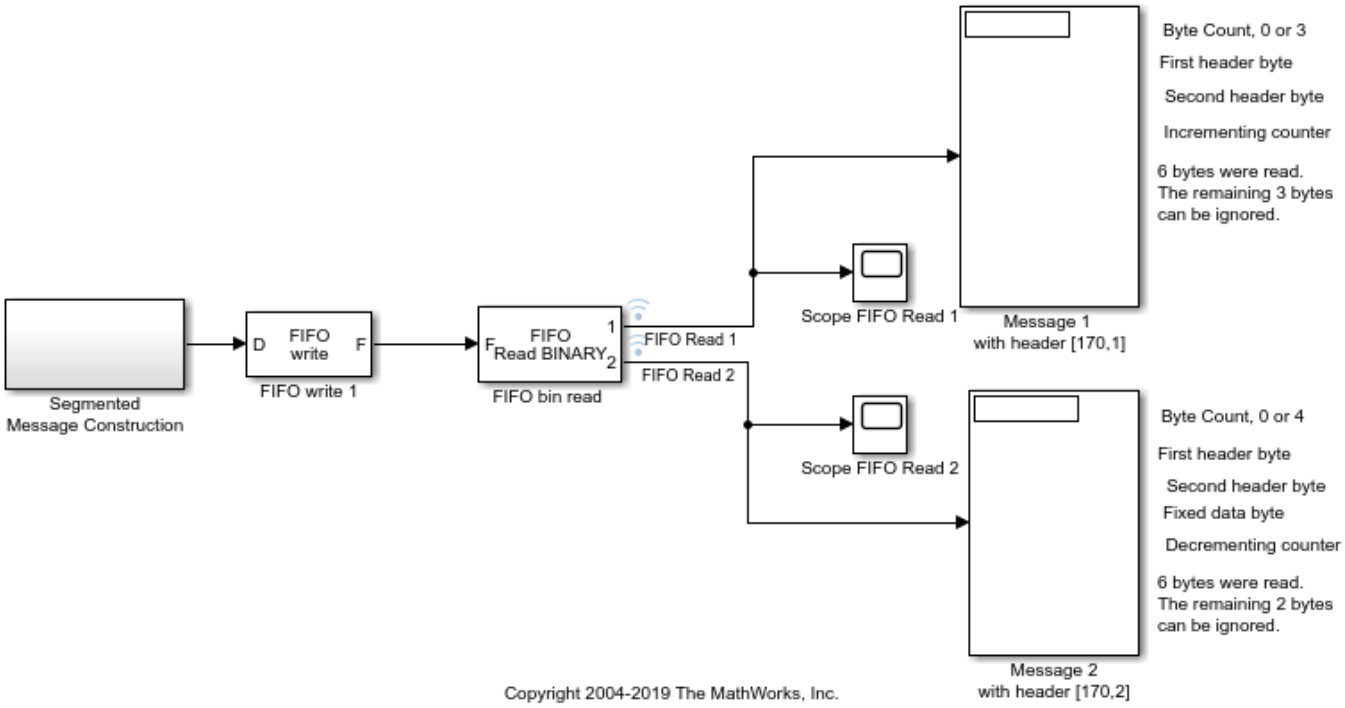
The Segmented Message Constructor subsystem contains blocks that prepare and send only parts of messages on each time step.

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 incremenenting 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','slrealtime','examples','slrt_ex_serialbinarysplit'));
```



See Also

- “RS-232 Serial Communication”
- “RS-232 Legacy Drivers”

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.

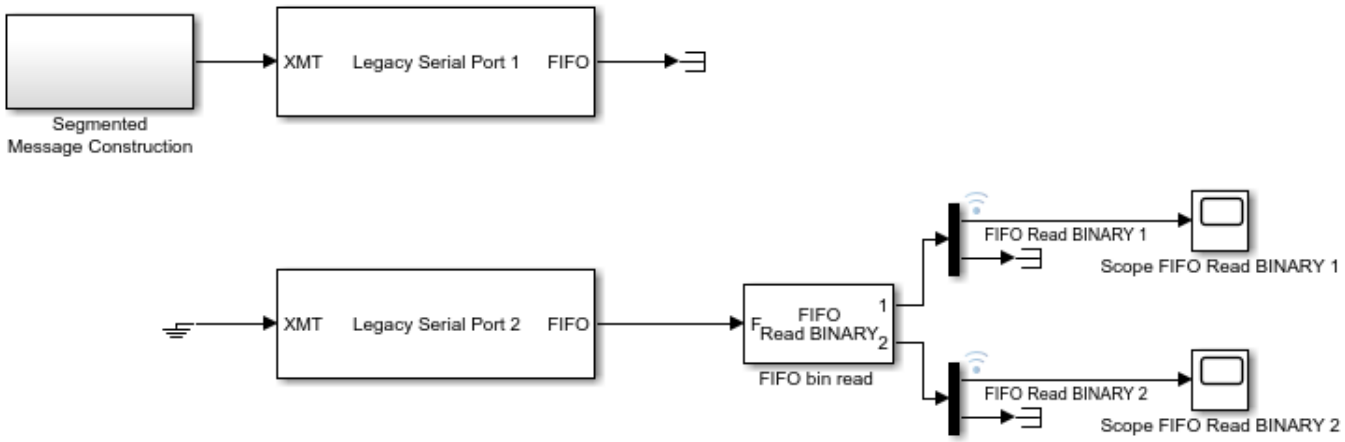
Scope 1 displays the received message 1 data. Scope 2 displays the received message 2 data.

To test this model:

1. The target computer must have two legacy serial ports.
2. Connect legacy serial port 1 to legacy serial port 2 with a null modem cable.

This example is configured to use baseboard serial ports (legacy serial port 1 and legacy serial port 2). You can also use legacy serial port 3 and legacy serial port 4 by changing the board setup in the Baseboard blocks. Other serial blocks could be used in place of the Baseboard blocks.

```
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_serialbaseboardbinarys
```



See Also

- “RS-232 Serial Communication”
- “RS-232 Legacy Drivers”

Target to Development Computer Communication by Using TCP

This example shows how to use TCP blocks to send data from the target computer to MATLAB running on the development computer.

The TCP Send block in the server real-time application `slrt_ex_target_to_host_TCP` sends data from the target computer to the TCP/IP object that is created in MATLAB on the development computer. The MATLAB m-script sends the received data back to the real-time application.

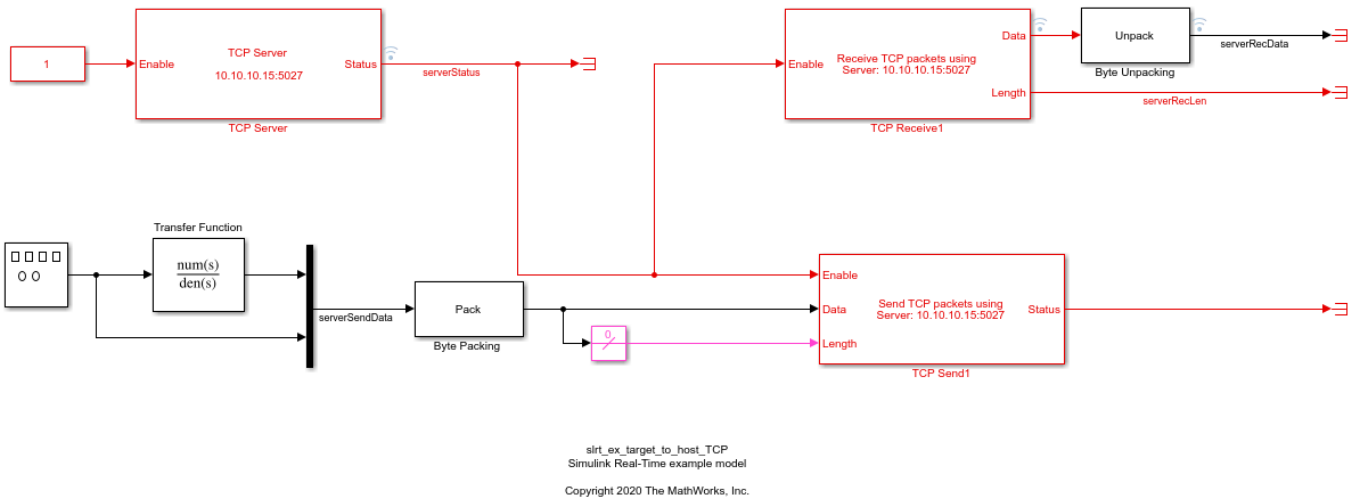
To open this example, in the MATLAB Command Window, type:

```
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_target_to_host_TCP'))
```

Open, Build, and Download Server Application

Open the model.

```
mdl = 'slrt_ex_target_to_host_TCP';
mdlOpen = 0;
systems = find_system('type', 'block_diagram');
if ~any(strcmp(mdl, systems))
    mdlOpen = 1;
    open_system(fullfile(matlabroot,'toolbox','slrealtime','examples',mdl));
end
```



Build Model and Download to Target Computer

```
set_param(mdl, 'RTWVerbose', 'off');
set_param(mdl, 'StopTime','10');
targetIP = '10.10.10.35';
set_param([mdl,'/TCP Server'],'serverAddress',targetIP);
rtwbuild(mdl);
tg = slrealtime;
load(tg,mdl);
```

```
### Successful completion of build procedure for: slrt_ex_target_to_host_TCP
### Created MLDATX ..\slrt_ex_target_to_host_TCP.mldatx
```

```
Build Summary

Top model targets built:

Model                        Action                        Rebuild Reason
========================================================================================
slrt_ex_target_to_host_TCP  Code generated and compiled  Global variables have changed.

1 of 1 models built (0 models already up to date)
Build duration: 0h 0m 37.964s
```

Close the model.

```
if (mdlOpen)
  bdclose(mdl);
end
```

Create TCP/IP Object in MATLAB on Development Computer

Create a TCP/IP object and connect the TCP/IP object to the development computer.

```
t = tcpip(targetIP,5027);
t.BytesAvailableFcnMode = 'byte';
t.BytesAvailableFcnCount = 16;
fopen(t);
```

Run Real-Time Application on Target Computer

```
start(tg);
pause(3);
```

Read Data Packets and Send Back to Target Computer

Read from the target computer and write back.

```
while (strcmp(t.Status,'open'))
   data = fread(t,16);
   fwrite(t,data);
end
```

Stop Real-Time Application on Target Computer

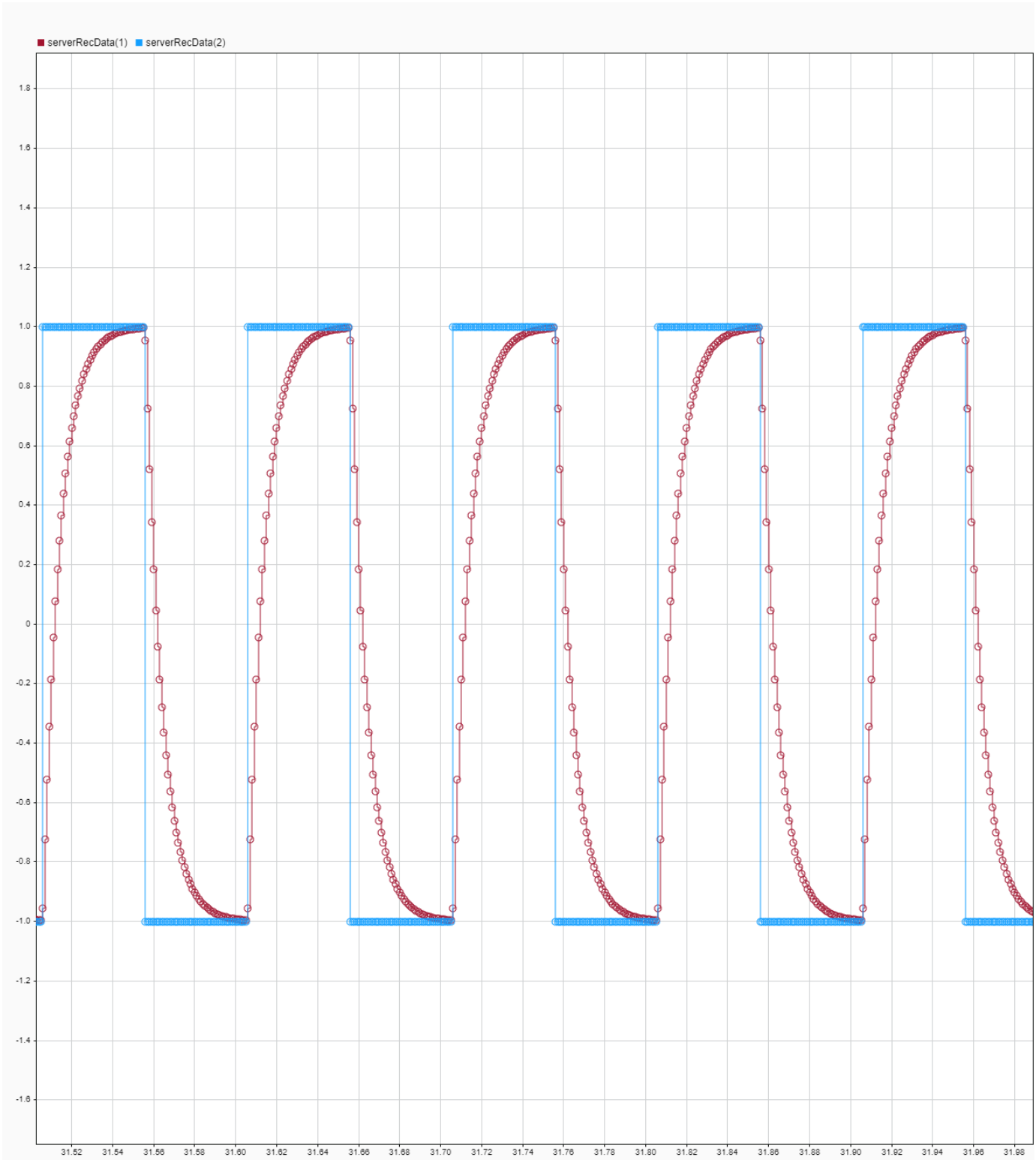
```
stop(tg);
```

Close TCP/IP Object on Development Computer

```
fclose(t);
delete(t);
clear t;
```

View Signal Received on Target Computer

```
Simulink.sdi.view();
```


serverRecData(1)
serverRecData(2)
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1.0
-1.2
-1.4
-1.6
31.52
31.54
31.56
31.58
31.60
31.62
31.64
31.66
31.68
31.70
31.72
31.74
31.76
31.78
31.80
31.82
31.84
31.86
31.88
31.90
31.92
31.94
31.96
31.98

Target to Host Transmission by Using UDP

This example shows how to use UDP blocks to send data from a target computer to a development computer. The transmit real-time application `slrt_ex_target_to_host_UDP` runs on the target computer and send signal data to the UDP object that the script creates in MATLAB on the development computer.

When using the UDP protocol for communicating data to or from the target computer, consider these issues:

- The Simulink model on the development computer runs as fast as it can. The model run speed is 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. These transmissions share bandwidth with the Ethernet link.

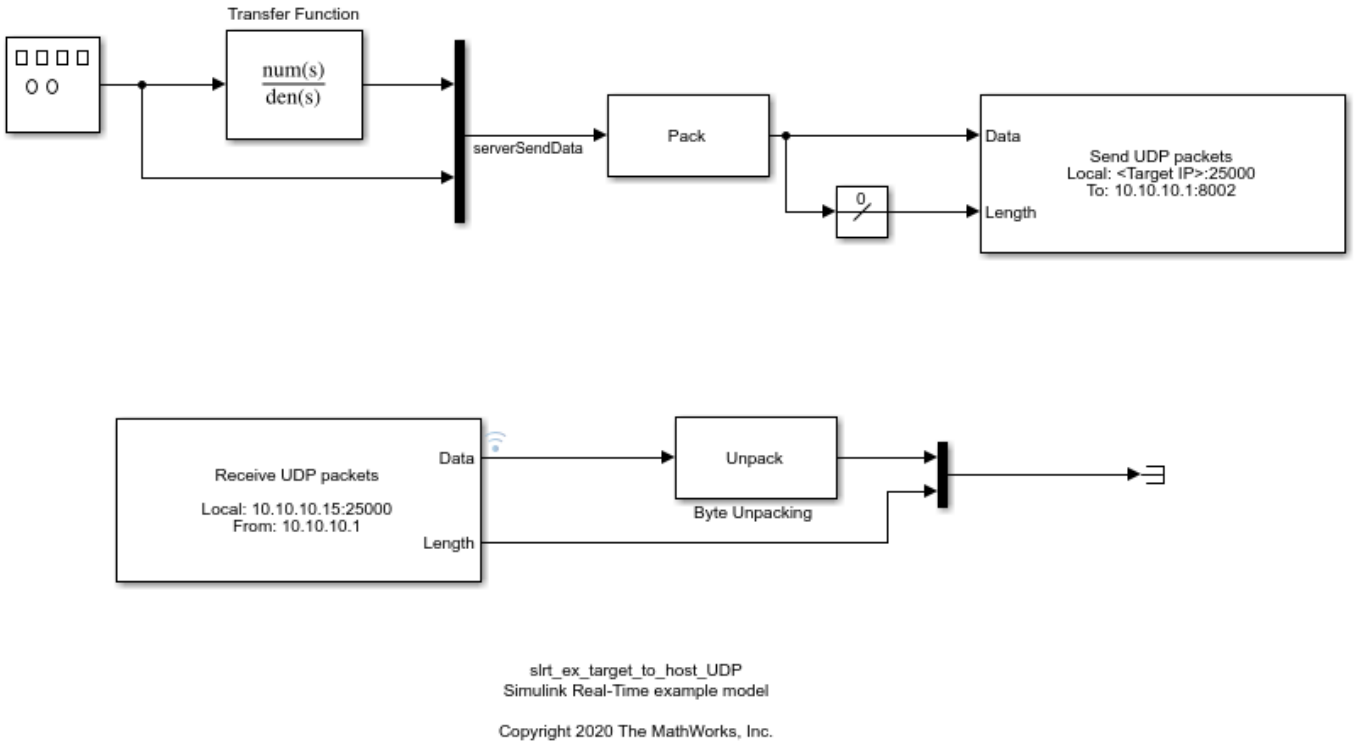
For more information about UDP and Simulink Real-Time, see “UDP Communication Setup”.

Open Model, Build, and Load Real-Time Application

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. To open the model, in the MATLAB Command Window, type:

```
open_system(fullfile(matlabroot,'toolbox','slrealtime','examples','slrt_ex_target_to_host_UDP'));
```

```
mdl = 'slrt_ex_target_to_host_UDP';
mdlOpened = 0;
systems = find_system('type', 'block_diagram');
if ~any(strcmp(mdl, systems))
  mdlOpened = 1;
  open_system(fullfile(matlabroot,'toolbox','slrealtime','examples',mdl));
end
```



Build the model and download to the target computer.

- Configure for a non-Verbose build.
- Mark the Byte Unpacking block 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(mdl,'RTWVerbose','off');
set_param(mdl,'StopTime','10');
targetIP = '10.10.10.35';
set_param([mdl,'/UDP Receive'],'ipAddress',targetIP);
hostIP = '10.10.10.128';
set_param([mdl,'/UDP Send'],'toAddress',hostIP)
set_param([mdl,'/UDP Receive'],'fmAddress',hostIP)
handle = get_param([mdl,'/Byte Unpacking '],'PortHandles');
Outport = handle.Outport(1);
Simulink.sdi.markSignalForStreaming(Outport,'on');
rtwbuild(mdl);
tg = slrealtime;
load(tg,mdl);

### Successful completion of build procedure for: slrt_ex_target_to_host_UDP
### Created MLDATX ..\slrt_ex_target_to_host_UDP.mldatx
```

```
Build Summary

Top model targets built:

Model                        Action                        Rebuild Reason
========================================================================================
slrt_ex_target_to_host_UDP  Code generated and compiled  Global variables have changed.

1 of 1 models built (0 models already up to date)
Build duration: 0h 0m 36.128s
```

Close the model if it is opened.

```
if (mdlOpened)
  bdclose(mdl);
end
```

Create UDP object in MATLAB on Development Computer

```
udpObj = udp('','LocalHost',hostIP,'LocalPort',8002,'RemoteHost',targetIP,'RemotePort',25000);
udpObj.InputBufferSize = 16;
fopen(udpObj);
```

Run Model on Target Tomputer

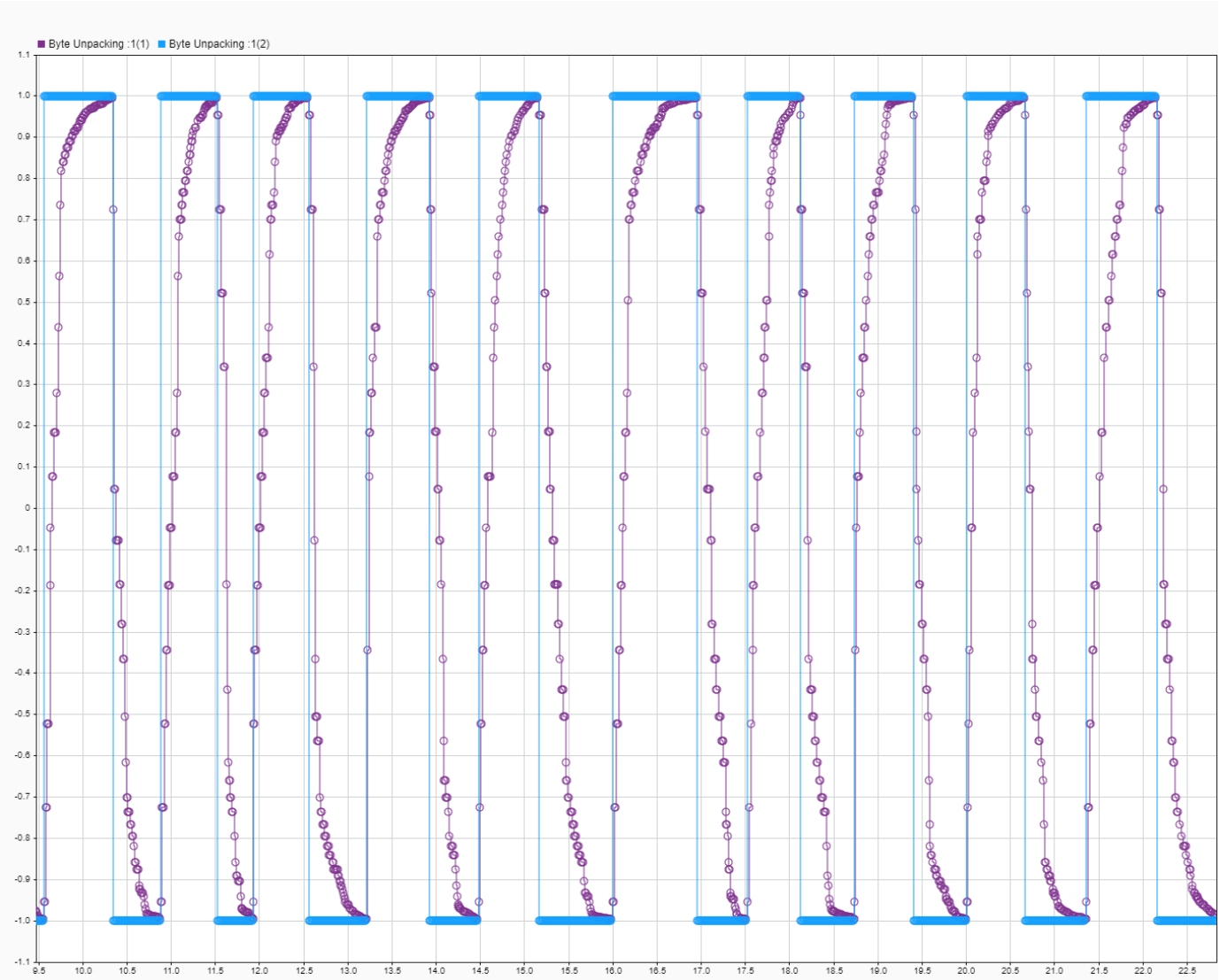
```
start(tg);
```

Read Data and Write Development Computer

```
[data,count,errmsg] = fread(udpObj,16);
while count~=0
    fwrite(udpObj,data);
    [data,count,errmsg] = fread(udpObj,16);
end
```

View Signals in Simulation Data Inspector

```
Simulink.sdi.view;
```



Disconnect UDP Object on Development Computer

```
fclose(udpObj);
```

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 provides a new model that has configuration parameters set up for building a real-time application.

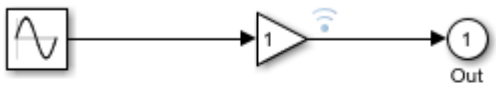
To see the Simulink Real-Time commands for each operation in this example, view the example code.

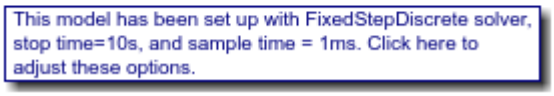
Create Simulink Model from Template

To create a Simulink 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 `exampleSlrealtimeApp` model. Or, in the Command Window, use the `Simulink.createFromTemplate` command. See code for this script for full syntax.





Blocks, Connections, and Data Logging in the Model

The Simulink Real-Time model template contains a Gain block that connects a Signal Generator to an Outport. The Gain block output is marked for logging with the Simulation Data Inspector (SDI).

Simulate Real-Time Application and View Logged Data

Build the real-time application, run it on the target computer, and view the logged data:

1. Make sure that the development computer has a connection to the target computer.

2. 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.

3. Run the real-time application and log data by using the `start` command.

4. 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 Real-Time Application by Using Simulink Real-Time Explorer”
- Simulation Data Inspector

Troubleshooting

Troubleshooting Basics

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.

Link Between Development and Target Computers

Troubleshoot Communication Failure Through Firewall

I see communication timeout errors or access errors with my target computer. Some issue with the Windows Defender Firewall or Symantec Endpoint Protection 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 the firewall software 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 for the Windows Defender firewall or Symantec Endpoint Protection firewall.

Configure Windows Defender Firewall

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 MATLAB 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` by using the Windows search.
3. To allow MATLAB to communicate with Public networks, in **Windows Defender Firewall** > **Allowed apps**, for `MATLAB R2020b`, select `allow Public network`.
4. Select **Inbound Rules** and **New Rule**.
5. For the **Rule Type**, select **Custom**, and click **Next**.
6. For the **Program**, select **All programs**, and click **Next**.
7. For the **Protocol and Ports**, select **Any**, and click **Next**.
8. 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?**.
9. For the **Action**, select **Allow the connection**, and click **Next**.
10. For the **Profile**, select the **Domain**, **Private**, and **Public** check boxes, and click **Next**.
11. For **Name**, provide a **Name** for this inbound rule (for example, `Simulink Real-Time inbound`), and click **Finish**.
12. Select **Outbound Rules** and click **New Rule**.
13. Repeat steps 4 through 10 for the custom outbound rule.

Configure Symantec Endpoint Protection Firewall

If you are using Symantec Endpoint Protection software and get an error message that Simulink Real-Time failed to connect to the target due to a timeout issue, try this workaround:

1. In the Windows Start menu Search, type `firewall and network protection` and open the selection.
2. Under the Domain network selection, click **Open Symantec Endpoint Protection**.
3. Select **Settings** > **Firewall** > **Program Control**.
4. Add the `matlab.exe` path to the list and select **Allow** access.

See Also

More About

- “Enable Development Computer Communication (Windows)”

External Websites

- MathWorks Help Center website

Troubleshoot Signal Data Logging from Nonvirtual Bus, Fixed-Point, and Multidimensional Signals

My models sometimes use signals in nonvirtual buses, signals with fixed-point data types, and mutidimensional signals that have a number of dimensions greater than two. I want to view signal data from these signals in the Simulation Data Inspector. I do not see data for these signals when I select them in Simulink Real-Time Explorer for streaming to the Simulation Data Inspector.

What This Issue Means

There are some guidelines to data logging signals in nonvirtual buses, signals with fixed-point data types, and mutidimensional signals that have a number of dimensions greater than two:

- When these signals are marked for logging with the Simulation Data Inspector, the signal data displays in the Simulation Data Inspector.
- When these signals are connected to File Log blocks, the signal data displays in the Simulation Data Inspector.
- When these signals are selected for dynamic streaming with an instrument object—either by selecting the signals in Simulink Real-Time Explorer or adding the signals by using the `Application` object API, the signal data does not display in the Simulation Data Inspector or in App Designer instrument panel applications.

Try This Workaround

There are workarounds to get signals in nonvirtual buses, signals with fixed-point data types, and mutidimensional signals (that have a number of dimensions greater than two) to display in the Simulation Data Inspector.

Signals in Nonvirtual Buses

To get signals in nonvirtual buses to display in the Simulation Data Inspector, mark the signals for data logging in the model or connect the signals to File Log blocks.

To instrument signals in nonvirtual buses to stream to an `Instrument` object, use the `BusElement` argument in the `addSignal`, `connectLine`, or `connectScalar` methods.

Signals with Fixed-Point Data Types

To get signals with fixed-point data types to display in the Simulation Data Inspector, mark the signals for data logging in the model or connect the signals to File Log blocks.

Multidimensional Signal

To get signals in mutidimensional signals (that have a number of dimensions greater than two) to display in the Simulation Data Inspector, mark the signals for data logging in the model or connect the signals to File Log blocks.

See Also

Bus Creator | `addSignal` | `connectLine` | `connectScalar` | `fixdt`

Related Examples

- “Parameter Tuning and Data Logging” on page 13-2

More About

- “Create Nonvirtual Buses with Bus Creator Blocks”
- “Fixed-Point Data in MATLAB and Simulink”
- “Signal Basics”
- “Variable-Size Signal Basics”

Troubleshoot Signal Data Logging from Send and Receive Blocks

My model uses Send and Receive blocks. I want to view signal data from the message line (output of send or input of receive) in the Simulation Data Inspector. I see unexpected data when I select a message line in Simulink Real-Time Explorer for streaming to the Simulation Data Inspector.

What This Issue Means

There are some guidelines to data logging message line signals:

- Message line signals that are marked for logging with the Simulation Data Inspector display the data accurately in the Simulation Data Inspector.
- Message line signals that are connected to File Log blocks display the data accurately in the Simulation Data Inspector.
- Message line signals that are selected for dynamic streaming with an instrument object—either by selecting the signals in Simulink Real-Time Explorer or adding the signals by using the `Application` object API—do not display the data accurately in the Simulation Data Inspector or in App Designer instrument panel applications.

For more information about message lines, see “Animate and Understand Sending and Receiving Messages”.

Try This Workaround

To get accurate display of message line signals in the Simulation Data Inspector, mark the signals for data logging in the model or connect the signals to File Log blocks.

See Also

File Log

Related Examples

- “Animate and Understand Sending and Receiving Messages”

More About

- “Data Logging with Simulation Data Inspector (SDI)” on page 6-13

Model Compilation

Troubleshoot Model Links to Shared Libraries

Some model build issues are caused by linking to shared object libraries (`.so`).

What This Issue Means

When building real-time applications, the Simulink Real-Time software supports links to QNX Neutrino static link libraries (`.a`) only, not links to shared object libraries (`.so`), unless the shared object is included in the model through an FMU block. Building a real-time application from a model with links to one or more SOs produces a build error.

Try This Workaround

When you build your models, make sure 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 shared object libraries. Each static library must be self contained. The static library must not be dependent on another external library.

See Also

FMU

More About

- “Build Support for S-Functions”
- “Compile Source Code for Functional Mockup Units” on page 3-3

External Websites

- MathWorks Help Center website

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 `slrt_ex_serialasciitest` model, change the Simulink mode to rapid accelerator, and run the model, Simulink displays this error:

```
Unable to build a standalone executable to simulate the model
'slrt_ex_serialasciitest' 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 the model 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 Real-Time Options Pane”

External Websites

- MathWorks Help Center website

Real-Time Application Performance

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:

- To profile execution of a real-time application , use the `startProfiler` command.
- To run the profiler and plot the results, use the `plot` function.

For more information, see “Execution Profiling for Real-Time Applications” on page 9-8.

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. Click **Configure Tasks**, and then select the **Enable explicit model partitioning for concurrent behavior** check box.
4. Create tasks and triggers, and then explicitly assign subsystem partitions to the tasks. See “Partition Your Model Using Explicit Partitioning” and “Multicore Programming with Simulink”.
5. Run the real-time application.

Note Do not use MATLAB System blocks in the top level of Simulink Real-Time models in which task execution is explicitly partitioned. These blocks generate a TLC error when building the real-time application, for example:

```
"Unable to find TLCBlockSID within the Block scope"
```

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. On the **Solver** pane, increase **Fixed-step size (fundamental sample time)**. Executing with a short sample time can overload the CPU.
2. Use polling mode. See “Execution Modes” on page 7-2.
3. 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.

See Also

Related Examples

- “Concurrent Execution on Simulink® Real-Time™” on page 13-6

More About

- “Execution Profiling for Real-Time Applications” on page 9-8
- “Partition Your Model Using Explicit Partitioning”
- “Execution Modes” on page 7-2
- “Find Simulink Real-Time Support” on page 18-2
- “Multicore Programming with Simulink”

External Websites

- MathWorks Help Center website
- www.speedgoat.com/products
- 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 Simulink Real-Time kernel halts model execution and the `Target` object property TargetStatus shows an error, for example:

```
mCPUOverload: Sub-rate exception: Overload limit (0) exceeded in 0.02s rate with 1 overloads
```

If you allow the overload, model execution continues until the allowed overload limit is reached. 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.

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.
- The model is too complex (algorithmic complexity).
- 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 SLRT Overload Options block to increase the number of startup time steps to ignore overloads. By default, only the first time step ignores overloads.

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 the SLRT Overload Options block only for diagnosis. When your diagnosis is complete, turn off these options.

See Also

Related Examples

- “Monitor CPU Overload Rate” on page 9-3

More About

- “CPU Overload” on page 9-2

External Websites

- MathWorks Help Center website

Troubleshoot Gaps in Streamed Data

A real-time application is producing a live streaming overload while attempting to stream signal data at a high rate.

What This Issue Means

Live streaming from a real-time application does not guarantee all the data appears in the Simulation Data Inspector. Live stream instrumentation runs at a lower priority than the real-time application. So, data sent by live streaming could be dropped if the host-target connection cannot keep up.

If a live stream overload occurs, you could see noticeable gaps in the data in the Simulation Data Inspector or see that some timesteps are lost when you export data from the Simulation Data Inspector.

Try This Workaround

The issue is caused by high data rates and live streaming of data.

To workaround the issue:

- Modify the real-time application to decrease the data rate for live streaming data. To do this, you could increase the sample rate, instrument fewer signals, or increase the decimation of instrumented signals.
- Change the real-time application to use file logging instead of live streaming. File logging is capable of logging higher data rates without dropping data.

See Also

Related Examples

- “Parameter Tuning and Data Logging” on page 13-2

More About

- “Trace or Log Data with the Simulation Data Inspector” on page 6-22

External Websites

- MathWorks Help Center website

Simulink Real-Time Support

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:

- For online or phone support, contact MathWorks technical support 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 the software version to your development computer.
3. Install and integrate the new release software.

After updating Simulink Real-Time, to re-create your Simulink Real-Time target settings:

1. In the MATLAB Command Window, type `slrtExplorer`.
2. On the **Targets Tree** pane, select a target computer node.
3. Click the **Target Configuration** tab.
4. Click **Change IP Address** and select the IP Address and Netmask for communication method between your development and target computer. For more information, see “Target Computer Settings”. Click **OK**.
5. Click the **Disconnected** link, toggling it to **Connected**.
6. Repeat steps 2 through 5 for each target computer.
7. Build each model that you want to execute. In the Simulink Editor, on the **Real-Time** tab, click **Run on Target**.

See Also

More About

- “Target Computer Settings”

External Websites

- www.mathworks.com/downloads
- www.speedgoat.com/support