RTI and RTI-MP

Implementation Guide

Release 2021-A – May 2021

dSPACE

How to Contact dSPACE

Mail:	dSPACE GmbH Rathenaustraße 26 33102 Paderborn Germany
Tel.:	+49 5251 1638-0
Fax:	+49 5251 16198-0
E-mail:	info@dspace.de
Web:	http://www.dspace.com

How to Contact dSPACE Support

If you encounter a problem when using dSPACE products, contact your local dSPACE representative:

- Local dSPACE companies and distributors: http://www.dspace.com/go/locations
- For countries not listed, contact dSPACE GmbH in Paderborn, Germany.
  Tel.: +49 5251 1638-941 or e-mail: support@dspace.de

You can also use the support request form: http://www.dspace.com/go/supportrequest. If you are logged on to mydSPACE, you are automatically identified and do not need to add your contact details manually.

If possible, always provide the relevant dSPACE License ID or the serial number of the CmContainer in your support request.

Software Updates and Patches

dSPACE strongly recommends that you download and install the most recent patches for your current dSPACE installation. Visit http://www.dspace.com/go/patches for software updates and patches.

Important Notice

This publication contains proprietary information that is protected by copyright. All rights are reserved. The publication may be printed for personal or internal use provided all the proprietary markings are retained on all printed copies. In all other cases, the publication must not be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form, in whole or in part, without the prior written consent of dSPACE GmbH.

© 1999 - 2021 by:
dSPACE GmbH
Rathenaustraße 26
33102 Paderborn
Germany

This publication and the contents hereof are subject to change without notice.

AUTERA, ConfigurationDesk, ControlDesk, MicroAutoBox, MicroLabBox, SCALEXIO, SIMPHERA, SYNECT, SystemDesk, TargetLink and VEOS are registered trademarks of dSPACE GmbH in the United States or other countries, or both. Other brand names or product names are trademarks or registered trademarks of their respective companies or organizations.

Contents

About This Guide 9

Safety Precautions 13

General Warning ... 13

Introduction to RTI and RTI-MP 15

Development Steps ... 15
Throttle Control Example ... 16

Working with RTI and RTI-MP 21

How to Start RTI for a Particular Platform ... 22
How to Activate a Specific Platform Support ... 22
How to Access RTI Blocks ... 23
How to Customize the MATLAB Start-Up ... 25
How to Adapt the Default Configuration for Model Properties ... 26
How to Specify RTI-Specific Settings for a Model ... 27
How to Remove the RTI-Specific Settings from a Model ... 30
How to Attach a Configuration Reference ... 31

Defining I/O Interfaces 33

Naming Conventions ... 33
PHS-Bus Address and Board Number ... 35
Vectorized Versus Scalar I/O Blocks ... 36
Data Type Basics ... 37
How to Add I/O Blocks to Models ... 38

Handling Tasks 41

Introducing Task Handling ... 42
Real-Time Kernel and Scheduler ... 42
Priorities and Task-Switching Time ... 43
Task States and Execution Order ... 44
Overrun Situation and Turnaround Time ... 45

Timer Tasks ... 48
Timer Task Basics ... 48
Single Timer Task Mode ... 50
Multiple Timer Task Mode ... 51
Example Applications Requiring the Multiple Timer Task Mode ... 53
Comparing the Execution Modes ... 54
How to Connect Timer Tasks to an External Interrupt Source ... 55

Tasks Driven by Interrupt Blocks ... 57

Basics on Triggered Subsystems ... 57
Triggered Subsystems ... 57

Preparing a Function-Call Subsystem ... 60
Introduction to Interrupt-Block-Driven Tasks and Task Transitions ... 60
How to Prepare Function-Call Subsystems ... 61

Implementing Interrupt-Block-Driven Tasks ... 62
How to Use Hardware Interrupts ... 62
How to Use Software Interrupts ... 65
How to Branch Interrupt Signals ... 67
Example of a Task Driven by a Hardware Interrupt Block ... 68

Rate Transitions ... 70
Basics of Rate Transitions ... 70
How to Implement Sample-Rate Transitions ... 75
How to Implement Protected Task Transitions ... 75
How to Implement Unprotected Task Transitions ... 77
How to Implement Task Transitions for Multiple Receiving Blocks ... 79

Assigning Parts of A Model to The Background Task ... 81
Basics of the Background Task ... 81
How to Assign Parts of Models to Background Tasks ... 82
Example of a Subsystem Assigned to the Background Task ... 84

Configuring Tasks ... 86
Basics on Task Priorities ... 86
How to Change the Task Priorities ... 89
How to Specify Overrun Strategies for Tasks ... 90
How to Configure Tasks via Scripts ... 92
How to Combine Several Interrupt Sources ... 94
How to Subschedule Tasks ... 96

Time-Triggered Tasks and Timetables ... 99
Basics of Time-Triggered Tasks and Timetables ... 99
How to Implement Time-Triggered Tasks ... 102
How to Implement Timetables ... 104

Distributing the Model for MP Systems 107

Basics of Distributing the Model for MP Systems..... 108
Details on MP Systems..... 108
How to Configure Models for MP Systems..... 111
Standard Network Topology..... 113
Model Distribution Guidelines..... 114

Interprocessor Communication..... 115

Introducing Interprocessor Communication..... 115
Gigalink Connection..... 116
Swinging Buffer Protocol..... 118
Virtual Shared Memory Protocol..... 119
Model Evaluation..... 120
Direct Feedthrough..... 122

Implementing Interprocessor Communication Using IPC Blocks..... 124
Communication with a Sample Time Transition..... 124
Interprocessor Communication Using IPC Blocks..... 130
How to Implement Interprocessor Communication Using IPC Blocks..... 132
How to Make IPCx Blocks Use a Different Channel..... 133
How to Redirect all Member IPCs of a Channel to a New CPU..... 134
How to Define a Default CPU for Unconnected Model Parts..... 136

Implementing Interprocessor Communication Using Gigalink Blocks..... 137
Interprocessor Communication Using Gigalink Blocks..... 138
How to Implement Interprocessor Communication Using Gigalink Blocks..... 141

Interprocessor Interrupts..... 143
How to Implement Interprocessor Interrupts..... 143
How to Change the Task Priorities in RTI-MP Models..... 145

Referencing Models 149

Basics on Model Referencing..... 150
How to Create Model References..... 154
Creating a Model Reference from an Atomic Subsystem..... 155
Configuring Tasks with Model Referencing..... 156
How to Execute Referenced Models in Interrupt-Driven Tasks..... 157
Simulating and Building Models Containing Model Blocks..... 160

Inserting Custom C/C++ Code 163

Implementing S-Functions ... 164
- Basics of S-Functions ... 164
- Purpose of the C Methods ... 166
- S-Functions for Real-Time and Simulink Simulations ... 167
- Accessing PHS-bus-based systems in S-Functions ... 168
- How to Access dSPACE I/O Devices in S-Functions ... 169

Implementing User-Code ... 171
- Basics on User-Code ... 171
- User-Code Interprocessor Communication ... 173
- How to Include User-Code in Simulink Models ... 174
- How to Generate a Template for the User-Code ... 177
- How to Implement the User-Code ... 178

Integrating C++ Code ... 180
- Using C++ Code in an RTI Application ... 180

Comparing S-Functions and User-Code ... 186
- Execution Order of S-Functions and User-Code ... 186
- Tips and Tricks for Custom C Code ... 188

Examples ... 190
- Flow Chart for the Custom C Code of the Throttle Control Example ... 190
- C Code for the Throttle Control Example's S-Function ... 191
- C Code for the Throttle Control Example's User-Code ... 194

Advanced Techniques ... 196
- How to Make Custom Variables Accessible ... 196
- How to Send Messages to the Message Module ... 197
- Example of a Slave Application as User-Code ... 199
- Example of a Slave Application as an S-Function ... 201

Managing the Simulation 205

- Simulation Control (RUN/STOP Mechanism) ... 205
- How to Set the Simulation State ... 208
- How to Read the Simulation State ... 209
- Time-Stamping and Data Acquisition ... 210
- How to Modify Data Acquisition Behavior ... 213
- Reaction to Run-Time Errors ... 215
- Different Results from Different Simulation Modes ... 216

Building and Downloading the Model 219

Basics on the Build and Download ... 220
Introducing the Build and Download Process ... 220
Basics on the Generation of TRC Files ... 224
How to Specify Options for the Build Process ... 226
How to Start the Build and Download ... 228
How to Automate the Build Process ... 230
How to Back Up Real-Time Applications ... 231
How to Remove the Temporary Files of the Build Process ... 232

Debugging an Application ... 234
The RTI-MP Diagnostic Tool ... 234
Handling Exceptions ... 236
How to Trace an Exception ... 237
How to Find the Simulink Block Related to an Exception ... 239

Creating an Application Without Connected dSPACE System ... 241
How to Create an Application Without a Connected dSPACE System ... 241

User-Specific Files, Source Files and Libraries ... 243
Adding User-Specific Files, Source Files and Libraries ... 243

Advanced Techniques 245

Optimizing the Build Process and Simulation via Atomic Subsystems ... 246
Virtual Versus Atomic Subsystems ... 246
How to Bundle the Execution of Blocks ... 247
How to Reduce the Function Sizes via Atomic Subsystems ... 248
How to Reduce the Files Sizes via Atomic Subsystems ... 249

Excluding Subsystems from the TRC File ... 250
Advantages of a Reduced TRC File ... 250
How to Exclude a Subsystem from the TRC File ... 253

Modifying the Block Execution Order ... 256
Determining the Execution Order ... 256
How to Check for Illogical Block Priorities ... 258

Tuning Parameters of Transfer Fcn, Zero-Pole and State-Space Blocks ... 259
MATLAB's State-Space Representation for a Transfer Function ... 259
How to Tune Transfer Function Parameters via Manual Conversion ... 261
How to Tune Transfer Function Parameters via Parameter Inlining ... 261
How to Keep Zero Parameters of State-Space Blocks ... 263

Using Simulink's Model Verification Blocks 265
Basics of Simulink's Model Verification Blocks 265
How to Use Simulink's Model Verification Blocks 265

Conventions and Limitations 267

Conventions 268
General Conventions 268
Data Type Restrictions 269

Limitations 270
General Limitations 270
Unsupported Simulink and Simulink Coder Features 272
Limitations with Stateflow 275
Limitations in Task Handling 276
Limitations with RTI-MP 276
Limitations with the RTI Gigalink Blockset 279
Limitations with Model Referencing 279
Limitations for Custom Code 282
Limitations Regarding Simulink Variables in the TRC File 283
Limitations in Building Models 285
Miscellaneous Limitations 286

Troubleshooting 289

Implementing the Model 290
Automatic Program Building 293
Checking the Model 295
Generating the Code 296
Compiling and Linking the Code 297
Using Model Referencing 299
Using Subsystem Referencing 302
Initializing the Application 302
Running the Application 304
Running ControlDesk 308

Index 311

About This Guide

Content

This guide introduces you to Real-Time Interface (RTI and RTI-MP) for dSPACE systems.

- *RTI* acts as the link between Simulink®, Simulink® Coder™ and the dSPACE hardware (DS1006, DS1007, DS1104, MicroAutoBox II, MicroLabBox).
- *RTI-MP* enables you to partition your system model and allocate the parts to the different CPUs of a multiprocessor system or to the CPU cores of a multicore system.

After reading through this guide you will be able to use RTI and RTI-MP to add I/O interfaces, interrupts, etc. to your Simulink model, build real-time code, and download and execute it on dSPACE hardware.

All the descriptions and screenshots in this guide are based on MATLAB R2021a. If there are major changes working with supported earlier MATLAB versions, these changes are described too.

Note

There are a few limitations concerning RTI's support for Simulink blocks. For details, refer to Conventions and Limitations on page 267.

For a brief overview of the new features of RTI and RTI-MP, refer to RTI/RTI-MP and RTLib (New Features and Migration 🕮).

Supported MATLAB versions RCP and HIL software from dSPACE Release 2021-A supports the following MATLAB versions:

- R2021a
- R2020b
- R2020a
- R2019b

Note

- For information on the compatibility of all products from dSPACE Release 2021-A with the current MATLAB release and MATLAB releases that came out later than dSPACE Release 2021-A, visit www.dspace.com/go/compatibility.
- This document describes Real-Time Interface for all hardware platforms. Not all sections apply to all platforms. You can skip all paragraphs that are not relevant to your platform.

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
DANGER	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
WARNING	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
CAUTION	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a hazard that, if not avoided, could result in property damage.
Note	Indicates important information that you should take into account to avoid malfunctions.
Tip	Indicates tips that can make your work easier.
	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.
	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

%name% Names enclosed in percent signs refer to environment variables for file and path names.

< > Angle brackets contain wildcard characters or placeholders for variable file and path names, etc.

Examples:

- Where you find terms such as `rti<XXXX>` replace them by the RTI platform support you are using, for example, `rti1007`.

- Where you find terms such as `<model>` or `<submodel>` in this document, replace them by the actual name of your model or submodel. For example, if the name of your Simulink model is `smd_1007_sl.slx` and you are asked to edit the `<model>_usr.c` file, you actually have to edit the `smd_1007_sl_usr.c` file.

RTI block name conventions All I/O blocks have default names based on dSPACE's board naming conventions:

- Most RTI block names start with the board name.
- A short description of functionality is added.
- Most RTI block names also have a suffix.

Suffix	Meaning
B	Board number (for PHS-bus-based systems)
M	Module number (for MicroAutoBox II)
C	Channel number
G	Group number
CON	Converter number
BL	Block number
P	Port number
I	Interrupt number

A suffix is followed by the appropriate number. For example, DS2201IN_B2_C14 represents a digital input block located on a DS2201 board. The suffix indicates board number 2 and channel number 14 of the block. For more general block naming, the numbers are replaced by variables (for example, DS2201IN_Bx_Cy).

Special folders

Some software products use the following special folders:

Common Program Data folder A standard folder for application-specific configuration data that is used by all users.

`%PROGRAMDATA%\dSPACE\<InstallationGUID>\<ProductName>`

or

`%PROGRAMDATA%\dSPACE\<ProductName>\<VersionNumber>`

Documents folder A standard folder for user-specific documents.

`%USERPROFILE%\Documents\dSPACE\<ProductName>\<VersionNumber>`

Local Program Data folder A standard folder for application-specific configuration data that is used by the current, non-roaming user.

`%USERPROFILE%\AppData\Local\dSPACE\<InstallationGUID>\<ProductName>`

Accessing dSPACE Help and PDF Files

After you install and decrypt dSPACE software, the documentation for the installed products is available in dSPACE Help and as PDF files.

dSPACE Help (local) You can open your local installation of dSPACE Help:

- On its home page via Windows Start Menu
- On specific content using context-sensitive help via **F1**

dSPACE Help (Web) You can access the Web version of dSPACE Help at www.dspace.com/go/help.

To access the Web version, you must have a *mydSPACE* account.

PDF files You can access PDF files via the icon in dSPACE Help. The PDF opens on the first page.

Safety Precautions

Introduction

To avoid risk of injury and/or property damage, read and ensure compliance with the safety precautions given.

General Warning

Danger potential

Using dSPACE software can be dangerous. You must observe the following safety instructions and the relevant instructions in the user documentation.

Improper or negligent use can result in serious personal injury and/or property damage

Using the dSPACE software can have a direct effect on technical systems (electrical, hydraulic, mechanical) connected to it.

The risk of property damage or personal injury also exists when the dSPACE software is controlled via an automation interface. The dSPACE software is then part of an overall system and may not be visible to the end user. It nevertheless produces a direct effect on the technical system via the controlling application that uses the automation interface.

- Only persons who are qualified to use dSPACE software, and who have been informed of the above dangers and possible consequences, are permitted to use this software.
- All applications where malfunctions or operating errors involve the danger of injury or death must be examined for potential hazards by the user, who must if necessary take additional measures for protection (for example, an emergency off switch).

Liability

It is your responsibility to adhere to instructions and warnings. Any unskilled operation or other improper use of this product in violation of the respective safety instructions, warnings, or other instructions contained in the user documentation constitutes contributory negligence, which may lead to a limitation of liability by dSPACE GmbH, its representatives, agents and regional

dSPACE companies, to the point of total exclusion, as the case may be. Any exclusion or limitation of liability according to other applicable regulations, individual agreements, and applicable general terms and conditions remain unaffected.

Data loss during operating system shutdown

The shutdown procedure of Microsoft Windows operating systems causes some required processes to be aborted although they are still being used by dSPACE software. To avoid data loss, the dSPACE software must be terminated manually before a PC shutdown is performed.

Introduction to RTI and RTI-MP

Introduction

RTI and RTI-MP are used in the development phases of plant models and controllers.

Where to go from here

Information in this section

Development Steps.....15
Throttle Control Example.....16

Development Steps

Control design and Simulink simulation

At this stage of development the simulation is performed in Simulink. The major feature of this type of simulation is that the computer has as much time as needed to calculate the behavior of the system. Thus, if your model is simple, the results can be calculated quickly. If your model is complex, much more time is needed to carry out the necessary calculations. However, because you do not need to fulfill a strict time requirement for Simulink simulation, you do not need to reduce the complexity of the model.

RCP with dSPACE Prototyper

The whole situation is different once the simulated control system meets your expectations and you want to test the controller on the actual plant. Since the controller might need further modifications, you do not want to produce it in hardware at this stage of development. Therefore, you have to connect the real plant to a controller that is simulated in real time – this technique is called rapid control prototyping (RCP). The major feature of this real-time simulation is that the simulation has to be carried out as quickly as the real system would actually run, thereby allowing you to combine the simulation and the real plant.

HIL with dSPACE Simulator

When your simulated controller is able to control your real plant, you typically produce the actual controller. For the final tests you usually connect the real controller to a model of the plant, which, of course, has to be simulated in real time. This way you can ensure that the controller does not contain any errors that could damage the real plant. This technique is called hardware-in-the-loop simulation (HIL).

For both RCP and HIL the real-time simulation is rather important. The computing power required by real-time simulation highly depends on the characteristics of the simulated model: If it contains very demanding calculations you have to provide a lot of computing power because the timing cannot be satisfied otherwise. dSPACE systems fulfill this demand for computing power.

Automatic code generation

Because real-time simulation is such a vital aspect for control engineering, the same is true for the automatic generation of real-time code, which can then be implemented on the hardware. For dSPACE systems, Real-Time Interface (RTI/RTI-MP) carries out this linking function. Together with Simulink® Coder™ from MathWorks® it automatically generates real-time code from Simulink models and implements this code on dSPACE real-time hardware. Therefore, you save time and effort twice:

- You do not need to manually convert your Simulink model into another language such as C.
- You do not need to be concerned about a real-time program frame and I/O function calls, or about implementing and downloading the code onto your dSPACE hardware.

RTI carries out these steps for you. You just have to add the required dSPACE blocks (for example, I/O interfaces) to your Simulink model.

Related topics

Examples

Throttle Control Example..... 16

Throttle Control Example

Introduction

The following example demonstrates the steps from Simulink to real-time simulation.

Control design and Simulink simulation

Suppose you want to develop an electronic throttle control (ETC or drive-by-wire) for a throttle valve like the one shown below.

ETC departs from conventional mechanical systems by replacing the Bowden cable or linkage with an ECU and the electric motor for throttle valve control. The system can thus control and modify the throttle valve aperture with reference to numerous operating parameters, performing tasks such as torque reduction for electronic traction control.

The first step in developing an ETC is to model the throttle valve itself. Then you have to add the model of the actual electronic throttle control. You can use the resulting Simulink model to optimize the ETC in Simulink simulation. An appropriate Simulink model is shown in the illustration below:

Rapid control prototyping

The next step is to build the real-time code for the ETC (including I/O interfaces) and to download it onto the dSPACE hardware. You can then connect it to the real throttle valve for further optimization (rapid control prototyping). The following illustration shows the arrangement according to the MicroAutoBox II demonstration system:

Hardware-in-the-loop simulation

If you want to control the simulated throttle valve at a later stage with the ETC (hardware-in-the-loop simulation), the arrangement is the other way round.

This guide will show how to implement I/O interfaces in your models below.

Build & download

For a single-processor system such as MicroAutoBox II, the following steps are necessary to generate real-time code for the model, to download the code onto the dSPACE hardware and to simulate it. To carry out these steps you have to open the Configuration Parameters of your model.

- Choose a fixed-step solver and specify a fixed-step size in the Solver dialog.

- Specify the correct settings in the **Code Generation** dialog for MicroAutoBox II.

- When you start the build and download procedure, Simulink Coder and RTI generate the real-time code and download it to your dSPACE hardware – provided that the hardware is registered correctly with the Platform Manager, for example, using ControlDesk.

Related topics

Basics

Development Steps..... 15

Working with RTI and RTI-MP

Introduction

When you start working with RTI and RTI-MP, you must know important aspects like selecting the platform or customizing the environment.

Note

- Models with RTI can be easily reused from project to project, as new RTI versions can automatically migrate blocks and configuration data from most of the preceding RTI versions.
- RTI provides consistency checks, so potential errors can be avoided.

Where to go from here

Information in this section

How to Start RTI for a Particular Platform.....22
How to Activate a Specific Platform Support.....22
How to Access RTI Blocks.....23
How to Customize the MATLAB Start-Up.....25
How to Adapt the Default Configuration for Model Properties.....26
How to Specify RTI-Specific Settings for a Model.....27
How to Remove the RTI-Specific Settings from a Model.....30
How to Attach a Configuration Reference.....31

How to Start RTI for a Particular Platform

Objective

Your dSPACE RCP and HIL software installation includes several dSPACE platforms, but you can work with one platform only at the same time.

Method

To start RTI for a particular platform

1 When MATLAB is started, RTI displays a dialog that lets you choose the desired platform.

Tip

If you select the Do not show this dialog again checkbox, MATLAB will always start with the platform that was last active.
This option can be reset by clicking the *Click here to re-enable RTI platform selection for next MATLAB start link* in the MATLAB Command Window.

Related topics

HowTos

How to Activate a Specific Platform Support 22

References

rti1xxx (RTI and RTI-MP Implementation Reference 🕮)

How to Activate a Specific Platform Support

Objective

You cannot work with a platform which is inactive. You must first activate it.

Method

To activate a different platform

1 Change to the MATLAB prompt and type `rti<XXXX>` to activate the desired platform (replace `rti<XXXX>` with the platform you want to use: for example, `rti1007`).

Result

The corresponding RTI library opens automatically.

Related topics

HowTos

How to Access RTI Blocks 23
How to Specify RTI-Specific Settings for a Model 27

References

rti1xxx (RTI and RTI-MP Implementation Reference 🕮)

How to Access RTI Blocks

Objective

The various RTI and RTI-MP blocks allow you to access the dSPACE hardware.

Accessing blocks

There are two ways to get a specific block: You can use the **Simulink Library Browser**, or the RTI and RTI-MP block libraries.

Note

- You must not break the library links of the RTI blocks. Otherwise, the blocks cannot be updated when you migrate your model to a new version of RTI and it is very likely that your model will not operate correctly.
- When you work with RTI blocks note that Simulink models containing RTI blocks cannot be downgraded to an older Simulink version.

Possible methods

You can access RTI blocks as follows:

- You can use the **Simulink Library Browser**. Refer to Method 1.
- You can use the **RTI block library**. Refer to Method 2.

Method 1

To access an RTI block via the Simulink Library Browser

1 Open the Simulink Library Browser.

2 Expand the relevant dSPACE library, and find the block in one of the sublibraries it contains.

3 Drag and drop the block into your model.

Example

The following illustration shows the dSPACE RTI1401 library in the Simulink Library Browser.

Method 2

To access an RTI block via the RTI block library

1 Open the RTI block library by typing `rti` at the MATLAB prompt, open the corresponding sublibrary, and find the block.

To access multiprocessor-specific blocks you have to open the RTI-MP library by typing `rtimp` at the MATLAB prompt.

If you want to use only a single processor board of a multiprocessor system, you do not need to add any RTI-MP-specific blocks to your model.

2 Drag and drop the block into your model.

Example

The following illustration shows the dSPACE RTI1401 library.

Related topics

HowTos

How to Activate a Specific Platform Support 22

References

General RTI and RTI-MP Libraries (RTI and RTI-MP Implementation Reference 🕮)
Overview of RTI1006 (DS1006 RTI Reference 🕮)
Overview of RTI1007 (DS1007 RTI Reference 🕮)
Overview of RTI1104 (DS1104 RTI Reference 🕮)
Overview of RTI1202 (MicroLabBox RTI Reference 🕮)
Overview of RTI1401 (MicroAutoBox II RTI Reference 🕮)

How to Customize the MATLAB Start-Up

Objective

To customize the MATLAB start-up, you must use custom scripts.

Execution order of the initialization phase

The execution order is:

- dSPACE initialization
- `dsstartup.m`
- `dspoststartup.m`
- `startup.m`

Method

To customize the MATLAB start-up

1 Add the necessary commands in the `dsstartup.m` file, for example, to automatically open a library or model whenever MATLAB is started. The `dsstartup.m` file has to be placed in the MATLAB search path.

For more information, refer to dsstartup.m (RTI and RTI-MP Implementation Reference 🕮).

Tip

To specify further commands to be executed after the dSPACE initialization, you can use **`dspoststartup.m`**.

Example

Suppose you want MATLAB to automatically change to your working folder `d:\work` and open the Simulink model `my_model.slx`. Create the dsstartup.m file and write the following commands to it:

```
cd d:\work
my_model
```

Customizing the MATLAB shutdown

You can customize the MATLAB shutdown by using the `dsfinish.m` file to carry out user-specific commands before MATLAB is shut down. Place this M file in the MATLAB search path, for example, in MATLAB's working folder. You can specify several `dsfinish.m` files. All `dsfinish.m` files located in the MATLAB search path are executed before MATLAB is shut down.

Related topics

HowTos

How to Adapt the Default Configuration for Model Properties..... 26

References

dsfinish.m (RTI and RTI-MP Implementation Reference 🕮)
dspoststartup.m (RTI and RTI-MP Implementation Reference 🕮)
dsstartup.m (RTI and RTI-MP Implementation Reference 🕮)
startup.m (RTI and RTI-MP Implementation Reference 🕮)

How to Adapt the Default Configuration for Model Properties

Objective

To set the configuration preferences, you have to use the Simulink Default Model Template for your RTI platform.

Configuration sets

Simulink provides different types of configuration sets. The following configuration sets can be distinguished:

- Model-specific configuration sets
- Referenced configuration sets

The `new_system` command creates a model based on the default template. The New command in the user interface (Simulink Start Page) creates a model based on the custom template.

The settings in the templates are configured when you switch the RTI platform and use the Set template file automatically command in the Unsuitable Default Model Template dialog.

Model-specific configuration sets Your model can have several configuration sets. You can activate a specific configuration set for a simulation or code generation.

Referenced configuration sets Referenced configuration sets are special model-specific configuration sets. Your model can have several configuration references that link to configuration sets in the MATLAB base workspace. The referenced configuration sets in the MATLAB base workspace contain the simulation parameters which are used for your model. Referenced configuration sets can be linked by any number of models. I.e., changing the value of any parameter in the configuration set changes it for every model that references this configuration set.

Default Model Template set for use with RTI

RTI checks the settings in the current default Simulink template file. If the settings does not suit to the selected RTI platform, the following message dialog is displayed.

Tip

You can access the model-specific preference settings via the Model Explorer.

How to Specify RTI-Specific Settings for a Model

Model-specific configuration sets

Simulink provides model-specific configuration sets which are saved together with the model file. A configuration set is a set of values for your models configuration parameters.

You can specify different configuration sets for your model, for example, one for RTI1401, and one for RTI1007. In this case, for example, if you change from

RTI1401 to RTI1007 platform support, you can simply activate the corresponding configuration set and continue working with the same model without altering the configuration used for RTI1401. For RTI single-processor models, you can configure RTI options in active and inactive configuration sets for which you have specified an RTI<xxxx> target.

Note

- For RTI single-processor models, you can specify the RTI options for the model-specific configuration either in the Model Explorer or in the Configuration Parameters dialog. Both methods lead to the same result.
- To specify RTI options for multiprocessor systems, you have to use the Multiprocessor Setup dialog.

The following illustration displays, for example, a model which contains an active configuration set named *Configuration* and two inactive ones named *Configuration1* and *Configuration2*.

Note

- You can specify an RTI-specific configuration independently of the RTI platform that is currently activated. For example, while RTI1007 is the currently active platform support, you can create a new configuration set and configure it for RTI1401 (or any other RTI platform support) without the need to activate the RTI1401 platform support.
- Additionally, you can open a model that is configured for a specific RTI platform while another RTI platform is activated. For example, while RTI1007 is the currently active platform, you can open a model that is configured for RTI1401.
- You can start an RTI build process only if the RTI platform specified in the active configuration set is the same as the active RTI platform. Otherwise, an error message is displayed.
- Task configuration cannot be specified for each configuration set separately since all configurations refer to the same model that contains the tasks to be configured. Thus, RTI task configuration is the same for all RTI-specific configuration sets.
- You can open the RTI Task Configuration dialog only for an active configuration set and if the specified target is the same as the activated platform.

Target-specific default values If you change the RTI<xxxx> target in the Configuration Preferences set or in a model-specific configuration set, target-specific values are reset to their target-specific default values. All other values remain unchanged.

Precondition

You must have specified different model-specific configuration sets. For details, refer to the Simulink documentation.

Method

To specify RTI-specific settings for a model

1. In your model, open the Model Explorer.
2. In the Contents pane, right-click the desired configuration and select Activate from the context menu.

MATLAB activates the selected configuration set.

Tip

- You might also need to replace the I/O blocks of the model with I/O blocks for the new platform.
- It is possible to use configuration references in addition to configuration sets. For details, refer to How to Attach a Configuration Reference on page 31.

Related topics

HowTos

How to Activate a Specific Platform Support 22
How to Remove the RTI-Specific Settings from a Model 30

References

Model Configuration Parameters Dialogs (RTI and RTI-MP Implementation Reference 🕮)
set_rti (RTI and RTI-MP Implementation Reference 🕮)

How to Remove the RTI-Specific Settings from a Model

Objective

When a model is configured for use with RTI, it contains not only RTI-specific blocks but also block-independent, RTI-specific settings. These remain in the model even if you remove all the RTI-specific blocks. If you open such a model with a non-RTI environment, MATLAB issues appropriate warnings. Therefore,

you should use the `rti_mdlcleanup` function to remove all the RTI-specific settings from the model and avoid the warnings.

Method

To remove the RTI-specific settings from a model

1. Open the model you want to convert.
2. Delete all RTI blocks from the model.
3. Enter `rti_mdlcleanup` in the MATLAB Command Window.

Result

This function removes all RTI-specific settings, and closes and reopens the model.

Related topics

HowTos

How to Specify RTI-Specific Settings for a Model..... 27

References

rti_mdlcleanup (RTI and RTI-MP Implementation Reference 🕮)

How to Attach a Configuration Reference

Working with configuration references

You can link from your model to a configuration set which is stored in the MATLAB base workspace and serves as a referenced configuration set, i.e., it provides the configuration parameters for your model. This referenced configuration set can be used by any number of models. All those models which share the referenced configuration set contain the same configuration parameters. These simulation parameters can be modified in the referenced configuration set without changing the model files.

To link to a referenced configuration set you must first of all attach a configuration reference to your model. You can attach any number of configuration references to it. Each must have a unique name.

Note

If you change the MATLAB version and/or the dSPACE Release, configuration sets stored in a MAT file of an earlier version might cause problems. Therefore, you are recommended to create these configuration sets again when you change the Release version.

Precondition

A configuration set that should serve as a referenced configuration set must exist in the MATLAB base workspace. It can be created by using the command:

```
myCfgSet = Simulink.ConfigSet
```

For details, refer to the Simulink documentation.

Method

To attach a configuration reference

1. In your model, open the Model Explorer.
2. In the Model Explorer, select the node of the model to which the configuration reference should be attached.
3. From the menu bar select Add – Configuration Reference.

   A new configuration reference with the default name Reference is attached to the selected model. This is visualized by a Reference node in the Contents pane of the Model Explorer.

4. In the Model Explorer, select or enter a name of an existing configuration you want to reference.
5. If the specified configuration set is available, you can click Open to open the Configuration Reference dialog.
6. Specify the parameters of the configuration reference according to your needs.

Result

You have linked to a configuration set in the MATLAB workspace from the attached configuration reference of your selected model.

Defining I/O Interfaces

Introduction

To connect the real-time simulation to the physical world, you need to introduce I/O interfaces into your model. These allow you to replace parts of your simulated model with real hardware. In RTI and RTI-MP, the I/O of each dSPACE board is supported by one or more I/O libraries that contain the corresponding I/O blocks.

Note

If you want to generate an S-function from a model or subsystem using the RTW S-Function (**C/C++ Code - Generate S-Function**), you cannot use any I/O blocks in that model.

Where to go from here

Information in this section

Naming Conventions.....33
PHS-Bus Address and Board Number.....35
Vectorized Versus Scalar I/O Blocks.....36
Data Type Basics.....37
How to Add I/O Blocks to Models.....38

Naming Conventions

Board naming conventions

All I/O blocks have default names based on dSPACE's board naming conventions:

- Block names always start with the board name.
- The description of their functionality is added.

- In PHS-bus-based systems, the suffix B and the board number are added. If your system contains several boards of the same type, this board number enables RTI to distinguish between them. The board numbering restarts with 1 for each board type. For details on the relation between the PHS-bus address of an I/O board and the corresponding board number, refer to PHS-Bus Address and Board Number on page 35.
- For MicroAutoBox II, the suffix M and the module number are added. If your MicroAutoBox II contains several modules of the same type, this module number enables RTI to distinguish between them. The module numbering restarts with 1 for each module type.
- If the I/O unit allows channel-wise access, the suffix C and the selected channel number are added.
- There are further suffixes: G specifies a group number, CON a converter number, I an interrupt number, P a port number, and BL a channel block.

If you customize the name of a block, it does not correspond to dSPACE's naming conventions, and is therefore displayed in italics. Do not use the " character (quotation mark) in block names.

Note

When you copy an I/O block, Simulink might modify the board, channel, group number, etc. in order to keep the block names unique. This inherently modifies the settings of the copied block.

Example

The following illustration shows a few examples of I/O block names:

Related topics

Basics

PHS-Bus Address and Board Number 35

HowTos

How to Add I/O Blocks to Models 38

PHS-Bus Address and Board Number

Introduction

If you have a PHS-bus-based system that contains several I/O boards of the same type, you can distinguish them via their PHS-bus addresses, which make them unique in your system (the PHS-bus address of each board was specified during the hardware installation refer to *How to Set the PHS-Bus Address* in the corresponding *Installation and Configuration Guide* for details).

Sorting I/O boards

The Platform Manager, for example, of ControlDesk, sorts I/O boards of the same type in ascending order of their PHS-bus addresses. The same ascending order applies to the board numbers (the number after the suffix B) of the corresponding I/O blocks in Simulink.

Note

- As the hardware design of DS4201 and DS4201-S boards is very similar, the Platform Manager cannot handle them as different board types; thus, you have to consider their PHS-bus addresses to find the correct board numbers for RTI's I/O blocks.
- To create a model without having the dSPACE system available, follow the guidelines in How to Create an Application Without a Connected dSPACE System on page 241 to find I/O block settings that match the actual hardware.

Example for a PHS-bus-based system

The following table demonstrates how the Platform Manager lists the different boards of a fictional PHS-bus-based system (e.g., DS1007 modular system), and which board numbers result for RTI's I/O blocks:

Board Type	Platform Manager Display	PHS-Bus Offset	Board Number (RTI)
DS2004	DS2004	0x00	1
DS2004	DS2004	0x20	2
DS2004	DS2004	0x40	3
DS4201	DS4201	0x10	1
DS4201-S	DS4201	0x60	2
DS4201-S	DS4201	0x70	3
DS4201	DS4201	0x80	4
DS5001	DS5001	0x30	1
DS5001	DS5001	0x50	2

Example for MicroAutoBox II

The following table demonstrates how the Platform Manager lists the different modules of a fictional MicroAutoBox II and which module numbers result for RTI's I/O blocks:

Module Type	Platform Manager Display	Module Position	Module Number (RTI)
CAN Type 1	CAN_TYPE1_M1/Ser/LIN	2	1
CAN Type 1	CAN_TYPE1_M2/Ser/LIN	3	2
ECU Type 1	ECU_TYPE1_M1	11	1
ECU Type 1	ECU_TYPE1_M2	12	2
ECU Type 1	ECU_TYPE1_M3	13	3
SER Type 1	IP_TYPE1_M1	7	1
SER Type 1	IP_TYPE1_M2	9	2

Related topics

Basics

Accessing PHS-bus-based systems in S-Functions..... 168

HowTos

How to Add I/O Blocks to Models..... 38
How to Create an Application Without a Connected dSPACE System..... 241

Vectorized Versus Scalar I/O Blocks

Kinds of I/O blocks

RTI provides two kinds of I/O blocks: vectorized and scalar. Both types have specific characteristics and require different handling. The following table lists the main differences:

	Scalar I/O Blocks	Vectorized I/O Blocks
Access of I/O device	Individual (scalar) I/O channels	Groups of (vectorized) I/O channels
Type of ports	One or more scalar ports	One or more vectorized ports
Multiple blocks	A single I/O device may be accessed via several instances of the same I/O block	A single I/O device can be accessed by only a single I/O block
Sample times	Individual channels may have different sample times	The elements of a vector always inherit the same sample time

For details on the different I/O blocks and their settings, refer to the corresponding reference information in dSPACE Help.

Scalar I/O blocks

Channel numbering The numbering of the different I/O channels is top down with respect to the I/O block.

Multiple blocks This feature of scalar I/O blocks can help you avoid long connection lines because you can place several copies of the same block wherever you need them in your model. However, you have to ensure that you do not access any channel more than once.

Note

When you copy an I/O block, Simulink might modify the board, channel, group number, etc. in order to keep the block names unique. This inherently modifies the settings of the copied block.

Vectorized I/O blocks

I/O access In the dialog of the I/O block you can specify which vector elements (channels) you want to activate. The number of enabled elements defines the vector width of the I/O block's port(s).

Sample times If you connect a vectorized I/O block to Simulink blocks that have different sample times, the vectorized I/O block inherits the fastest sample time. This might lead to illegal sample-rate transitions in some cases. If you get such an error message while compiling the model, you have to follow the instructions provided in Rate Transitions on page 70.

Related topics

Basics

Basics of Rate Transitions.....70

HowTos

How to Add I/O Blocks to Models.....38

Data Type Basics

Supported data types

Simulink supports the following basic data types:

Name	Description
double	Double-precision floating point
single	Single-precision floating point
half[1)]	16-bit floating point

Name	Description
int8	Signed 8-bit integer
uint8	Unsigned 8-bit integer
int16	Signed 16-bit integer
uint16	Unsigned 16-bit integer
int32	Signed 32-bit integer
uint32	Unsigned 32-bit integer
int64[2)]	Signed 64-bit integer
uint64[2)]	Unsigned 64-bit integer
Boolean	(0 or 1) represented internally by uint8 values

1) This data type can be used with RTI and RTI-MP. However, variables of this type are not generated into the variable description.

2) To enable this data type, you must set the **Support long long** option in the **Hardware Implementation** dialog, refer to Hardware Implementation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮).

RTI uses the following data types:

- Single-bit I/O blocks use the Boolean data type. Therefore, you can connect them directly to all logical operators.
- Byte I/O blocks use the uint8 data type.
- Encoder Index Search blocks use the int8 data type.
- DAC and ADC blocks use the double data type.

Note

There are a few restrictions concerning the use of data types. For details, refer to Data Type Restrictions on page 269.

Related topics

Basics

Data Type Restrictions 269

How to Add I/O Blocks to Models

Objective

To use the functionality of dSPACE I/O boards in your Simulink model, you must add RTI I/O blocks to it.

Example

Consider the throttle control example (refer to Throttle Control Example on page 16) and suppose you want to carry out rapid control prototyping. You have to replace the simulated throttle valve with the real one, which entails exchanging the corresponding Simulink blocks of your model with suitable I/O blocks. To run the real-time simulation of this model you have to connect the corresponding I/O to the real throttle valve. Some further adjustments might also be necessary in the model, for example, adding linearization, normalization and saturation.

If you have a MicroAutoBox II, the resulting Simulink model should be similar to the simplified model shown in the illustration below.

Basics

The blocks used for input and output in this example have different characteristics: The ADC block acting as input provides a single vectorized port. The output is done via a PWM generation block where the output frequency is constant but the duty cycle is varied.

Note

You must not break the library links of the RTI blocks. Otherwise, the blocks cannot be updated when you migrate your model to a new version of RTI and it is very likely that your model will not operate correctly under such a condition.

Method

To add I/O blocks to a model

1. In the **Library Browser**, expand the dSPACE I/O library that corresponds to the particular I/O unit you want to address.
2. Drag and drop the appropriate block from the I/O library to your Simulink model. You can place RTI's I/O blocks on the root level or in any subsystem.
3. Connect the block to your model as you would with any standard Simulink block and open the **Block Parameter** dialog.
4. Edit the settings of the block to meet your requirements: for example, the board number, channel number, and range. For details on the different I/O blocks and their settings, refer to the corresponding reference information in dSPACE Help.

Tip

- Most of RTI's I/O blocks apply automatic scaling to their Simulink ports. An input block provides floating-point values in the range -1 … +1. An output block requires values in the same range. You should make maximum use of this -1 … +1 range to exploit the full resolution of your converters.
- In order to optimize the performance of your application, RTI does not generate any code for unused channels. (This does not apply to several I/O blocks of RTI1401 where I/O code is not generated but implemented as S-functions.) If you leave unused channels open, Simulink produces a warning message by default. To avoid warnings due to unconnected ports, connect all unused ports to Ground or Terminator blocks from the Simulink library.

Result

You added I/O blocks to your model.

Related topics

Basics

Naming Conventions.....33
PHS-Bus Address and Board Number.....35
Vectorized Versus Scalar I/O Blocks.....36

References

Overview of RTI1006 (DS1006 RTI Reference 🕮)
Overview of RTI1007 (DS1007 RTI Reference 🕮)
Overview of RTI1104 (DS1104 RTI Reference 🕮)
Overview of RTI1202 (MicroLabBox RTI Reference 🕮)
Overview of RTI1401 (MicroAutoBox II RTI Reference 🕮)

Handling Tasks

Introduction

If you want to calculate parts of your Simulink model periodically or aperiodically, you need to create tasks in your model. With RTI, models commonly consist of one background task and several periodic and aperiodic real-time tasks.

Where to go from here

Information in this section

Introducing Task Handling.....42
Timer Tasks.....48
Tasks Driven by Interrupt Blocks.....57
Rate Transitions.....70
Assigning Parts of A Model to The Background Task.....81
Configuring Tasks.....86
Time-Triggered Tasks and Timetables.....99

Introducing Task Handling

Introduction

Your Simulink model usually consists of parts that have to be calculated periodically and others that have to be calculated at aperiodic points in time. Therefore, the real-time code is not executed as a whole, but Simulink Coder and RTI subdivide it according to the model structure and assign the parts to tasks. The tasks are scheduled individually so that they execute when necessary, without a fixed execution order. A request for the execution of a particular task is called an interrupt and can come from a timer device on the processor, an I/O device etc. Interrupts can be internal or external events, and they can be periodic or aperiodic. Each model has at least one timer task (executed with the base sample time) or a task driven by an interrupt block.

Where to go from here

Information in this section

Real-Time Kernel and Scheduler.....42
Priorities and Task-Switching Time.....43
Task States and Execution Order.....44
Overrun Situation and Turnaround Time.....45

Real-Time Kernel and Scheduler

Real-Time Kernel

RTI uses dSPACE Real-Time Kernel (RTK) to manage tasks, handle interrupts and schedule application's tasks efficiently. The RTK provides a priority-based preemptive scheduler that supports the rate-monotonic scheduling (RMS) strategy. By means of the priority of a task, the scheduler decides whether it should be started immediately or if it has to wait until a higher-priority task is completed. Hence, higher-priority tasks interrupt lower-priority tasks. Interrupt handling and scheduling cause a small time overhead.

Related topics

Basics

Overrun Situation and Turnaround Time.....45
Priorities and Task-Switching Time.....43
Task States and Execution Order.....44

Priorities and Task-Switching Time

Priorities

The priorities of the tasks are stored in variables of the real-time application and can be displayed in ControlDesk.

A background task is not a task in the common sense, meaning that it has no priority.

Task-switching time

A crucial value in task scheduling is the task-switching time, which defines the delay between the occurrence of an interrupt on the hardware and the execution of the corresponding task's first statement. In the example mentioned below, this is the time that passes until the control unit responds to a person pressing the emergency stop button.

Example

Consider the following situation in the throttle control example (refer to Throttle Control Example on page 16): Someone presses the emergency stop button. Thus, the corresponding task of the control unit that closes the throttle valve must have the highest priority. All other tasks of the control unit have to wait. When the valve is closed, the suspended tasks resume execution.

Therefore, if the execution request for the high-priority task occurs, the scheduler of the control unit has to carry out the following steps:

- Interrupt (preempt) the currently running (lower-priority) task.
- Execute the requested high-priority task.
- When the high-priority task is finished, resume the interrupted (lower-priority) task.

Values of task-switching times

In a multiple-task environment, a triggered task has the following task-switching times:

Board	Best Case	Worst Case
DS1006 @ 2.6 GHz	2.13 µs	4.26 µs
DS1006 @ 2.8 GHz	1.62 µs	3.23 µs
DS1007 @ 2.0 GHz	1.25 µs	2.38 µs
DS1104 @ 250 MHz	2.4 µs	4.8 µs
MicroLabBox @ 2.0 GHz	1.39 µs	2.78 µs

Note

- The stated task-switching times apply only if the involved routines are already located in the cache. Otherwise, the task-switching times can be significantly longer.
- The values in the above table do only apply for tasks that are triggered by the Base Rate Timer. Latency times for tasks triggered by other interrupt sources can be significantly longer.

Related topics

Basics

Overrun Situation and Turnaround Time.....45
Real-Time Kernel and Scheduler.....42
Task States and Execution Order.....44

References

priority (RTI and RTI-MP Implementation Reference 🕮)

Task States and Execution Order

Assigning tasks

In order to identify which task is active, which one is suspended, and so on, the scheduler assigns a state to each task as shown in the following illustration:

The dSPACE Real-Time Kernel uses the definitions listed in the table below:

State	Value	Meaning
Idle	0	The task is inactive, waiting to be triggered.
Ready	1	The task has been triggered but could not start due to a high-priority task that is currently running. It is waiting to start the execution.
Running	2	The task has started running. This state is true until the task finishes running, regardless of whether it is suspended by a high-priority task or not.

The states of the tasks are stored in variables of the real-time application and can be displayed in ControlDesk.

Executing tasks

Depending on the priorities and current states of the tasks, the scheduler executes them according to the following rules:

- A high-priority task that is triggered always suspends a low-priority task that is currently running.

- If no high-priority task is triggered, the suspended low-priority task resumes execution.
- Tasks of the same priority do not suspend each other if they are triggered, but follow a first come, first served policy.
- As long as all other tasks are idle, the background task is executed.

Related topics

Basics

Overrun Situation and Turnaround Time.....45
Priorities and Task-Switching Time.....43
Real-Time Kernel and Scheduler.....42

References

RTI and RTI-MP Variable Reference (RTI and RTI-MP Implementation Reference 🕮)
state (RTI and RTI-MP Implementation Reference 🕮)

Overrun Situation and Turnaround Time

Overrun situation

An overrun situation occurs if a task is requested to start but has not finished its previous execution yet. Consider the situation shown below:

Between the first and the second timer interrupt there is no overrun, but if the hardware-interrupt-block-driven task T_{HW} needs to be calculated, there is not enough time for the timer task T_{Timer} to finish until it is requested to start again. If such an overrun situation occurs for a task, the scheduler detects it, and RTI responds according to the overrun settings for the task.

Avoiding overrun situations

To avoid overrun situations you have to consider the time values listed below. For each task of your model, ensure that its sample time is greater than the sum of these time values.

- Depending on the dSPACE board used, you have to take into account a different task-switching time (see Priorities and Task-Switching Time on page 43).

- The turnaround time for a task is the time that passes between the triggering and the end of its execution. It can include the time required by higher-priority tasks that interrupt it.
- Conditional execution paths in the model code can cause the turnaround time of a task to vary from one execution step to another. For example, if you run your model in the multiple timer task mode, the fastest task actually contains only blocks that are executed with this sample time. Nevertheless the turnaround time of this task might have periodical peaks occurring with the period of a slower task, which result from the interaction with the slower timer tasks.

  In the real-time application, you can measure the turnaround time of each task via ControlDesk. To get a continuous display of the turnaround time, you can use a plotter instrument.

Example

The illustration below shows the turnaround times for a number of tasks.

In this example, the *Timer interrupt* task requires always the same time to complete. Due to the varying interrupts, the *Timer* task requires different times to complete. If it is not interrupted, it requires the time *t1* to complete, otherwise it requires *t1* + *t2* (*t2* = time to finish the *Timer interrupt* task).

Tip

You can configure the behavior towards overrun situations for each task individually, see How to Specify Overrun Strategies for Tasks on page 90.

Related topics

Basics

Priorities and Task-Switching Time 43
Real-Time Kernel and Scheduler 42
Task States and Execution Order 44
Timer Tasks 48

HowTos

How to Change the Task Priorities 89
How to Specify Overrun Strategies for Tasks 90

Examples

Example Applications Requiring the Multiple Timer Task Mode 53

References

turnaroundTime (RTI and RTI-MP Implementation Reference 🕮)

Timer Tasks

Introduction

If a model part is not driven by an interrupt block, it is part of a timer task by default. Your model can contain several timer tasks. RTI provides two execution modes for the timer tasks in your model.

Where to go from here

Information in this section

Timer Task Basics....48
Single Timer Task Mode....50
Multiple Timer Task Mode....51
Example Applications Requiring the Multiple Timer Task Mode....53
Comparing the Execution Modes....54
How to Connect Timer Tasks to an External Interrupt Source....55

Information in other sections

Interprocessor Interrupts....143
Tasks Driven by Interrupt Blocks....57
How to Change the Task Priorities....89
How to Change the Task Priorities in RTI-MP Models....145

Timer Task Basics

Sample time

If a model part is not driven by an interrupt block, it is part of a timer task by default. The model code for timer tasks is bound implicitly to the timer interrupt source of the processor.

Usually your model consists of blocks that have different sample times. The sample time of a block indicates when the block produces outputs or updates its internal state during simulation.

Some Simulink blocks allow you to explicitly specify a sample time or switch to sample-based operation: for example, the Sine Wave block. If you specify a sample time or switch to sample-based operation, this block also changes its symbol to indicate that it has become a discrete block.

A block for which you specified an inherited (-1) sample time, inherits its sample time from its context, for example from the blocks that are connected to it. The sample time of such a block is determined when the model is initialized. This process is called *Simulink's Sample Time Propagation*.

Single-rate and multirate models

In general, one or more blocks in the model require periodic execution at the base sampling rate (model fixed step size). All these blocks are grouped in the first timer task, which is bound to a timer interrupt. If the model also contains at least one time-discrete block with a sampling rate different from the base sampling rate, the model is multirate (otherwise it is *single-rate*). The different sampling rates of a multirate model can be calculated in one single timer task or in several timer tasks – one for each sampling rate. The following formula applies to all sampling rates:

sampling rate = 1/n · base sampling rate

where

n	=	integer multiple
base sampling rate	=	1/fixed step size of model

Example

Consider the throttle control example (refer to Throttle Control Example on page 16). To measure the flow rate of the throttle valve at a slow sample time, you have to alter the model as shown in the next illustration. The upper part operates with the base sample time. The lower part that monitors the flow rate operates at a slow sample time of just one sample per second.

Single Timer Task Mode

Single timer task mode

The single timer task mode is the default execution mode. RTI places all calculations performed in timer tasks in one single timer task. This is executed with the fastest sample time of your model, that is, the base sample time.

If your model is a multirate system, a sample hit control mechanism ensures that the discrete blocks are executed at their specified sampling steps. The example below demonstrates this case.

Example Two discrete blocks (B_{Fast} and B_{Slow}) are executed in a single timer task T_{Timer} with the block B_{Slow} being executed only every second step:

Limitations of the single timer task mode

There is no preemption mechanism for the timer tasks of the model in this execution mode. Therefore, the base sample time must be long enough to allow the execution of all blocks within one interval. If this is not possible because you require a smaller base sample time, you get an overrun situation. This is illustrated in the next example.

Example If the base sample time is too small, an overrun situation occurs:

Tip

You can configure the behavior towards overrun situations for each task individually, see How to Specify Overrun Strategies for Tasks on page 90.

Related topics

Basics

Comparing the Execution Modes.....54
Multiple Timer Task Mode.....51

Overrun Situation and Turnaround Time.....45

HowTos

How to Specify Overrun Strategies for Tasks.....90

References

CPU's Page (Multiprocessor Setup Dialog) (RTI and RTI-MP Implementation Reference 🕮)
Solver Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮)

Multiple Timer Task Mode

Multiple timer task mode

RTI creates several timer tasks, one for each sample time/sample time offset of your model. Therefore, the execution of a slow block can be interrupted (preempted) for the execution of a fast (higher-priority) block.

Example If block B_{Fast} is assigned to task T_{Timer1} and block B_{Slow} to task T_{Timer2}, the execution sequence in the multiple timer task mode runs as shown below:

After T_{Timer1} has finished, T_{Timer2} starts. The next time T_{Timer1} is ready for execution, T_{Timer2} is interrupted until task T_{Timer1} has finished. T_{Timer2} then resumes its execution.

Sample time offset

If you use sample time offsets, each offset is treated as a new task: The smaller the sample time offset, the higher the priority of a task. For details, refer to *Using Simulink* by MathWorks®.

Advantages of the multiple timer task mode

As long as all tasks can be executed within the time interval given by the base sample time, the model behaves in a manner similar to the execution in the single timer task mode.

In the multiple timer task mode, RTI manages the priorities of the different timer tasks according to a rate-monotonic scheme: The faster the sample time of a task, the higher its priority.

Note

Because there are no true aperiodic interrupts in a Simulink simulation, Simulink and Simulink Coder use the terms SingleTasking and MultiTasking instead of single timer task mode and multiple timer task mode. However, with dSPACE real-time programs you can place all calculations performed in timer tasks in a single timer task and still have additional tasks triggered by hardware or software interrupt blocks. Therefore, RTI uses the terms single timer task mode and multiple timer task mode – which both allow aperiodic interrupts.
To achieve data consistency in multiple timer task mode, you have to insert a Simulink **Rate Transition** block into the connection of two blocks executed with different sample times. For further information, refer to Rate Transitions on page 70.

Restricted execution mode

In the multiple timer task mode, RTI only supports the *real-time* execution mode. The *time-scaled* and *as fast as possible* execution modes are not supported. For more information, refer to Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮).

Related topics

Basics

Comparing the Execution Modes.....54
Overrun Situation and Turnaround Time.....45
Rate Transitions.....70
Single Timer Task Mode.....50

HowTos

How to Change the Task Priorities.....89

Examples

Example Applications Requiring the Multiple Timer Task Mode.....53

References

CPU's Page (Multiprocessor Setup Dialog) (RTI and RTI-MP Implementation Reference 🕮)
Solver Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮)

Example Applications Requiring the Multiple Timer Task Mode

Fast Fourier Transform (FFT)

The multiple timer task mode is required if you want to use the Signal Processing Blockset to run an FFT in addition to regular continuous or discrete blocks. The FFT is not started until the required data (for example, 1024 points) is available, and it does not finish in just one interval, as illustrated below:

Aperiodic interrupt

Another example where the multiple timer task mode can avoid overrun situations is illustrated below. Here you can see an aperiodic interrupt in a multirate application that delays the execution of the timer task so that it cannot finish before the next timer interrupt occurs:

If you choose the multiple timer task mode for this model, the fast block B_{Fast} can interrupt the slow block B_{Slow} so that no task is required to start executing while it is still running. Therefore, there is no overrun situation, which is illustrated below:

Related topics

Basics

Multiple Timer Task Mode.....51

Comparing the Execution Modes

Execution mode comparison

The following table gives an overview of the advantages and disadvantages of both execution modes:

	Single Timer Task Mode	**Multiple Timer Task Mode**
Execution Time Overhead	Smaller	Bigger
Target Application	▪ Single-rate ▪ Multirate with a predominating timer task that has a short sample time	▪ Multirate with a predominating timer task that has a long sample time ▪ Multirate with a predominating task driven by an interrupt block
Conclusion	Standard applications	Solving specific overrun problems

Related topics

Basics

Multiple Timer Task Mode.....51
Single Timer Task Mode.....50

How to Connect Timer Tasks to an External Interrupt Source

Using an Interrupt block

By default, the timer tasks of a model are connected to a timer interrupt of the dSPACE hardware. As a result the model is executed at fixed time steps. However, sometimes you might want to compute the model synchronously to external events, for example, an external timer device. Therefore, you have to connect the timer task(s) to an external interrupt source. This is possible via the Timer Task Assignment block.

Note

The computations of the timer task are executed as normal using the time value of the simulation step size, as specified in the Solver dialog of the Model Explorer or in the Multiprocessor Setup dialog. However, the points in time when the computations are performed depend on the external trigger events. RTI cannot check whether the simulation step size of the model and the step size of the external trigger are equal. You must ensure identical step sizes yourself. Otherwise, the model is not calculated in real time.

Method

To connect timer task(s) to an external interrupt source

1. Place the desired hardware interrupt block from the appropriate I/O library on the root level of your Simulink model.

2. Open the Block Parameters dialog, and specify the settings you require, for example, the interrupt type. If you use a PHS-bus-based system, you also have to specify the correct board number.
3. Place the Timer Task Assignment block from RTI's TaskLib on the root level of your model.

4. Connect the output port of the hardware interrupt block to the input port of the Timer Task Assignment block.
5. Open the Solver dialog in the Model Explorer or the Multiprocessor Setup dialog, and make sure that the models Fixed Step Size or Basic Step Size and Step Size Multiple match the period of the external interrupt source.

Example

For a demo of this task type, see the Timer Task Assignment demo model, which you can access via the RTI block library – Demos – TaskLib Demos.

Related topics

References

Hardware Interrupt Block (RTI and RTI-MP Implementation Reference 🕮)
Timer Task Assignment Block (RTI and RTI-MP Implementation Reference 🕮)

Tasks Driven by Interrupt Blocks

Introduction

RTI provides interrupt blocks which can be used to calculate parts of your model at asynchronous points in time, for example if an external trigger event occurs.

Where to go from here

Information in this section

Basics on Triggered Subsystems.....57
Preparing a Function-Call Subsystem.....60
Implementing Interrupt-Block-Driven Tasks.....62

Information in other sections

Interprocessor Interrupts.....143
Timer Tasks.....48
How to Change the Task Priorities.....89
How to Change the Task Priorities in RTI-MP Models.....145

Basics on Triggered Subsystems

Triggered Subsystems

Introduction

To drive a part of the model by a hardware or software interrupt block, the part has to be placed in a specially triggered subsystem. Simulink provides several types of triggered subsystems:

- Subsystem without Trigger or Enable block
- Subsystem with a Simulink Enable block
- Subsystem with a Simulink Trigger block

Subsystem without Trigger or Enable block

This simply helps to structure your model layout; for instance, to group parts that logically belong together, such as a controller. If declared atomic, a subsystem also influences the block execution order and thus the generated real-time code.

For details, refer to Optimizing the Build Process and Simulation via Atomic Subsystems on page 246.

If you add an Enable or Trigger block to a subsystem it becomes a conditionally executed subsystem as shown in the following paragraphs.

Subsystem with a Simulink Enable block

This executes whenever the enable signal is greater than zero. This type of conditionally executed subsystem does not form a separate task in the real-time program but is part of its parent subsystem's task. The enable port of the subsystem requires a Simulink signal and cannot be used with hardware or software interrupts.

Subsystem with a Simulink Trigger block

This executes once whenever an appropriate trigger signal is applied to the trigger port of the subsystem.

- If the **Trigger type** is set to *rising*, *falling*, or *either*, the trigger port of the subsystem requires a Simulink signal and cannot be used with hardware or software interrupts. This type of conditionally executed subsystem does not form a separate task in the real-time program but is part of its parent subsystem's task.

- If the **Trigger type** is set to *function-call*, the trigger port of the subsystem accepts only function-call connections (for example, from RTI's Interrupt blocks, or the Simulink Function-Call Generator block) and no standard Simulink signals. Throughout this RTI and RTI-MP Implementation Guide, this is referred to as a function-call subsystem.

  If the trigger port is connected to a Timer, Hardware, or Software Interrupt block, the function-call subsystem forms a separate task in the real-time program. Therefore, a typical application of this trigger type is for building tasks driven by interrupt blocks. If the trigger port is connected to RTI's Background block, the subsystem is assigned to the background task.

Note

The Trigger block provides the following options:

- States when enabling
- Sample time type
- Sample time

These options have no impact on function-call subsystems that are driven by RTI interrupt blocks. In detail this means that:

- The States when enabling option has no effect. Independently of its setting (*inherit*, *held*, or *reset*) the states of a function-call subsystem triggered by an RTI interrupt block will never be reset when an interrupt occurs. This corresponds to the *held* behavior and is the same as in previous releases.
- The Sample time type option must always be set to *triggered* (the Sample time option is then disabled). Otherwise, you get the following error message from Simulink:

  "Block 'test/Function-Call Subsystem1/function' has a discrete sample time. Only constant blocks or blocks with an inherited (-1) sample time are allowed in triggered subsystems."

Tip

You can also use the function-call subsystem from the Simulink library or from RTI's TaskLib.

Note

- Suppose your model contains timer tasks and interrupt-driven tasks. Time `t = 0` is defined as the point in time when the first timer task (Timer Task 1) is executed. Interrupt-driven tasks which were started by hardware interrupts before this point in time then also read time 0. This also applies to triggered subsystems which are connected to an RTI Background block.
- Numeric errors previously caused by interrupt-driven tasks reading the time variable value of Timer Task 1 no longer occur.

Related topics

Basics

Optimizing the Build Process and Simulation via Atomic Subsystems..... 246

References

Function-Call Subsystem Block (RTI and RTI-MP Implementation Reference 🕮)

Preparing a Function-Call Subsystem

Introduction

If you want to drive a particular group of blocks by an interrupt block or assign it to the background task, the blocks must be placed in a function-call subsystem. Data transfer to/from other tasks must be implemented via particular rate transitions.

Where to go from here

Information in this section

Introduction to Interrupt-Block-Driven Tasks and Task Transitions..... 60

How to Prepare Function-Call Subsystems..... 61

Introduction to Interrupt-Block-Driven Tasks and Task Transitions

Interrupt-Driven Tasks

If you want to drive a particular group of blocks by an interrupt block, these blocks must be placed inside a function-call subsystem and the subsystem must be connected to the interrupt source. A function-call subsystem must not be located inside another function-call subsystem.

Note

RTI lets you work with the enhanced time and elapsed time support in Simulink Coder. RTI supports Simulink Coder time counter variables for all interrupt-driven tasks. RTI assigns values which are derived from the dSPACE Real-Time Kernel or the Time Stamping module to these variables. This ensures accurate time information with high resolution for interrupt-driven tasks.

Data transfer problems

Since separate tasks can interrupt each other, the transfer of data vectors between them can also be interrupted, producing inconsistent data unless the data transfer is buffered.

To handle task transitions in your application, you have to use the Simulink Rate Transition block. For further information, refer to Rate Transitions on page 70.

Example

For a demonstration on how to model valid task transitions, refer to the Task Transition demo model, which you can access via the RTI block library – Demos – TaskLib Demos.

Related topics

Basics

Rate Transitions.....70

How to Prepare Function-Call Subsystems

Objective

All the blocks that are to be driven by an interrupt block or the Background block must be located in a function-call subsystem.

Method

To prepare a function-call subsystem

1 Place the function-call subsystem from RTI's TaskLib inside your Simulink model.

Note

A function-call subsystem must not be located inside another function-call subsystem.

2 Open the function-call subsystem, and add the blocks you want to drive with the interrupt or assign to the background task.

Example

The following illustration shows what your model could look like:

Related topics

Basics

Introduction to Interrupt-Block-Driven Tasks and Task Transitions..... 60

References

Function-Call Subsystem Block (RTI and RTI-MP Implementation Reference 🕮)

Implementing Interrupt-Block-Driven Tasks

Introduction

Interrupt-block-driven tasks can be driven by hardware interrupts or software interrupts. Hardware interrupts are generated by designated hardware or timer interrupt devices of the dSPACE boards, for example, if a specific input signal occurs. Software interrupts are generated within the model itself, for example, if a Simulink signal exceeds a specific value.

Where to go from here

Information in this section

How to Use Hardware Interrupts..... 62

How to Use Software Interrupts..... 65

How to Branch Interrupt Signals..... 67

Example of a Task Driven by a Hardware Interrupt Block..... 68

How to Use Hardware Interrupts

Hardware interrupts

RTI provides hardware interrupt blocks for the different dSPACE boards. There are various types of hardware interrupts depending on the type of board you consider, for example, external interrupts used for the emergency stop button in the above-mentioned example. For details on the different hardware interrupts, refer to the corresponding reference material.

Precondition

A function-call subsystem must be prepared. For instructions, refer to How to Prepare Function-Call Subsystems on page 61.

Method

To use hardware interrupts

1 Place the **Hardware interrupt** block from the appropriate I/O library in your Simulink model.

This block is the trigger source for the function-call subsystem.

> **Note**
>
> Do not place this block inside a triggered subsystem or an enabled subsystem. It always activates its function-call subsystem regardless of whether its parent subsystem is active or not.

2 Open the **Block Parameters** dialog, and specify the settings you require, for example, the interrupt type.

If you use a PHS-bus-based system, you also have to specify the correct board number.

3 Connect the outport of the trigger source to the trigger port of the function-call subsystem.

> **Note**
>
> Do not connect multiple function-call subsystems to the same trigger source.

You may include the following blocks in the connection: **Mux/Demux**, **Inport/Outport**, and **From/Goto**.

> **Note**
>
> RTI-MP
> To transfer an interrupt from one CPU to another, you have to connect the trigger source and the function-call subsystem via an **IPI** block. No other blocks are allowed for this connection. The IPI block must list all target CPUs involved in the execution of the function-call subsystem. However, for local interrupts you do not need to include the IPI block in the connection. For details on IPI blocks, refer to How to Implement Interprocessor Interrupts on page 143.

> **Tip**
>
> For multiprocessor systems, you can also assign different parts within the subsystem to different CPUs: Just add the required **IPC** blocks and/or **Default CPU** blocks. For details, refer to Interprocessor Communication Using IPC Blocks on page 130 and How to Define a Default CPU for Unconnected Model Parts on page 136).

Example

For a demo of this task type, see the **Hardware and Software Interrupt** demo model, which you can access via the RTI block library – Demos – TaskLib Demos.

Hardware interrupts from a timer device

Each dSPACE controller/processor board provides a number of timer devices. By default, RTI implicitly uses one of these to trigger the timer task(s) of the model.

Via the **Timer Interrupt** block from RTI's TaskLib, you can use the free timer devices of the controller/processor board to periodically trigger subsystems in your model. The resulting tasks are not synchronized to the fixed step size of the model and are not tied to the rate-monotonic priority scheme of the model. For example, you can create a periodic task with a very slow sampling rate but a priority higher than the fastest timer task, or create a task with a sampling rate that is not a multiple of the fixed step size.

Note

Tasks driven by the **Timer Interrupt** block are executed periodically but behave like tasks driven by interrupt blocks.

Related topics

Basics

Preparing a Function-Call Subsystem..... 60

HowTos

How to Change the Task Priorities..... 89
How to Connect Timer Tasks to an External Interrupt Source..... 55
How to Implement Interprocessor Interrupts..... 143
How to Prepare Function-Call Subsystems..... 61
How to Use Software Interrupts..... 65

Examples

Example of a Task Driven by a Hardware Interrupt Block..... 68

References

Function-Call Subsystem Block (RTI and RTI-MP Implementation Reference 🕮)
Hardware Interrupt Block (RTI and RTI-MP Implementation Reference 🕮)
IPI (RTI and RTI-MP Implementation Reference 🕮)
Timer Interrupt Block (RTI and RTI-MP Implementation Reference 🕮)

How to Use Software Interrupts

Software interrupts

In some cases you might want to trigger the execution of a task by means of a Simulink signal that does not have the function-call type. However, it is not possible to connect a non function-call Simulink signal to the trigger port of a function-call subsystem. RTI provides the Software Interrupt block, which generates an interrupt signal depending on the enable/trigger signals and lets you achieve very flexible interrupt handling.

Precondition

A function-call subsystem must be prepared. For instructions, refer to How to Prepare Function-Call Subsystems on page 61.

Method

To use software interrupts

1 Place the **Software Interrupt** block from RTI's TaskLib in your Simulink model.

> **Tip**
>
> You can place this block inside any subsystem, for single-processor systems also inside triggered and enabled subsystems. However, the subsystem may not be connected to a Background Task block or triggered from within a background task subsystem.

2 Open the **Block Parameters** dialog of the Software Interrupt block, and select the desired trigger/enable port.

3 Connect the Simulink signal(s) you want to use as the trigger/enable to the trigger/enable port(s) of the **Software Interrupt** block.

4 Connect the outport of the trigger source to the trigger port of the function-call subsystem.

> **Note**
>
> Do not connect multiple function-call subsystems to the same trigger source.

You may include the following blocks in the connection: **Mux/Demux**, **Inport/Outport**, and **From/Goto**.

Note

RTI-MP
To transfer an interrupt from one CPU to another, you have to connect the trigger source and the function-call subsystem via an **IPI** block. No other blocks are allowed for this connection. The IPI block must list all target CPUs involved in the execution of the function-call subsystem. However, for local interrupts you do not need to include the IPI block in the connection. For details on IPI blocks, refer to How to Implement Interprocessor Interrupts on page 143.

Tip

For multiprocessor systems, you can also assign different parts within the subsystem to different CPUs: Just add the required **IPC** blocks and/or **Default CPU** blocks. For details, refer to Interprocessor Communication Using IPC Blocks on page 130 and How to Define a Default CPU for Unconnected Model Parts on page 136).

Example

For a demo of this task type, see the **Hardware and Software Interrupt** demo model, which you can access via the RTI block library – Demos – TaskLib Demos.

Related topics

Basics

Preparing a Function-Call Subsystem 60

HowTos

How to Change the Task Priorities 89
How to Implement Interprocessor Interrupts 143
How to Prepare Function-Call Subsystems 61
How to Use Hardware Interrupts 62

References

Function-Call Subsystem Block (RTI and RTI-MP Implementation Reference 🕮)
IPI (RTI and RTI-MP Implementation Reference 🕮)
Software Interrupt Block (RTI and RTI-MP Implementation Reference 🕮)

How to Branch Interrupt Signals

Objective

In some cases you might want to trigger multiple function-call subsystems by means of one interrupt block. The Simulink **Function-Call Split** block is used to branch the outport of an interrupt block and connect it to multiple function-call subsystems or models.

Precondition

The function-call subsystems which you want to trigger must be prepared. For instructions, refer to How to Prepare Function-Call Subsystems on page 61.

Restrictions

With RTI-MP, the **Function-Call Split** block does not have to be inserted before or after an **IPI** block. For details, refer to How to Implement Interprocessor Interrupts on page 143.

For further information on this limitation, refer to Limitations with RTI-MP on page 276.

Method

To branch an interrupt signal

1. Place the **Function-Call Split** block from Simulink's **Ports & Subsystems** library in your Simulink model.
2. Open the **Block Parameters** dialog of the block and configure it.
3. Connect the trigger source (for example, an outport of an interrupt block) to the inport of the block.
4. Connect the outports of the **Function-Call Split** block to the trigger ports of the function-call subsystems.

Example

The following illustration shows how you can branch the signal from a **Software Interrupt** block to two function-call subsystems. The outport that is marked with a bullet will be triggered first.

Related topics

Basics

Preparing a Function-Call Subsystem..... 60

HowTos

How to Change the Task Priorities..... 89
How to Implement Interprocessor Interrupts..... 143
How to Prepare Function-Call Subsystems..... 61
How to Use Hardware Interrupts..... 62

References

Function-Call Subsystem Block (RTI and RTI-MP Implementation Reference 🕮)
IPI (RTI and RTI-MP Implementation Reference 🕮)
Software Interrupt Block (RTI and RTI-MP Implementation Reference 🕮)

Example of a Task Driven by a Hardware Interrupt Block

Generating an interrupt signal

Consider the throttle control example (refer to Throttle Control Example on page 16). To close the throttle valve when the emergency stop button is operated, the emergency stop button has to generate a hardware-interrupt signal. The following illustration shows the necessary changes:

However, the interrupt signal does not need to come from the hardware. If you want to generate an interrupt signal by the Simulink model itself, you can use a software interrupt.

Note

In the example shown above, the hardware interrupt sets just the output of the function-call subsystem to zero. However, the Switch and DIO_TYPE3_PWM_BL1 blocks are processed in the timer task, which delays the output signal until the next execution step of the timer task.

Related topics

Basics

Preparing a Function-Call Subsystem..... 60

HowTos

How to Use Hardware Interrupts..... 62

Rate Transitions

Introduction

If your model provides tasks with different sample times, the consistency of the data exchanged between the tasks is not guaranteed. To achieve data consistency, you have to implement rate transitions.

Where to go from here

Information in this section

Basics of Rate Transitions.....70
How to Implement Sample-Rate Transitions.....75
How to Implement Protected Task Transitions.....75
How to Implement Unprotected Task Transitions.....77
How to Implement Task Transitions for Multiple Receiving Blocks 79

Basics of Rate Transitions

Introduction

To achieve data consistency in multiple timer task mode, you have to insert a Simulink **Rate Transition** block into the connection of two blocks executed with different sample times.

Sample-rate transitions

In the multiple timer task mode, each sample time forms a separate timer task and these timer tasks can interrupt each other. If data vectors were transferred directly between blocks with different sample times (called sample-rate transition), data inconsistency could result.

Example Suppose a timer task with a fast sample time (T_{Fast}) provides the data for a timer task with a slow sample time (T_{Slow}). T_{Fast} might be executed while T_{Slow} is busy reading the data vector – marked with (1) in the following illustration. Thus, T_{Slow} is interrupted and partly reads old data (2) and new data (3) when it resumes its execution. A similar problem arises if a slow timer task provides data for a fast one. Both of these cases of data inconsistency are illustrated below:

Achieving data consistency

Because of this data inconsistency you cannot connect blocks that are executed with different sample times directly. Instead, you must insert the **Rate Transition** block from the Simulink library. This block is executed with the priority of the faster block, but with the sample time of the slower block. It automatically acts as a *Fast to slow* or *Slow to fast* data rate transition depending on the sample rates of the sending and receiving task involved, respectively:

- If the block acts as a *Fast to slow* data rate transition, the block is executed after the fast block and before the slow block – marked with a (1) in the following illustration – thereby providing consistent data for the slow block (2).
- If the block acts as a *Slow to fast* data rate transition, the block is executed after the slow block and before the fast block (3), thereby providing consistent data for the fast block (4).

The block is valid for all kinds of task transitions including periodic to periodic, asynchronous to asynchronous and periodic to asynchronous task transitions.

Example The illustration below demonstrates the data transfer for both *Fast to slow* (f2s) and *Slow to fast* (s2f) data rate transitions:

Simulink Rate Transition block options

After inserting a Simulink Rate Transition block to your model, you have to configure it. The block's most important setting is the Ensure data integrity during data transfer option.

If selected, the data transfer between sending and receiving task is buffered and data consistency is ensured (protected data transfer).

If cleared, data transfer is non-buffered (unprotected data transfer). This leads to potentially inconsistent data in the receiving task.

For further details on the block options, refer to the Simulink documentation.

Invalid task transitions

Because you need to declare the type of data transfer for each task transition explicitly, Simulink can detect the task transitions that you might have unintentionally left undeclared. Therefore, you do not get an unsafe or problematic model design by mistake.

Note

If your model contains invalid task transitions, for example, a direct data connection between blocks located in different asynchronously executed function-call subsystems, Simulink issues an error message when you attempt to simulate or build the model. Make sure that you use Simulink's Rate Transition block to implement valid task transitions.

Unprotected data transfer

If you select the **Ensure data integrity during data transfer** option and the **Rate Transition** block display shows *Copy*, the data transfer is unprotected.

The following illustration shows the **Rate Transition** block when unprotected data transfer is implemented between two interrupt-driven tasks:

The data transfer might behave as shown in the following illustration:

The high-priority task (T_{Fast}) provides data for the low-priority task (T_{Slow}). If T_{Fast} is triggered (1) while T_{Slow} is busy reading a part of T_{Fast}'s last data vector (2), T_{Slow} is interrupted and partly reads old data – it reads new data when it resumes its execution (3).

Protected data transfer

If you select the **Ensure data integrity during data transfer** option the data transfer is protected. The method, for example, *Db_buf*, is automatically set by Simulink. For further information, refer to the Simulink documentation.

The following illustration shows the Simulink **Rate Transition** block when protected data transfer is implemented between two interrupt-driven tasks:

The data transfer might behave as shown in the following illustration:

If T_{Fast} is triggered (1) while T_{Slow} reads the old data (3) from buffer A, it writes the new data in buffer B (2). When T_{Slow} resumes execution after T_{Fast} has finished (5), it continues reading the old data from buffer A (4). The new data of T_{Fast} is not included in this execution of T_{Slow}, but the next time T_{Slow} is executed, it reads the new data from buffer B (6).

Limitation of task transitions

Each task transition can have receivers in only one task. Therefore, if you need multiple receivers in different tasks, the signal must be branched first and a valid task transition, for example, a Simulink **Rate Transition** block, must be implemented for each branch.

An alternative to using Rate Transition blocks

As an alternative to using Simulink's **Rate Transition** block, you can select the **Automatically handle rate transition for data transfer** option in the **Model Configuration Parameters - Solver** dialog. For details, refer to Solver Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮).

Example

For a demonstration on how to model valid task transitions, refer to the **Task Transition** demo model, which you can access via the RTI block library – **Demos – TaskLib Demos**.

Related topics

Basics

Multiple Timer Task Mode 51

HowTos

How to Change the Task Priorities 89
How to Change the Task Priorities in RTI-MP Models 145
How to Implement Unprotected Task Transitions 77

How to Implement Sample-Rate Transitions

Objective

If you want data to be transferred directly between timer tasks (i.e. blocks in your model that operate at different rates in multiple timer task mode), you must implement valid sample-rate-transitions. You can implement sample-rate transitions in protected or unprotected mode for data transfer.

Detecting sample rate transitions

Via the Format – Sample Time Display menu command, you can make Simulink display blocks and signal lines in different colors for the different sample times. As a result, you can easily see where two adjacent blocks have different colors, indicating a sampling rate transition. You should add the Rate Transition block at this position.

Method

To implement sample-rate transitions

1 Insert the Rate Transition block from the Simulink library into the connection of the two blocks.

2 Open the Block Parameters dialog and specify the parameters as required.

Result

You implemented a valid sample-rate transition.

Related topics

Basics

Multiple Timer Task Mode.....51

HowTos

How to Change the Task Priorities.....89
How to Change the Task Priorities in RTI-MP Models.....145
How to Implement Unprotected Task Transitions.....77

How to Implement Protected Task Transitions

Objective

To achieve data consistency between interrupt driven tasks or between an interrupt driven task and a timer task, you must implement protected task transitions in your model. They ensure that the transferred data read in the

receiving task is always consistent, even if the reading task is interrupted while data is read.

Method

To implement a protected task transition

1 Place a Simulink **Rate Transition** block directly before the function-call subsystem if the subsystem receives data. If the function-call subsystem is sending data, connect the Simulink **Rate Transition** block directly to the function-call subsystem's Output block.

2 Open the **Block Parameter** dialog and make sure that the **Ensure data integrity during data transfer** option is selected.

Example

The following illustrations show how you can implement protected task transitions:

- Protected data transfer between a block that is executed in the timer task and a block that belongs to an asynchronous task:

- Protected data transfer between blocks that belong to different asynchronous tasks:

Related topics

Basics

Basics of Rate Transitions 70
Introduction to Interrupt-Block-Driven Tasks and Task Transitions 60

HowTos

How to Implement Task Transitions for Multiple Receiving Blocks 79
How to Implement Unprotected Task Transitions 77

How to Implement Unprotected Task Transitions

Objective

If you want the data between tasks to be transferred as soon as possible, but at the potential loss of data integrity, you must implement unprotected task transitions in your model.

Method

To implement an unprotected task transition

1. Place a Simulink **Rate Transition** block directly before the function-call subsystem if the subsystem receives data. If the function-call subsystem is sending data, connect the Simulink **Rate Transition** block directly to the function-call subsystem's Output block.

2 Open the **Block Parameter** dialog and make sure that the **Ensure data integrity during data transfer** option is deselected.

> **Note**
>
> If the sending and the receiving tasks do not have the same priority and the lower-priority task is interrupted by the higher-priority task, a data inconsistency can occur.

Example

The following illustrations show how you can implement unprotected task transitions:

- Unprotected data transfer between a block that is executed in the timer task and a block that belongs to an asynchronous task:

- Unprotected data transfer between blocks that belong to different asynchronous tasks:

Related topics

Basics

Introduction to Interrupt-Block-Driven Tasks and Task Transitions..... 60
Rate Transitions..... 70

HowTos

How to Implement Protected Task Transitions..... 75
How to Implement Task Transitions for Multiple Receiving Blocks 79

How to Implement Task Transitions for Multiple Receiving Blocks

Objective

If you want to transfer data by the same input signal to multiple receiving blocks that belong to different tasks, you must implement a task transition for each branch of the signal.

Method

To implement task transitions for multiple receiving blocks

1. Branch the signal inside the sending task for as many receivers as you need.
2. For each branch, implement a protected or an unprotected task transition. For instructions, refer to How to Implement Protected Task Transitions on page 75 or How to Implement Unprotected Task Transitions on page 77 respectively.

Example

The following illustration shows how you can branch a signal and send it to multiple receivers. The top branch implements an unprotected data transfer whereas the bottom branch is protected.

Related topics

Basics

Basics of Rate Transitions 70
Introduction to Interrupt-Block-Driven Tasks and Task Transitions 60

HowTos

How to Implement Protected Task Transitions 75
How to Implement Unprotected Task Transitions 77

Assigning Parts of A Model to The Background Task

Introduction

Your real-time application contains one background task which is executed whenever no other real-time task is currently running. It is possible to assign parts of your model to the background task via the Background block provided by RTI's TaskLib.

Where to go from here

Information in this section

Basics of the Background Task.....81

How to Assign Parts of Models to Background Tasks.....82

Example of a Subsystem Assigned to the Background Task.....84

Information in other sections

Preparing a Function-Call Subsystem.....60

Background Block (RTI and RTI-MP Implementation Reference 🕮)
To execute a function-call subsystem in the background task of the real-time application.

Function-Call Subsystem Block (RTI and RTI-MP Implementation Reference 🕮)
To insert a ready-to-use function-call subsystem triggered by a software or hardware interrupt.

Basics of the Background Task

Background task

The background task is realized as an endless for-loop in the main program of your real-time application. It is not accessible in the **RTI Task Configuration** dialog because it does not have a priority. Therefore, you cannot predict when or how often it is executed. The calculations made in it can be either distributed over several base sampling steps or executed several times between two base sampling steps. There is no Task Info group in the variable description, therefore `turnaroundTime`, `taskCallCount`, etc. are not available for the background task.

The background task remains active no matter what the current simulation state is (unless you stop the real-time processor or reset the hardware). If you want the connected function-call subsystem to be conditionally executed whenever a

specific simulation state is active, you can specify the desired condition (RUN, PAUSE, STOP) in the Block Parameters dialog of the Background block.

Note

The *PAUSE* simState is not supported by DS1007 and MicroLabBox (DS1202).

Background block

RTI's Tasklib contains a Background block which lets you calculate parts of your model in the background task. You should only place parts of your model in the background task if they are uncritical with regard to timing.

If desired, you can use several instances of the Background block in a single model or subsystem.

Note

- If you use the Background block extensively, it might take longer to update the instruments of ControlDesk because the corresponding host service for ControlDesk is executed in the background task as well.
- Do not connect multiple subsystems to the same Background block.
- With multiprocessor systems, no blocks are allowed in this connection. With single-processor systems, you may include the following blocks in the connection: Mux/Demux, Inport/Outport, and From/Goto.

Related topics

HowTos

How to Assign Parts of Models to Background Tasks..... 82

Examples

Example of a Subsystem Assigned to the Background Task..... 84

How to Assign Parts of Models to Background Tasks

Objective

The Background block allows you to assign parts of your model to the background task. You might want to place blocks in the background task if their execution timing is uncritical.

Precondition

The parts of the model you want to assign to the background task must be placed in a function-call subsystem. For instructions on how to prepare function-call subsystems, refer to How to Prepare Function-Call Subsystems on page 61.

Method

To assign a part of the model to the background task

1 Place the **Background** block from RTI's TaskLib in your model. This block is the caller for the function-call subsystem.

2 Open the **Block Parameters** dialog of the Background block and specify the desired condition(s) (`RUN`, `PAUSE`, `STOP`) on which the subsystem should be executed in the background task.

> **Note**
>
> The *PAUSE* simState is not supported by DS1007 and MicroLabBox (DS1202).

3 Connect the output port of the Background block to the trigger port of the function-call subsystem.

> **Tip**
>
> To assign the entire subsystem to a single CPU, add a Default CPU block (refer to How to Define a Default CPU for Unconnected Model Parts on page 136) or place the entire subsystem together with its trigger source inside another parent subsystem that is assigned to a single CPU (see How to Define a Default CPU for Unconnected Model Parts on page 136).

Related topics

Basics

Basics of the Background Task 81

HowTos

How to Prepare Function-Call Subsystems 61

Examples

Example of a Subsystem Assigned to the Background Task 84

References

Background Block (RTI and RTI-MP Implementation Reference 🕮)
Function-Call Subsystem Block (RTI and RTI-MP Implementation Reference 🕮)
IPI (RTI and RTI-MP Implementation Reference 🕮)

Example of a Subsystem Assigned to the Background Task

Using a background task

Consider the throttle control example (refer to Throttle Control Example on page 16) and suppose you want to monitor the flow rate. This model part has nothing to do with the actual control problem, so its timing is not critical. Especially if the CPU load is rather high, you might want to place such uncritical subsystems in the background task so that they are executed only if the CPU is idle. A possible model design for the example is shown below:

Related topics

Basics

Basics of the Background Task..... 81

HowTos

How to Assign Parts of Models to Background Tasks..... 82

Configuring Tasks

Introduction

Your model usually consists of several tasks that are executed at different points in time. Each task needs to be configured with regard to its priority and overrun strategy. Tasks can be combined or subscheduled.

Where to go from here

Information in this section

Basics on Task Priorities.....86
How to Change the Task Priorities.....89
How to Specify Overrun Strategies for Tasks.....90
How to Configure Tasks via Scripts.....92
How to Combine Several Interrupt Sources.....94
How to Subschedule Tasks.....96

Basics on Task Priorities

Basics

If you want to configure task priorities, you must specify first whether you want to execute your model in the single timer task mode or multiple timer task mode. Depending on the execution mode chosen, a different constellation of timer tasks results. The priority of a timer task is fixed in relation to the priorities of the other timer tasks. This is due to the rate-monotonic priority scheme. For detailed information on the execution modes refer to Timer Tasks on page 48.

Example

The RTI Task Configuration dialog can, for example, list the following distribution of priorities.

RTI Task Configuration dialog

The RTI Task Configuration dialog shows all the tasks of your model in decreasing order of their priorities (highest priority on top). The entries of the task list show the priority number, followed by the name of the task. The priority number can range from 1 as the highest priority to 127 as the lowest priority. The background task has no priority; thus, can be interrupted by any other task and it is not displayed in the RTI Task Configuration dialog. RTI derives the name of a task as follows:

- For timer tasks from the sample time and offset (if any), for example, Timer Task 2 [1] s
- For tasks driven by interrupt blocks from the name of the driving Timer, Hardware or Software Interrupt block

The Task configuration frame displays the task configuration details for the task that is currently highlighted in the task list.

Note

In some cases, inappropriate distribution of priorities might lead to overrun situations – and even to deadlock for multiprocessor systems.

Non real-time simulation mode

For testing purposes you might want to run the real-time program in a non real-time mode. If you use the as-fast-as-possible simulation mode, you should assign the lowest priority to the timer task. Otherwise virtually no time would be left for the lower-priority tasks driven by interrupt blocks because the timer task is called at a very fast pace in the background task of the model.

Displaying task configuration information

Once your real-time application is running on the dSPACE hardware, you can find the task configuration information (priority, overrunChecktype etc.) of each task in the Task Info group of the application's variable description. You can display task configuration information using ControlDesk.

Note

Simulink provides the Higher priority value indicates higher task priority option on the Solver dialog of the Model Explorer.

For RTI, you must clear this checkbox. RTI checks for the correct setting of this checkbox when starting the build process.

Related topics

Basics

Overrun Situation and Turnaround Time 45
Tasks Driven by Interrupt Blocks 57
Timer Tasks 48

HowTos

How to Change the Task Priorities in RTI-MP Models 145
How to Specify Overrun Strategies for Tasks 90

References

Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮)
Main Page (Multiprocessor Setup Dialog) (RTI and RTI-MP Implementation Reference 🕮)
RTI Task Configuration Dialog (RTI and RTI-MP Implementation Reference 🕮)

How to Change the Task Priorities

Objective

If you want to modify the general execution order of your real-time application, you must change the priorities of its tasks.

Precondition

You must have specified whether you want to execute your model in the single timer task mode or multiple timer task mode. For detailed information on the execution modes, refer to Timer Tasks on page 48.

Method

To change a task priority

1 Open the RTI Task Configuration dialog:
   - For single-processor systems you can access it from the Model Explorer's Code Generation dialog of your model's configuration set (RTI simulation options dialog).
   - For multiprocessor systems you can access it from the individual CPUs' pages of the Multiprocessor Setup dialog.

2 You can modify the priority of one task in relation to another and change its position in the task list via the Up, Down and As Previous buttons.

> **Note**
>
> If you changed the number of tasks in your model, i.e. you added or removed a task driven by an RTI interrupt block, the RTI Task Configuration dialog opens automatically during automatic program building so that you can confirm the task configuration.

Related topics

Basics

Overrun Situation and Turnaround Time 45
Tasks Driven by Interrupt Blocks 57
Timer Tasks 48

HowTos

How to Change the Task Priorities in RTI-MP Models 145
How to Specify Overrun Strategies for Tasks 90

References

Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮)
Main Page (Multiprocessor Setup Dialog) (RTI and RTI-MP Implementation Reference 🕮)
RTI Task Configuration Dialog (RTI and RTI-MP Implementation Reference 🕮)

How to Specify Overrun Strategies for Tasks

Objective

To control how RTI behaves if an overrun occurs for a specific task in your running real-time application, you must specify overrun strategies for tasks.

Available overrun strategies

Stop simulation (default) The simulation is stopped whenever a task overrun occurs.

Queue task before simulation stop When a task overrun occurs, the following steps are performed:

- The overrun-count variable is incremented for the task concerned.
- The task instance is queued for later execution, and the queue-count variable is incremented. You can specify a maximum value for the queue-count variable. When this limit is reached, the simulation is stopped.
- When the previous task instance finishes execution, the task is immediately scheduled again. The queue-count variable is decremented.

Ignore overrun When a task overrun occurs, the overrun-count variable is incremented for the task concerned. However, the task is not scheduled, i.e., task instances are discarded in overrun situations.

Note

- If one of the overrun strategies *Queue task before simulation stop* or *Ignore overrun* is selected, a task overrun might lead to unexpected simulation results since tasks are not executed in real-time.
- Even though stopping the application due to an overrun means that a run-time error has occurred, the application remains in the STOP simulation state and can be restarted.

See How to Set the Simulation State on page 208. For details on run-time errors, see Reaction to Run-Time Errors on page 215.

Method

To specify a overrun strategy for a task

1 Open the RTI Task Configuration dialog:
   - For single-processor systems you can access it from the Model Explorer's Code Generation dialog of your model's configuration set (RTI simulation options dialog).
   - For multiprocessor systems you can access it from the individual CPUs' pages of the Multiprocessor Setup dialog.

The RTI Task Configuration dialog opens:

2 In the task list, highlight the task you want to modify.

3 Select the desired overrun strategy in the Overrun strategy frame.

If you select the queue task before simulation stop setting, you also have to specify the number of max. queued task instances.

Analyzing overrun situations

You can analyze any overrun situation you encounter by means of the following variables:

- The `overrunCount` variable of a task counts the total number of task overruns that occurred for it.
- The `overrunQueueCount` variable displays the number of queued task instances. The number is incremented for each queued task request and decremented each time the task finishes execution.
- The `taskCallCount` variable of a task is incremented whenever an interrupt triggers a task registration. It does not include the task calls that caused overrun situations.

These variables can be displayed in ControlDesk (refer to the variable descriptions in the *RTI and RTI-MP Implementation Reference*).

For further information on overrun situations, refer to Running the Application on page 304.

Result

You specified an overrun strategy for a particular task.

Related topics

Basics

Overrun Situation and Turnaround Time 45
Reaction to Run-Time Errors 215

HowTos

How to Change the Task Priorities 89
How to Change the Task Priorities in RTI-MP Models 145

References

RTI Task Configuration Dialog (RTI and RTI-MP Implementation Reference 🕮)
RTI Task Configuration Dialog (Multiprocessor Setup Dialog) (RTI and RTI-MP Implementation Reference 🕮)

How to Configure Tasks via Scripts

Objective

RTI and RTI-MP let you configure the tasks in your model. You can do this either manually with the Task Configuration dialog or automatically by using the `TaskConfiguration` API.

Basics

The `TaskConfiguration` API is an object-oriented programming interface that provides functions that are equivalent to the settings in the Task Configuration dialog. You can enter the API methods directly in the MATLAB Command Window or write them into an M file and execute the file afterwards.

The `TaskConfiguration` API provides certain `Get` methods to return the current task settings in the model, such as the `GetTaskPriority` method.

To modify the task settings, the API provides `Set` methods for the following main actions:

- To configure the priority of tasks.
- To specify the overrun behavior of tasks.

For an overview of the available methods, refer to RTI and RTI-MP Task Configuration API Reference (RTI and RTI-MP Implementation Reference 🕮).

TaskManager classes

The API provides different task manager classes for single-processor models used with RTI and multiprocessor models used with RTI-MP. For RTI, you have to create an instance of the `TaskManager` class. For RTI-MP, you have to create one `MainModelTaskManager` instance for the main model that contains the Multiprocessor Setup block, and `SubModelTaskManager` instances for each submodel whose tasks you want to configure.

The handling of the `TaskManager` class and the `SubModelTaskManager` class are similar. The `MainModelTaskManager` class is used to get information on the contained submodels and to create the `SubModelTaskManager` instances.

Error handling

The error handling of the `TaskConfiguration` API uses the MATLAB exception handling mechanism.

The constructors for the task manager instances and the `Commit` method are not exception-safe. If these methods generate an exception, the internal data might be invalid and the affected object instance should not be used. All other methods are exception-safe. They can still be used after an exception was generated.

Template files

In your RCP and HIL installation, you find template files that contain some basic methods for using the `TaskConfiguration` API. You can open the files by using the `edit <FileName>` command in MATLAB.

RTI/RTI-MP	File Name
RTI	rti_task_api_template.m
RTI-MP	rtimp_task_api_template.m

Note

Save the open template files under a new name in your working directory before you modify them to avoid overwriting them.

Restrictions

While the task configuration is running, you must not edit the open Simulink model.

Method

To configure tasks via scripts

1. Open the Simulink model whose tasks you want to configure. The model must be in a consistent state.
2. Create the task manager instance.
   - RTI:

     ```
     <ObjectName> =
     RTI.TaskConfiguration.TaskManager('<ModelName>');
     ```

   - RTI-MP:

     ```
     <ObjectName> =
     RTIMP.TaskConfiguration.MainModelTaskManager('<ModelName>');
     ```

3. Use a `Get` method to get certain values, such as the task names available in the model. These output values are often required to specify the input value of another method.

   When you use RTI-MP, create a `SubModelTaskManager` instance.

4 Use a `Set` method to specify a certain value.

5 Use the `Commit` method to write the modifications of the task configuration to the model file.

6 Save the model. This also saves the modified task configuration.

Result

You modified the task configuration in your model via the `TaskConfiguration` API.

Related topics

References

RTI and RTI-MP Task Configuration API Reference (RTI and RTI-MP Implementation Reference 🕮)

How to Combine Several Interrupt Sources

Example

Consider the throttle control example (refer to Throttle Control Example on page 16). Suppose you want to close the throttle valve when someone presses the emergency stop button or if the flow rate in the throttle valve exceeds a certain limit. You have to combine these interrupts via a **Mux** block to trigger a single function-call subsystem as shown in the illustration below:

Combining interrupt sources

Via the Mux block, each interrupt source can then create its own task from the function-call subsystem, which therefore might execute in multiple calls. Since each call has its own overrun check, no overrun situation occurs – provided each call is triggered by a different interrupt. If an interrupt with high priority occurs, it suspends the call executed as a task with a lower priority and creates a new call.

Limitations

In combining interrupts to trigger the same subsystem, the following limitations apply:

- The code generated for the function-call subsystem is not safe on reentrance. That is, the data of the first call of a task is lost if another call starts. This is due to the global definition of block outputs in the function-call subsystem. Therefore, if two tasks act simultaneously on the same function-call subsystem, you cannot predict which block outputs will result. Even serious memory corruption can result depending on the blocks calculated in the task.

  For this reason you should use identical priorities when combining interrupt sources: Tasks of the same priority do not interrupt each other.
- You cannot combine the function-call signals from RTI interrupt blocks with those of other blocks, for example, a Function-Call Generator block or Stateflow chart.
- You cannot use the `ssEnableSystemWithTid` and `ssDisableSystemWithTid` macros to enable and disable a function-call subsystem driven by an RTI interrupt block because the subsystem will always be executed when it is triggered by an RTI interrupt block.
- With RTI-MP you can combine only local interrupts to trigger the same subsystem, i.e., all interrupts must originate from the same CPU. None of the function-call signal lines may contain an IPI block.

Method

To combine several interrupt sources

1. Place a Mux block from the Simulink library on the same level as the function-call subsystem you want to trigger.

2. Connect all Timer, Hardware or Software Interrupt blocks that you want to use as the triggers to the input ports of the Mux block.
3. Connect the output port of the Mux block to the trigger port of the function-call subsystem.
4. Assign the same priority to all the resulting tasks.

Next step

If you wish to detect which interrupt source triggered the function-call subsystem, open the function-call subsystem and select the Show output port checkbox in the Block Parameters dialog of the subsystem's Simulink Trigger block.The Simulink Trigger block then provides a vectorized (multiplexed) signal in which all vector elements show zero, except for the one that triggered the function-call subsystem.

Related topics

Basics

General Conventions.....268
Preparing a Function-Call Subsystem.....60

HowTos

How to Change the Task Priorities.....89
How to Subschedule Tasks.....96
How to Use Hardware Interrupts.....62
How to Use Software Interrupts.....65

References

Function-Call Subsystem Block (RTI and RTI-MP Implementation Reference 🕮)
Hardware Interrupt Block (RTI and RTI-MP Implementation Reference 🕮)
Software Interrupt Block (RTI and RTI-MP Implementation Reference 🕮)

How to Subschedule Tasks

Example

Consider the throttle control example (refer to Throttle Control Example on page 16). Suppose you want two actions to be carried out in a specific sequence (called subscheduling) after the emergency-stop button was pressed. You can achieve this behavior by means of additional priorities, which you can define via Software Interrupt blocks as illustrated below. The blocks forming the actual control loop have been left out for clarity.

Subscheduling tasks

For this approach to function correctly, the interrupt block of the master subsystem must have a higher priority than the Software Interrupt blocks used for the subscheduling. Otherwise the behavior is unpredictable because the execution order of the Software Interrupt blocks (in the master subsystem) is not predefined.

Note

With RTI-MP, subscheduling is possible only for local interrupts, i.e., if the source and target CPUs are identical.

Method

To subschedule a task

1. Build up the level triggered by the **hardware interrupt**. This acts as the master subsystem.
2. Place as many **Software Interrupt** blocks from RTI's TaskLib in the master subsystem as you require for subscheduling.

3. Add the same number of function-call subsystems from RTI's TaskLib to the model. These act as slave subsystems.

4. Connect the slave subsystems to the **Software Interrupt** blocks of the master subsystem.

   You may include the following blocks in the connection: **Mux/Demux**, **Inport/Outport**, and **From/Goto**.

Related topics

Basics

Preparing a Function-Call Subsystem 60

HowTos

How to Combine Several Interrupt Sources 94
How to Use Hardware Interrupts 62

How to Use Software Interrupts.....65

References

Function-Call Subsystem Block (RTI and RTI-MP Implementation Reference 🕮)
Hardware Interrupt Block (RTI and RTI-MP Implementation Reference 🕮)
Software Interrupt Block (RTI and RTI-MP Implementation Reference 🕮)

Time-Triggered Tasks and Timetables

Introduction

Time-triggered tasks and timetables let you implement task trigger times and even complete time-dependent schedules (i.e., timetables) for tasks.

Where to go from here

Information in this section

Basics of Time-Triggered Tasks and Timetables.....99
How to Implement Time-Triggered Tasks.....102
How to Implement Timetables.....104

Basics of Time-Triggered Tasks and Timetables

Scheduling tasks

Time-triggered tasks and timetables are features of RTI and RTI-MP that let you schedule one or more tasks with predefined delay times in relation to a single trigger event. Via time-triggered tasks and timetables, you can implement more flexible task handling in your model, for example, for time-based data processing. You can include multiple independent time-triggered tasks and/or timetables in your model.

Example Suppose you want to take a measurement n milliseconds after the ignition of an engine. The value of n depends on several parameters such as engine speed and acceleration. This can be implemented via a time-triggered task. To start several measurements, you can implement a timetable.

Restrictions

- The total number of time-triggered tasks and timetable tasks that can be simultaneously scheduled is limited to 127 (for DS1006, DS1007, MicroAutoBox II and MicroLabBox).
- Not available for DS1104 platform support.

Time-triggered tasks

When a time-triggered task is triggered (for example, by a hardware interrupt), the actual task is started with a trigger delay time.

The trigger delay time is specified via a Simulink signal, which may vary during the simulation. If you specify a vector as the trigger delay time, the actual task is started once for each vector element.

Example The following illustration shows the timing for a time-triggered task with three trigger delay times having the values 0.1 ms, 0.2 ms and 0.5 ms on occurrence of the trigger event:

Timetables

A timetable consists of a set of timetable tasks, each of which has its own predefined trigger delay time (scalar or vector). In addition, the entire timetable has a common timetable start delay time.

When the timetable is triggered (for example, by a hardware interrupt), each timetable task is started with its effective trigger delay time, i.e., timetable start delay time (variable) + predefined trigger delay time of the timetable task (predefined and fixed).

The timetable start delay time is specified via a scalar Simulink signal, which may vary during the simulation.

Example The following illustration shows the timing for a timetable with two timetable tasks: Task A has a trigger delay time of 2 ms and Task B has the trigger delay times 1 ms and 4 ms. The timetable start delay time is 1.5 ms on occurrence of the trigger event. Therefore, the effective trigger delay times are 3.5 ms for Task A, and 2.5 ms and 5.5 ms for Task B.

Self-triggered task

If desired, you can implement a self-triggered task or timetable by using a time-triggered task or timetable task as trigger event for the task or timetable itself. To initiate the first trigger event, you must use the simulation start as a trigger event by selecting the **Execute time-triggered task once at simulation start** option in the **Time-Trigger Set** block or the **Start timetable once at simulation start** option in the **Timetable Start** block.

Note

- When a time-triggered task is started at simulation start, its task trigger delay time is ignored for the first execution step and the actual task is started immediately.
- When a timetable is started at simulation start, its timetable start delay time is ignored for the first execution step and the timetable tasks are started with only their individual task trigger delay times.

Example The following illustration shows the timing for a time-triggered task and a timetable that use the simulation start (T=0) as trigger event. The task trigger delay time(s) of the time-triggered task and the timetable start delay time for the timetable tasks are ignored for the first execution step.

Executing the timer tasks as time-triggered or timetable tasks

You can execute the timer tasks of your model as time-triggered tasks or timetable tasks by connecting the Time-Triggered Task block or Timetable Task block to the Timer Task Assignment block.

Example Suppose you want to implement an application that performs data sampling at a fast sampling rate and feeds the sampled data vectors to the timer tasks for further data processing. In this case you can implement two time-triggered tasks with appropriate trigger delays. Connect the Timer Task Assignment block to one of the time-triggered task blocks. This time-triggered task must be configured to use the simulation start as trigger event.

Related topics

HowTos

How to Implement Timetables..... 104
How to Implement Time-Triggered Tasks..... 102

How to Implement Time-Triggered Tasks

Objective

If you want to trigger a task with one or more trigger delay times, you must implement a time-triggered task. The trigger delay time is given by a Simulink signal.

Basics

A time-triggered task is implemented by one **Time-Trigger Set** block and one **Time-Triggered Task** block, which are associated by means of a unique identifier (time-trigger ID).

- When the **Time-Trigger Set** block is executed, it sets the trigger delay time for its task according to the signal at the **Trigger delay** inport. When the trigger delay time is reached the task is scheduled.

- The actual time-triggered task consists of a function-call subsystem that is connected to the **Time-Triggered Task** block.

Prerequisites

A function-call subsystem must be prepared. For instructions, refer to How to Prepare Function-Call Subsystems on page 61.

Restrictions

- The total number of time-triggered tasks and timetable tasks that can be simultaneously scheduled is limited to 127 (for DS1006, DS1007, MicroAutoBox II and MicroLabBox).
- Not available for DS1104 platform support.

Method

To implement a time-triggered task

1. Place the **Time-Trigger Set** block from RTI's **TaskLib/Time-Trigger** library in the model. You can place it anywhere but in the background task.

2. Open the **Block Parameters** dialog of the **Time-Trigger Set** block and enter a unique **Time-Trigger ID**.

Close the dialog.

Tip

To implement a self-triggered task, place the block in the function-call subsystem driven by the associated **Time-Triggered Task** block. In this case, select the **Execute time-triggered task once at simulation start** checkbox.

3 Connect the desired Simulink signal to the **Time-Trigger Set** block, which serves as the task trigger delay time. To specify more than one task trigger delay time, use a vector signal.

4 Place the **Time-Triggered Task** block from RTI's **TaskLib/Time-Trigger** library in your model. This block is the trigger source for the time-triggered task.

Note

Do not place this block inside a triggered subsystem or an enabled subsystem. It always activates its function-call subsystem regardless of whether its parent subsystem is active or not.

5 Open the **Block Parameters** dialog of the Time-Triggered Task block and enter the same **Time-Trigger ID** as in step 1 on page 102. Afterwards close the dialog.

6 Connect the outport of the trigger source to the trigger port of the function-call subsystem.

Note

Do not connect multiple function-call subsystems to the same trigger source.

You may include the following blocks in the connection: **Mux/Demux**, **Inport/Outport**, and **From/Goto**.

Note

RTI-MP
To transfer an interrupt from one CPU to another, you have to connect the trigger source and the function-call subsystem via an **IPI** block. No other blocks are allowed for this connection. The IPI block must list all target CPUs involved in the execution of the function-call subsystem. However, for local interrupts you do not need to include the IPI block in the connection. See How to Implement Interprocessor Interrupts on page 143 for details on IPI blocks.

Tip

For multiprocessor systems, you can also assign different parts within the subsystem to different CPUs: Just add the required **IPC** blocks and/or **Default CPU** blocks (see Interprocessor Communication Using IPC Blocks on page 130 and How to Define a Default CPU for Unconnected Model Parts on page 136).

Tip

To execute the timer tasks of the model as time-triggered tasks, connect the **Timer Task Assignment** block to the outport of the **Time Triggered Task** block.

Related topics

Basics

Basics of Time-Triggered Tasks and Timetables.....99

References

Time-Trigger Set Block (RTI and RTI-MP Implementation Reference 🕮)
Time-Triggered Task Block (RTI and RTI-MP Implementation Reference 🕮)

How to Implement Timetables

Objective

To start a set of timetable tasks in relation to a single trigger event, you must implement a timetable. The start delay time for a timetable is a Simulink signal.

Basics

A timetable is implemented by one **Timetable Start** block and at least one **Timetable Task** block, which are associated by means of a unique identifier (timetable ID).

- When the **Timetable Start** block is executed, it triggers the timetable with the timetable start delay time according to the signal at the **Start delay** inport.

- Each timetable task consists of a function-call subsystem that is connected to a **Timetable Task** block. When the timetable start delay time plus the task trigger delay time of a **Timetable Task** block is reached, the actual task of that **Timetable Task** block is executed.

Prerequisites

Prepare as many function-call subsystems as needed for timetable tasks. For instructions, refer to How to Prepare Function-Call Subsystems on page 61.

Restrictions

- The total number of time-triggered tasks and timetable tasks that can be simultaneously scheduled is limited to 127 (for DS1006, DS1007, MicroAutoBox II and MicroLabBox).
- Not available for DS1104 platform support.

Method

To implement a timetable

1 Place the **Timetable Start** block from RTI's **TaskLib/Time-Trigger** library in the model. You can place it anywhere but in the background task.

Tip

To implement a self-triggered task, place the block in the function-call subsystem driven by an associated **Timetable Task** block. In this case, select the **Start timetable once at simulation start** checkbox.

2 Open the **Block Parameters** dialog of the **Timetable Start** block and enter a unique **Timetable ID**.

If you want to specify a start delay time for the timetable, select the **Enable timetable start delay inport** checkbox.

Close the dialog.

3 If you selected the **Enable timetable start delay inport** checkbox, connect the desired Simulink signal to the **Timetable Start** block. The signal must be a scalar.

4 Place the **Timetable Task** block from RTI's **TaskLib/Time-Trigger** library in your model. This block is the trigger source for the timetable task.

Note

Do not place this block inside a triggered subsystem or an enabled subsystem. It always activates its function-call subsystem regardless of whether its parent subsystem is active or not.

5 Open the **Block Parameters** dialog of the Timetable Task block. Enter the same **Timetable ID** as in step 1 on page 105.

Enter the desired **Task trigger delay time**. To specify more than one task trigger delay time, specify a vector.

Close the dialog.

6 Connect the outport of the trigger source to the trigger port of the function-call subsystem.

Note

Do not connect multiple function-call subsystems to the same trigger source.

You may include the following blocks in the connection: **Mux/Demux**, **Inport/Outport**, and **From/Goto**.

Note

RTI-MP
To transfer an interrupt from one CPU to another, you have to connect the trigger source and the function-call subsystem via an **IPI** block. No other blocks are allowed for this connection. The IPI block must list all target CPUs involved in the execution of the function-call subsystem. However, for local interrupts you do not need to include the IPI block in the connection. See How to Implement Interprocessor Interrupts on page 143 for details on IPI blocks.

Tip

For multiprocessor systems, you can also assign different parts within the subsystem to different CPUs: Just add the required **IPC** blocks and/or **Default CPU** blocks (see Interprocessor Communication Using IPC Blocks on page 130 and How to Define a Default CPU for Unconnected Model Parts on page 136).

Tip

To execute the timer tasks of the model as a timetable task, connect the **Timer Task Assignment** block to the outport of one of the **Timetable Task** blocks.

7 Repeat steps 3 and 4 as often as required to create the desired timetable.

Related topics

Basics

Basics of Time-Triggered Tasks and Timetables 99

References

Timetable Start Block (RTI and RTI-MP Implementation Reference 🕮)
Timetable Task Block (RTI and RTI-MP Implementation Reference 🕮)

Distributing the Model for MP Systems

Introduction

To gain enough computing power to calculate even large models in real-time, you must distribute them for multiprocessor systems.

Where to go from here

Information in this section

Basics of Distributing the Model for MP Systems..... 108
Interprocessor Communication..... 115
Interprocessor Interrupts..... 143

Information in other sections

Limitations with RTI-MP..... 276

Basics of Distributing the Model for MP Systems

Introduction

When distributing models for multiprocessor systems you must consider the following basic principles.

Where to go from here

Information in this section

Details on MP Systems 108
How to Configure Models for MP Systems 111
Standard Network Topology 113
Model Distribution Guidelines 114

Details on MP Systems

Multiprocessor systems

Real-time simulation can be very demanding in terms of computing power – this is especially true for rather large models. If the computing power of single-processor systems does not meet your requirements you can use a dSPACE multiprocessor system. This allows you to break your large model into several smaller parts and to assign these parts to different CPUs. Thus, you can gain enough computing power to calculate even large models in real-time.

Example

Consider the throttle control example (refer to Throttle Control Example on page 16) and suppose it would require more computing power than a single processor system can provide. You could then divide the model and assign the parts to the different CPUs of a multiprocessor system so that one board simulates the plant and the other board the controller. To get the desired multiprocessor model you have to add a **Multiprocessor Setup** block that holds the configuration parameters for multiprocessor models. You also have to modify all connection lines between the plant and controller so that they are passed through Interprocessor Communication blocks (**IPC** blocks). Hence, the **IPC** blocks mark the boundaries between the different CPUs. The following illustration shows the modified model for a multiprocessor system:

Optional CPUs

The optional CPU feature of RTI-MP allows you to:

- Specify certain CPUs – and the corresponding real-time applications – as optional in RTI-MP
- Work with topological subsets of the currently registered multiprocessor system in ControlDesk

> **Note**
>
> The optional CPU feature is not supported by DS1007 and MicroLabBox.

This allows you to download a multiprocessor application either to the entire multiprocessor system or to a topological subset of it without having to rebuild the application.

Refer to Working with Subsets of a Multiprocessor Topology (DS1006 Features 🕮) for details on this technique.

Including submodels in multiprocessor models

By default, the **Always create a new submodel** option in the **Multiprocessor Setup** dialog is set. In this modeling mode, the multiprocessor model, that contains the **Multiprocessor Setup** block and the interprocessor blocks for communication and interrupt handling (IPC and IPI blocks), is always separated into submodels when you start the build process, open the dialog for task configuration, or only show a submodel. The separation is done according to the configurations of the IPC and IPI blocks.

With the **Always create a new submodel** option in the **Multiprocessor Setup** dialog, you can change this behavior. If you clear this option, you can include an already existing submodel in your multiprocessor system.

The Always create a new submodel option is available for each processor (CPU) in your Multiprocessor Setup dialog.

The submodel must be stored in the current working folder of your multiprocessor model. To integrate it in the multiprocessor model, you have to enter its name via the Application name setting on the CPU's page and implement the required interprocessor communication and interprocessor interrupts manually.

Interprocessor communication The submodel has to provide IPC Send and IPC Receive blocks for data to be exchanged. The corresponding IPC blocks in the main model have to provide the configuration for these blocks.

Interprocessor interrupts To implement interprocessor interrupts, the configuration of the IPI Send and IPI Receive blocks in the submodel has to match the configuration of the IPI blocks in the main model.

If you select Update Submodel on the CPU's page, the configuration of the current submodel is checked. If required blocks for interprocessor communication and interprocessor interrupts are not available in the submodel, a subsystem containing these blocks is added to the submodel with preconfigured settings for copying. The created RTI-MP subsystem has the block display *Contains blocks created during 'Update submodel'*.

If you start a build process, the update is done for the entire multiprocessor model with the model workspace, MP-specific settings of the Model Configuration Parameters and the configuration of the IPC and IPI blocks. If there are generated RTI-MP subsystems in the submodels with unused IPC Send/IPI Receive and IPI Send/IPI Receive blocks, they are ignored in the build process.

How to Configure Models for MP Systems

Objective

Models need to be configured for use in a multiprocessor system.

Simulation parameters

The **Multiprocessor Setup** dialog (available via the Multiprocessor Setup block) holds the majority of the global simulation parameters for the multiprocessor model and all the local options for each CPU, including the name for each CPU and its submodel.

However, if you simulate the RTI-MP model in Simulink you must specify the desired simulation parameters via the **Solver** dialog of your model's configuration set in the **Model Explorer** instead.

You must additionally configure the following options in the main model via the **Model Explorer**:

- All Simulink data import/export options
- All Simulink and Simulink Coder optimization options
- All Simulink diagnostic options
- All Simulink and Simulink Coder hardware implementation options
- The following Simulink Coder options:
  - The Documentation options on the **General** page
  - All options on the **Comments** page
  - All options on the **Symbol** page
  - All options on the **Custom Code** page
  - All options on the **Debug** page
  - All options on the **Interface** page
  - The **Generate code only** option of Simulink Coder

When the model is separated, RTI-MP ensures that the configuration of the main model is transferred to all submodels properly.

For example, **Data Capture** and **TRC Exclusion** blocks in the main model are copied to all submodels.

If you implement interprocessor communication to a new CPU, the corresponding CPU's page is automatically added to the **Multiprocessor Setup** dialog, where you can edit the properties of the new CPU and the name of the new submodel according to your requirements.

Note

- The board names used by the **Platform Manager** of ControlDesk and by RTI-MP must be identical.
- RTI-MP automatically selects the CPU with the name `master` as the master CPU. If no CPU is named `master`, RTI-MP selects the CPU with the name that is the first in the alphabet. The master CPU is marked with an **(M)** on the **Main** page of the **Multiprocessor Setup** dialog.

Tip

RTI-MP provides a hook function to customize the model separation and the update of submodels. To use the hook function, there must be a specific M file available when applying settings in the **Multiprocessor Setup** dialog or during the build process. For this M file, there is a template available in the **Templates** sublibrary of the **RTI-MP** blockset. There you also find an example that shows you how to modify the template.

For more information, refer to rti1xxx_template_rtimphook.m (RTI and RTI-MP Implementation Reference 🕮).

Method

To configure a model for an MP system

1. Drag and drop the **Multiprocessor Setup** block from the RTI-MP Blocks library onto the root level of your model.

   If you double-click the Multiprocessor Setup block in the RTI-MP Blocks library, a preconfigured multiprocessor model is created. It contains a Multiprocessor Setup block and an IPC block.

2. Whenever you create a communication connection to a new CPU, you have to adapt the name of the new submodel on the corresponding CPU's page of the **Multiprocessor Setup** dialog (this CPU's page has the same name as the new CPU that you specified for the communication connection).

3. To build and download the RTI-MP model, you have to adapt all the settings of the **Multiprocessor Setup** dialog according to the instructions given in How to Specify Options for the Build Process on page 226.

4. Ensure that the Gigalink topology is defined via the **Multiprocessor Topology Setup** dialog of the model.

   To make a slave CPU optional, select the **Treat as optional CPU** checkbox on the CPU's page of the Multiprocessor Setup dialog (see CPU's Page (Multiprocessor Setup Dialog) (RTI and RTI-MP Implementation Reference 🕮) and Working with Subsets of a Multiprocessor Topology (DS1006 Features 🕮).

Result

You configured a model for an MP system.

Related topics

HowTos

How to Specify Options for the Build Process 226

References

Multiprocessor Setup (RTI and RTI-MP Implementation Reference 🕮)
rti1xxx_template_rtimphook.m (RTI and RTI-MP Implementation Reference 🕮)

Standard Network Topology

Example for a standard network topology

You might want to use a standard network topology for your multiprocessor models so that you can run all models on the same hardware without reconfiguration. The following illustration shows you a solution for connecting up to five boards so that each board has a direct connection to all other boards:

For multiprocessor models with a preconfigured network topology, refer to the **Templates** library in the **RTI-MP** blockset.

Related topics

Basics

Gigalink Connection 116

Model Distribution Guidelines

Efficient MP applications

The execution time of a multiprocessor application generally does not decrease linearly with the number of CPUs involved. This is due to the time overhead needed for interprocessor communication and to the delay that might result if you use the Synchronized Swinging Buffer protocol. Therefore, the execution time of a multiprocessor application (and thus its efficiency) depends highly on the structure of the application, meaning the amount and type of interprocessor communication used. The following guidelines show how to structure your model to achieve efficient multiprocessor application.

- To achieve an efficiently parallelized execution, you should keep the amount of interprocessor communication as low as possible.
- To avoid possible waiting times due to interprocessor communication, you might want to use the Unsynchronized Swinging Buffer protocol or the Virtual Shared Memory protocol instead of the Synchronized Swinging Buffer protocol (see Swinging Buffer Protocol on page 118 and Virtual Shared Memory Protocol on page 119). However, these protocols do not guarantee synchronization between sender and receiver, and the Virtual Shared Memory protocol does not even guarantee data consistency.
- You should avoid direct feedthrough terms (such as IPC block – Gain block – IPC block) if you operate the corresponding communication channels with the Synchronized Swinging Buffer protocol. Otherwise, the involved CPUs might have to wait for each other due to a sequential working order. Direct feedthrough might even lead to deadlock situations. Refer to Model Evaluation on page 120 for details on the timing of the Synchronized Swinging Buffer protocol.
- You should distribute the calculations equally over all CPUs.

Tip

You can free up execution time of the real-time application by assigning model parts (including IPC blocks) to the background task if these model parts are non-critical with regard to time. See Assigning Parts of A Model to The Background Task on page 81 for details.

Related topics

Basics

Assigning Parts of A Model to The Background Task.....81
Model Evaluation.....120
Swinging Buffer Protocol.....118
Virtual Shared Memory Protocol.....119

Interprocessor Communication

Introduction

To simulate your model on a dSPACE multiprocessor system, you have to define the necessary interprocessor communication between the model parts you want to assign to the different CPUs. By implementing interprocessor communication you implicitly specify which model part belongs to each CPU.

For dSPACE systems, the different tasks of a real-time application require separate communication channels due to different timing characteristics. However, it is possible to group all data of a specific sample time and send it over just one communication channel. The same is true for the data of a task driven by an interrupt block that is executed on multiple CPUs. Usually, you need to transfer data of a single sample time only, so you do not need to use several communication channels.

Where to go from here

Information in this section

Introducing Interprocessor Communication..... 115

Implementing Interprocessor Communication Using IPC Blocks..... 124

Implementing Interprocessor Communication Using Gigalink Blocks..... 137

Information in other sections

Interprocessor Interrupts..... 143

Limitations with RTI-MP..... 276

RTI-MP Blockset Reference (RTI and RTI-MP Implementation Reference 🕮)

Introducing Interprocessor Communication

Introduction

To implement interprocessor communication, you must know its basic principles.

Where to go from here

Information in this section

Gigalink Connection 116
Swinging Buffer Protocol 118
Virtual Shared Memory Protocol 119
Model Evaluation 120
Direct Feedthrough 122

Gigalink Connection

Gigalink modules

Several DS1006 or DS1007 boards can communicate by means of the DS910 or DS911 Gigalink Modules respectively. The DS91<x> Gigalink Module provides four bidirectional Gigalinks that allow a flexible network topology.

Note

RTI-MP supports multiprocessor systems built from processor boards that are connected via Gigalinks. Such a multiprocessor system must consist of either only DS1006 boards or only DS1007 boards. RTI-MP also supports the DS1006MC board, DS1007 board and MicroLabBox as multicore systems. For multicore systems no Gigalink connections are necessary.

Each Gigalink provides 16 bidirectional data channels with 8 channels reserved for the Swinging Buffer Protocol and 8 channel reserved for the Virtual Shared Memory Protocol. Each channel can transfer several signals (up to 8 kB). For details on the different protocols, refer to Swinging Buffer Protocol on page 118 and Virtual Shared Memory Protocol on page 119.

RTI-MP uses logical communication channel numbers (for example, `master to slave on channel 1`) and manages the mapping to the available physical communication channels. Throughout this Implementation Guide the term *communication channel* refers to the logical channels, as can be seen in the IPC blocks of the Simulink model.

Note

The `<model>_tp.m` file holds the information about the existing processor network topology and needs to be updated whenever you change the physical Gigalink connections of your multiprocessor system. This can be done via the Multiprocessor Topology Setup dialog (see *Multiprocessor Topology Setup Dialog* (*RTI and RTI-MP Implementation Reference* 🕮)). Via this dialog you can also set up a virtual network topology, for example, if the hardware is not available yet.

Example

The following illustration shows a communication channel between two processor boards, which are connected via the Gigalinks `1` (sender) and `2` (receiver). RTI-MP automatically maps the logical channel numbers in the IPC block (for example, `master to slave on channel 1`) to this physical connection so that the Simulink model does not contain any hardware-dependent information. Therefore, in the Simulink model you cannot see which physical connections are used for interprocessor communication. In this example, physical channel `0` is used.

Related topics

Basics

DS911 Gigalink Module (DS1006 Features 🕮)
DS911 Gigalink Module (DS1007 Features 🕮)
Interprocessor Communication Using IPC Blocks..... 130
Standard Network Topology..... 113

References

Multiprocessor Topology Setup Dialog (RTI and RTI-MP Implementation Reference 🕮)

Swinging Buffer Protocol

Basics

The swinging buffer protocol defines three buffers between the sender and receiver, which can hold a single data vector each. The sender and receiver allocate and lock a separate buffer for writing or reading data. Because of this locking, there is no risk of data inconsistency for this protocol.

The following illustrations show how the swinging buffer protocol works.

- `t[i]`: At the beginning the sender fills the first buffer.

  When the sender finishes writing it unlocks its buffer and locks the next one. The receiver reads the previously written data.

- `t[i+1]`: The procedure described above is repeated using the next buffer: The sender fills its current buffer after which the receiver reads it and so on.

In order to reduce unnecessary waiting times, RTI-MP computes the output signals of a submodel as early as possible. It also carries out all computations that are independent of input signals before reading the input signals, which in turn are read as late as possible.

Synchronized and unsynchronized

The swinging buffer protocol is available in synchronized and unsynchronized mode:

Synchronized If no new data is available the receiver waits for the sender to deliver new data. Data consistency and synchronized data transfer are especially important in control applications where a delay might cause stability problems.

Unsynchronized If no new data is available the receiver reads the data vector of the previous sampling step. As a result there are no waiting times but the

signal is delayed by one sampling step. Depending on the characteristics of the application, usually one of the following two cases applies:

- The receiver always gets the current buffer. This behavior is identical to the synchronized swinging buffer mode and is described above.
- The receiver always gets the buffer that comes from the previous sampling step. This behavior corresponds to the synchronized swinging buffer mode with an additional Unit Delay block.

  This is shown in the next illustration.

Note

In some cases the synchronized swinging buffer protocol might cause overrun or deadlock situations when the receiver waits for new data. For details on how a deadlock situation can arise, see Model Evaluation on page 120.

Tip

If you want to avoid waiting times due to interprocessor communication you can select the unsynchronized swinging buffer protocol or virtual shared memory protocol. However, simulation robustness might decrease with these protocols, because they do not provide synchronized data transfer.

Related topics

Basics

Direct Feedthrough 122
Interprocessor Communication Using IPC Blocks 130
Model Distribution Guidelines 114
Model Evaluation 120
Swinging Buffer Mode (DS1006 Features 🕮)
Swinging Buffer Mode (DS1007 Features 🕮)
Virtual Shared Memory Protocol 119

Virtual Shared Memory Protocol

Basics

The virtual shared memory protocol defines only one buffer for the data vector to be transmitted. Neither the sender nor the receiver locks the buffer but accesses it as if it were its local memory. There is no synchronization between the

sender and receiver, so the receiver might read the same data vector several times or it might not catch each new data vector. Because writing and reading can be simultaneous, this protocol might result in data inconsistency, meaning that the input buffer (as seen by the receiver) contains data resulting from different sampling steps of the sender. The following illustration shows how the virtual shared memory protocol works:

Related topics

Basics

Interprocessor Communication Using IPC Blocks..... 130
Model Distribution Guidelines..... 114
Model Evaluation..... 120
Swinging Buffer Protocol..... 118
Virtual Shared Memory Mode (DS1006 Features 🕮)
Virtual Shared Memory Mode (DS1007 Features 🕮)

Model Evaluation

Basics

Whenever the submodel of a CPU is extracted from the overall multiprocessor model (for example, when building the model), RTI-MP replaces the IPCx blocks of the overall model with either an IPC Receive block or an IPC Send block – depending on whether the CPU receives or sends data via this IPC channel.

Synchronized swinging buffer protocol

If a communication channel uses the synchronized swinging buffer protocol, the receiving CPU must wait until new data is available before it can calculate the blocks depending on this data. Therefore RTI-MP computes the output signals of a submodel as early as possible so that unnecessary waiting times for other CPUs are avoided. For the same reason, RTI-MP reads the input signals as late as possible, meaning that all computations independent of the input data are done in advance until the CPU cannot continue the computation without input data from another CPU.

Assigning priorities

RTI-MP achieves this execution order by means of a high block priority for IPC Send blocks and a low block priority for IPC Receive blocks. However, in the overall RTI-MP model, RTI-MP cannot assign the priorities until the submodels of the individual CPUs are separated from the overall multiprocessor model. Whenever the submodel of a CPU is separated from the overall multiprocessor

model, RTI-MP uses the following two techniques to assign the highest priority to IPC Send blocks:

- The IPC Send blocks are set to a priority value of -10,000, which represents a very high priority.
- The IPC Receive blocks are set to a priority value of 10,000, which represents a very low priority.

If you include submodels in your multiprocessor model (the **Always create a new submodel** option in the **Multiprocessor Setup** dialog is cleared), you can change the priority defaults, because they will not be replaced by updating the model or building the real-time application.

Example

The following example shows how a user-specific priority can cause a block priority violation in the extracted submodel while being unproblematic in the overall model.

Consider the demo model `pipt1_dual1007_sl.slx` and suppose you assigned a priority value of `1` to Sum block `e`. This causes no priority conflict in the overall multiprocessor model – you can start the Simulink simulation without a problem:

However, in the extracted submodel `pi_1007_sl` this priority value conflicts with the RTI-MP-specific high priority of the IPC Send block (-10,000). Due to topological dependencies, Sum block `e`, which has a low priority, must be computed before the IPC Send. Therefore, Simulink issues a warning when you start the Simulink simulation or code generation for the submodel.

Direct feedthrough

Direct feedthrough terms can lead to waiting times and even cause deadlocks. Refer to Direct Feedthrough on page 122.

Checking block priorities

The execution order of the individual blocks can be controlled via block priorities and level 2 S-functions. Refer to Modifying the Block Execution Order on page 256.

A block priority violation can lead to warnings. Refer to How to Check for Illogical Block Priorities on page 258.

Related topics

Basics

Model Distribution Guidelines..... 114
Modifying the Block Execution Order..... 256
Swinging Buffer Protocol..... 118
Virtual Shared Memory Protocol..... 119

Direct Feedthrough

Basics

If the signal path from one IPCx block to another does not contain any block that corresponds to a state-space description with zero D matrix (for example, integrator, unit delay, or memory) this signal path is called a direct feedthrough term. Its output is directly dependent on its input; thus, the output cannot be calculated before the input data is available. An example of such a direct feedthrough term is shown below:

In conjunction with the Synchronized Swinging Buffer protocol, such direct feedthrough terms can cause undesired waiting times, as shown below, where the master processor contains a direct feedthrough term:

To eliminate such waiting times you have to avoid the direct feedthrough terms, as shown below:

For this model there is no waiting time, as you can see in the following illustration. This is due to the outputs that do not depend directly on the inputs. The time that was freed up is colored in gray:

Note

Direct feedthrough terms in conjunction with the Synchronized Swinging Buffer protocol can cause deadlock situations.

Example

Consider the following model (with both IPC channels using the Synchronized Swinging Buffer protocol):

In this model there are direct feedthrough terms for both CPUs. Such a signal path is called an algebraic loop and causes a deadlock situation because each CPU is waiting for input data from the other before being able to send output data.

Related topics

Basics

Interprocessor Communication Using IPC Blocks..... 130
Model Evaluation..... 120
Swinging Buffer Protocol..... 118
Virtual Shared Memory Protocol..... 119

Implementing Interprocessor Communication Using IPC Blocks

Introduction

When you implement interprocessor communication, you must know how to distribute models to different CPUs.

Where to go from here

Information in this section

Communication with a Sample Time Transition..... 124

Interprocessor Communication Using IPC Blocks..... 130

How to Implement Interprocessor Communication Using IPC Blocks..... 132

How to Make IPCx Blocks Use a Different Channel..... 133

How to Redirect all Member IPCs of a Channel to a New CPU..... 134

How to Define a Default CPU for Unconnected Model Parts..... 136

Communication with a Sample Time Transition

The step size

For each communication channel you can specify the step size as an integer multiple of the basic real-time step size. The step size defines when data exchange on this communication channel takes place.

Note

The step size of a communication channel must be an integer multiple of the step sizes of the connected source and destination CPUs.

Standard operation and downsampling

If both CPUs have identical step size multiples and you want to transfer data at a specific sample time (this corresponds to the behavior of a direct connection or a Zero Order Hold block), you have to enter identical numbers for the source and the destination step sizes (enter **1** for a direct connection). See also Example of standard operation of an IPCx block on page 126 and Example of downsampling via an IPCx block on page 126.

Different CPU step sizes

You can implement model parts with different sample times on various CPUs so that each CPU runs only one timer task and the single timer task mode is sufficient. As a result, you do not need the multiple timer task mode as often as you do with single-processor systems. The major benefit is that you do not need the Unit Delay block for the transition from the slow to the fast sampling rate, which avoids time delays. In this case, the source and destination step sizes are different. See also Rate Transitions on page 70 and Example of an IPCx block with different CPU step sizes on page 128.

Note

The IPCx blocks with their *port-based sample time* concept automatically adapt the sample times of their inports and outports to the step size multiples of their related CPUs.

Buffer and unbuffer mode

You can also use the **IPCx** block to achieve data buffering or unbuffering. You must then specify a value greater than 1 in the **Buffered samples** edit field of the **IPCx** block. The IPC buffer or IPC unbuffer mode is selected automatically for an IPC connection as follows:

- Step size of source CPU < step size of destination CPU

  The IPC connection is operated in IPC buffer mode. For example, if the step size multiples are 1 for the source CPU and 10 for the destination CPU, and the number of buffered samples is 10, a vector of 10 input values is collected and passed to the destination CPU at every 10th simulation step.
- Step size of source CPU > step size of destination CPU

  The IPC connection is operated in IPC unbuffer mode. For example, if the step size multiples are 10 for the source CPU and 1 for the destination CPU, and the number of buffered samples is 10, a vector of 10 values is passed from the source to the destination CPU at every 10th simulation step. In each basic simulation step, the destination CPU uses one of the ten values as the model input.

You can use the **Buffered samples** edit field of the **Communication Channel Setup** dialog to define how many signal samples are to be buffered or unbuffered on an IPC connection.

Note

Make sure that you transfer signals of the double data type and use the swinging buffer protocol. The IPCx block does not support other data types or the virtual shared memory protocol if operated in the IPC buffer or IPC unbuffer mode.

See also Example of IPC buffer/unbuffer mode on page 129.

Example of standard operation of an IPCx block

The following illustration shows the behavior of an IPC channel for standard operation:

- Step size multiple of source CPU = 1
- Step size multiple of destination CPU = 1
- Step size multiple of channel for source CPU = 1
- Step size multiple of channel for destination CPU = 1
- Buffered samples = 1

Example of downsampling via an IPCx block

The following illustration shows the behavior of an IPC channel for downsampling:

- Step size multiple of source CPU = 1
- Step size multiple of destination CPU = 1
- Step size multiple of channel for source CPU = 4
- Step size multiple of channel for destination CPU = 4
- Buffered samples = 1

Example application with downsampling

Suppose you want to operate the two CPUs of a multiprocessor system with the same step sizes (100 µs, the same as the base step size). There are two signals that you want to transfer from CPU master to CPU slave – one signal has to be transferred every 100 µs, the other at least every 500 µs. The following illustration shows a simplified model (the two signals have been replaced by a Sine Wave block). The different colors correspond to the sample times:

Because you want to operate the two IPCs at different sample times, you have to use two separate communication channels. Their settings are as follows:

- `master to slave on channel 0` (IPC1) has to operate at 100 µs. Because this is the base step size, you have to enter a step size multiple of `1` for the communication channel.
- `master to slave on channel 1` (IPC2) has to operate at 500 µs. This is five times the base step size, so you have to enter a step size multiple of `5` for the communication channel.

The following illustration shows the Communication Channel Setup dialogs for both IPCx blocks:

IPC1 uses the step size multiple 1

IPC2 uses the step size multiple 5

This illustration shows the resulting signal plots on the CPU slave for both signals, along with the turnaround time, which shows peaks whenever the slow IPC transfers the data:

Example of an IPCx block with different CPU step sizes

The following illustrations show the behavior of an IPC channel for CPUs operated with different step sizes.

Source CPU faster than destination CPU

- Step size multiple of source CPU = 1
- Step size multiple of destination CPU = 4
- Step size multiple of channel for source CPU = 1
- Step size multiple of channel for destination CPU = 4
- Buffered samples = 1

Source CPU slower than destination CPU

Note

In the example below, the destination CPU cannot read a current value until the 4th step, which is the worst case. The destination CPU tries to read a current value in each step. If no current value is available, the destination CPU uses the value of the preceding step.

- Step size multiple of source CPU = 4
- Step size multiple of destination CPU = 1
- Step size multiple of channel for source CPU = 4
- Step size multiple of channel for destination CPU = 1

- Buffered samples = 1

Example of IPC buffer/unbuffer mode

The following illustrations show the behavior of an IPC channel for CPUs operated with different step sizes and the IPC channel operated in the buffer/unbuffer mode.

Source CPU faster than destination CPU and IPCx block in buffer mode

- Step size multiple of source CPU = 1
- Step size multiple of destination CPU = 4
- Step size multiple of channel for source CPU = 1
- Step size multiple of channel for destination CPU = 4
- Buffered samples = 4

Source CPU slower than destination CPU and IPCx block in unbuffer mode

- Step size multiple of source CPU = 4
- Step size multiple of destination CPU = 1
- Step size multiple of channel for source CPU = 4
- Step size multiple of channel for destination CPU = 1
- Buffered samples = 4

For further examples refer to the IPC buffer/unbuffer mode demo model of RTI-MP.

Related topics

Basics

Interprocessor Communication..... 115
Interprocessor Communication Using IPC Blocks..... 130

Examples

Example Applications Requiring the Multiple Timer Task Mode..... 53

References

IPCx (RTI and RTI-MP Implementation Reference 🕮)

Interprocessor Communication Using IPC Blocks

IPCx block

An IPCx block defines the boundary between the different CPUs. Its Block Parameters dialog, the Communication Channel Setup dialog, is shown below:

Adding IPCx blocks

For each communication channel, you can specify the protocol to be used, the step sizes and whether you want to operate the channel in buffered/unbuffered mode. For details, refer to Swinging Buffer Protocol on page 118, Virtual Shared

Memory Protocol on page 119 and Communication with a Sample Time Transition on page 124.

You can also specify the block width and the data type to be used. Technically, there is no difference if you connect several signal lines via one single IPCx block (using multiple ports or a vectorized port) or via several separate blocks with the same channel number. The **Communication Channel Setup** dialog shows the total **Number of signals** for all member IPC blocks.

Connection, channel and IPCx block

The following illustration shows the relationship between the Communication Connection, Communication Channel and IPCx blocks. It summarizes the effects if you change the connection settings in the Communication Channel Setup dialog or in the Change Connection subdialog of an IPCx block:

Related topics

Basics

Direct Feedthrough 122
Gigalink Connection 116
Swinging Buffer Protocol 118
Virtual Shared Memory Protocol 119

HowTos

How to Configure Models for MP Systems 111
How to Define a Default CPU for Unconnected Model Parts 136
How to Implement Interprocessor Communication Using IPC Blocks 132
How to Make IPCx Blocks Use a Different Channel 133

How to Redirect all Member IPCs of a Channel to a New CPU..... 134

References

IPCx (RTI and RTI-MP Implementation Reference 🕮)

How to Implement Interprocessor Communication Using IPC Blocks

Objective

To implement interprocessor communication, you must add IPCx blocks from the RTI-MP Blocks library to your model.

Method

To implement interprocessor communication using IPCx blocks

1 Add as many IPCx blocks as you need from the RTI-MP Blocks library to the root level of your model.

You can also add IPCx blocks to function-call subsystems that are driven by interprocessor interrupts. For details, refer to How to Implement Interprocessor Interrupts on page 143.

2 Open the Communication Channel Setup dialog for the different IPCx blocks.

The currently selected IPCx block is highlighted in the model and its name in the Member IPC blocks frame of the Communication Channels Setup dialog is displayed in a bold font.

You can easily locate the member IPC blocks via Show Block.

3 Specify the settings according to your requirements: for each member IPC block, specify the number of ports, the widths of the ports and the data type.

4 Check if the settings of the communication channel (the connection to be used, the protocol and the number of buffered samples) meet your requirements.

Note

If you specify a number greater than 1 for the buffered samples, you have to make all member IPC blocks use the double data type.

Result

You implemented interprocessor communication by adding IPCx blocks to your model and specifying them.

Next step

You can make a single IPCx block use a different communication channel. For instructions, refer to How to Make IPCx Blocks Use a Different Channel on page 133.

Related topics

Basics

Direct Feedthrough 122
Gigalink Connection 116
Swinging Buffer Protocol 118
Virtual Shared Memory Protocol 119

HowTos

How to Configure Models for MP Systems 111
How to Define a Default CPU for Unconnected Model Parts 136

References

IPCx (RTI and RTI-MP Implementation Reference 🕮)

How to Make IPCx Blocks Use a Different Channel

Objective

If you want to make an IPCx block use a different channel, you must modify its connection.

Method

To make an IPCx block use a different channel

1. Open the Communication Channel Setup dialog of the IPCx block you want to modify.
2. Open the Change Connection dialog of the IPCx block you want to modify.

Enter the source and destination CPU for this IPC

3 Specify the desired **Source CPU**, **Destination CPU** and **Channel** for the IPCx block. The logical channel numbers are mapped to the channels of the available Gigalink connections via the network topology file.

If this communication channel already exists, the IPCx block uses the existing channel and appears in the **Member IPC blocks** frame of the corresponding **Communication Channel Setup** dialog.

If this communication channel is new to the model, the IPCx block uses a new channel without further member IPC blocks.

Note

If you add a new CPU to the model, you also need to configure the corresponding CPU's page of the Multiprocessor Setup dialog (see How to Configure Models for MP Systems on page 111).

Result

You changed the channel of a single IPCx block.

Next step

You can redirect all member IPCs of a specific communication channel to a new source or destination CPU. For instructions, refer to How to Redirect all Member IPCs of a Channel to a New CPU on page 134.

Related topics

Basics

Direct Feedthrough 122
Gigalink Connection 116
Swinging Buffer Protocol 118
Virtual Shared Memory Protocol 119

HowTos

How to Configure Models for MP Systems 111
How to Define a Default CPU for Unconnected Model Parts 136

References

IPCx (RTI and RTI-MP Implementation Reference 🕮)

How to Redirect all Member IPCs of a Channel to a New CPU

Objective

If you want to use all member IPCs of a specific channel with a new CPU, you must redirect them.

Method

To redirect all member IPCs of a channel to a new CPU

1 Open the Communication Channel Setup dialog of one of the member IPC blocks.

2 Enter the new settings for the desired Source CPU, Destination CPU and Channel in the Communication Channel frame. The logical channel numbers are mapped to the channels of the available Gigalink connections that are specified in the network topology file.

The communication channel is redefined according to your settings. If you enter a CPU that is new to the model, the corresponding CPU's page is added to the Multiprocessor Setup dialog.

Note

- If you add a new CPU to the model, you also need to configure the corresponding CPU's page of the Multiprocessor Setup dialog. For instructions, refer to How to Configure Models for MP Systems on page 111.
- The field for the application name is initially empty. Here you must insert the name of the object file to be generated. All other fields for the new CPU are initialized with the following default values:
  - Step size multiple = 1
  - Solver = ode1

Result

You redirected all member IPCs of a specific channel to a new CPU.

Example

Suppose you want to use a third processor board (named `slave_b`) for a multiprocessor model (CPU names `master` and `slave`). Ensure that the Multiprocessor Setup dialog is closed. Copy an existing IPCx block, and open its Change Connection dialog. Enter the name `slave_b` in the Source CPU or Destination CPU edit field. The Communication Channel Setup dialog is adapted accordingly, and in the current Simulink model, the IPCx block displays the name of the new CPU. The next time you open the Multiprocessor Setup dialog, a new CPU's page named slave_b is available in this dialog.

Related topics

Basics

Direct Feedthrough 122
Gigalink Connection 116
Swinging Buffer Protocol 118

Virtual Shared Memory Protocol..... 119

HowTos

How to Define a Default CPU for Unconnected Model Parts..... 136

References

IPCx (RTI and RTI-MP Implementation Reference 🕮)

How to Define a Default CPU for Unconnected Model Parts

Objective

If you want to assign unconnected model ports to a default CPU, you must use a **Default CPU** block.

The Default CPU block

You can use the Default CPU block to assign any stand-alone part of your model to a default CPU. A model part is stand-alone if it is left without a connection to an IPCx block.

Note

If there are stand-alone parts in your model that are not assigned to a default CPU, RTI-MP assigns them to the master CPU by default.

Method

To define a default CPU for unconnected model parts

1 The **Default CPU** block is located in the **RTI-MP Blocks** library.
   - To affect all stand-alone blocks of the entire model, including the subsystems, place a single **Default CPU** block on the root level.
   - To define a separate default CPU for a specific subsystem and all of its subordinate subsystems, place an additional **Default CPU** block inside that subsystem. You can use one **Default CPU** block per subsystem. All

subsystems holding a **Default CPU** block have to be placed on the root level.

2 Open the **Block Parameters** dialog and specify the desired **CPU**.

Note

If you add a new CPU to the model, you also need to configure the corresponding CPU's page of the **Multiprocessor Setup** dialog (see How to Configure Models for MP Systems on page 111).

Related topics

Basics

Interprocessor Communication Using IPC Blocks..... 130

HowTos

How to Configure Models for MP Systems..... 111

References

Default CPU (RTI and RTI-MP Implementation Reference 🕮)
IPCx (RTI and RTI-MP Implementation Reference 🕮)

Implementing Interprocessor Communication Using Gigalink Blocks

Introduction

If you want to transfer data between real-time applications running on DS1006, DS1007, or SCALEXIO platforms, you can use Gigalink blocks.

Where to go from here

Information in this section

Interprocessor Communication Using Gigalink Blocks..... 138

How to Implement Interprocessor Communication Using Gigalink Blocks..... 141

Interprocessor Communication Using Gigalink Blocks

RTI Gigalink blockset

Interprocessor communication can be realized via Gigalink blocks. They are contained in the RTI Gigalink Blockset, which is part of the RTI-MP library. The RTI Gigalink Blockset is available for the DS1006 and the DS1007 platform and allows you to transmit signals between real-time applications via a Gigalink connection. You can use the blockset to synchronize an application running on a DS1006 or DS1007 with an application running on a SCALEXIO system.

Use cases

Data transmission With Gigalink blocks, signals can be transmitted in blocking or non-blocking communication mode between applications using processor boards of the same type or of different types. In a multiprocessor system, the Gigalink blocks can use any Gigalink connection that is specified in the MP network topology, even if the connection is realized via a virtual Gigalink of a multicore board. Signals can also be transmitted between applications running on different processor boards. Thus, it is possible to connect a DS1006 multiprocessor system with a DS1007 processor board running a single-processor application, for example.

With the RTI Gigalink blockset you can implement the following communication scenarios.

Connection Scenarios	From …	To …
Data Connection Without Task Synchronization		
The non-blocking receive mode is applicable. In this connection scenarios the timer tasks between communicating applications are not synchronized. Between non-synchronized tasks a blocking communication will lead to task overruns.	DS1006	DS1006
	DS1006	DS1007
	DS1006	SCALEXIO
	DS1007	DS1007
	DS1007	DS1006
	DS1007	SCALEXIO
	SCALEXIO	DS1006
	SCALEXIO	DS1007
	SCALEXIO	SCALEXIO[1)]
Data Connection With Synchronization of Timer Tasks		
The timer tasks in the DS1006/DS1007 application are synchronized by a timer event sent by the connected SCALEXIO system. The blocking receive mode is applicable.	SCALEXIO	DS1006
	SCALEXIO	DS1007

[1)] RTI and RTI Gigalink Blockset are not used in this scenario.

Synchronized communication between SCALEXIO and PHS-bus applications The Gigalink_Interrupt block can be used to receive a synchronization interrupt from a SCALEXIO system via Gigalink connection. This allows you to synchronize the timer task execution between one application running on a DS1006 or DS1007 system and the other application running on a SCALEXIO. The Gigalink_Interrupt block can only be connected to a Timer

Task Assignment Block. The related Gigalink_Receive block has to be set to *blocking* communication mode. The Gigalink blocks used for the synchronization have to be configured with the same Gigalink port, channel number and interrupt number.

Note

If you use the Timer Task Assignment block, you must ensure that the simulation step size of the model and the step size of the external trigger are identical.

These features are not applicable for PHS-bus to PHS-bus communication.

For information on implementing the Gigalink connection with the SCALEXIO system, refer to Building the Signal Chain for Gigalink Communication (ConfigurationDesk Real-Time Implementation Guide 🕮).

Gigalink block types

The RTI Gigalink Blockset contains the following blocks:

- Gigalink_Send block
  Sends data to a specified Gigalink channel.
- Gigalink_Receive block
  Receives data from a specified Gigalink channel.
- Gigalink_Status block
  Checks the connection status of the specified Gigalink number.
- Gigalink_Interrupt block
  Receives an interrupt from a specified Gigalink channel.

For details on the Gigalink blocks, refer to RTI Gigalink Blockset Reference (RTI and RTI-MP Implementation Reference 🕮).

Implementing interprocessor communication with Gigalink blocks

If you want to implement interprocessor communication with Gigalink blocks, you need a block pair consisting of a Gigalink_Send block and a Gigalink_Receive block. The Gigalink_Send block must be placed in the model you want to transmit data from and the Gigalink_Receive block in the model you want to send data to. In a multiprocessor model, the blocks must be assigned to a CPU according to the standard model distribution guidelines of RTI-MP. For details, refer to Model Distribution Guidelines on page 114.

Note

To make data transmission between models possible, you must specify the same channel number and signal width for the Gigalink_Send and Gigalink_Receive block pair. Be aware that there is no consistency check.

If wanted, you can place an optional Gigalink_Status block in one of your models to check whether data transmission is possible at all. Depending on the

availability of Send and Receive blocks for your specified Gigalink number, the outputs of the Gigalink_Status block can be as follows:

- Output is 1: The specified Gigalink channel is ready for data transmission. The **Gigalink_Receive** block using this Gigalink channel has a counterpart that sends data.
- Output is 0: The specified Gigalink channel is not ready for data transmission. The **Gigalink_Receive** block using this Gigalink channel has no counterpart that sends data. The **Gigalink_Receive** block returns an old value from the buffer or 0 if the buffer is empty.

Limitations with the Gigalink Blockset

When implementing Gigalink connections, note the following limitations:

- All processor boards must be equipped with Gigalink modules.
- Consistency of data transfer is ensured by the swinging buffer protocol in blocking or non-blocking mode. For details, refer to Swinging Buffer Protocol on page 118.

  Blocking communication should be combined with the synchronization of timer tasks. This use case is only supported by SCALEXIO to DS1006/DS1007 connections.
- Signals must be scalar or vectorial and of `double` data type. Otherwise an error message appears during model initialization.
- At most 1024 signals of `double` data type can be transmitted via a Gigalink channel.
- Each combination of Gigalink number and channel number can be used for a block only once. Otherwise an error message appears during the build process.
- If Gigalink_Send and Gigalink_Receive blocks and RTI-MP blocks for interprocessor communication use the same Gigalink number, they cannot use the same channel numbers. The RTI Gigalink Blockset and RTI-MP do not check the channel numbers and the related Gigalink number. You must ensure that the channel numbers used for RTI-MP interprocessor communication are not the same channel numbers used by the RTI Gigalink Blockset Send and Receive blocks.
- The RTI Gigalink Blockset does not support MicroLabBox.

Related topics

Basics

Gigalink Connection..... 116
Model Distribution Guidelines..... 114

HowTos

How to Implement Interprocessor Communication Using Gigalink Blocks..... 141

References

RTI Gigalink Blockset Reference (RTI and RTI-MP Implementation Reference 🕮)

How to Implement Interprocessor Communication Using Gigalink Blocks

Objective

To implement interprocessor communication via Gigalink blocks, you must add them from the RTI Gigalink blockset to your models. The **Gigalink_Send** and **Gigalink_Receive** blocks are mandatory for implementing interprocessor communication. The **Gigalink_Status** and **Gigalink_Interrupt** blocks, however, are optional.

Method

To implement interprocessor communication using Gigalink blocks

1. Add a **Gigalink_Send** block from the RTI Gigalink blockset to the model you want to transmit data from.
2. Open the **Block Parameters** dialog of the **Gigalink_Send** block and specify the settings according to your requirements. A possible model design could look like this:

3. Add a **Gigalink_Receive** block from the RTI Gigalink blockset to the model you want to send data to.
4. Open the **Block Parameters** dialog of the **Gigalink_Receive** block and specify the settings according to your requirements. A possible model design could look like this:

Note

The **Channel number** and **Signal width** must be identical for the Gigalink block pair (Send block and Receive block).

5. If wanted, you can add a **Gigalink_Status** block from the RTI Gigalink Blockset to one of your models. For example, add a **Gigalink_Status** block to the model you want to send data to.
6. Open the **Block Parameters** dialog of the **Gigalink_Status** block and specify the Gigalink number which you use for your interprocessor communication. A possible model design could look like this:

7 If you want to synchronize a DS1006 or DS1007 application with a SCALEXIO application via Gigalink channel, you can add a **Gigalink_Interrupt** block from the RTI Gigalink Blockset to the DS1006 or DS1007 model and connect it to a **Timer Task Assignment** block. The **Timer Task Assignment** block uses the Gigalink event as trigger source for the base rate task. The timer event of the SCALEXIO application has to be sent via the same Gigalink channel. To achieve synchronization, the related **Gigalink_Receive** block has to be set to *blocking* communication mode, otherwise the communication is unsynchronized.

Result

You implemented interprocessor communication by inserting Gigalink blocks in two different models and specifying them. If you use a **Gigalink_Status** block, its output depends on the availability of Send and Receive blocks for the specified Gigalink channel. For the synchronization with a SCALEXIO application, the **Gigalink_Interrupt** block is used. For details on the different outputs, refer to Interprocessor Communication Using Gigalink Blocks on page 138.

Related topics

Basics

Gigalink Connection..... 116
Interprocessor Communication Using Gigalink Blocks..... 138

References

Gigalink_Interrupt (RTI and RTI-MP Implementation Reference 🕮)
Gigalink_Receive (RTI and RTI-MP Implementation Reference 🕮)
Gigalink_Send (RTI and RTI-MP Implementation Reference 🕮)
Gigalink_Status (RTI and RTI-MP Implementation Reference 🕮)

Interprocessor Interrupts

Introduction

Two types of tasks exist in real-time applications: timer tasks and tasks driven by interrupt blocks. For dSPACE multiprocessor systems, you usually do not need to change anything for the timer tasks. However, tasks driven by interrupt blocks require a multiprocessor-specific treatment.

Where to go from here

Information in this section

How to Implement Interprocessor Interrupts..... 143

How to Change the Task Priorities in RTI-MP Models..... 145

Information in other sections

Handling Tasks..... 41

Timer Tasks..... 48

Tasks Driven by Interrupt Blocks..... 57

Interprocessor Communication..... 115

Limitations with RTI-MP..... 276

How to Implement Interprocessor Interrupts

Interprocessor interrupts

For all hardware, software or timer interrupts that are transferred from one CPU to another, you have to insert an Interprocessor Interrupt block (IPI block) into the connection lines between the interrupt block and the corresponding function-call subsystem.

The IPI block is not required for local interrupts, i.e., if the interrupt source and destination are located on the same CPU.

Example

Consider the throttle control example (refer to Throttle Control Example on page 16). Suppose you want the emergency stop button (simulated by a step on the slave processor) to trigger the corresponding task on the master board. The necessary configuration is shown in the illustration below (the actual control loop is left out for clarity):

Preconditions

The following preconditions must be fulfilled:

- A **Hardware**, **Software** or **Timer Interrupt** block must be configured as usual. For instructions, refer to How to Use Hardware Interrupts on page 62 and How to Use Software Interrupts on page 65.
- The interrupt block must be on the root level of the model.

Method

To implement interprocessor interrupts

1 Add an **IPI** block from the RTI-MP Blocks library to the root level of your model and connect it directly to the interrupt source.

2 Add a function-call subsystem from RTI's TaskLib to the root level of your model and connect it directly to the IPI block.

Note

No block other than a single IPI block is allowed between the interrupt source and function-call subsystem. You can drive only one subsystem with a single interrupt.

3 Open the function-call subsystem, and add the desired part of your model. If you want to assign different parts of this task to different CPUs, you can use **IPC** blocks and **Default CPU** blocks as though the function-call subsystem was the root level of the model.

4 Open the **Block Parameters** dialog of the IPI block and specify the **Source CPU** and all Target CPUs involved in the execution of this task. You can find the involved Target CPUs by analyzing the IPC blocks and/or Default CPU blocks that the function-call subsystem might contain. If there are too few Target CPU edit fields you can expand the dialog by means of the **Maximum number of Target CPUs** drop-down list.

The following illustration shows an example:

Result

When you start the build process, **IPI Send** and **IPI Receive** blocks are automatically added to the created submodels of the target CPUs. They are configured according to the related **IPI** block in the main model.

Related topics

HowTos

How to Use Hardware Interrupts 62
How to Use Software Interrupts 65

References

Function-Call Subsystem Block (RTI and RTI-MP Implementation Reference 🕮)
Hardware Interrupt Block (RTI and RTI-MP Implementation Reference 🕮)
IPI (RTI and RTI-MP Implementation Reference 🕮)
Software Interrupt Block (RTI and RTI-MP Implementation Reference 🕮)
Timer Interrupt Block (RTI and RTI-MP Implementation Reference 🕮)

How to Change the Task Priorities in RTI-MP Models

Objective

If your model contains one or more real-time tasks, you must specify priorities for each of them. If a task belongs to more than one CPU, it can have different priorities on the various CPUs.

Overruns and deadlocks

By choosing illogical priorities for the tasks of your model you might cause overrun or even deadlocks. To avoid this, it is important to assign the same priority order to tasks that are calculated on multiple CPUs.

Notes on modeling

The submodels of the target CPUs contain IPI Receive blocks instead of copies of the related interrupt blocks. Timer Task Assignment blocks are also not copied.

There are therefore some points to be considered:

- A task is represented by its IPI Receive block in the RTI Task Configuration dialog.
- By specifying an intuitive name for the IPI block in the main model, you can easily identify the corresponding task.
- Because the related Gigalink module is specified as the interrupt source, the task configuration does not need to be adapted when you replace the related interrupt in the source CPU model.

Specifying the task configuration

You have to specify the task configuration individually for each CPU. The handling of the settings is identical to that of single-processor systems. For details, refer to How to Change the Task Priorities on page 89. However, you cannot open them via the Solver dialog of your model's configuration set in the Model Explorer because this dialog is not valid for real-time simulation on multiprocessor systems. Instead, you must carry out the following instructions:

Method

To change the task priorities in an RTI-MP model

1. Open the Multiprocessor Setup dialog.

2. Change to the individual CPU's page, and open the RTI Task Configuration dialog for the current CPU.
3. Arrange the priorities according to your requirements, and close the RTI Task Configuration dialog.
4. Repeat steps 2 and 3 for each CPU included in your model.

Related topics

Basics

Overrun Situation and Turnaround Time 45
Tasks Driven by Interrupt Blocks 57
Timer Tasks 48

HowTos

How to Configure Tasks via Scripts 92

How to Specify Overrun Strategies for Tasks.....90

References

RTI Task Configuration Dialog (Multiprocessor Setup Dialog) (RTI and RTI-MP Implementation Reference 📖)

Referencing Models

Introduction

When you use model referencing, you must note some specifics to ensure your models are properly implemented, simulated and built.

Where to go from here

Information in this section

Basics on Model Referencing.....150
How to Create Model References.....154
Creating a Model Reference from an Atomic Subsystem.....155
Configuring Tasks with Model Referencing.....156
How to Execute Referenced Models in Interrupt-Driven Tasks.....157
Simulating and Building Models Containing Model Blocks.....160

Information in other sections

Limitations with Model Referencing.....279
File Specifics for Model Referencing (RTI and RTI-MP Implementation Reference 🕮)
Available Variables in the Variable Description File (RTI and RTI-MP Implementation Reference 🕮)
To show rules that define whether a variable is included in the variable description file.

Basics on Model Referencing

Defining model referencing

Model referencing means that you include models in other models by using **Model** blocks. Each Model block represents a reference to another model and displays the root level inputs and outputs of the model it references. Model blocks are provided by Simulink. The following example shows a Model block that represents a reference to a model with one root level input and one root level output port respectively. For a Model block, you can specify the simulation mode. In *Accelerator* simulation mode, the edges of the Model block are filled, in *Normal* simulation mode, they are unfilled.

RTI and RTI-MP also support code generation of protected models (SLXP files) and model hierarchies that contain referenced models as protected models. The following figure shows a **Model** block that represents a reference to a protected model.

A protected model can be additionally protected by password.

Note the following limitations when you use protected models:

- A model that contains a protected model can only be built if the protected model contains its generated code. The build options of the protected model, especially the system target file, must suit to the parent model.
- RTI and RTI-MP do not support **Binaries code generation** option for protected models.
- In Simulink, protected models can contain other protected referenced models. Nested protected models are not supported.
- You cannot use USR TRC files and USR Make files for protected models.

A model containing a Model block can be the top-level model in a model referencing hierarchy. Alternatively, it can be located on a lower level in the hierarchy and fulfill the role of a parent model which references one or more other models.

A parent model can reference the same model multiple times. However, it is not possible for a Model block to refer back to the model that it references.

As shown below, model referencing allows you to organize large models hierarchically. During simulation and code generation, a referenced model effectively replaces the Model block that references it.

Model referencing is useful if it is possible to subdivide a large model into smaller parts which can be implemented as separate models. You can then easily include these models in other models via Model blocks. Additionally, the protection feature of referenced models allows you to hide the intellectual property of the model.

The same model that is referenced within an RTI model can also be referenced within an RTI-MP model. This is possible without modifications for the referenced model if all models are generated for the same hardware platform.

Note

- Each referenced model can also, of course, be used as a top-level model and can be built accordingly.
- The code built for a referenced model will not be reused if the same model is later used as a top-level model, and vice versa.
- With RTI-MP, the names of the submodels and the referenced models must be unique.

Advantages of model referencing

If you organize large models by means of model referencing, you can benefit from advantages like:

- Modular development

  Complex models can be developed modularly, i.e. each referenced model can be implemented independently of the model that references it.
- Inclusion by reference

  The functionality of a model can be included in one or several different models multiple times without having to make redundant copies.
- Incremental loading

  Simulink reduces load times and memory usage on the host PC by using incremental loading for referenced models. When you open a top-level model, any models to which it refers are not loaded into memory until they are needed or you open them.

- Incremental code generation

  Depending on the rebuild options for referenced models, the code of your referenced models is rebuilt only if your models have changed. This feature, called incremental code generation, prevents the unnecessary code generation of models which are up-to-date.

  Note

  In certain situations the Simulink Coder is not able to detect relevant modifications.

  For example:

  - Installing hotfixes, patches or service packs for the active RCP & HIL installation.
  - Renaming of signal labels

  To force generating the entire code, you must delete the build folders `slprj` and `<mdl>_rti1xxx` for your models before starting a new build process. You must also delete the <mdl>.slxc file. To do so, you can use `rti_build2` or `rtimp_build2` with the `CleanUpAll` method.

- Modular build process

  During the build process for a top-level model, this model is not generated and compiled as a whole, but all the models in the model referencing hierarchy are generated and compiled individually. Therefore, less memory capacity is needed on your host PC for code generation and compiling processes.

RTI block support for model referencing

The use of RTI blocks is supported for top-level models only. RTI blocks are not supported in referenced models except for the TRC Exclusion block from RTI's Extras Library.

With RTI-MP, it is also not possible to use IPI, IPC and other RTI-MP blocks in a referenced model. A Model block can therefore be assigned to one CPU only. The same referenced model can be assigned to several CPUs by inserting several instances of this model (Model blocks) into the MP main model.

Tunable block parameters

RTI treats the **Default parameter behavior** option similarly for top-level models and referenced models in a model referencing hierarchy.

Entries for signals and block parameters in the referenced model are generated into the variable description only, if the **Total number of instances per model** option in the **Model Referencing** dialog is set to *One*.

For details on tunable parameters in the variable description file, refer to *Rules for model referencing* in Available Variables in the Variable Description File (RTI and RTI-MP Implementation Reference 🕮).

Parameterizing referenced models

You can use model arguments in your instances of Model blocks. They enable you to define different behavior for various references to the same model. Model arguments are available in the variable description.

For further information, refer to the Simulink documentation.

Model dependency viewer

To facilitate your work with model referencing models, you can use the model dependency viewer. It displays model referencing hierarchies graphically as shown in the following example:

To display a dependency view for a model, open the model and select **View – Model Dependency Viewer - Referenced Model Instances** from the model editor's menu bar.

Alternatively, the command **`find_mdlrefs`** finds all models directly or indirectly referenced by a given model.

Note

With RTI-MP, the Dependency Viewer should be used separately for the main model and the submodels to get a sufficient information.

For further information, refer to the Simulink documentation.

Related topics

Basics

Available Variables in the Variable Description File (RTI and RTI-MP Implementation Reference 🕮)
Configuring Tasks with Model Referencing 156
Creating a Model Reference from an Atomic Subsystem 155

Simulating and Building Models Containing Model Blocks..... 160
Using Model Referencing..... 299

HowTos

How to Create Model References..... 154
How to Execute Referenced Models in Interrupt-Driven Tasks..... 157

References

File Specifics for Model Referencing (RTI and RTI-MP Implementation Reference 🕮)

How to Create Model References

0bjective

If you want to include a model in another model, you must create a model reference. A model can contain multiple model references which are realized via Model blocks.

Preconditions

The model you want to reference must be on the MATLAB path.

Method

To create a model reference

1 Open the model in which you want to create a model reference.

2 From Simulink's **Ports & Subsystems** block library, drag a Model block to your model.

3 Open the new Model block's parameter dialog and enter the name of the referenced model in the **Model name** edit field.

> **Note**
>
> With RTI-MP, the names of the submodels and the referenced models must be unique.

4 Also specify the other settings, if you require them.

> **Note**
>
> If the referenced model contains any root-level inputs or outputs, Simulink displays corresponding input and output ports on the Model block instance you have created. The same applies to trigger ports.

Result

You have created a reference to another model in your model. During simulation and code generation, the referenced model effectively replaces the Model block that references it.

Tip

If you want to open a referenced model, select the Model block that references it. Then select Open Model from the block's context menu.

Related topics

Basics

Basics on Model Referencing..... 150
Creating a Model Reference from an Atomic Subsystem..... 155

Creating a Model Reference from an Atomic Subsystem

Converting atomic subsystems to model references

You can convert your atomic subsystem to a model reference in the following ways:

- You can use the Convert to Model Block command from the atomic subsystem's context menu.
- Alternatively, you can use the Simulink command `Simulink.Subsystem.convertToModelReference`.

For further information, refer to the Simulink documentation.

Related topics

Basics

Basics on Model Referencing..... 150
Simulating and Building Models Containing Model Blocks..... 160

References

Diagnostics Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮)

Configuring Tasks with Model Referencing

Configuring tasks in a model referencing hierarchy

Task configuration with model referencing is generally performed in the same way as for models without Model blocks. The task configuration is, however, always performed from the top-level model of a model referencing hierarchy. Therefore open the **RTI Task Configuration** dialog from the top-level model's **Configuration Parameters** dialog on the **RTI simulation options** page.

For RTI-MP models, you must perform the task configuration for each submodel via the **Multiprocessor Setup** dialog.

Note that it is still possible to open the RTI Task Configuration dialog for referenced models. However, the task configuration settings made in referenced models are independent and not considered by the top-level model.

Sample time constraints

The actual sample time of a Model block depends on the model referenced by it. The referenced model can either have specific sample times or it can inherit its sample time from its parent model. This sample time constraint limits the use of a referenced model:

- If the referenced model defines specific sample times, its use in a parent model is restricted. The corresponding Model block cannot be placed in a triggered, function-call, or an iterator subsystem. In addition, you must ensure that blocks connected to it have the same sample time as your referenced model. This is necessary to avoid rate transition errors. Furthermore, the sample times of a referenced model need to be integer multiples of the fixed step size of the top-level model.
- If the referenced model inherits its sample time, the corresponding Model block can be placed anywhere in the parent model. It can particularly be used in periodic tasks and in interrupt-driven tasks.

To determine whether a model inherits its sample times, set the **Periodic sample time constraint** option on the **Solver** dialog of the **Configuration Parameters** dialog to *Ensure sample time independent*. If your model is unable to inherit sample times, this setting causes Simulink to display an error message during model initialization or within the RTI build process. For details, refer to the Simulink documentation.

Multiple asynchronous function calls

The **Asynchronous Task Specification** block allows you to implement an asynchronous function-call input to a **Model** block. With this feature, RTI interrupt blocks (hardware interrupts, timer interrupts, and software interrupts) can be used to drive input ports of a Model block.

The required **Inport** block on top level of the referenced model must be configured to *Output function call*. The **Task priority** parameter of the **Asynchronous Task Specification** block must be set to `[]` (empty value) to be used with RTI. You set the task priority and other task properties in the **RTI Task Configuration** dialog.

The following figure shows you an implementation example.

With RTI-MP, you have to ensure that signals and function-call inputs to a Model block belong to the same destination CPU.

Related topics

Basics

Basics on Model Referencing..... 150

HowTos

How to Execute Referenced Models in Interrupt-Driven Tasks..... 157

References

Model Configuration Parameters Dialogs (RTI and RTI-MP Implementation Reference 🕮)
Multiprocessor Setup Dialog (RTI and RTI-MP Implementation Reference 🕮)
RTI Task Configuration Dialog (RTI and RTI-MP Implementation Reference 🕮)

How to Execute Referenced Models in Interrupt-Driven Tasks

Objective

If you want to execute referenced models in interrupt-driven tasks, they must be triggered asynchronously via RTI Interrupt blocks.

Restriction

With RTI-MP, each Model block can only be assigned to one CPU. When using an RTI Interrupt Block to directly trigger a Model block, the connected IPI block has to be configured for one target CPU only.

Precondition

The Periodic sample time constraint option on the Solver dialog of the Configuration Parameters dialog must be set to *Ensure sample time independent* for the referenced model.

Possible methods

To execute referenced models in interrupt-driven tasks, you can choose between the following two methods:

- You can implement your referenced models as directly triggered. This means that you add a function-call port on the root level for each referenced model. For details, refer to Method 1.
- You can implement your referenced models as indirectly triggered. This means that the corresponding Model block is placed in a function-call subsystem that is triggered by an RTI Interrupt block. For details, refer to Method 2.

Method 1

To execute a referenced model in interrupt-driven tasks as directly triggered

1. Add a Trigger block to the root level of the referenced model and configure it as function-call.
2. In the top-level model, add a Hardware Interrupt block from RTI's block library.
3. Connect the output of the Hardware Interrupt block to the trigger port of the Model block.
4. If the referenced model contains root-level input and output ports, implement the data transfer between your referenced model and your top-level model in the following way:

   Insert a Simulink Rate Transition block in the top-level model. The Ensure data integrity during data transfer option controls whether the data transfer between the top-level model and the referenced model is protected. The top-level model could look like this:

   For a multiprocessor model it could look like this:

   For details, refer to Introduction to Interrupt-Block-Driven Tasks and Task Transitions on page 60.

Method 2

To execute a referenced model in interrupt-driven tasks as indirectly triggered

1. In the top-level model, place the Model block in a function-call subsystem from RTI's TaskLib.
2. Connect the Model block's input and output ports to the inports and outports of the function-call subsystem.

3 In the top-level model, add a Hardware Interrupt block from RTI's block library.

4 Connect the output of the Hardware Interrupt block to the trigger port of the function-call subsystem.

5 Implement the data transfer between the function-call subsystem and the surrounding top-level model. All the standard possibilities apply as described in How to Implement Unprotected Task Transitions on page 77. The top-level model could look like this:

For a multiprocessor model it could look like this:

Next step

You can now configure your tasks via the **RTI Task Configuration** dialog in the top-level model.

Related topics

Basics

Basics on Model Referencing 150
Configuring Tasks with Model Referencing 156
Handling Tasks 41

References

RTI Task Configuration Dialog (RTI and RTI-MP Implementation Reference 🕮)

Simulating and Building Models Containing Model Blocks

Simulation targets and code generation targets

For the simulation and code generation of models containing Model blocks, specific files are necessary. These files are the following:

- When you simulate in Normal mode, the models are directly interpreted and SIM targets are not generated.

  When you simulate in Accelerator mode, there must be simulation targets (SIM targets) available for each simulation on the host PC. A SIM target is a kind of S-function that computes the outputs of a referenced model during the simulation of its parent model. If the SIM target does not exist at the start of the simulation, Simulink automatically creates it.
- For code generation Simulink Coder targets (RTW targets) are required. An RTW target consists of C code which is generated for a referenced model. If the RTW target is not available at the start of the build process, Simulink Coder automatically creates it.

SIM targets and RTW targets are both stored in the Simulink project directory named **`slprj`**. It is a subdirectory of your current working directory.

The creation of targets, for example, for the initial build of a large model referencing hierarchy, takes time, but once the SIM and RTW targets are created and up to date, they need not be rebuilt. This makes all the subsequent build processes for the hierarchy faster.

You can easily control whether existing targets should always or never be rebuilt via the **Rebuild options** on the **Configuration Parameters - Model Referencing** dialog.

Starting the build process

The RTI or RTI-MP build process for models containing Model blocks can be started in the same way as for models without Model blocks. For RTI models, you can start it from the top-level model by clicking the **Build** command in the menu of the model, using the **`Ctrl + B`** shortcut key, or by entering the **`rti_build2`** command in the MATLAB Command Window. For details, refer to rti_build2 (RTI and RTI-MP Implementation Reference 🕮) or rtimp_build2 (RTI and RTI-MP Implementation Reference 🕮).

Building referenced models separately

It is possible to build a referenced model separately from its top-level model. This reference build can be started by using one of the following Simulink commands:

- `slbuild('<ModelName>', 'ModelReferenceSimTarget')`

  The simulation target (SIM) is built for your referenced model.
- `slbuild('<ModelName>', 'ModelReferenceRTWTarget')`

  The SIM target and RTW target are built for your referenced model.

Note

During reference builds, no separate application is created for referenced models. However, the code generated of the referenced model will be reused whenever the respective top-level model is built.

Building referenced models in parallel

If your host PC has a multicore CPU, you can use the Parallel Computing Toolbox from MathWorks® to let the build process for referenced models of a model referencing hierarchy execute in parallel.

For further information, refer to the Simulink user documentation from MathWorks.

Ensuring compatibility between top-level and referenced models

Model referencing requires that the **Include only Simulink.Parameter and Simulink.Signal objects with global storage class** option is identical between all models in a model referencing hierarchy.

During the build process, Simulink Coder compares the parameters between the top-level model and its referenced models. If they differ, Simulink Coder aborts the build process and displays an error message.

Tip

Compatibility between top-level and referenced models can be easily achieved via referenced configuration sets. All models that contain an activated configuration reference to a specific configuration set in the base workspace will share the same configuration settings. Using referenced configuration sets centralizes and simplifies the management of model configurations if several models are employed. For details, refer to How to Attach a Configuration Reference on page 31.

For details on the available options, refer to Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮).

For details on the variable description file, refer to File Specifics for Model Referencing (RTI and RTI-MP Implementation Reference 🕮) and Available Variables in the Variable Description File (RTI and RTI-MP Implementation Reference 🕮).

Related topics

Basics

Available Variables in the Variable Description File (RTI and RTI-MP Implementation Reference 🕮)
Building and Downloading the Model 219

Limitations with Model Referencing..... 279

HowTos

How to Attach a Configuration Reference..... 31

References

Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮)
File Specifics for Model Referencing (RTI and RTI-MP Implementation Reference 🕮)
Model Referencing Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮)

Inserting Custom C/C++ Code

Introduction

RTI is designed to provide automatic handling of dSPACE I/O board capabilities from a Simulink model. However, some specific hardware features are not supported by RTI. In this case, you can handle the appropriate hardware feature via a C-coded S-function or via RTI's User-Code in C or C++.

Where to go from here

Information in this section

Implementing S-Functions 164
Implementing User-Code 171
Integrating C++ Code 180
Comparing S-Functions and User-Code 186
Examples 190
Advanced Techniques 196

Information in other sections

Defining I/O Interfaces 33
Adding User-Specific Files, Source Files and Libraries 243

Implementing S-Functions

Introduction

S-functions are a feature of Simulink that allow you to implement user-defined functionality, for example, to incorporate existing C code into your Simulink model.

Where to go from here

Information in this section

Basics of S-Functions.....164
Purpose of the C Methods.....166
S-Functions for Real-Time and Simulink Simulations.....167
Accessing PHS-bus-based systems in S-Functions.....168
How to Access dSPACE I/O Devices in S-Functions.....169

Information in other sections

Implementing User-Code.....171
Comparing S-Functions and User-Code.....186
Execution Order of S-Functions and User-Code.....186
Limitations for Custom Code.....282
C Code for the Throttle Control Example's S-Function.....191

Basics of S-Functions

Definition of S-functions

S-functions (*Simulink functions*) provide a computer language description for Simulink's S-Function blocks. S-functions are written in a high-level language such as M, C, or Fortran.

Note

If you want to use an S-function with RTI, it must be written in C or C++. If you want to use C++, refer to Using C++ Code in an RTI Application on page 180.

S-functions can hold continuous, discrete and hybrid systems and interact with the equation solvers of MATLAB and Simulink.

Creating S-functions from scratch

If you require user-specific functionality that is not based on an S-function predefined by RTI, you have to create it from scratch by using Simulink's template `sfuntmpl_basic.c`.

Consider, for example, the throttle control example (refer to Throttle Control Example on page 16) and suppose you want to include a checking routine that issues a message if the throttle valve position exceeds a certain value. It is no problem to implement the checking routine in the Simulink model. However, no Simulink block provides message output from a real-time application. Thus, you have to use custom C code to achieve the desired behavior.

S-functions can be included in timer tasks or tasks driven by an interrupt block because you can place Simulink's S-Function blocks anywhere in your model depending on their sample time setting. In conjunction with the dSPACE real-time simulation you can only use C-coded S-functions.

Types of S-functions

Simulink knows two types of S-functions:

Noninlined S-functions This type of S-function (also called C-MEX S-function) is used by Simulink and Simulink Coder, which call the various S-function methods during the model execution. Both Simulink and Simulink Coder use the same S-function code. This guide deals exclusively with this type of S-function.

Wrapper and fully inlined S-functions These types of S-functions make use of the Target Language Compiler code generation tool. Fully inlined S-functions must be implemented separately for Simulink (C-MEX S-function) and Simulink Coder (TLC file). These types of S-functions are not described in this guide. For information, refer to the documentation of MathWorks®.

S-function templates

Since all S-functions must have a particular set of methods, you should always start your programming with an S-function template.

- If you want to create a new S-function (such as in the level control example), take the S-function template `sfuntmpl_basic.c` file from the `<MATLAB_ROOT>\simulink\src\` folder.
- If you want to modify an RTI-specific S-function, take the desired original S-function C file from the `<RCP_HIL_InstallationPath>\MATLAB\RTI\RTI<XXXX>\SFcn\` folder.

Note

The original S-functions are restored and updated during any new installation. So before you modify any template, you should copy it to the working folder that holds your model and rename the template. It is then available for all models located in that folder.

Related topics

Basics

Execution Order of S-Functions and User-Code..... 186

Purpose of the C Methods

Available C methods

An S-function consists of a fixed set of `mdl<...>()` methods that are called by the `Mdl<...>()` model functions, which Simulink Coder generates from the overall model. The set of `mdl<...>()` methods in an S-function can be considered as a model within the model.

The following C methods are available:

C Method	Purpose
mdlStart()	Contains the initialization code of the S-function, which is executed once when simulation is started. You can use it for I/O initialization, for example `dsXYZ_init()` or `dsXYZ_range_set()`.
mdlInitializeConditions()	Sets the initial values of the state variables of the S-function. It is executed once for each simulation start. If the S-Function block is in an enabled subsystem that is configured as "reset states when enabled", it is also executed whenever the subsystem becomes enabled. Therefore, this method is generally not suitable for I/O initialization code.
mdlInitializeSizes()	Defines the size characteristics of the S-function, such as the number of inputs, outputs, states and sample times. Simulink reads the characteristics to configure the S-Function block.
mdlInitializeSampleTimes()	Defines the sample times for the S-function.
mdlOutputs()	Contains the code for calculating the block outputs of the S-function. You can use it for read/write access of I/O devices.
mdlUpdate()	Contains the algebraic code for calculating the discrete states of the S-function.
mdlDerivatives()	Contains the code for calculating the derivatives of time-continuous state variables within the integration routine.
mdlTerminate()	Contains the termination code of the S-function, which is executed once when simulation is stopped. You can use it for writing the termination values to the output devices.

Note

If an I/O device is initialized more than once, for example, if you place the initialization in `mdlInitializeConditions()` by mistake, the behavior of the device can become unpredictable. Therefore, use `mdlStart()` for initializations that must occur only during simulation start.

Related topics

Basics

Execution Order of S-Functions and User-Code..... 186

S-Functions for Real-Time and Simulink Simulations

Including or excluding code

Since S-functions are a feature of Simulink, you can use them not only for real-time simulation, but also for standard Simulink simulation. Even though the majority of the code of an S-function is valid for both real-time and Simulink simulations, you usually need to conditionally include or exclude specific code segments for one of the two purposes. As an example, consider dSPACE I/O statements that do not make sense for Simulink simulation. On the host PC system there is no dSPACE hardware available and dSPACE I/O functions are not valid.

Implementing code inclusion or exclusion

You can achieve this conditional inclusion or exclusion of a code segment via the `MATLAB_MEX_FILE` C preprocessor variable, which is defined during the MEX DLL file generation and not defined when real-time code is generated. Use the syntax shown below:

```
#ifdef MATLAB_MEX_FILE
   /* Code only used for Simulink simulation that          */
   /* you want to exclude from the real-time object file,   */
   /* for example, an output on the host computer's screen. */
#else
   /* Code only used for real-time simulation that         */
   /* you want to exclude from the Simulink S-function,    */
   /* for example, dSPACE-specific I/O code.               */
   /* Instead of the #ifdef ... #else construct you can also*/
   /* use the line #ifndef MATLAB_MEX_FILE; for example,    */
   /* if there is no need for Simulink-only code.          */
#endif
```

Any line of code that is not enclosed in such an `#ifdef` or `#ifndef` construct is included in both real-time code generation and Simulink simulation.

Example

Consider the throttle control example (refer to Throttle Control Example on page 16): The statements needed for the real-time simulation that send a message to the Message Module (i.e., the Log Viewer of ControlDesk) are functional only for the real-time simulation. Therefore, you have to tag them with the `#ifndef MATLAB_MEX_FILE` statement. On the other hand, you have to conditionally include a statement that sends the message to the MATLAB Command Window if the model is simulated in Simulink. Thus, you have to enclose it within an `#ifdef MATLAB_MEX_FILE` construct. The corresponding code segment is shown below:

```
static void mdlOutputs(SimStruct *S, int_T tid)
{
  /* Get pointer for input port one.
    Note: "uPtrs" is a vector of void pointers. It is
    absolutely crucial to cast the indexed element from void
    pointer to the proper type of pointer. Only after the
    cast is it safe to get the value via indirection. */
  InputPtrsType uPtrs =
    (InputPtrsType) ssGetInputPortSignalPtrs(S, 0);
  boolean_T Limit = *((boolean_T *) uPtrs[0]);
  /* Check if limit is reached. */
  if (Limit)
  {
#ifdef MATLAB_MEX_FILE
    /* Warning for Simulink simulation. */
    mexWarnMsgTxt("Limit reached!");
#else
    /* Warning for real-time simulation. */
    msg_warning_set(MSG_SM_USER, 1, "Limit reached!");
#endif
  }
}
```

Related topics

Examples

C Code for the Throttle Control Example's S-Function..... 191

Accessing PHS-bus-based systems in S-Functions

PHS-bus-based systems

For PHS-bus-based systems there is a distinct relationship between the PHS-bus address of an I/O board and its board number as shown in RTI. When accessing I/O boards via RTLib functions, you must always provide the PHS-bus base address as a parameter. You can obtain this address via the `DSXXXX_n_BASE` macros that are made available via the hardware-specific `brtenv.h` header.

Example

Suppose you want to start the conversion of ADC channel 1 that is located on the second DS2004 board of your system. In this case, specify the following include statement in the header of your S-function:

```
#include <brtenv.h>
#include <ds2004.h>
```

You can then use the `DS2004_2_BASE` macro to retrieve the board address needed for further RTLib I/O functions, in this case:

```
ds2004_sw_trigger(DS2004_2_BASE, DS2004_CONV_START_1);
```

The `ds2004.h` header file contains the board-specific defines, for example, `DS2004_CONV_START_1`.

Board number as parameter

A more general approach is to pass the board number to the S-function as a parameter, which makes the S-function more flexible towards changing the dSPACE system. As a result, you do not need to modify the S-function when adding further I/O boards. In this case, use the following macro:

```
PHS_BOARD_BASE_GET(board_type, board_number)
```

where

`board_type`	=	PHS board type macro, for example, `PHS_BT_2004` for a DS2004 board
`board_number`	=	board number to distinguish between boards of the same type. If `board_number` is passed to the S-function by argument, you can access the argument via the `ssGetSFcnParam` statement.

Related topics

Basics

PHS-Bus Address and Board Number 35

HowTos

How to Access dSPACE I/O Devices in S-Functions 169

How to Access dSPACE I/O Devices in S-Functions

Objective

Using the dSPACE real-time hardware from within C-coded S-functions lets you perform system initialization and access the I/O features.

Method

To access dSPACE I/O devices in S-functions

1. To access dSPACE I/O devices in an S-function, you must include the following lines in the S-function header:

```
#ifndef MATLAB_MEX_FILE
#  include <brtenv.h>     /* basic real-time environment */
#  include <dsXXXX.h>     /* dsXXXX Board, e.g. ds2004.h */
#endif
```

2. Make sure to also enclose all subsequent RTLib statements such as I/O access within the `#ifndef MATLAB_MEX_FILE ... #endif` statements.

Result

Since the `MATLAB_MEX_FILE` macro is only defined if the S-function is compiled for a Simulink simulation (i.e., when building a MEX DLL file), the dSPACE-specific statements are not executed for Simulink simulations but only for real-time simulation.

Tip

If you want to provide extra code for Simulink simulations that is not relevant to dSPACE real-time simulations, for example, screen output for the MATLAB Command Window, you can use the `#ifdef MATLAB_MEX_FILE ... #endif` statements.

Related topics

Basics

Accessing PHS-bus-based systems in S-Functions..... 168

Implementing User-Code

Introduction

RTI allows you to customize your code, for example, by including User-Code to your model's execution code or generating User-Code templates.

Where to go from here

Information in this section

Basics on User-Code.....171
User-Code Interprocessor Communication.....173
How to Include User-Code in Simulink Models.....174
How to Generate a Template for the User-Code.....177
How to Implement the User-Code.....178

Information in other sections

Implementing S-Functions.....164
Comparing S-Functions and User-Code.....186
Execution Order of S-Functions and User-Code.....186
Limitations for Custom Code.....282
How to Include User-Code in Simulink Models.....174
How to Generate a Template for the User-Code.....177
C Code for the Throttle Control Example's User-Code.....194

Basics on User-Code

Customizing code

With User-Code, RTI offers a set of adaptable C functions. They allow you to customize the following sections in your model execution:

- The initialization segment
- The evaluation sequence of the fastest timer task
- The background task
- The termination segment

User-Code can be accessed in the Simulink model by means of Simulink's Data Store Memory blocks.

User-Code template

If you did not build real-time code for your Simulink model yet, there is no User-Code template C file. However, if you want to program the User-Code, you need the User-Code template C file for the model. The User-Code template file `<[sub]model>_usr.c` is generated in your current working directory by RTI when your model is built the first time. An existing User-Code template C file is never overwritten by RTI. The User-Code template file `<[sub]model>_usr.c` contains a set of empty C functions. Each C function has a specific purpose. You have to enter your user-specific C code in these C functions according to their purposes.

For RTI-MP models there are several User-Code files – one for each CPU of the model. These User-Code files are named according to the applications of the different CPUs: `<submodel>_usr.c`.

Data transfer

If you want to transfer data

- From the User-Code to the Simulink model (for example, from an input device) you have to use the function `usr_input()`
- From the Simulink model to the User-Code (for example, to send it to an output device) you have to use the function `usr_output()`

The additional User-Code function `usr_sample_input()` can be used to start the conversion process for an ADC. This process is then finished before the `usr_input()` function starts, which reduces delays in execution.

Communication with Simulink

The User-Code and Simulink can communicate via a global structure variable.

The Data Store blocks of a Simulink model whose Simulink Coder Storage Class is set to **ExportedGlobal** are available via the names specified as Data store name, for example, `Limit`. If such a communication variable is a data vector, you can access its individual elements by means of an index, for example, `Limit[0]`.

Note

If you use such vectorized communication variables, the corresponding index in the User-Code runs from `0` to n-1 if n is the number of vector elements. Do not use any other indices, because this corrupts the memory of the real-time processor.

Related topics

Basics

User-Code Interprocessor Communication 173

HowTos

How to Generate a Template for the User-Code 177
How to Implement the User-Code 178

How to Include User-Code in Simulink Models..... 174

Examples

C Code for the Throttle Control Example's User-Code..... 194

User-Code Interprocessor Communication

Data channels

Between two processor boards of the same type, you can use as many user-defined data channels as are left unused by the Simulink model. Each GigaLink provides eight shared memory (SMEM) and eight swinging buffer (SBUF) channels in total. You can get the numbers of the GigaLink and channel via the function `rtimp_get_usr_channel`. This function has the following syntax:

```
rtimp_get_usr_channel("<CPU>", <protocol>, <No>, &gl, &chan);
```

The numbers of the free IPC channels (the GigaLink and channel numbers) are assigned dynamically, depending on the channels used by the Simulink model.

The table below lists the purposes of the terms:

Term	Purpose
`<CPU>`	Specifies the name of the CPU that you want to communicate with.
`<protocol>`	Specifies the desired protocol. `IPC_SBUF` means the swinging buffer protocol, `IPC_VSM` means the virtual shared memory protocol.
`<No>`	Specifies the desired relative index number for the channel. For example, if you specify `IPC_VSM` as the protocol and `1` as the relative index number, you get the numbers for the first free virtual shared memory channel, regardless of how many virtual shared memory channels are used by the IPC blocks of the Simulink model.
`&gl`	This refers to the `gl` variable that returns the number of the GigaLink to be used.
`&chan`	This refers to the `chan` variable that returns the number of the channel to be used.

Example

Suppose you want to transfer the content of the `out_data` variable from the User-Code of the master to the `in_data` variable of the slave's User-Code. You want to use the first available swinging buffer channel. The following code excerpt shows how to initialize and access the required channel in the User-Code of the master board:

```
int out_data = 0;
int *ch_ptr;
int gl;
int chan;
static void usr_initialize(void)
{
   /* get number of gigalink and channel */
   rtimp_get_usr_channel("slave",IPC_SBUF,1,&gl,&chan);
   /* pointer to the user channel */
   ch_ptr = dsgl_ptr_get(gl,chan);
}
__INLINE void usr_output(void)
{
   /* calculate output data*/
   out_data++ ;
   /* send output and switch buffer */
   *ch_ptr = out_data;
   dsgl_write_buffer_switch(gl,chan);
}
```

The corresponding code excerpt in the User-Code of the slave board looks like this:

```
int in_data;
int *ch_ptr;
int gl;
int chan;
static void usr_initialize(void)
{
   /* get number of gigalink and channel */
   rtimp_get_usr_channel("master",IPC_SBUF,1,&gl,&chan);
   /* pointer to the user channel */
   ch_ptr = dsgl_ptr_get(gl,chan);
}
__INLINE void usr_output(void)
{
   /* switch buffer, if buffer has been updated */
   if (dsgl_read_buffer_is_updated(gl,chan)){
      dsgl_read_buffer_switch(gl,chan);
   }
   /* get input data */
   in_data = *ch_ptr;
}
```

How to Include User-Code in Simulink Models

Objective

If you want to include User-Code in your Simulink model, you must prepare the communication between the model and the User-Code via global variables, e.g., by using Data Store Memory blocks.

Example

Consider the throttle control example (refer to Throttle Control Example on page 16) and suppose you want to receive a message if the throttle valve position exceeds 50°. The Boolean value (whether the limit is reached or not) can be calculated in the Simulink model. You can then let the User-Code output the appropriate message.

To transfer the Boolean value from the Simulink model to the User-Code you have to change your model as illustrated below:

Simulink and User-Code

As you can see from this model, the interface between Simulink and User-Code is implemented via Simulink's Data Store Memory blocks:

- Each **Data Store Memory** block defines and initializes a single communication variable.

- The **Data Store Read** block represents the transition from the User-Code to the Simulink model.

- The **Data Store Write** block denotes the transition from the Simulink model to the User-Code.

Since these **Data Store** blocks do not have any function on their own (apart from the communication), you have to hand-code the desired function in C.

Method

To include User-Code in a Simulink model

1. For each communication variable needed, place a separate **Data Store Memory** block from the Simulink library into your model.

2. For each communication variable, open the **Block Parameters** dialog of the corresponding **Data Store Memory** block and specify the following settings.
   - Enter the name of the communication variable (for example, `Limit`) as **Data store name**.
   - Enter the desired initial value and – via the vector width – its size.
   - Set the **Storage Class** to **ExportedGlobal**.
3. To transfer data from the User-Code to the Simulink model, place a **Data Store Read** block from the Simulink library in your model. To transfer data from the Simulink model to the User-Code, use the **Data Store Write** block instead. Again, you need separate blocks for each communication variable you want to access.

4. For each block, open the **Block Parameters** dialog and specify the data store name of the communication variable you want to access: for example, `Limit`. This name must match the name you gave to the corresponding **Data Store Memory** block. You can also specify a **Sample time** for the communication variable.

5 The User-Code in the `usr_sample_input()`, `usr_input()`, and `usr_output()` functions of the USR.C file are always executed with the fastest timer task and, therefore, have the same task priority. If the Simulink part of a communication variable does not belong to the timer task driven with the base sample time, it might have a different task priority. You can avoid data inconsistency between the **Data Store Memory** blocks and the User-Code. To do so, set the sample time of the **Data Store Write** and **Data Store Read** blocks to the base sample time of the model. If necessary, use **Rate Transition** blocks before the **Data Store Write** and after the **Data Store Read** blocks to model the required data integrity.

Result

You included User-Code in your Simulink model.

Related topics

HowTos

How to Generate a Template for the User-Code 177

How to Generate a Template for the User-Code

Objective

To program User-Code, you must first generate a template for it.

Method

To generate a template for the User-Code

1 Change to the working folder that holds your model.

2 Start a build process for the Simulink model you want to work on. For instructions, refer to How to Start the Build and Download on page 228.

Result

Once you have built your model you will find the `<model>_usr.c` file in its working folder. This user-code template contains a set of empty C functions into which you can incorporate your custom C code.

Note

RTI generates a new User-Code template C file only if there is no `<model>_usr.c` file in the working path of the model yet. Thus, if you already added user-specific code to the User-Code file of the model, RTI does not replace this file.

RTI-MP For RTI-MP models there are several User-Code files – one for each CPU of the model. These User-Code files are named according to the applications of the different CPUs: `<submodel>_usr.c`.

Related topics

HowTos

How to Include User-Code in Simulink Models..... 174
How to Start the Build and Download..... 228

How to Implement the User-Code

Objective

To implement your User-Code efficiently, you must implement the User-Code statements according to their purposes.

Method

To implement the User-Code

1 Fill the empty functions of the `<[sub]model>_usr.c` User-Code template C file with your user-specific C code. Place the code statements in the appropriate functions according to their purposes as specified below:

Purpose	Target Function
Initialization, e.g., write initialization values to output devices	`usr_initialize()`
Start conversion, e.g., for A/D devices	`usr_sample_input()`
Read access, e.g., for input devices	`usr_input()`
Write access, e.g., for output devices	`usr_output()`
Shut down, e.g., write termination values to output devices	`usr_terminate()`
Background task	`usr_background()`

2 Where required, implement the communication with the Simulink model:
- You can access the **Data Store Read** and **Data Store Write** blocks of the Simulink model via the **Data store names** specified in the blocks.
- Do not exceed the defined index if you use vectorized variables.
- If the Simulink part of a communication variable does not belong to the timer task driven with the base sample time, it might belong to a different task. If this is the case and if the variable is a vector, you have to avoid data inconsistency:
  - If the User-Code has the lower priority, you have to enclose the critical code segments that access the communication variables with the `global_disable()` and `global_enable()` statements. Refer to the *RTLib Reference* (available in dSPACE Help) for further information on these functions.
  - If the User-Code has the higher priority, this is not necessary. Take precautions in the Simulink model as described in How to Include User-Code in Simulink Models on page 174.

Example

Consider the throttle control example. The following statements form the basic function that outputs the message to the Message Module if the throttle valve position exceeds 50°.

```
__INLINE void usr_output(void)
{
   /* check if limit is reached */
   if (Limit!=0.0)
   {
      /* warning for real-time simulation */
      msg_warning_set(MSG_SM_USER, 1,
         "Throttle Control Example: Limit reached!");
   }
}
```

To keep this code segment simple it does not contain administrative code (generating a single message output even if the throttle valve position remains above the limit). The complete source code is included in C Code for the Throttle Control Example's User-Code on page 194.

Related topics

Examples

C Code for the Throttle Control Example's User-Code..... 194

Integrating C++ Code

Introduction

Basically, RTI is supporting C code. To integrate C++ code, the following aspects are to be considered.

If you want to integrate C++ code into a hand-coded application or if you want to build a library, please contact dSPACE Support for further information.

Using C++ Code in an RTI Application

Introduction

RTI provides different ways to use C++ code in your RTI application.

Basics

You can add C++ code as user sources, objects and run-time libraries to be linked to the real-time application. The C code calling the C++ code and the makefile used for the user code must be adapted.

Limitations

For technical reasons C++ cannot be used in some parts of the application. It is explicitly not possible to use:

- C++ code within the RTI User-Code file (USR.C file)
- C++ code in inlined S-functions using Target Language Compiler™ (TLC) files
- C++ code within Custom Code blocks from Simulink Coder, since it is also inlined code using TLC files
- C++ modules or libraries defined via Simulink Coder's Make Customization (rtwmakecfg.m) file

For all these, only plain C code is possible. However, the following descriptions show you how to combine these files with C++ code from additional modules.

Note

On the Code Generation page of the Configuration Parameters dialog, the Language option must be set to *C*.

Supported Platforms C++ support is applicable with all platforms supported by RTI and RTI-MP.

Using DS1006 The constructors and destructors of global C++ objects are not executed if the global C++ object is defined in an object module that is part of a run-time library. This can result in unwanted run-time behavior of the real-time application.

This limitation does not apply if the respective object module is linked directly to the application executable. It also does not apply to non global C++ objects, e.g. if the object is defined within a function scope or created with the `new` operator.

This limitation is due to the behavior of the linker of the GNU compiler collection on which the dSPACE DS1006 Compiler is based.

Note

Using model referencing and C++ user code

The complete code of a referenced model is placed in a run-time library, including user code. If the referenced model contains C++ user code, the above limitation applies.

Using iostream It is possible to include `<iostream>`, but using the `cout` or `cin` streams will cause an error during the application's run time on all platforms except for DS1006. All data written to `cout` and `cerr` on a DS1006 is redirected to the ControlDesk log handler. Reading from `cin` on a DS1006 will not produce any data.

Overview of features and limitations The following table shows you the basic features and limitations according to the compiler used.

Feature	Microtec C++ Compiler for PowerPC Platforms[1)]	DS1006 C++ Compiler[2)]	DS1007/MicroLabBox Compiler[3)]
Exception handling[4)]	–	–	–
Run-Time Type Information (RTTI)[4)]	–	–	–
Standard Template Library (STL)	✓	✓	✓[5)]
I/O streams support[6)]	–	✓	–

1) The Microtec C++ Compiler does not support C++11.
2) The DS1006 C++ Compiler supports some C++11 features but these are not tested by dSPACE.
3) The C++11 support is set by default with the compiler switch `-std=gnu++11`.
4) Any C++ objects and libraries which are to be linked to the real-time application must not require these features. Linking a C++ library which requires C++ exception handling and/or RTTI support will cause the application build process to fail during the linker stage.
5) The STL is contained in the compiler libraries and is not tested by dSPACE. It is not guaranteed that each functionality is supported.
6) See *Using iostream* for detailed information.

Adapting the User Makefile

For adding C++ code to your application you have to adapt the User Makefile.

- Enable the C++ support
- Add C++ source files, C++ object files and C++ libraries

Example:

```
# Enable C++ support
USER_BUILD_CPP_APPL = ON
...
# Additional C/C++ source files to be compiled
USER_SRCS = MyModule1.c MyModule2.cpp
...
# Additional user object files to be linked
USER_OBJS = MyModule3.o50 MyModule4.cppo50
...
# Additional user libraries to be linked
USER_LIBS = MyCLib.lib MyCppLib.lib
```

For further information on the User Makefile, refer to User Makefile (USR.MK File) (RTI and RTI-MP Implementation Reference 🕮).

Integration C++ code in your C code

To allow a C module calling a function of a C++ module, you have to provide a C linkage identifier.

To achieve type-safe linkage, the function names must be mangled.

```
#ifdef __cplusplus
// Use C linkage identifier instead
// of mangled C++ names for
// all functions defined in this scope.
extern "C" {
#endif

<Enter functions, variable and pointer declarations and
definitions here.>

#ifdef __cplusplus
  // End of extern "C" scope.
}
#endif
#ifdef __cplusplus
extern "C" { // Use C linkage identifier instead of mangled C++
             // names for all functions defined in this scope.
#endif
<Enter functions, variable and pointer declarations and definitions
here.>
#ifdef __cplusplus
} // End of extern "C" scope
#endif
```

In this way, you can call a function of a C++ module within your C code.

If the C++ code cannot be adapted in this way, you might provide an interface for this module.

In this case, you have to add both CPP files to the `USER_SRCS` entry in the makefile.

Using C++ code in an S-Function block

To build a real-time application for a model containing an S-Function block with an C++ function, you have to additionally provide the compiled MEX DLL and the C++ module, the function is contained in.

Example:

The function `sfcn1_cpp`, that is used in an S-Function block, belongs to the `module_cpp` C++ module.

The required files to be added in the user makefile are:

- `sfcn1_cpp.cpp`
- `module_cpp.cpp`

Simulink demo

As an example, you can refer to a Simulink demo that comes with a C++ S-function. To access the demo model, enter `sfundemos` in the MATLAB Command Window. Select **C++ S-functions** in the model and then **C++ Objects**.

To configure the demo model for RTI, you will need to specify the respective system target file in the **Simulation Parameter** dialog. Open the dialog and go to the **Code Generation** page. Select the system target file matching your RTI platform, for example, `rti1007.tlc`.

With the explanations in Adapting the User Makefile on page 181, you should be able to build the demo model.

Comparing S-Functions and User-Code

Introduction

Both S-functions and User-Code have specific advantages and disadvantages, which are listed below. These can help you decide whether you want to use an S-function or User-Code.

	Advantages	Disadvantages
S-function	Can be executed in any task of the real-time program. Easy to use via Simulink's S-function block.	No control over the time of initialization/execution in the overall execution order of the model. Source code not as easy as that of User-Code.
User-Code	Precise control over the time of initialization/execution in the overall execution order of the model. Source code simpler than that of S-functions.	Always executed with the fastest timer task.

Where to go from here

Information in this section

Execution Order of S-Functions and User-Code..... 186
Tips and Tricks for Custom C Code..... 188

Information in other sections

Implementing S-Functions..... 164
Implementing User-Code..... 171
Limitations for Custom Code..... 282

Execution Order of S-Functions and User-Code

Executing S-functions and User-Code

The following two flow charts summarize when the individual C functions or methods of User-Code and S-functions are executed in relation to the Simulink model.

Note

- Certain RTI blocks are implemented as S-functions. They are executed like any other user-specific S-functions. The execution order of S-functions is demonstrated via the fictional Sfcn A and Sfcn B blocks. All other RTI blocks are explicitly shown in the flow chart.
- In the diagram, flow chart blocks are grouped in a box (as shown below) if their execution order cannot be predefined.

Execution order of the tasks of a model

This flow chart shows the execution order of the initializations and blocks that are contained in the timer tasks. Whenever all these tasks are finished, the processor executes the background task, which is shown in the next flow chart.

Execution order of the background task of a model

This flow chart shows the execution order of the blocks that are assigned to the background task. The background task is executed only if no other task (shown in the previous flow chart) needs to be executed.

Related topics

Basics

Purpose of the C Methods 166

References

User-Code File (USR.C File) (RTI and RTI-MP Implementation Reference 🕮)

Tips and Tricks for Custom C Code

Introduction

The following tips might help you implement your custom C code.

Holding C files separately

If you want to hold your S-function C files in a separate folder, you can use the settings in the Include list of additional frame in Code Generation's Custom Code page instead of, or in addition to, the settings in the RTI User Makefile (USR.MK). Refer to User Makefile (USR.MK File) (RTI and RTI-MP Implementation Reference 🕮) and to *Custom Code page* on the Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮).

Including files and libraries

If you want to include and use custom source files, include files, or libraries in S-functions or User-Code, you can use the settings in the Include list of the additional frame in Code Generation's Custom Code page instead of, or in addition to, the settings in the RTI User Makefile (USR.MK). Refer to User Makefile (USR.MK File) (RTI and RTI-MP Implementation Reference 🕮) and to *Custom Code page* on the Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮).

Rebuilding custom C code	You do not always need to start a complete build process after you modify certain parts of the custom C code. Instead, you can run the make utility by hand. Refer to How to Automate the Build Process on page 230.
Adding custom code to your real-time application	You can use Code Generation's **Custom Code** page to add hand-written code to your real-time application. For further information, refer to Adding User-Specific Files, Source Files and Libraries on page 243.

Examples

Introduction

To help you implement S-functions and User-Code, several examples are provided below:

Where to go from here

Information in this section

Flow Chart for the Custom C Code of the Throttle Control Example..... 190

C Code for the Throttle Control Example's S-Function..... 191

C Code for the Throttle Control Example's User-Code..... 194

Flow Chart for the Custom C Code of the Throttle Control Example

Implementing S-functions or User-Code

The following flow chart shows the behavior that has to be implemented in an S-function or User-Code to solve the required job for the throttle control example. The white blocks issue the actual warning if the throttle valve position exceeds the limit, the light-colored blocks ensure that the warning is issued only once, and the dark-colored block counts how often the warning was issued.

Related topics

Basics

Inserting Custom C/C++ Code..... 163

Examples

C Code for the Throttle Control Example's S-Function..... 191
C Code for the Throttle Control Example's User-Code..... 194

C Code for the Throttle Control Example's S-Function

Example

The following C source code forms the required S-function:

```
/*****************************************************************************
 FILE: issue_warning.c
 DESCRIPTION:
   Example S-function for the throttle control example from the RTI
   Implementation Guide. The S-function issues a warning if the input is
   logical true.
 REMARKS:
   This file is based on the Simulink S-function template sfuntmpl_basic.c and
   contains only the most common S-function methods. Comments on example-
   specific changes are included.
   See <MATLAB_ROOT>/simulink/src/sfuntmpl_doc.c for more details.
 (c) Copyright 2005, dSPACE GmbH. All rights reserved.
*****************************************************************************/
#define S_FUNCTION_LEVEL  2
#define S_FUNCTION_NAME   issue_warning
#include "simstruc.h"
/* When compiling for dSPACE target, include the brtenv.h file.*/
#ifndef MATLAB_MEX_FILE
# include <brtenv.h>
#endif
/*  Declare global variables. These can be made available in ControlDesk by
    adding them to the <model>_usr.trc file.
    NOTE:
    Due to the use of global variables, this S-function is not reentrant */
int warning_flag = 0;  /* Control issuing of warning */
int counter      = 0;  /* Counter for warnings */
```

```
/* Function: mdlInitializeSizes ========================== */
static void mdlInitializeSizes(SimStruct *S)
{
  /* This S-function has no parameters. */
  ssSetNumSFcnParams(S, 0);  /* No. of expected parameters */
  if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
    return;
  }
  /* Set the number of states to zero. */
  ssSetNumContStates(S, 0);
  ssSetNumDiscStates(S, 0);
  /* Set one input port. */
  if (!ssSetNumInputPorts(S, 1)) return;
  /* Set width of input port to one. */
  ssSetInputPortWidth(S, 0, 1);
  /* Input is accessed in mdlOutputs() routine,
     therefore set direct feedthrough flag. */
  ssSetInputPortDirectFeedThrough(S, 0, 1);
  /* Set the input port data type to boolean. */
  ssSetInputPortDataType(S, 0, SS_BOOLEAN);
  /* We have no output ports, therefore we must not
     call ssSetOutputPortWidth()! */
  if (!ssSetNumOutputPorts(S, 0)) return;
  /* This S-function has one sample time. */
  ssSetNumSampleTimes(S, 1);
  /* No work vectors and other special features are used. */
  ssSetNumRWork(S, 0);
  ssSetNumIWork(S, 0);
  ssSetNumPWork(S, 0);
  ssSetNumModes(S, 0);
  ssSetNumNonsampledZCs(S, 0);
  /* No additional options are used. */
  ssSetOptions(S, 0);
}
/* Function: mdlInitializeSampleTimes ===================== */
static void mdlInitializeSampleTimes(SimStruct *S)
{
  /* Use inherited sample time. */
  ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
  /* Ensure that this S-function will not execute in intermediate integration
     steps. */
  ssSetOffsetTime(S, 0, FIXED_IN_MINOR_STEP_OFFSET);
}
#define MDL_START  /* Change to #undef to remove function */
#if defined(MDL_START)
  /* Function: mdlStart ================================== */
  static void mdlStart(SimStruct *S)
  {
    /* Reset control variables when switching RTI simulation state from STOP to
       RUN. */
    warning_flag = 0;
    counter = 0;
  }
#endif /* MDL_START */
```

```
static void mdlOutputs(SimStruct *S, int_T tid)
{
  /* Get pointer for input port one.
     Note: "uPtrs" is a vector of void pointers. It is absolutely crucial to
     cast the indexed element from void pointer to the proper type of pointer.
     Only after the cast is it safe to get the value via indirection. */
  InputPtrsType uPtrs = (InputPtrsType) ssGetInputPortSignalPtrs(S, 0);
  boolean_T Limit = *((boolean_T *) uPtrs[0]);
  /* Check if limit is reached. */
  if (Limit)
  {
    /* Check if warning has already been issued. */
    if (!warning_flag)
    {
      /* Set flag to remember that warning has been issued. */
      warning_flag = 1;
      /* Increment the number of warnings. */
      counter++;
#ifdef MATLAB_MEX_FILE
      /* Warning for Simulink simulation. */
      mexWarnMsgTxt(
        "Throttle Control Example: Limit reached!");
#else
      /* Warning for real-time simulation. */
      msg_warning_set(MSG_SM_USER, 1, "Throttle Control Example: Limit reached!");
#endif
    }
  }
  else
  {
    /* Reset the warning flag. */
    warning_flag = 0;
  }
}
/* Function: mdlTerminate ================================*/
static void mdlTerminate(SimStruct *S)
{
}
#ifdef MATLAB_MEX_FILE /* Compile as a MEX file? */
#include "simulink.c"  /* MEX file interface mechanism */
#else
#include "cg_sfun.h"   /* Code generation registration fcn */
#endif
```

Related topics

Basics

S-Functions for Real-Time and Simulink Simulations..... 167

Examples

Flow Chart for the Custom C Code of the Throttle Control Example..... 190

C Code for the Throttle Control Example's User-Code

Example

The following C source code forms the required User-Code.

```
/*****************************************************************************
   Include file throttlecontroldemo_usr.c:
   Definition of functions for user-defined initialization,
   system I/O, and background process code.
 *****************************************************************************/
/* ========================================================================== */
/* =====  Define file version macro. Never change this value.  ============= */
/* ========================================================================== */
#define USER_CODE_FILE_VERSION 5
/* ========================================================================== */
/* Insert #include directives for header files here. */
#if defined(_INLINE)
# define __INLINE static inline
#else
# define __INLINE static
#endif
/* declare global variables */
int warning_flag = 0;      /* to control issuing of warning  */
int counter = 0;           /* counter for warnings */
static void usr_initialize(void)
{
   /* reset warning flag if simulation starts (STOP -> RUN) */
   warning_flag = 0;
   /* reset counter if simulation starts */
   counter = 0;
}
__INLINE void usr_sample_input(void)
{
}
__INLINE void usr_input(void)
{
}
__INLINE void usr_output(void)
{
   /* check if limit is reached */
   if (Limit!=0.0)
   {
      /* check if warning has already been issued */
      if (warning_flag == 0)
      {
         /* Set flag to remember that warning has been issued. */
         warning_flag = 1;
         /* Increment the number of warnings. */
         counter++;
         /* warning for real-time simulation */
         msg_warning_set(MSG_SM_USER, 1,
            "Throttle Control Example: Limit reached!");
      }
   }
   else
   {
      /* reset warning flag */
      warning_flag = 0;
   }
}
```

```
static void usr_terminate(void)
{
}
static void usr_background(void)
{
}
#undef __INLINE
/****** [EOF] ***************************************************************/
```

Related topics

HowTos

How to Implement the User-Code..... 178

Examples

Flow Chart for the Custom C Code of the Throttle Control Example..... 190

Advanced Techniques

Introduction

To learn about advanced techniques when inserting custom C code, look at the following instructions and examples.

Where to go from here

Information in this section

How to Make Custom Variables Accessible..... 196

How to Send Messages to the Message Module..... 197

Example of a Slave Application as User-Code..... 199

Example of a Slave Application as an S-Function..... 201

How to Make Custom Variables Accessible

Accessing custom variables

Suppose you want to capture custom variables of an S-function or User-Code. By default, RTI does not include the internal variables of S-functions or User-Code in the variable description file, only the output of S-function blocks or the global Data Store Memory communication variables.

Note

If you are using more than one instance of a specific S-function in your model, avoid unexpected side-effects in your application due to the use of global variables.

Method

To make custom variables accessible

1. In the corresponding S-function or User-Code, declare the custom variables as global non-static variables or as global non-static pointer variables.
2. Add a user variable description file to the folder of the model (`<model>_usr.trc` for single-processor systems or `<submodel>_usr.trc` for multiprocessor systems).
3. List the desired custom variables in the user variable description file using the syntax of the variable description file.
4. Build your real-time application and download it to the real-time hardware.

   During the build process, RTI adds the contents of the `<model>_usr.trc` file to the `<model>.trc` file.

5 Load the corresponding system description file (`<model>.sdf`) to the specified dSPACE board, for example, using ControlDesk.

Result

ControlDesk's Variables controlbar displays the internal variables in an extra group called User Variables.

Example

Consider the throttle control example (refer to Throttle Control Example on page 16). Suppose you want to implement a counter in the user-defined code that counts how often a message is issued that warns that the throttle valve position has exceeded its limit. The corresponding code could read as follows:

```
/* Declare global variables. These can be made available in
   ControlDesk by adding them to the <model>_usr.c file. */
int warning_flag = 0;  /* Control issuing of warning */
int counter      = 0;  /* Counter for warnings */
```

The contents of `<model>_usr.trc` would read as follows:

```
-- ******** Warning control variables********************
warning_flag
{
   type: int(32)
   desc: "Control issuing of warning"
}
counter
{
   type: int(32)
   desc: "Counter for warnings"
}
```

Related topics

Basics

Building and Downloading the Model..... 219

References

Syntax of the TRC File (RTI and RTI-MP Implementation Reference 🕮)
type (Data Type, Data Format and Type Definition) (RTI and RTI-MP Implementation Reference 🕮)

How to Send Messages to the Message Module

Objective

If you want to generate messages in certain situations and display them via the Message Module of the host computer, you have to use special C-function calls. You can use these C-function calls in any S-function or User-Code.

Reporting errors

For reporting errors you can use the `ssSetErrorStatus()` function in your S-function, which works for both Simulink and real-time simulation. To report errors from your User-Code during real-time simulation you can use the function `msg_error_printf(MSG_SM_USER, 1, <...>)`. If your S-function or User-Code reports an error during real-time simulation, the message used with this function is displayed in the Log Viewer of ControlDesk, and RTI stops the application. Then the real-time application is terminated via the `exit()` function (see also Reaction to Run-Time Errors on page 215).

If you want to report a problem without terminating the real-time application, you cannot use the functions `ssSetErrorStatus()` and msg_error_printf(MSG_SM_USER, 1, <...>). Instead use the functions `msg_warning_set(MSG_SM_USER, 1, <...>)` or `msg_warning_printf(MSG_SM_USER, 1, <...>)` for the real-time application or the function `ssWarning(S, <...>)` to report a warning during Simulink simulation.

For details on `ssSetErrorStatus` and `ssWarning` see *Writing S-Functions* by MathWorks®.

Method

To send messages to the Message Module

1. Include the `#include <brtenv.h>` statement in the program header of your S-function or User-Code.
2. Use the access C functions according to the table shown below to send information, a warning or an error to the Message Module. An error message automatically terminates the simulation by switching the `simState` variable to STOP.

Desired Operation	Message Function
Display information	`msg_info_set()`
Display warning	`msg_warning_set()`
Display error, terminate simulation	`msg_error_set()`
Return last error number	`msg_last_error_number()`

Note

You have to use these access C functions together with the submodule identifier `MSG_SM_USER`.

For details on the Message Module, refer to Message Handling in the *RTLib Reference* that corresponds to your hardware.

Related topics

Basics

Reaction to Run-Time Errors..... 215

References

errorNumber (RTI and RTI-MP Implementation Reference 🕮)
Message Handling (MicroLabBox RTLib Reference 🕮)
Message Handling (DS1006 RTLib Reference 🕮)
Message Handling (MicroAutoBox II RTLib Reference 🕮)
Message Handling (DS1104 RTLib Reference 🕮)
Message Handling (DS1007 RTLib Reference 🕮)

Example of a Slave Application as User-Code

Including a slave application

Suppose you run a DS2201 board and want to use its slave processor to run a 4-channel square wave generation. Include the `4df.c` file in the User-Code of your model via the `#include <<RCP_HIL_InstallationPath>\Slave\DS2201\4df.c>` C include statement.

The corresponding Simulink model could look like this:

The DS2201-specific User-Code for this application could read:

```
/* ========================================================= */
/* = Define file version macro. Never change this value. = */
/* ========================================================= */
#define USER_CODE_FILE_VERSION 5
/* ========================================================= */
/* Insert #include directives for header files here. */
/* Include basic real-time environment header. */
#include <brtenv.h>
/* Include the slave application source code to be found in
   <RCP_HIL_InstallationPath>\slave\ds2201 */
#include <<RCP_HIL_InstallationPath>\slave\ds2201\4DF.c>
#if defined(_INLINE)
# define __INLINE static inline
#else
# define __INLINE static
#endif
static void usr_initialize(void)
{
  /* Initialize the DS2201 I/O board. */
  ds2201_init(DS2201_1_BASE);
  /* Load the slave application. */
  load_slave_4df(DS2201_1_BASE);
  /* Enable signal generation. */
  ds2201_dtof_enable(DS2201_1_BASE, DS2201_DTOF_ENABLE);
}
__INLINE void usr_sample_input(void)
{
}
__INLINE void usr_input(void)
{
}
__INLINE void usr_output(void)
{
  /* Write frequency value for channel 1. */
  ds2201_dtof_4(DS2201_1_BASE, 1, <model>_DWork_freq_out[0]);
  /* Write frequency value for channel 2. */
  ds2201_dtof_4(DS2201_1_BASE, 2, <model>_DWork_freq_out[1]);
  /* Write frequency value for channel 3. */
  ds2201_dtof_4(DS2201_1_BASE, 3, <model>_DWork_freq_out[2]);
  /* Write frequency value for channel 4. */
  ds2201_dtof_4(DS2201_1_BASE, 4, <model>_DWork_freq_out[3]);
}
static void usr_terminate(void)
{
}
static void usr_background(void)
{
}
#undef __INLINE
/****** [EOF] ****************************************************************/
```

The following actions are carried out in this User-Code:

- The source code of the 4df slave application is included. This code also incorporates the necessary code for loading the application from the master CPU to the slave CPU, which makes it possible to load and start the slave application with the call `load_slave_4df()`.

- Via the Data Store Write block, the frequency value of the generated square-wave signal is passed to the User-Code to let you control the frequency parameter from within the Simulink model. In the User-Code the corresponding variables are:

```
<model>_DWork_freq_out[0]
<model>_DWork_freq_out[1]
<model>_DWork_freq_out[2]
<model>_DWork_freq_out[3]
```

Related topics

HowTos

How to Send Messages to the Message Module..... 197

Examples

Example of a Slave Application as an S-Function..... 201

Example of a Slave Application as an S-Function

S-function source file

Suppose you run a DS2201 board and want to use its slave processor to run a 4-channel square wave generation. The S-function source file is `sfcn_ds2201_4df.c` for the signal generation.

The corresponding Simulink model could look like this:

The DS2201-specific S-function implementing the 4-channel square wave generation could read:

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME sfcn_ds2201_4df
#include "simstruc.h"
/* Include basic real-time environment header. */
/* Include the slave application source code to be found in
<RCP_HIL_InstallationPath>\slave\ds2201 */
#ifndef MATLAB_MEX_FILE
# include <brtenv.h>
# include <<RCP_HIL_InstallationPath>\slave\ds2201\4df.c>
#endif
#define NUM_INPUT_PORT          (1)
#define INPUT_PORT_WIDTH_1      (4)
/* Function: mdlInitializeSizes =========================== */
static void mdlInitializeSizes(SimStruct *S)
{
  /* This S-function has no parameters. */
  ssSetNumSFcnParams(S, 0); /* No. of expected parameters */
  if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S))return;
  /* Set the number of states to zero. */
  ssSetNumContStates(S, 0);
  ssSetNumDiscStates(S, 0);
  /* We have one input port with width 4*/
  if (!ssSetNumInputPorts(S, NUM_INPUT_PORT)) return;
  ssSetInputPortWidth(S, 0, INPUT_PORT_WIDTH_1);
  /* We have no output ports.*/
  if (!ssSetNumOutputPorts(S, 0)) return;
  /* This S-function has one sample time. */
  ssSetNumSampleTimes(S, 1);
  /* No work vectors and other special features are used. */
  ssSetNumRWork(S, 0);
  ssSetNumIWork(S, 0);
  ssSetNumPWork(S, 0);
  ssSetNumModes(S, 0);
  ssSetNumNonsampledZCs(S, 0);
  /* No additional options are used. */
  ssSetOptions(S, 0);
}
/* Function: mdlInitializeSampleTimes ===================== */
static void mdlInitializeSampleTimes(SimStruct *S)
{
  /* Use 5 ms sample time and no offset. */
  ssSetSampleTime(S, 0, 0.005);
  ssSetOffsetTime(S, 0, 0.0);
}
#define MDL_START /* Change to #undef to remove function */
#if defined(MDL_START)
/* Function: mdlStart ================================== */
static void mdlStart(SimStruct *S)
{
#ifndef MATLAB_MEX_FILE
  /* Initialize the DS2201 I/O board. */
  ds2201_init(DS2201_1_BASE);
  /* Load the slave application. */
  load_slave_4df(DS2201_1_BASE);
#endif
}
#endif /* MDL_START */
```

```
/* Function: mdlOutputs =================================*/
static void mdlOutputs(SimStruct *S, int_T tid)
{
/* Get pointer to S-function input port. */
#ifndef MATLAB_MEX_FILE
  int               j;
  InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
  for (j = 0; j < INPUT_PORT_WIDTH_1; j++) {
    /* Write frequency value for channel j+1. */
    ds2201_dtof_4(DS2201_1_BASE, j+1, *uPtrs[j]);
  }
#else
  /* do nothing */
#endif
}
/* Function: mdlTerminate ===================================*/
static void mdlTerminate(SimStruct *S)
{
}
#ifdef MATLAB_MEX_FILE /* Compile as a MEX file? */
#include "simulink.c" /* MEX file interface mechanism */
#else
#include "cg_sfun.h" /* Code generation registration fcn */
#endif
```

Related topics

HowTos

How to Send Messages to the Message Module..... 197

Examples

Example of a Slave Application as User-Code..... 199

Managing the Simulation

Introduction

To manage simulations, you must be familiar with basic principles such as simulation control, data acquisition, and error handling.

Where to go from here

Information in this section

Simulation Control (RUN/STOP Mechanism).....205
How to Set the Simulation State.....208
How to Read the Simulation State 209
Time-Stamping and Data Acquisition.....210
How to Modify Data Acquisition Behavior.....213
Reaction to Run-Time Errors.....215
Different Results from Different Simulation Modes.....216

Simulation Control (RUN/STOP Mechanism)

Simulation state

For both RTI and RTI-MP the simulation state can be preset via the **Initial simulation state** option (except for DS1007 and MicroLabBox). When the application is compiled it inherits this setting. For details, refer to **Code Generation Dialog (Model Configuration Parameters Dialogs)** (for RTI) and **Main Page (Multiprocessor Setup Dialog)** (for RTI-MP).

Note

- If the initial simulation state is *PAUSE*, the output devices are set to the Initialization values. If the initial simulation state is *STOP*, they are set to the Termination values.
- The *PAUSE* simulation state is not supported by DS1007 and MicroLabBox.

Changing the simulation state

For both RTI and RTI-MP the simulation state can be changed during run time via the `simState` variable:

- Manually, for example, via ControlDesk. The `simState` variable is available in the Main group of the variable description file.
- Automatically by the simulation itself via the simState SET block.

Tip

The simState variable can also be used in the simulation via the simState READ block.

Note

If you use the simState READ and simState SET blocks in a Simulink simulation to read or specify the simulation state note the following restrictions:

- You can use the simState SET block to change the simulation state from *RUN* to *STOP*. You cannot set the simulation state to *PAUSE*. A warning message is displayed at the MATLAB command prompt. After changing the simulation state to *STOP* you cannot restart the simulation with the simState SET block as there is no background task running in a Simulink simulation.
- As in real-time simulation, the simState READ block outputs a value of 2 during Simulink simulation.

If an error occurs during the real-time simulation, the simulation is stopped automatically. After that the real-time application is terminated via the `exit()` function. See also Reaction to Run-Time Errors on page 215.

Results of simulation state changes

When the simulation state is being changed, the following actions are carried out:

Change to	Value	Result
STOP	0	The termination code is executed and the `currentTime` variable reset.
PAUSE	1	The simulation is paused (the `currentTime` variable is not reset).
RUN	2	▪ If the simulation was previously in the PAUSE mode, the execution of the application is reenabled.

Change to	Value	Result
		▪ If the previous state was STOP, the states of the model are reset to their initial values and the execution of the application is restarted.

The simulation state is processed in the background task. As long as you do not reset the hardware, the background task continues running.

Note

Using, for example, ControlDesk, you can use the following commands to stop a real-time application:

- Stop RTP (ControlDesk Platform Management 🕮) (DS1006, DS1104, MicroAutoBox II platforms)
- Stop RTPs (ControlDesk Platform Management 🕮) (Multiprocessor System platforms)
- Stop (ControlDesk Platform Management 🕮) (DS1007 and MicroLabBox platforms)

  The application is not unloaded automatically from the DS1007 or MicroLabBox platform when it is stopped. You can unload it manually. Refer to Unload (ControlDesk Platform Management 🕮).

When you use the **`Stop RTP`** or **`Stop RTPs`** commands to stop an application on the above mentioned platforms, the related processor is stopped immediately. This is not the same as setting the **`simState`** variable to *STOP*. The termination code therefore is not executed and the background task is no longer active. As a result, the I/O does not output the termination values of the application and you have to reload the application again via the **Reload Application** command if you want to restart it.

When you use the **`Stop`** command to stop an application on a DS1007 or MicroLabBox, this is the same as setting the **`simState`** variable to *STOP*. To restart an application, you have to use the related **`Start`** command or set the **`simState`** variable to *RUN*.

Related topics

HowTos

How to Set the Simulation State 208

References

Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮)
Main Page (Multiprocessor Setup Dialog) (RTI and RTI-MP Implementation Reference 🕮)
simState (RTI and RTI-MP Implementation Reference 🕮)
simState READ Block (RTI and RTI-MP Implementation Reference 🕮)
simState SET Block (RTI and RTI-MP Implementation Reference 🕮)

How to Set the Simulation State

Objective

If you want to set the simulation state during real-time simulation, you can do so via the simState variable.

Possible methods

The simulation state can be set as follows:

- Manually, for example, via ControlDesk. Refer to Method 1.
- Automatically via the simState SET block. Refer to Method 2.

Method 1

To set the simulation state via ControlDesk

1 Connect the simState variable from the Main group of the variable description file to a suitable instrument in a layout.

Method 2

To set the simulation state from within the Simulink model

1 Add the simState SET block from RTI's Extras library to your model. Several instances of this block are allowed.

2 Open the Block Parameters dialog, and select the desired action in the Set simState to drop-down list (RUN, PAUSE, STOP). You can also choose the trigger type and whether the block should have an enable port.

3 Connect the port(s) of the block to the signal(s) that should change the simulation state.

Note

- If you want to change the simulation state while the simulation is paused or stopped, you have to place the involved blocks in the background task of the model.
- The **simState SET** block writes just the new value to the `simState` variable. The actual simulation state is switched in the background task. Therefore, the current computation step is always finished before the simulation pauses or stops. If the background task is complex to compute or if there is very little time for the background task, several further computation steps might be performed before switching takes place.
- With RTI-MP, the **simState SET** block can only be used for the Master CPU.
- Setting the simulation state to *STOP* does not stop the real-time processor, but sets the application to a different condition: The termination values are written to the I/O, after which only the background task of the application remains active.

  In contrast, when you use ControlDesk's `Stop RTP` or `Stop RTPs` commands to stop an application, the real-time processor is stopped immediately: The termination values are not written to the I/O, and the background task does not remain active.

Related topics

HowTos

How to Read the Simulation State 209

References

simState READ Block (RTI and RTI-MP Implementation Reference 🕮)
simState SET Block (RTI and RTI-MP Implementation Reference 🕮)

How to Read the Simulation State

Objective

To read the current simulation state in the Simulink model, you must work with the **simState Read** block.

Method

To read the simulation state

1 Add the **simState READ** block from RTI's Extras library to your model. Several instances of this block are allowed.

2 Connect the output port of the block. The data type is `int32`. The meanings of the output values are as follows:

Value	Simulation State
0	STOP
1	PAUSE
2	RUN

Note

If you want to read the simulation state while the simulation is paused or stopped, you have to place the involved blocks in the background task of the model (in a foreground task the **simState READ** block would constantly read the simulation state RUN).

Related topics

HowTos

How to Set the Simulation State..... 208

References

simState READ Block (RTI and RTI-MP Implementation Reference 🕮)
simState SET Block (RTI and RTI-MP Implementation Reference 🕮)

Time-Stamping and Data Acquisition

Time-stamping support

dSPACE boards support time-stamping, where a time stamp is assigned to each execution step of each task. As a result, ControlDesk can use the actual time values to plot the signals of tasks driven by interrupt blocks. One benefit is that you can see when the aperiodic interrupts occur in relation to each other.

Generating time stamps

The time stamps are generated in different ways for each task type:

Timer tasks For timer tasks, the dSPACE Real-Time Kernel calculates the time stamps from the sample time of the tasks in the same way as the Simulink Coder timing engine calculates the time for timer tasks (currentTime T_t):

$T_t = K \cdot T_{sample_time} + T_{offset}$

where

T_t= `currentTime` variable or RTK time stamp for timer task
K = execution counter

This ensures consistency between the time of the real-time application and the time displayed in plots in ControlDesk.

Tasks driven by hardware interrupt blocks For tasks that are driven by hardware interrupt blocks, the dSPACE Real-Time Kernel derives the time stamps (T_{hw}) from a hardware clock. This ensures exact time stamps that are independent of the fixed step size of the real-time application:

If the real-time application also contains a timer task (this is the case for all multiprocessor applications), that task acts as a reference task for the asynchronous tasks. The dSPACE Real-Time Kernel derives the time stamps for asynchronous tasks in relation to the time stamps of the reference task to ensure time consistency, especially if a simulation runs over a long period of time:

$T_{hw} = T_t + \Delta T_{hw}$

where

T_{hw} = time of task driven by a hardware interrupt block
T_t = `currentTime` variable or RTK time stamp for timer task
ΔT_{hw} = time that elapsed since the last increment of T_t

Tasks driven by Software Interrupt blocks For tasks driven by Software Interrupt blocks, the dSPACE Real-Time Kernel uses the time stamps from the task that issues the software interrupt.

Note

The `currentTime` variable of the real-time simulation is calculated by Simulink Coder using floating-point numbers, whereas the time stamps are calculated by the dSPACE Real-Time Kernel using integer numbers. Since the precision of floating-point numbers decreases the larger the numbers become, the `currentTime` variable and the associated time stamps might differ slightly.

Limitations

For information on the limitations with the time-stamping feature, refer to Limitations on page 270.

Time-stamping and RTI-MP distributed tracing

Via the distributed tracing feature, ControlDesk can capture the data from the CPUs in a multiprocessor system individually and display it all in a single plot window with a common time base.

Note

Distributed tracing is available only for boards that support time-stamping and multiprocessing, namely the DS1006, DS1007 boards, and MicroLabBox.

Related topics

Basics

Limitations 270
ControlDesk Measurement and Recording

HowTos

How to Modify Data Acquisition Behavior 213

References

currentTime (RTI and RTI-MP Implementation Reference 🕮)

How to Modify Data Acquisition Behavior

Observing variable changes

Suppose you want to observe variable changes that occur in a task driven by an interrupt block. If the host service code for real-time data acquisition is placed in a timer task, some variable changes can be lost as illustrated below:

In contrast, if the host service code is placed inside the task driven by an interrupt block, all variable changes from this task can be captured as illustrated below:

This effect applies especially to the task information variables:

Some variables show different behavior depending on where the host service is executed. Consider the `overrunQueueCount` and `state` variables of a particular task, for example:

- `overrunQueueCount` is always `0` if read in the background task. This is due to the fact that the background task has the lowest priority and is therefore executed only if no other task is queued. If `overrunQueueCount` is read from within its own task, its value is always `1` or higher because it is incremented when the task is called and decremented when the task is finished.
- `state` is always `idle` (0) if read in the background task. If `state` is read from within its own task, it is always `running` (2), and if read from any other task, it can be `running` (2), `ready` (1) or `idle` (0).

If you use one of ControlDesk's instruments to access a variable, ControlDesk automatically uses the host service that is executed in the background task.

If you trace a variable via ControlDesk instruments, you can specify the desired host service via the **Measurement Configuration** controlbar.

> **Note**
>
> The default host service code for data acquisition is executed with the fastest timer task. This code is omitted automatically if you place at least one **Data Capture** block in your model.

Method

To modify the data acquisition behavior

1 Place up to 27 **Data Capture** blocks from RTI's Extras library in your model, and open the **Block Parameters** dialog.

- To achieve a host service that is executed in a task driven by an interrupt block, place the **Data Capture** block inside a function-call subsystem. Specify **-1** as the **Sample time**.
- To achieve a host service that can be turned on or off via a control variable, place the **Data Capture** block inside an enabled subsystem which is not located inside a triggered or function-call subsystem. Specify the desired **Sample time** in the **Block Parameters** dialog.
- To achieve a downsampling compared to the default host service code, place the **Data Capture** block in your model so that it is not inside a conditionally executed subsystem. Specify the desired **Sample time**. You can also achieve a downsampling via ControlDesk without having to modify your model.
- To achieve a host service that is executed on specific signal conditions, place the **Data Capture** block inside a triggered subsystem (trigger type: *rising*, *falling* or *either*). Specify **-1** as the **Sample time**.

> **Note**
>
> Do not place the **Data Capture** block in the background task or in a function-call subsystem driven by a **Software Interrupt** block from within the background task. This leads to unpredictable results because there are no useful time stamps for the background task.

2 In the **Block Parameters** dialog, specify a unique **Service number** and **Service name**.

3 Select the desired service for each data acquisition instrument in ControlDesk.

Result

You modified the data acquisition behavior.

Related topics

Basics

Time-Stamping and Data Acquisition 210

References

Data Capture Block (RTI and RTI-MP Implementation Reference 🕮)

Reaction to Run-Time Errors

Run-Time errors

When you run a real-time simulation, a run-time error might occur, meaning a situation where the real-time application can no longer be executed. Such run-time errors can result, for example, from incorrect code, task overrun or user-specific function calls to certain Message Module functions or the `ssSetErrorStatus` function.

Types of run-time errors

RTI responds to run-time errors by stopping the simulation. The termination code is executed and the termination values are written to the I/O.

- If the run-time error was caused by a nonfatal error such as an overrun situation, the application remains in the simulation state STOP and can be restarted. For instructions, refer to How to Set the Simulation State on page 208.
- All other run-time errors are most likely fatal errors. Therefore, RTI also terminates the real-time application via the `exit()` function. As a result, you cannot restart the application via the simulation state, but you have to reload the application.

Related topics

HowTos

How to Send Messages to the Message Module 197
How to Specify Overrun Strategies for Tasks 90

References

errorNumber (RTI and RTI-MP Implementation Reference 🕮)

Different Results from Different Simulation Modes

Simulation modes

With RTI and RTI-MP, you can simulate your model in different modes. The following simulation modes are available:

- Simulink simulation
- Real-time simulation

For details on the simulation modes, refer to Control design and Simulink simulation and RCP with dSPACE Prototyper in Development Steps on page 15.

Reasons for different simulation results

The results of the Simulink simulation provide a reference for real-time simulation results. However, there can be differences between the results from different simulation modes. In the following you are informed of the problems that can arise when you run simulations in different modes. Due to the complexity of this issue, it is not possible to give precise instructions on preventing or minimizing the differences. You should be aware of the facts given below when comparing the results of Simulink and real-time simulation:

- Different behavior caused by interrupt-driven tasks
- Different rules for modeling
- Different task scheduling in multiple timer tasks mode
- Different output values of I/O blocks
- Incongruent signals from signal generators

Different behavior of interrupt-driven tasks

In Simulink simulation mode, modeled interrupt-driven tasks are executed in a different way than in real-time simulation.The Simulink simulation executes these tasks in fixed steps of periodic tasks. That means they are executed when the RTI interrupt block connected to the function-call port of a subsystem is executed by Simulink. There are also no task interrupts in Simulink simulations.

Multiple timer tasks mode

In Simulink simulation, the tasks are executed one after another. In real-time simulation, tasks can be preempted by higher priority tasks. Thus, in real-time simulation mode, a task can interrupt another task it exchanges signals with. In this case, signals can differ from signals in Simulink simulations.

Output of I/O blocks

You have to use RTI I/O input blocks to connect the real-time simulation to the physical world. In real-time simulations, these blocks provide valid output values. If you perform a Simulink simulation, however, the RTI I/O blocks do not read physical values. Thus, their output values are always 0.

Incongruent signals of signal generators

The signals of signal generators may not be exactly congruent in Simulink simulation mode and in real-time simulation mode. For example, in real-time simulation mode, an edge of a square wave may occur one calculation step

earlier or later than in Simulink simulation mode. The algorithms which are calculated in the two simulation modes are not 100% identical. Thus, different rounding errors occur. This basically affects every Simulink block. However, for a square wave generator where an edge shifts by one sampling step, the effect of the different rounding errors is more easily seen than for other blocks.

Building and Downloading the Model

Introduction

When the modeling of an RTI or RTI-MP model in Simulink is complete, real-time code can be generated and downloaded to the dSPACE system.

Where to go from here

Information in this section

Basics on the Build and Download.....220
Debugging an Application.....234
Creating an Application Without Connected dSPACE System.....241
User-Specific Files, Source Files and Libraries.....243

Basics on the Build and Download

Introduction

To ensure that your RTI or RTI-MP model is properly built and downloaded, make yourself familiar with the build and download process.

Where to go from here

Information in this section

Introducing the Build and Download Process.....220
Basics on the Generation of TRC Files.....224
How to Specify Options for the Build Process.....226
How to Start the Build and Download.....228
How to Automate the Build Process.....230
How to Back Up Real-Time Applications.....231
How to Remove the Temporary Files of the Build Process.....232

Introducing the Build and Download Process

Registering platforms

Before you can start the build and download procedure, you have to ensure that your dSPACE hardware is registered correctly, for example, by using ControlDesk's Platform Manager. Refer to Basics of Handling Platforms (ControlDesk Platform Management 🕮).

Code format

RTI and RTI-MP use the RealTime Code Format of Simulink Coder for real-time applications. See the Simulink Coder user documentation.

File handling

RTI and RTI-MP always generate a system description file (SDF), which contains all the relevant information about the various object files and TRC files. For example, with ControlDesk, you use this file to download the application to your dSPACE system or to access the signals and parameters of the simulation model.

Several intermediate files are generated during or required by the build process. These are located in the `\<[sub]model>_<target>\` build folder(s), where `<target>` is your active platform support. The current working folder contains user-supplied files (for example, `<[sub]model>_usr.*`) and the files that are needed for the real-time application (such as the real-time object, TRC file, map file, etc.).

A2L file generation

RTI can additionally generate an A2L file (ASAM-MCD 2MC file format) for a Simulink model. The A2L file generation is based on Simulink Coder's ASAP2 file generation feature.

You can enable the generation of an A2L file with the following methods:

- By selecting the TRC and A2L option as the Variable description file format in Configuration Parameters - Code generation - RTI variable description file options.
- By assigning TRC and A2L to the VDFType model parameter.

The generated A2L file is named `<ModelName>.a2l` and is created in the current MATLAB working directory.

RTI deletes an existing A2L file when a build or a make process is started, to ensure that no A2L file from a previous build is located in the working directory. An existing A2L file will also be deleted, if you specified to generate the TRC file only.

The A2L file generation is available for MicroAutoBox II, DS1007, MicroLabBox, and MP systems based on DS1007 and MicroLabBox. It is not available for DS1006 and DS1104.

Using model referencing If the model for which you want to generate the A2L file contains referenced models, RTI creates an A2L file that contains the complete model hierarchy.

As of MATLAB R2021a, the ASAP2Interface option is no longer available in the Code generation - Interface page. If an already specified option differs from the specified VDFType or Variable description file format, a warning is generated during the initial build check.

Application download

RTI The real-time application is downloaded to the platform that you have specified in RTI. The platform has to be registered in your experimentation software with the same platform name used in RTI.

For example, if RTI1007 is running, the platform name is set to *ds1007*. To download the real-time application, a DS1007 PPC Processor Board must be registered in ControlDesk with *ds1007* as the platform name.

RTI sets the following default platform names, if you create a new model or switch the RTI platform:

- ds1006
- ds1007
- ds1104
- ds1202 (for MicroLabBox)
- ds1401 (for MicroAutoBox II)

RTI-MP The MP real-time application is downloaded to the platform that you have specified in RTI-MP. The multiprocessor platform must be registered in your experimentation software with the same platform name used in RTI-MP.

RTI-MP sets the following default platform names, if you create a new MP model:

- Multiprocessor
- ds1007
- ds1202 (for MicroLabBox)

Multiprocessor hardware is registered with *Multiprocessor* as the default platform name.

You have to adapt the platform names for RTI and RTI-MP if the names of the registered platforms differ from the default ones. See Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮) (for RTI) and Advanced Page (Multiprocessor Setup Dialog) (RTI and RTI-MP Implementation Reference 🕮) (for RTI-MP).

Overview of the build process

When you initiate the build process, a number of utilities are invoked sequentially to build the real-time program and download it to the hardware.

The build process can be divided into five separate phases:

- Code generation
- Precompilation
- Compilation
- Postcompilation
- Download

RTI-MP separates the submodel for each CPU from the overall model and carries out a similar build process for each submodel.

Code generation phase In this phase all the C code needed for the real-time application is generated. The `make_rti` command is started and performs the following steps sequentially:

- Simulink Coder is called to generate an intermediate file, `<model>.rtw`, from the Simulink model. This file is used as the basis for code generation by the Target Language Compiler (see next step).
- The Target Language Compiler (TLC) is invoked. It reads the `<model>.rtw` file as well as the TLC files provided by Simulink Coder and RTI to generate the C code for the model and the dSPACE I/O blocks used in the model.

Precompilation phase In this phase the results of the previous build process are removed.

Compilation phase In this phase the C source is compiled. The application-specific `<model>.mk` makefile is invoked by the `dsmake` make utility. The utility calls the compiler/linker, which compiles the C sources of the model and the

dSPACE real-time simulation environment. It also links the object files and libraries into an executable file.

Postcompilation phase In this phase the C sources are translated into the final application, which can be executed on the dSPACE real-time hardware.

Download phase In this phase the dSPACE loader program, which downloads the executable file to the real-time processor, starts its execution and checks whether it was started correctly. With RTI-MP the application is not downloaded automatically.

Note

- Whenever you start a build and download, RTI and RTI-MP update the corresponding system description file (`<model>.sdf`). Therefore, you should not edit this file. If you need to customize the download procedure, you can use the user system description file (`<model>_usr.sdf`).
- If you run a multiprocessor system and experience an overrun in the first simulation step of the fastest timer task of a CPU, you might be able to solve this problem via increasing the first simulation steps. Refer to Configure Initial Step Size Dialog (Multiprocessor Setup Dialog) (RTI and RTI-MP Implementation Reference 🕮).

Tip

- You can automate the build and download procedure via the `rti_build2` or `rtimp_build2` command.
- RTI supports the *Incremental Code Generation* feature of the Simulink Coder. If the model is unchanged and RTI-specific settings have not been modified, Simulink does not generate new code.
- With RTI-MP you can use the `rtimpdiag` diagnostic utility to get information on the real-time program. For more information, refer to The RTI-MP Diagnostic Tool on page 234.

Note

Simulink Coder copies the generated code to the Simulink Cache (SLXC) file. To enforce a rebuild, you have to delete the build folder, the slprj folder, and the Simulink Cache file. If you use the `rti_build2` or `rtimp_build2` command with its `CleanUpAll` method, this is automatically done.

Automatic build procedure

Using the `rti_build2` and `rtimp_build2` commands, you can run build processes virtually unattended or even create build scripts that carry out several actions for you. In addition, the commands provide the following customization options:

- You can be prompted to check the task configuration or leave the task configuration as it is to avoid interaction.
- You can log the messages of the build process to a file.

- You can create a backup of all the run-time files directly after successfully building a model. This lets you restore the last successful build whenever the run-time files are deleted.
- You can delete all the temporary files created during the build process.

For RTI-MP models, some additional options are available:

- If an error occurs while building one submodel, the build process can be continued with the remaining submodels.
- You can build either all submodels or just a subset of submodels.

For a detailed description of the commands and their options, refer to rti_build2 (RTI and RTI-MP Implementation Reference 🕮) and rtimp_build2 (RTI and RTI-MP Implementation Reference 🕮).

Related topics

Basics

Basics on Multiprocessor System Platforms (ControlDesk Platform Management 🕮)

References

Advanced Page (Multiprocessor Setup Dialog) (RTI and RTI-MP Implementation Reference 🕮)
Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮)
File Overview (RTI and RTI-MP Implementation Reference 🕮)

Basics on the Generation of TRC Files

Introduction

A TRC file provides information on the variables that are used in a simulation application. This variable description is a data interface between the simulation application executed on dSPACE hardware or VEOS and the dSPACE experimentation software.

In the experimentation software, you get access to the variables by loading a system description file (SDF file). However, the variable description itself is stored in a trace file (TRC file), that is then also loaded. The following description refers to the contents of variable description files in TRC format.

Generating and consuming software products

Build processes that include the generation of TRC files can be started via:

- RTI and RTI-MP
- ConfigurationDesk
- Model Interface Package for Simulink

  The generated intermediate files are used by VEOS and ConfigurationDesk to create a platform-specific TRC file.

dSPACE products that need a generated TRC file to access variables in a simulation application are, for example, ControlDesk, AutomationDesk, ModelDesk, and also custom applications based on dSPACE XIL API.

Data source of the variable description

When you build a simulation application, Simulink Coder provides information on the model's variables and parameters in an internal format. This data is used by dSPACE within the build process to generate a variable description in TRC file syntax.

The contents of a variable description depend on the following points:

- The way you have implemented and prepared your model. For example, variables in a subsystem can be excluded by using the subsystem omission tag (`DsVdOmit`).
- Simulink Coder version (represented by the MATLAB Release number).
- Configuration settings for code generation, refer to Simulink configuration settings on page 225.

Configuration settings for the TRC file generation

The contents of the TRC file can be configured by various settings. The most relevant settings are described below.

Simulink configuration settings The behavior of Simulink Coder can be influenced generally by the **Code Generation** options, that you can find in the model's **Configuration Parameters**. Also the optimization settings, such as the **Default parameter behavior** setting or the **Signal storage reuse** setting, have a direct effect on the available variables in the generated code and therefore on the TRC file generation.

> **Note**
>
> Not all configuration settings lead to the expected behavior, for example, variables can be missing in the TRC file as a result of the Simulink Coder internal optimizations.

dSPACE configuration settings If you use a system target file for a dSPACE platform, you can use the additional code generation options in the model's **Configuration Parameters**. Especially the **RTI variable description file options** page provides settings to configure the contents of a generated TRC file. For further information, refer to Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮).

How to Specify Options for the Build Process

Objective

You can configure the build process according to your requirements by specifying build options provided by RTI.

Available options

Simulink, Simulink Coder, RTI and RTI-MP provide options for:

- Basic simulation settings
- Task configuration and data typing
- Code optimization options
- Configuration of the variable description file
- Non real-time simulation and processor board selection
- Make process customization
- Build and download options

For a complete list of options, see *RTI Dialog Reference* and *RTI-MP Dialog Reference* in the RTI and RTI Implementation Reference.

dSPACE single-processor systems You have to specify the majority of real-time relevant settings in the model-specific configuration set in the Model Explorer. For details, refer to *Model Configuration Parameters Dialogs* in the *RTI and RTI-MP Implementation Reference*. You can access this dialog via the **View – Model Explorer** menu command. Open the active configuration to display the **Configuration Parameters** dialog.

dSPACE multiprocessor systems You have to specify the majority of real-time relevant settings in the **Multiprocessor Setup** dialog. For details, refer to Multiprocessor Setup Dialog (RTI and RTI-MP Implementation Reference 🕮). You can access this dialog via the **Multiprocessor Setup** block, which is located on the root level of your multiprocessor model.

Method

To specify options for the build process

1 Set the options for the build process. The default values are sufficient for most applications.

> **Tip**
>
> If you run a single-processor system, you can configure your model by activating a suitable configuration set. For instructions, refer to How to Specify RTI-Specific Settings for a Model on page 27.

Example

Consider the throttle control example (refer to Throttle Control Example on page 16), and suppose you want to build it for a MicroAutoBox II or a DS1007

multiprocessor system with solver type ode2, a fixed-step size of 0.00065 sec, and using the single timer task mode.

MicroAutoBox II The following illustrations show the corresponding settings for the Configuration Parameters dialog (for other single board systems the settings are similar):

DS1007 multiprocessor system The following illustrations show the corresponding settings for the Multiprocessor Setup dialog:

Related topics

HowTos

How to Adapt the Default Configuration for Model Properties..... 26
How to Configure Models for MP Systems..... 111
How to Create an Application Without a Connected dSPACE System..... 241
How to Specify RTI-Specific Settings for a Model..... 27
How to Start the Build and Download..... 228

References

RTI-MP Dialog Reference (RTI and RTI-MP Implementation Reference 🕮)
set_rti (RTI and RTI-MP Implementation Reference 🕮)

How to Start the Build and Download

Objective

Once you have specified the required options, you can start the build and download procedure.

Preconditions

- Your hardware must be registered, for example, via the Platform Manager in ControlDesk. Ensure that the platform name is specified correctly. For details, refer to Basics of Handling Platforms (ControlDesk Platform Management 🕮).
- In MATLAB, you set the desired working folder.
- The model you want to build and download is open or activated.

Possible methods

There are different instructions for the build and download procedure of RTI models and RTI-MP models:

- For building and downloading RTI models, refer to Method 1 on page 229.
- For building and downloading RTI-MP models, refer to Method 2 on page 229.

Note

When you start a build process, the existing PPC/X86/RTA, MAP, TRC, and TRZ files are deleted to guarantee the consistency of the newly generated files.

Method 1

To start the build and download for an RTI model

1 Start the build process by using one of the following methods:
   - Clicking the **Build** command in the menu of the Simulink model dialog.

     You find the command in **C CODE - GENERATE CODE - Build**. The **C CODE** ribbon is available, after you have activated the Simulink Coder in the **APPS** ribbon.
   - Entering **`Ctrl`** + **`B`** as the shortcut key.

   By default, the **Load application after build** option in the **Code Generation - RTI load options** page of the **Configuration Parameters** dialog is set. The built application is therefore automatically downloaded to the hardware.

Method 2

To start the build and download for an RTI-MP model

1 Generate a new `<model>_tp.m` file whenever you build a model for the first time. The file must be regenerated if you changed or added any Gigalink connection, or if you are building an application for a virtual hardware system that is not connected to your PC.

   The `<model>_tp.m` file contains the topology information of your multiprocessor system. You can generate it via the **Multiprocessor Topology Setup** dialog, which you can open via the **MP network topology** button on the Advanced page of the **Multiprocessor Setup** dialog.

2 - Open the **Multiprocessor Setup** dialog, and select the **Main** page.
   - To generate object files for all CPUs at once, click **Build All**.
   - To generate an object file for a single CPU, click **Build** for the corresponding CPU.
   - When an up-to-date object file for a CPU exists, the **Build** button label reads **Build OK**.

   When all the object files are available, the **Download** button is enabled.

3 Click the **Download** button.

Result

Your model is built and downloaded.

For automating the build process, refer to How to Automate the Build Process on page 230.

Related topics

Basics

Basics of Handling Platforms (ControlDesk Platform Management 🕮)

HowTos

How to Create an Application Without a Connected dSPACE System..... 241
How to Specify Options for the Build Process..... 226

References

Build Options Page (CPU Options Dialog) (RTI and RTI-MP Implementation Reference 🕮)
Multiprocessor Topology Setup Dialog (RTI and RTI-MP Implementation Reference 🕮)

How to Automate the Build Process

Objective

When working with models that are very large and take a long time to build, you might want to automate the build procedure.

Method

To automate the build process

1. Open the RTI or RTI-MP model you want to build.
2. Change to the MATLAB Command Window and enter the `rti_build2` or `rtimp_build2` command with the desired build command and options using the following syntax:

```
[errorFlag, errorMsg] = rti_build2(mdlName, 'Command',...
<CommandValue>, <Parameter1>, <Value1>, ...
<Parameter2>, <Value2>, ...)
```

or:

```
[errorFlag, errorMsg] = rtimp_build2(mdlName, 'Command', ...
<CommandValue>, <Parameter1>, <Value1>, ...
<Parameter2>, <Value2>, ...)
```

Note

Optional parameters are set in bold.

where `<commands>` is one of:

- `'CodeGen'` (or `'C'` for short)
- `'CodeGen&Load'` (or `'CL'` for short)
- `'CodeGen&Make'` (or `'CM'` for short)
- `'CodeGen&Make&Load'` (or `'CML'` for short)
- `'Load'` (or `'L'` for short)

Result

Your model is built according to the options you specified.

Tip

Combinations of the options `CodeGen`, `Make` and `Load` should be used as follows:

- Rebuild the entire model, especially if you changed the size information of any S-function (sample time, number of inputs, outputs, states, etc.) or the dimensions of data store memories used for data transfer between the model and the User-Code:
  Recompile the MEX DLL files of the modified S-functions. Then use the `'CodeGen&Make&Load'` command to rebuild the entire model.
- If it is sufficient to load the real-time application without generating new code from the model, use the `'Load'` command.

Related topics

Basics

The RTI-MP Diagnostic Tool 234

HowTos

How to Start the Build and Download 228

References

rti_build2 (RTI and RTI-MP Implementation Reference 🕮)
rtimp_build2 (RTI and RTI-MP Implementation Reference 🕮)

How to Back Up Real-Time Applications

Objective

You can create a backup of all the real-time application files directly after successfully building a model. This allows you to restore the last successful build whenever the real-time application files are deleted.

Precondition

The model you want to back up must be successfully built. For instructions, refer to How to Start the Build and Download on page 228.

Method

To backup a real-time application

1 Create the `<location>` folder where you want to put the backup.

2 Change to the MATLAB Command Window and enter the following command:

```
[errorFlag, errorMsg] = rti_build2(<mdlName>,
'Command','BackUp','BackUpFolder',<location>)
```

or:

```
[errorFlag, errorMsg] = rtimp_build2(<mdlName>,
'Command','BackUp','BackUpFolder',<location>)
```

Result

A backup copy of the generated real-time application files that are necessary to load the application is saved to the specified folder.

Note

Any existing files in the `<location>` folder are overwritten without a warning.

How to Remove the Temporary Files of the Build Process

Objective

You can remove all the temporary files created during a build process, for example, to save memory capacity.

Temporary files

Temporary files are files that are not needed to load the application, i.e. they are files that are, for example, located in the build directory of your model.

Precondition

Temporary files of a build process must be available.

Method

To remove the temporary files of the build process

1 Change to the MATLAB Command Window and to the working folder of the model.

2 Enter the following command:

```
[errorFlag, errorMsg] = rti_build2(<mdlName>,
'Command','CleanUp')
```

or:

```
[errorFlag, errorMsg] = rtimp_build2(<mdlName>,
'Command','CleanUp',['ApplNames',{<appl1>,<appl2>,...}])
```

Via the `'ApplNames',{<appl1>,<appl2>,...}` parameter-value pair of the `rtimp_build2` command, you can remove the temporary files of specific applications.

Result

The temporary files of the build process, i.e. the current build directory including its contents are removed.

Tip

If you replace the `'CleanUp'` command with `'CleanUpAll'`, not only the temporary files in your current build directory are deleted, but also files generated by RTI in your current working directory, including the generated real-time application(s). In this case, only the Simulink model and the user files will remain.

Related topics

HowTos

How to Start the Build and Download 228

Debugging an Application

Introduction

If you experience problems during the build and download or simulation, it is useful to debug your application.

For further information, see also the FAQ section of our website: http://www.dspace.com/go/FAQ.

Where to go from here

Information in this section

The RTI-MP Diagnostic Tool 234
Handling Exceptions 236
How to Trace an Exception 237
How to Find the Simulink Block Related to an Exception 239

Information in other sections

How to Start the Build and Download 228

The RTI-MP Diagnostic Tool

Starting the diagnostic tool

Note

If you have installed RCP and HIL software, the `rtimpdiag` command is available to start the RTI-MP Diagnostic Tool. This tool is based on dSPACE XIL API .NET, that must be installed from the same dSPACE Release.

After you built and downloaded a multiprocessor application, you might want some information on its status; for example, whether it was initialized correctly. RTI-MP provides an easy-to-use MATLAB-based tool that can provide you with such a basic set of information about your multiprocessor application. You can start the RTI-MP diagnostic tool by typing `rtimpdiag [<model>]` at the MATLAB prompt. If there is only a single SDF file in the working folder, you do not have to specify the SDF file name.

Diagnostic information

`rtimpdiag` provides detailed information on the running program, possible error codes and initialization failures. In addition, the number of simulation terminations and reinitializations, as well as the number of background cycles,

evaluations of the base rate, and idle cycles are reported. Refer to Troubleshooting on page 289 if `rtimpdiag` reports any problems.

If you want `rtimpdiag` to provide profiling information for the interprocessor communication that is operated with the synchronized swinging buffer protocol, you have to specify the **Profile Synchronized Swinging Buffer Communication** option for the involved CPUs (this option is available on the CPU's page of the Multiprocessor Setup dialog). As a result, `rtimpdiag` includes information on how often data was read, the associated waiting times and whether the waiting times caused an overrun situation.

Example

`rtimpdiag` displays a table similar to the following one in the MATLAB Command Window.

```
>> rtimpdiag
     Current connection = 'BUS'.
 CPU name                                          :     master       slave
 Board name                                        :     ds1007      ds1007_2
-----------------------------------------------------------------------------
 Program status                                    :    started      started
 Basic hardware initialization                     :      ok           ok
 Initialization of I/O Boards                      :      ok           ok
 Installation of interprocessor communication      :      ok           ok
 Registration of Simulink model                    :      ok           ok
-----------------------------------------------------------------------------
 Application successfully initialized              :      ok           ok
 Error code                                        :       0            0
-----------------------------------------------------------------------------
 Simulation terminated                             :      1 x          1 x
 Simulation re-initialized                         :      1 x          1 x
-----------------------------------------------------------------------------
 Number of cycles in background process            :    2742740      3067065
 Number of evaluations of the base sample rate     :     215792       216537
 Number of idle cycles                             :     41907        41907
```

- The output lines from **Basic hardware initialization** up to **Application successfully initialized** indicate which initialization steps have been completed successfully by the application.
- Errors during initialization are frequently caused by I/O boards that have been used in the models but cannot be accessed on the hardware.
- `rtimpdiag` also displays how many times the simulation has been stopped and restarted.
- After a successful initialization, some diagnostic counters are always incremented by RTI-MP. These counters are displayed in the last three lines of the `rtimpdiag` table.
  - The first counter displays the number of cycles that have been performed in the background task. If this counter stops, the real-time processor might have stopped or the application might not respond.
  - The second counter is incremented each time one step of the Simulink model is performed in the base rate service routine, which corresponds to the fastest timer task.
  - If the model is in the `STOP` or `PAUSE` state, the idle counter is incremented instead of the base rate cycle counter.

If you specified the **Profile synchronized swinging buffer communication** option, an output similar to the following results:

```
CPU name: master
 ---------------------------------------------------------------------------------------
  IPC channel | input calls | wait time | max. wait time [usec] | overrun (caused by waiting)
       1           41192         0.5              1.85                        no
CPU name: slave
 ---------------------------------------------------------------------------------------
  IPC channel | input calls | wait time | max. wait time [usec] | overrun (caused by waiting)
       0           41937         2.8               3.6                        no
```

Related topics

Basics

Handling Exceptions..... 236
Troubleshooting..... 289

HowTos

How to Start the Build and Download..... 228

References

CPU's Page (Multiprocessor Setup Dialog) (RTI and RTI-MP Implementation Reference 🕮)

Handling Exceptions

Exception mechanism

The exception mechanism allows the processor to change to the supervisor state as a result of external signals, errors or unusual conditions arising in the execution of instructions. When exceptions occur, information about the state of the processor is saved to certain registers and the processor begins execution at an address predefined for each exception. The exception message is displayed in ControlDesk. The application stops in most cases when an exception occurs. Using RTLib functions, you can handle exceptions in such a way that the system is set to a predefined state before terminating.

Arithmetic exceptions

The CPU can generate exceptions if it cannot complete a given instruction. By default, arithmetic exceptions are disabled so that the CPU continues the computation. However, the computation results are unpredictable. It is not possible to make the processor correct arithmetic exceptions.

Localization of an error

When an exception occurs (e.g. data access error, alignment error), sometimes the error message includes the address that the last operation was executed on.

The location of the address may be a clue as to which function provoked the error. However, this is not necessarily the underlying cause of the run-time error, which may have occurred in a previous function.

Information about the function associated with the address are available from the MAP file, which is generated at each compilation. However, it is not possible to find all the addresses, because the functions are indicated by their start addresses only. To find an exception that occurred within a function is very difficult, because the functions are not sorted according to location, but according to addresses (DS1006). Line numbers for identifying the instruction in the source code are not displayed.

However, there is a way to get detailed information about the location of the exception.

Related topics

HowTos

How to Specify Options for the Build Process..... 226

References

Exception Handling (DS1104 RTLib Reference 🕮)
Exception Handling (MicroAutoBox II RTLib Reference 🕮)
Exception Handling (DS1006 RTLib Reference 🕮)

How to Trace an Exception

Objective

You can identify the source code where an exception occurs.

Note

Not applicable for DS1007 and MicroLabBox (DS1202).

Precondition

To trace an exception, you must compile your model in debug mode.

Tip

It is recommended to use the Command Prompt for dSPACE RCP and HIL shortcut in the Windows Start menu for calling `ppcobjdump` (for boards with PowerPC processor) or `i686-elf-objdump` (for the DS1006). The required paths and environment settings are then automatically set.

Method

To trace an exception

1. Recompile your application using the compiler option `-g` (valid for the Microtec PPC compiler (boards with PowerPC processor) and the DS1006 compiler). This option adds line number debug information to your generated PPC or X86 file.
2. Download the application via ControlDesk. ControlDesk issues the program address of the exception.
3. Invoke the GNU utility `ppcobjdump` (for boards with PowerPC processor) or `i686-elf-objdump` (for the DS1006) to disassemble the object file and retrieve line number debug information from your application:

   PowerPC: `ppcobjdump -S -l -D applname.ppc>debug.dmp`
   DS1006: `i686-elf-objdump -S -l -D applname.x86>debug.dmp`

   This command generates a `debug.dmp` file containing the following information for each source code line of the application:
   - Line number
   - Function name
   - Module name
   - Source code
   - Assembler listing
   - Program addresses

   Note

   The source code can be printed to the dump file only if the source file and the generated executable file are in the same folder.

   For detailed information about `ppcobjdump`, refer to:
   - PPCObjdump (DS1104 RTLib Reference 🕮)
   - PPCObjdump (MicroAutoBox II RTLib Reference 🕮)

   For detailed information about `i686-elf-objdump`, refer to:
   - i686-elf-objdump (DS1006 RTLib Reference 🕮)
4. For boards with PowerPC:

   Search the `debug.dmp` file for the address where the exception occurred (see step 2 on page 238). For this address you will find the source code line and the source code itself whenever possible.

   For DS1006:

   Analyze the MAP files to find the function where the exception occurred (via its absolute start address).

   Calculate the offset of the exception to the function's absolute start address via the formula:

   $Offset_{Exception} = AbsoluteAddress_{Exception} - AbsoluteAddress_{Function}$

   Search the `debug.dmp` file for the function name and retrieve the relative start address of the function.

   Calculate the relative address of the exception by adding its offset to the relative start address of the function.

Finally search the `debug.dmp` file for the relative address of the exception to find the corresponding source code line and the source code itself.

For DS1006, the assembler commands are different and the addresses are relative.

5 Derive the respective source code line from the C source line close above the disassembled code. Check whether these instructions could have caused the exception.

6 If the specified function is associated with a Simulink block, you can find the block in your model and change its parameters.

Further information

For more information, refer to FAQ 239 at our website (http://www.dspace.com/go/FAQ).

Related topics

HowTos

How to Find the Simulink Block Related to an Exception..... 239

How to Find the Simulink Block Related to an Exception

Objective

If you want to find the Simulink block related to an exception, you can use the `hilite_system` command.

Identification tags

Simulink Coder writes system/block identification tags in the generated code. The tags are designed to help you identify the block in your source model that generated a given line of code. You should use the `hilite_system` command to trace a tag back to the generating block. For further information on identification tags and code tracing, refer to the Simulink Coder user documentation.

Precondition

The affected model must be open.

Method

To find the Simulink block related to an exception

1 Investigate the model source code (`<model>.c`) for the Simulink block causing the problem: Simulink Coder begins the source code for each Simulink block with a token that indicates the block type and the actual block, for example,

```
/* Sum Block: <S1>/Sum */
rtB.s1_Sum = - rtB.s1_Product1 - rtB.s1_Product;
```

2 Change to the MATLAB Command Window and enter the **`hilite_system('token')`** command.

For the above-mentioned Sum block, this would read

`hilite_system('<S1>/Sum')`

Example

Further information on the subsystem identifiers S1, S2, etc. is held at the end of the `<model>.h` file, for example:

```
/*
 * Comments are included throughout the generated code which
 * allow you to trace directly back to the appropriate location
 * in model. The basic format is <system>/block_name, where
 * system is the system number (uniquely assigned by Simulink)
 * and block_name is the name of the block.
 * Use the MATLAB hilite_system command to trace the generated
 * code back to the model. For example,
 * hilite_system('<S3>')    - opens system 3
 * hilite_system('<S3>/Kp') - opens and selects block Kp
 *                            which resides in S3
 * Here is the system hierarchy for this model
 *
 * <Root> : smd_1007_sl
 * <S1>   : smd_1007_sl/Equation Block
 * <S2>   : smd_1007_sl/Model Parameters
 * <S3>   : smd_1007_sl/RTI Data
...
 */
```

Related topics

HowTos

How to Trace an Exception 237

References

Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮)

Creating an Application Without Connected dSPACE System

How to Create an Application Without a Connected dSPACE System

Objective

Suppose you want to create a real-time application for a dSPACE system that is not available at your developer PC and use another PC to download it to your dSPACE system. You have to use fictional board names and block numbers for this.

Method

To create an application without a connected dSPACE system

1. Create your model as normal using the usual board numbers.
2. In S-functions and custom C code as normal use the usual board names. For example, use `DS5001_1_BASE` to access the DS5001 board with the smallest PHS-bus address if you run a PHS-bus-based system.
3. Start the build process.
   - If you use RTI, build the model with the **Load application after build** option cleared (see Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮)).
   - If you use RTI-MP, build the object files for all CPUs as normal, but do not download it.

Result

You created an application for a dSPACE system that is not connected to your host PC.

Next step

You can download the real-time application to the dSPACE system, for example, via the **Platform Manager** of ControlDesk. The board numbers of the application are mapped to the PHS-bus addresses of the I/O boards in ascending order (see PHS-Bus Address and Board Number on page 35). If the real-time application does not match the available dSPACE system (for example, if the application references an I/O board that is not present), you get an error message.

Related topics

Basics

Naming Conventions..... 33
PHS-Bus Address and Board Number..... 35

HowTos

How to Start the Build and Download..... 228

References

Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮)

User-Specific Files, Source Files and Libraries

Where to go from here

Information in this section

Adding User-Specific Files, Source Files and Libraries.....243

Information in other sections

Inserting Custom C/C++ Code.....163

Adding User-Specific Files, Source Files and Libraries

Adding hand-written code

You can use Code Generation's **Custom Code** page to add hand-written code to your real-time application.

Tip

- You can use the settings in the **Include custom c code in generated** frame in Code Generation's **Custom Code** page in addition to the settings in the RTI User-Code file (`USR.C`).
- You can use the settings in the **Additional build information** frame in Code Generation's **Custom Code** page instead of, or in addition to, the settings in the RTI User Makefile (`USR.MK`).

For RTI-MP, you have to specify the settings in Code Generation's **Custom Code** page of the main model. To have the settings transferred to a submodel, you have to select the **Use Simulink Coder Custom Code settings** checkbox on the relevant CPU pages of the **Multiprocessor Setup** dialog.

If you use Code Generation's **Custom Code** page, you cannot specify different custom code for each submodel of a multiprocessor system. If this is required, you must use the RTI User-Code files (`USR.C`) and RTI User Makefiles (`USR.MK`).

Advanced Techniques

Introduction

To achieve optimized simulations, it is useful to make yourself familiar with the following advanced techniques.

Where to go from here

Information in this section

Optimizing the Build Process and Simulation via Atomic Subsystems.....246

Excluding Subsystems from the TRC File.....250

Modifying the Block Execution Order.....256

Tuning Parameters of Transfer Fcn, Zero-Pole and State-Space Blocks.....259

Using Simulink's Model Verification Blocks.....265

Optimizing the Build Process and Simulation via Atomic Subsystems

Introduction

Atomic subsystems are a feature of Simulink that can help you optimize the build process and simulation.

Where to go from here

Information in this section

Virtual Versus Atomic Subsystems.....246
How to Bundle the Execution of Blocks.....247
How to Reduce the Function Sizes via Atomic Subsystems.....248
How to Reduce the Files Sizes via Atomic Subsystems.....249

Virtual Versus Atomic Subsystems

Types of subsystems

Simulink and Simulink Coder distinguish between virtual subsystems and atomic subsystems.

Virtual subsystems

Virtual subsystems are mainly used to structure a model. In contrast to atomic subsystems, it makes no difference to the function of the model whether a block is located in a virtual subsystem or in the parent subsystem. The code that is generated for a virtual subsystem is embedded in the code of its parent subsystem. This is also true for nested virtual subsystems. The code generated for blocks inside a virtual subsystem can interleave with code generated for blocks outside it.

Atomic subsystems

In contrast to virtual subsystems, these can be used to influence the block execution order: The code generated for atomic subsystems does not interleave with code generated for blocks outside those subsystems. The resulting code resembles an unconditionally inlined function, which is called a code block. A pure atomic subsystem is always implemented as a code block. In the Simulink model, atomic subsystems are marked with a bold block frame.

Triggered and enabled subsystems Simulink automatically specifies triggered and/or enabled subsystems as atomic and disables the **Treat as atomic unit** checkbox for them, so the specification cannot be changed.

Subsystems driven by an RTI interrupt block Subsystems that are triggered by an RTI Interrupt block are always atomic. They are always generated into separate functions.

Related topics

HowTos

How to Bundle the Execution of Blocks 247
How to Reduce the Function Sizes via Atomic Subsystems 248

How to Bundle the Execution of Blocks

Objective

You might want to bundle the execution of certain blocks so that they are computed as a code block which is not interleaved with code from other subsystems. The execution of the code block still can be interrupted by higher-priority tasks.

Method

To bundle the execution of blocks

1 Place the blocks to be bundled in a subsystem and open the Block Parameters dialog of the subsystem.

2 Select the Treat as atomic unit checkbox to make the subsystem atomic.

Result

The atomic subsystem is marked with a bold block frame.

Related topics

Basics

Virtual Versus Atomic Subsystems 246

HowTos

How to Reduce the Function Sizes via Atomic Subsystems 248

How to Reduce the Function Sizes via Atomic Subsystems

Objective

Compilers are usually limited with regard to the code size (number of lines) of functions. Therefore, you should split large functions into smaller ones.

Method

To reduce the function size via atomic subsystems

1. Select the standard subsystem you want to optimize and open its Block Parameters dialog.
2. Select the Treat as atomic unit checkbox.

   The Function packaging drop-down list is enabled on the Code Generation page.

   Tip

   Triggered and enabled subsystems are always atomic and the Treat as atomic unit checkbox is disabled.

3. Select *Function* as Function packaging.

Result

Simulink Coder does not inline the code for the subsystem into the `MdlOutputs()` function of the parent subsystem or model, but creates a separate function. This reduces the function size. However, the file size of the `model.c` file remains nearly constant.

The function name and file name options are enabled.

Note

It is not possible to change the generation of functions for subsystems driven by RTI interrupt blocks: The settings relevant to the Function packaging option are ignored.

Related topics

Basics

Virtual Versus Atomic Subsystems 246

HowTos

How to Bundle the Execution of Blocks 247

How to Reduce the Files Sizes via Atomic Subsystems

Objective

Compilers are usually limited with regard to the code size (number of lines) of source code files. A large source code file may also have a significantly longer compilation time than several small files. You should, therefore, distribute the contents of a large source code file to several smaller files.

Method

To reduce the file size via atomic subsystems

1. Select the standard subsystem you want to optimize and open its Block Parameters dialog.
2. Select the Treat as atomic unit checkbox.

   The Function packaging drop-down list is enabled on the Code Generation page.

   Tip

   Triggered and enabled subsystems are always atomic and the Treat as atomic unit checkbox is disabled.

3. Select *Function* or *Reusable Function* as the function packaging.

   The function name and file name options are enabled.

Result

As a result, Simulink Coder generates a function for this subsystem into a separate source code file and thus reduces the size of the `<model>.c` file. The name of the separate source code file can be controlled by the file name options setting.

Related topics

Basics

Virtual Versus Atomic Subsystems..... 246

HowTos

How to Bundle the Execution of Blocks..... 247

Excluding Subsystems from the TRC File

Introduction

Excluding a subsystem from the TRC file is an optimization technique provided by RTI. It is, for example, useful if you want to reduce the size of the TRC file.

Where to go from here

Information in this section

Advantages of a Reduced TRC File.....250

How to Exclude a Subsystem from the TRC File.....253

Advantages of a Reduced TRC File

Reasons for excluding subsystems

There are two reasons why you might want to exclude a subsystem from the generation of the TRC file:

- To reduce the size of the TRC file
- To hide parts of a model from other users

Reduced file size

Normally, a generated TRC file contains all the signals defined in the model, which in the case of large models can result in a large TRC file. Subsequently, such a file requires more time for generation, which in turn results in increased loading times, for example, in ControlDesk.

By adding a TRC Exclusion block to your model at a selected location or using the subsystem omission tag (DsVdOmit), you can exclude all the signals from the blocks in the selected subsystem as well as all underlying subsystems. This means the signals are not included in the TRC file when it is generated, thus reducing the size of the file.

Hiding model parts by using the TRC Exclusion block

You can also use the TRC Exclusion block to hide parts of a model from other users, e.g., for protected models. Since the block signals of the subsystem and all underlying subsystems are excluded from the TRC file when the TRC Exclusion block is active, they are not available for selection, for example, in ControlDesk's Variables controlbar.

Examples of using TRC Exclusion blocks

The following illustration shows a subsystem with a TRC Exclusion block set inactive, which is the same as if the block was not present at all. As you can see, ControlDesk's Variables controlbar displays all the block groups of the subsystem.

The TRC Exclusion block is now set active. The **Variables** controlbar no longer displays the block groups of the subsystem. The only group available is the subsystem itself. All underlying subsystems are also not visible.

You can add more than one TRC Exclusion block to a model, which gives you full control over the parts of the model that are to be hidden. An extreme scenario would be to add the TRC Exclusion block to the root of the model. This would result in an empty Model Root group in the TRC file.

Tip

The mode of the TRC Exclusion block is set via the block dialog. You can enter a value directly or via a workspace variable. By using workspace variables, you could define different generation patterns or view scenarios and switch between them according to your needs.

Hiding model parts by using the subsystem omission tag

You can exclude a subsystem from the TRC file by using the subsystem omission tag (DsVdOmit). You must enter `DsVdOmit=1` in the **Tag** edit field in the **Block Properties** dialog of the subsystem.

Exclusion is applied recursively through the model hierarchy. To reinclude subsystems, you can enter `DsVdOmit=0`. By this, you can include the variables of a specific subsystem hierarchy. You can set the `DsVdOmit` tag by using workspace variables. For example, if WSVar1 is a workspace variable, you can use `set_param(subsystemHandle,'Tag','DsVdOmit=$(WSVar1)')` to let the `DsVdOmit` tag value being evaluated during build process.

If you set `DsVdOmit=-1`, the DsVdOmit settings in the included subsystems are ignored.

Setting	Description
`DsVdOmit` tag is set to 1	The subsystem contents including all blocks beneath this subsystem do not appear in the generated variable description file. Use `set_param(gcb,'Tag', 'DsVdOmit=1').`

Setting	Description
`DsVdOmit` tag is set to 0	The subsystem contents including all blocks beneath this subsystem do appear in the generated variable description file, even if a subsystem above this subsystem has set the `DsVdOmit` tag to 1. Use `set_param(gcb,'Tag', 'DsVdOmit=0').`
`DsVdOmit` tag is set to -1	The subsystem contents including all blocks beneath this subsystem do not appear in the generated variable description file. `DsVdOmit` settings of subsystems included in this subsystem are ignored. Use `set_param(gcb,'Tag', 'DsVdOmit=-1').`

The following example, shows how the `DsVdOmit` tags can control the exclusion of subsystem variables. The variables of the grayed subsystems are not generated in the variable description file.

```
Model Root
└─ Subsystem1
    └─ Subsystem2      (DsVdOmit = 1)
        ├─ Subsystem3
        │   └─ Subsystem4      (DsVdOmit = 0)
        │       ├─ Subsystem5
        │       │   └─ Subsystem6      (DsVdOmit = 1)
        │       └─ Subsystem7
        └─ Subsystem8
            ├─ Subsystem9
            └─ Subsystem10
```

The specified `DsVdOmit` tags are evaluated, if you set the **Apply subsystem omission tags** option in **RTI variable description file options** page of the **Code Generation** dialog. Then the variables of the excluded subsystems are not generated into the variable description file.

If this option is cleared, the `DsVdOmit` tags are ignored for all subsystems in the model. All subsystems and their contents appear in the generated variable description file.

The `DsVdOmit` tags can be used together with the TRC Exclusion block in the same model.

Related topics

HowTos

How to Exclude a Subsystem from the TRC File..... 253

References

TRC Exclusion Block (RTI and RTI-MP Implementation Reference 🕮)
Variable Description File (TRC File) (RTI and RTI-MP Implementation Reference 🕮)
Variable Description File Options Page (CPU Options Dialog) (RTI and RTI-MP Implementation Reference 🕮)

How to Exclude a Subsystem from the TRC File

Objective

By excluding a subsystem, all blocks in the configured subsystem and - if not differently configured - all underlying subsystems are not generated into the TRC file.

Possible methods

You can add a TRC Exclusion block to a subsystem, refer to Method 1.

You can specify the `DsVdOmit` tag in a subsystem's Properties dialog, refer to Method 2.

Method 1

To exclude a subsystem from the TRC file by using the TRC Exclusion block

1 Open the subsystem where the TRC Exclusion block is to reside.

2 Add the TRC Exclusion block from the Extras library to the subsystem.

TRC Exclusion

The block mode in the TRC Exclusion block dialog is set to 1 (active) by default. You can switch the block inactive by opening the block dialog and setting the block mode to 0 to reinclude the subsystem in the TRC file.

Method 2

To exclude a subsystem from the TRC file by using the DsVdOmit tag

1 Open the subsystem from which you want to start the exclusion.

2 Open the Block Properties dialog of the subsystem and enter `DsVdOmit=1` in the Tag edit field.

3 Open the Code Generation dialog of your model's configuration set and select the RTI variable description file options page.

4 Select the Apply subsystem omission tags option.

Result

You excluded a subsystem from the TRC file.

Related topics

Basics

Advantages of a Reduced TRC File..... 250

References

TRC Exclusion Block (RTI and RTI-MP Implementation Reference 🕮)
Variable Description File (TRC File) (RTI and RTI-MP Implementation Reference 🕮)
Variable Description File Options Page (CPU Options Dialog) (RTI and RTI-MP Implementation Reference 🕮)

Modifying the Block Execution Order

Introduction

By specifying block priorities, you can modify the execution order of the blocks in the overall execution of the Simulink model. Illogical distribution of block priorities can cause conflicts.

Where to go from here

Information in this section

Determining the Execution Order.....256

How to Check for Illogical Block Priorities.....258

Determining the Execution Order

Specifying block priority and execution order

The priority of Simulink blocks and thus their execution order can be specified by means of:

- The `SS_OPTION_PLACE_ASAP` option in a level 2 S-function
- The block priority

SS_OPTION_PLACE_ASAP

Level 2 S-functions are computed as soon as possible by setting the `ssSetOption(SS_OPTION_PLACE_ASAP);` option in the `mdlInitializeSizes` function.

Block priority

You can specify a priority for a block in its **Block Parameters** dialog. This allows you to influence the block execution order as long as there is no topological dependency. You can specify any integer value – even negative numbers and zero: the lower the number, the higher the priority of the block.

Note

If you run an RTI-MP model, you might cause priority conflicts by assigning block priorities. Such conflicts occur between user-specific and RTI-MP-specific blocks in the extracted submodels. See How to Check for Illogical Block Priorities on page 258 for information on how you can avoid them.

Resulting execution order

First, Simulink tries to satisfy the level 2 S-function demand to place the block as soon as possible (meaning that the `SS_OPTION_PLACE_ASAP` option has the highest priority). If a model contains multiple level 2 S-function blocks with the `SS_OPTION_PLACE_ASAP` option, the associated block priorities are used to determine which of these blocks is executed first. After that Simulink looks for user-specific block priorities in the model.

Priority conflicts

Due to topological dependencies or internal conflicts, MATLAB might not be able to apply user-specific block priorities. In that case, a warning is issued, and the conflicting priorities are ignored.

Priority conflicts with RTI-MP

When extracting the submodels of the individual CPUs, RTI-MP automatically sets block priorities of 10.000 for all IPC Input blocks and -10.000 for all IPC Output blocks to achieve the smallest possible waiting times for the interprocessor communication.

To satisfy the data dependencies, Simulink ignores the block priorities for direct feedthrough terms. As a result, the IPC Input block is calculated before the IPC Output block, and MATLAB issues a warning.

You should be careful when assigning user-specific priorities to blocks because the priorities might conflict with the priorities of the IPC Input/Output block in the extracted submodels.

Note

User-specific block priorities might influence the sequence of interprocessor communication, which might lead to a deadlock, even if they do not cause a warning! So be careful when you work with the `SS_OPTION_PLACE_ASAP` option in S-functions or with block priorities in your multiprocessor model.

Since such priority conflicts appear only in the extracted submodels, you should always check the submodels of all the CPUs for priority conflicts (refer to How to Check for Illogical Block Priorities on page 258) if you specify user-specific block priorities.

For information on how RTI-MP optimizes model evaluation via block priorities, see Model Evaluation on page 120.

Related topics

Basics

Model Evaluation 120

HowTos

How to Check for Illogical Block Priorities 258

How to Check for Illogical Block Priorities

Objective

Whenever you specify user-specific block priorities in a model, you should check it for possible priority conflicts. For single-processor systems, you have to check the model itself for priority conflicts, whereas for multiprocessor systems, you have to check the submodels of the individual CPUs.

Method

To check a model or submodel for possible priority conflicts

1. If you run a single-processor system, you can continue with step 2. Otherwise, extract the submodel you are interested in via the **Show submodel** button on the corresponding CPU's page.
2. Start the Simulink simulation of the model or extracted submodel to check if Simulink issues any warnings, or simply select the **Edit – Update Diagram** command from the menu bar.

Related topics

Basics

Model Evaluation 120
Modifying the Block Execution Order 256

Tuning Parameters of Transfer Fcn, Zero-Pole and State-Space Blocks

Introduction

Linear, time-invariant systems are mostly described as either transfer functions or state-space representations. MATLAB and Simulink support both mathematical forms. When Simulink Coder generates real-time code, all transfer functions are automatically transformed into state-space representations. As a result, the transfer function parameters (polynomial coefficients or poles and zeros) defined in Simulink are not available in the real-time simulation. Instead the real-time code contains the matrix values of the state-space vectors and matrices (A, B, C, and D). It is not very practical trying to correlate both sets of parameters, especially because the transformations carried out by Simulink Coder are not described within the documentation of MathWorks®.

Where to go from here

Information in this section

MATLAB's State-Space Representation for a Transfer Function.....259
How to Tune Transfer Function Parameters via Manual Conversion.....261
How to Tune Transfer Function Parameters via Parameter Inlining.....261
How to Keep Zero Parameters of State-Space Blocks.....263

MATLAB's State-Space Representation for a Transfer Function

tf2ss function

Depending on the chosen state-space variables, each transfer function can have several or even an infinite number of corresponding state-space representations. To achieve a definite result, a particular form of state-space representation is used, for example, the normalized form.

The conversion from transfer function to state-space representation can be performed via the `tf2ss` MATLAB function. This returns the state-space matrices in the *controller canonical form*.

Consider the following general transfer function in polynomial form (system of (n-1)th order) with the coefficient numbering following MATLAB's conventions:

$$F\left(s\right) = \frac{y(s)}{u(s)} = \frac{a_1 s^{m-1} + a_2 s^{m-2} + \ldots + a_m}{b_1 s^{n-1} + b_2 s^{n-2} + \ldots + b_n}$$

If you use the `tf2ss` function to convert this transfer function into a state-space representation, the following results:

$$\dot{x} = A \cdot x + B \cdot u$$

$$\dot{x} = \begin{bmatrix} -\frac{b_2}{b_1} & -\frac{b_3}{b_1} & \cdots & -\frac{b_n}{b_1} \\ 1 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot x + \begin{bmatrix} 1 \\ 0 \\ \cdots \\ 0 \end{bmatrix} \cdot u$$

$$y = C \cdot x + d \cdot u$$

$$y = \begin{bmatrix} 0 & \cdots & 0 \frac{a_1}{b_1} & \cdots & \frac{a_m}{b_1} \end{bmatrix} \cdot x + \frac{a_1}{b_1} \cdot u$$

Note

d is only different from zero if the degree of the numerator polynomial is equal to the degree of the denominator polynomial. In all other cases, d equals zero.

Example

Consider the following model:

Applying the conversion derived above will lead to the following state-space representation of this system:

$$A = \begin{bmatrix} -\frac{4}{3} & -\frac{5}{3} & -\frac{6}{3} & -\frac{7}{3} \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 0 & 0 & \frac{1}{3} & \frac{2}{3} \end{bmatrix}$$

B always has the form given above and d equals 0 in this case.

The code-generating process of Simulink Coder introduces the two vectors A and C (see TRC file), which can be accessed via ControlDesk, where these parameters are referenced as follows:

$$A[1,1] = -\frac{4}{3} \qquad C[1,1] = 0$$

$$A[2,1] = -\frac{5}{3} \qquad C[2,1] = 0$$

$$A[3,1] = -\frac{6}{3} \qquad C[3,1] = \frac{1}{3}$$

$$A[4,1] = -\frac{7}{3} \qquad C[4,1] = \frac{2}{3}$$

All the elements that are zero or one have been optimized out by Simulink Coder.

How to Tune Transfer Function Parameters via Manual Conversion

Objective

To derive the new values of the state-space matrices from the transfer function, you can use MATLAB's `tf2ss` function, which directly calculates all matrix positions. The state-space matrices are available in the variable description file and can be changed via ControlDesk.

Method

To tune transfer function parameters via manual conversion

1 In the MATLAB Command Window, enter the following command:

```
[A,B,C,D] = tf2ss(NUM,DEN)
```

using your desired numerator and denominator.

Tip

Enter `help tf2ss` to get further information on the command and its syntax.

2 Add the State-Space block from the Simulink library to your model and open its Block Parameters dialog. Set the parameters according to the results calculated via the `tf2ss` function.

Result

The state-space matrices of the State-Space block (A, B, C and if it exists, D) are available in the variable description file and can therefore be changed via ControlDesk.

Related topics

HowTos

How to Tune Transfer Function Parameters via Parameter Inlining..... 261

How to Tune Transfer Function Parameters via Parameter Inlining

Objective

It is possible to use parameter inlining for tuning transfer function parameters like any other parameters. This lets you access the transfer function parameters directly via ControlDesk.

Method

To tune transfer function parameters via parameter inlining

1 Place the vectors of the polynomial coefficients in the MATLAB workspace, for example,

```
myNum = [1 2]
myDenum = [3 4 5 6 7]
```

2 Specify the workspace parameters (for example, `myNum` and `myDenum`) in the Block Parameters dialog of the Transfer Fcn block.

3 Set the Default parameter behavior setting to *Tunable* in the Code Generation - Optimization dialog of the model's Configuration Parameters dialog, and specify tunable parameters via the Configure button.

Note

With RTI-MP, the tunable parameters are not CPU-specific, if you have set the Always create a new submodel option on the CPU's page in the Multiprocessor Setup dialog. Then, the tunable parameters declared in the overall model are used for all submodels. If the Always create a new submodel checkbox is cleared, you can define the tunable parameters per submodel.

4 In the **Source list** select the referenced workspace variables you want to tune, for example, "myNum" and "myDenum" and add them to the **Global (tunable) parameters** table.

Result

All the numerator and denominator coefficients in the generated code are implemented as variables that can be changed during real-time simulation.

Note

- Changing the highest-order coefficient of the denominator of a transfer function to zero causes a change in the model structure, which always requires that you rebuild the real-time code.
- If you changed this coefficient to zero, a run-time error such as division by zero could result. Changing zero coefficients to non-zero has no effect on the real-time simulation because they are not implemented in the real-time code.

Related topics

HowTos

How to Tune Transfer Function Parameters via Manual Conversion..... 261

References

Model Parameter Configuration Dialog (RTI and RTI-MP Implementation Reference 🕮)
Model Parameter Configuration Dialog (RTI and RTI-MP Implementation Reference 🕮)

How to Keep Zero Parameters of State-Space Blocks

Objective

Simulink Coder tries to optimize out unnecessary parameters of State-Space blocks. However, for parameter tuning you might want to keep all parameters. There is a simple workaround for keeping the parameters of the system matrices of a State-Space block.

Method

To keep zero parameters of a State-Space block

1 Enter nonzero values for all parameters of the State-Space block.

2 Set the **Initial simulation state** to *STOP*.

- For RTI, refer to Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮).

- For RTI-MP, refer to Main Page (Multiprocessor Setup Dialog) (RTI and RTI-MP Implementation Reference 🕮).

3 Build and download the model.

4 Download the desired parameters and switch the simulation state to run using ControlDesk.

Related topics

Basics

Simulation Control (RUN/STOP Mechanism)..... 205

References

Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮)
Main Page (Multiprocessor Setup Dialog) (RTI and RTI-MP Implementation Reference 🕮)

Using Simulink’s Model Verification Blocks

Introduction

RTI supports Simulink's model verification blocks. They allow you to observe signals and behavior of your model execution during real-time simulation.

Where to go from here

Information in this section

Basics of Simulink’s Model Verification Blocks.....265
How to Use Simulink's Model Verification Blocks 265

Basics of Simulink’s Model Verification Blocks

Model verification

RTI supports the Simulink Model Verification blocks for the real-time simulation. These blocks can observe the signals of the Simulink simulation and issue a warning or even stop the simulation if certain conditions occur. For details on the Model Verification blocks, refer to the Simulink documentation by MathWorks®.

To control the behavior of the Model Verification blocks in the real-time simulation, use RTI’s **Assertion mode** setting. Depending on how you set this, one of the following actions is carried out if a verified signal leaves the desired range, meaning that the assertion occurs:

- OFF: No reaction (default).
- WARN: Issue a warning message in the Log Viewer of ControlDesk.
- STOP: Issue an error message and stop the simulation, i.e., set the simulation state to *STOP*.

Note

In contrast to the Simulink simulation the behavior of the real-time simulation does not depend on the **Stop simulation when assertion fails** setting, which is available for each Model Verification block.

How to Use Simulink's Model Verification Blocks

Objective

To control the behavior of Simulink's Model Verification blocks, you must specify RTI's **Assertion mode** setting.

Method

To use a Simulink's Model Verification block

1. Add the desired Model Verification block to your model.
2. Connect it to the desired signal that you want to observe.
3. Specify the initial assertion mode that is active after starting the real-time simulation via the **Model Verification block enabling** setting located on the **Data Validity** page of the **Diagnostics** category the configuration parameters:
   - You can globally *Enable all* or *Disable all* Model Verification blocks, overruling the local **Enable assertion** settings of the individual Model Verification blocks.
   - You can select the *Use local settings* entry (default), which enables the Model Verification blocks whose **Enable assertion** setting is selected.
4. Specify the desired behavior of the real-time simulation in the event that an assertion occurs:
   - RTI:

     Open the Code Generation dialog of the model's configuration set in the Model Explorer and select the **RTI simulation options page**.
   - RTI-MP:

     Open a CPU's page of the **Multiprocessor Setup** dialog and open the **CPU Options** dialog on the **Build Options** page.
5. Select the **Assertion mode** setting: OFF, WARN or STOP.

Result

You prepared a signal to be observed during real-time simulation using a Simulink Model Verification block.

Related topics

References

Build Options Page (CPU Options Dialog) (RTI and RTI-MP Implementation Reference 🕮)
Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮)
Code Generation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮)
Diagnostics Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮)

Conventions and Limitations

Introduction

Even though RTI is designed to work seamlessly with Simulink and Simulink Coder, there are a few technically related conventions and limitations.

Where to go from here

Information in this section

Conventions.....268
Limitations.....270

Conventions

General

There are a number of general conventions for implementing models with RTI and RTI-MP.

Where to go from here

Information in this section

General Conventions.....268
Data Type Restrictions.....269

General Conventions

Conventions for implementing models

The following general conventions are available for implementing models with RTI and RTI-MP.

Naming conventions You have to follow the MATLAB naming conventions for files and directories. The naming conventions must also be fulfilled by automatically named folder paths, for example, the Documents folder on your PC.

Note, that you also have to follow the naming conventions of the C/C++ standard. Variable names must not contain double underscores. The names also must not start with an underscore followed by an uppercase letter.

For further restrictions, refer to General Limitations on page 270.

File names Ensure that you give your Simulink model a different name from the files that were initially installed by dSPACE or MATLAB; for example, do not use the name `ds1007.slx`. As a rule of thumb, you should not use file names that start with `ds`, `rt` or `rti`. This rule applies even if the file name extensions are different.

File names must not contain any blanks.

Naming of interrupt blocks Do not use identical names for different interrupt blocks in your model. The experimentation software, such as ControlDesk, requires unique names for all of the Task Info's subgroups.

Numerical integration For real-time simulation, use only fixed-step integration methods.

Sampling time For discrete blocks located in a timer task, you have to set the sample times to integer multiples of the base sample time.

Data Type Restrictions

Restrictions for data types

MATLAB, Simulink, Simulink Coder, RTI, and RTI-MP support various data types. However, there are a few restrictions:

- If you are uncertain whether an integer signal might fall outside the valid signal range, you should select the **Saturate on integer overflow** checkbox in the **Block Parameters** dialog of the corresponding block. However, selecting this checkbox results in code that is computationally more intensive. If you use this option for a large number of integer blocks, the real-time code might become slower than if you were not using data typing.
- If you clear the **Saturate on integer overflow** checkbox for an integer block, an overflow leads to an unexpected result that depends on both the compiler and hardware.
- ControlDesk displays the data types according to their representation on the hardware; thus the data type displayed in ControlDesk's **Variables** controlbar can differ from that defined in Simulink. This is not an error.

  The following data types are used:

  - All integer data types have the length specified in Simulink.
  - The Boolean data type is represented by the unsigned 8-bit integer data type.
  - The single data type is a 32-bit floating-point variable.
  - The double data type is a 64-bit floating-point variable.

Related topics

Basics

Data Type Basics 37

Limitations

Introduction

There are a number of general limitations for implementing models with RTI and RTI-MP.

Where to go from here

Information in this section

General Limitations....270
Unsupported Simulink and Simulink Coder Features....272
Limitations with Stateflow....275
Limitations in Task Handling....276
Limitations with RTI-MP....276
Limitations with the RTI Gigalink Blockset....279
Limitations with Model Referencing....279
Limitations for Custom Code....282
Limitations Regarding Simulink Variables in the TRC File....283
Limitations in Building Models....285
Miscellaneous Limitations....286

General Limitations

Limitations for implementing models

The following general limitations are available for implementing models with RTI and RTI-MP.

Unsupported blocks

Blocks not supported by Simulink Coder There are a few blocks which are not supported by Simulink Coder, for example, the Algebraic Constraint block. For details, refer to the Simulink Coder user documentation.

Using templates

The properties for the RTI platform support are configured in model templates.

You have to note the following limitations:

- You cannot use dSPACE blocks, such as RTI and RTI-MP blocks or blocks from board-specific RTI blocksets, in a model template.
- Using templates from previous dSPACE Releases is not supported.

Excluding blocks from simulation

Commenting out or commenting through RTI and RTI-MP blocks in your model to exclude them from simulation is not supported.

Restricted character encoding

Block names, signal labels and annotations using the full range of UTF-16 encoding are not supported by RTI and RTI-MP. Such characters might lead to problems when you generate code. Use only ASCII characters to avoid this.

Additionally, you should not use the following characters to avoid conflicts in the TRC file generation:

- "
- /
- .

To check the character encoding used in your model, type the following command in the MATLAB Command Window:

```
modeladvisor(<ModelName>, 'mathworks.design.characterEncoding')
```

Nonfunctional signal sinks

Most blocks from the Simulink Sinks or Additional Sinks library have no function when simulated on dSPACE hardware. You can leave these blocks in your Simulink model – but keep in mind that RTI does not create C code for them. The affected signal sinks are shown below:

Simulation parameters

Fixed-step size “auto” is not supported.

Time-stamping

The code generated by Simulink Coder does currently not support the time-stamping feature. Blocks that need the simulation time for their calculations use the `currentTime` variable instead.

The dSPACE Real-Time Kernel calculates the time stamps using integer numbers, whereas Simulink Coder calculates the `currentTime` within the fastest timer task using floating-point numbers. The increment of the `currentTime` is the fixed step size. Since the precision of floating-point numbers decreases the larger the numbers become, the `currentTime` variable and the associated time stamps might differ slightly.

Combining interrupts

In combining interrupts to trigger the same subsystem, the following limitations apply:

- The code generated for the function-call subsystem is not safe on reentrance. That is, the data of the first call of a task is lost if another call starts. This is due to the global definition of block outputs in the function-call subsystem. Therefore, if two tasks act simultaneously on the same function-call subsystem, you cannot predict which block outputs will result. Even serious memory corruption can result depending on the blocks calculated in the task.

  For this reason you should use identical priorities when combining interrupt sources: Tasks of the same priority do not interrupt each other.
- You cannot combine the function-call signals from RTI interrupt blocks with those of other blocks, for example, a Function-Call Generator block or Stateflow chart.
- You cannot use the `ssEnableSystemWithTid` and `ssDisableSystemWithTid` macros to enable and disable a function-call subsystem driven by an RTI interrupt block because the subsystem will always be executed when it is triggered by an RTI interrupt block.
- With RTI-MP you can combine only local interrupts to trigger the same subsystem, i.e., all interrupts must originate from the same CPU. None of the function-call signal lines may contain an IPI block.

Absolute time support in triggered subsystems The scaling of the absolute time might be incorrect. For more information, refer to the Mathworks website: http://www.mathworks.com/support/bugreports/1274779.

Related topics

Basics

Limitations for Custom Code 282
Limitations in Building Models 285
Limitations in Task Handling 276
Limitations with RTI-MP 276
Limitations with Stateflow 275
Miscellaneous Limitations 286

Unsupported Simulink and Simulink Coder Features

Introduction

There is no support, or only very restricted support, for the following features:

Blocks not supported by RTI

The following blocks are not supported by RTI:

- Initialize Function block
- Terminate Function block

Branching of output ports of asynchronous Rate Transition blocks is not supported

Simulink allows to branch the output signal of a Rate Transition block to drive multiple asynchronous subsystems. This is not supported by RTI. The following error message occurs when building or simulating the model: The rate transition block '<model name>/Rate Transition' has multiple destinations. All destinations must be in asynchronously executed function-call subsystems with the same priority.

Restricted support of Model Workspaces

MATLAB provides Model Workspaces, which means that each model has its own workspace for storing variable values. For RTI, you have to take several limitations concerning Model Workspaces into account:

- You cannot configure RTI blocks via MATLAB variables which are specified in Model Workspaces.

  These variables need to reside in the MATLAB workspace instead.
- Parameters defined in the model workspace of referenced models can be accessed via the **Model Parameters** group of the referenced model. Block-level references to these parameters can be accessed via the referenced model group of the respective model.

Restricted support of Simulink task features

The following limitations apply:

- RTI does not make the task priorities known to Simulink and Simulink Coder.
- RTI does not support the **Specified** setting of the **Periodic sample time constraint** option in the Solver dialog.

  If you select the above setting, RTI issues an error message and aborts processing.

Restricted support of multidimensional Simulink block parameters and signals

The parameters and input and output signals of many Simulink blocks can be multidimensional, i.e., they can have more than two dimensions. RTI supports Simulink models with multidimensional parameters and signals. These models can be built as a real-time application. However, RTI blocks and RTI-MP blocks support only one-dimensional (vector) or two-dimensional (matrix) signals.

The following rule applies for the variable description file entries:

- For $n_1 x n_2 x \ldots n_m$ dimensional parameters and signals, RTI generates a 1-D array of the length $n_1 * n_2 * \ldots * n_m$ in the TRC file.

Option for displaying subsystem port labels is not fully supported

The **Show port labels** option lets you define the following settings:

- **None**
- **FromPortIcon**
- **FromPortBlockName**
- **Signal Name**

In RTI, you must be aware that TRC Files are always generated with the **FromPortBlockName** setting regardless of the settings made in the **Show port labels** option. If you select **FromPortIcon** or **SignalName** in MATLAB, the port

	names in your subsystem may be displayed differently in the Simulink model and the TRC file.
Restricted support of Variant Sink and Variant Source blocks	RTI supports Simulink's Variant Sink and Variant Source blocks, but there must be no RTI blocks connected to these blocks, neither directly nor via other blocks and subsystems.
Restricted support of Variant Subsystem block	RTI supports Simulink's Variant Subsystem block, but there must be no RTI blocks in a Variant Subsystem or any of its subsystems, whether activated or deactivated.
Restricted support of Simulink Function Subsystem block	▪ You cannot use dSPACE blocks, such as RTI and RTI-MP blocks (except for the Default CPU block) or blocks from board-specific RTI blocksets, in a Simulink Function Subsystem block. ▪ The contents of a Simulink Function Subsystem is not generated into the TRC file.
Data dictionaries not supported	Simulink is able to store design data that your model uses in a data dictionary as a persistent repository. Data dictionaries are not supported by RTI and RTI-MP.
Restriction when using the Evenly spaced breakpoints option	If you use the Evenly spaced breakpoints option for look-up tables, it depends on the MATLAB version used whether the parameters of the related block are generated into the variable description. Even if these parameters are available in the variable description, you can access only a single value, but you cannot connect them to the Table Editor in ControlDesk.
Recursive functions not supported	Code generation of recursive functions is supported by Simulink Coder, but it should not be used for generating applications executed on real-time hardware.
Code obfuscation not supported	Simulink Coder provides the `ObfuscateCode` option. This option is not supported by RTI and RTI-MP.
Codebuild function not supported	As of MATLAB R2020b, the `codebuild` function is available to configure and start the compilation of code. This function is not supported by RTI and RTI-MP.
Parameter override not supported	The parameter override feature for referenced configuration sets is not supported.

Simulink Coder quick start wizard not supported

The quick start wizard automatically sets options that are not supported by RTI. It is therefore recommended to not use the wizard for creating RTI models.

Message signal type not supported

You must note the following limitations when using message signals:

- The message signals are not generated into the variable description, or do not point to a readable variable, such as the message output of Stateflow blocks.
- Message signals cannot be connected to dSPACE RTI blocks.
- The model separation for a multiprocessor model might lead to invalid settings if messages are to be sent between different models.

Restrictions when using referenced subsystems

You can add a subsystem by reference. Similar to the handling of referenced models, RTI and RTI-MP support subsystem references in the model. However, using RTI blocks in a referenced subsystem is not allowed.

Half data type not supported

Variables of the half data type are not generated into the variable description.

Limitations with Stateflow

Introduction

If you include Stateflow in your models, additional limitations apply.

States of Stateflow charts

By default, the states of Stateflow charts are not accessible via the variable description file. If you need to trace the state activity for a Stateflow chart, you can use the **Output State Activity** option for the states you need to observe. The outputs of the individual Stateflow charts are available via the Model Root group. The State Machine Data group collects all data that is available in the Stateflow data dictionary and parented by the state machine.

For details on the Output State Activity option, refer to *Stateflow* by MathWorks®.

Related topics

Basics

General Limitations 270
Limitations for Custom Code 282
Limitations in Building Models 285
Limitations in Task Handling 276
Limitations with RTI-MP 276
Miscellaneous Limitations 286

Limitations in Task Handling

Introduction

The following limitations apply to task handling.

Unsupported Simulink Coder feature

Simulink Coder provides the option `Allow tasks to execute concurrently on target`. RTI and RTI-MP do not support this feature. However, concurrent execution of tasks can be done by using RTI-MP for running tasks on multiprocessor or multicore systems.

Subsystems triggered by several interrupts

If a subsystem is triggered by several RTI interrupts (hardware, software, and/or timer interrupts), Simulink issues the following warning:

Warning: The function-call subsystem 'combined_interrupts/Subsystem' has multiple asynchronous triggers that do not specify priority. Data integrity will not be maintained if these triggers can preempt one another.

Note

Simulink always issues the above warning, because RTI does not make the task priorities known to Simulink. Refer to Restricted support of Simulink task features on page 273. With a modeling constellation like this, you must configure the priorities of the tasks involved to the same value.

Related topics

Basics

General Limitations 270
Limitations for Custom Code 282
Limitations in Building Models 285
Limitations with RTI-MP 276
Limitations with Stateflow 275
Miscellaneous Limitations 286

Limitations with RTI-MP

Introduction

If you implement RTI-MP models, the following additional limitations apply.

Mixed multiprocessor systems

RTI-MP does not support mixed multiprocessor systems that contain DS1006 and DS1007 boards.

Using MicroLabBox	RTI-MP supports MicroLabBox only as multicore system.
Propagating labels over CPU boundaries	It is not possible to propagate label strings over CPU boundaries (across IPC blocks).
Maximum number of configurable CPUs	The number of CPUs (number of processor cores) that you can configure for an MP system is restricted: ▪ In the RTI-MP configuration dialogs such as the dialog of the **Multiprocessor Setup** block, at most 25 entries for CPUs are provided. Because the dialogs cannot be scrolled, the visibility of the entries depends on the monitor resolution. For example, with a resolution of 1024*768, you will only see the first 16 CPU entries. ▪ In the IPI block dialog, you can configure at most 12 target CPUs.
Maximum number of connectable ports	There might be wrongly connected ports of IPC blocks in a submodel that was generated by model separation. After model separation, there might be unconnected or wrongly connected ports of **IPC Send** and **IPC Receive** blocks in the submodel. If the number of ports is ten or higher, there is no space in the block display to connect the signals. You have to scale the block to allow correct signal connections at the ports.
Subscheduling only for local software interrupts	RTI-MP supports subscheduling only for local software interrupts. An interrupt is called local if its source CPU is the same as its target CPU.
Combining interrupts	With RTI-MP you can combine only local interrupts to trigger the same subsystem, i.e., all interrupts must originate from the same CPU. None of the function-call signal lines may contain an IPI block.
Branching interrupts	With RTI-MP, the Simulink Function-Call Split block cannot be used in conjunction with IPI blocks.
Transferring matrix signals via IPC block	2-dimensional matrix signals cannot be transferred via an IPC Connector block. To transfer them, you can use the Simulink Reshape block to convert a matrix to a vector and vice versa.

Using blocks from the Simscape, SimDriveline, SimElectronics, SimHydraulics and SimMechanics blocksets

You must not place blocks from the Simscape, SimDriveline, SimElectronics, SimHydraulics and SimMechanics blocksets on the topmost level of your RTI-MP model. Otherwise the following problems can occur:

- Signal lines between those blocks are lost when the model is separated for RTI-MP.
- Model separation fails, and an error message is displayed.

You can avoid the above problems by placing blocks from these blocksets in a subsystem.

Initializing the workspace of MATLAB workers

If you use the Parallel Computing Toolbox from MathWorks®, which allows the parallel build of referenced models on a multicore PC, you can create MATLAB workers by executing the `parpool` command. The workspaces of the MATLAB workers have to be initialized if the referenced models are using variables from the MATLAB workspace.

With RTI-MP, the workspace of these MATLAB workers cannot be initialized by the top level model's **`PreLoadFcn`** function. However you can configure the workspace initialization with the **MATLAB worker initialization for builds** option. The **Load top model** setting is similar to the **`PreLoadFcn`** function. Instead of this, you can choose **Copy base workspace** to correctly initialize the workspace of the MATLAB workers.

For further information, refer to Model Referencing Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮).

Using Signals & Scope Manager objects

Objects defined with the Signals & Scope Manager are not supported. The build process will stop with an error message.

Code generation setting in MATLAB R2020b

As of MATLAB R2020b, code generation settings for Simulink signals and **Data Store Memory** blocks can be configured only via the Code Mappings Editor. Settings of the Code Mappings Editor are not copied to the models created during RTI-MP model separation. You must configure the code generation settings via the Code Mappings Editor also for the separated models.

Related topics

Basics

General Limitations 270
Interprocessor Interrupts 143
Limitations in Building Models 285
Limitations with Model Referencing 279
Miscellaneous Limitations 286

Limitations with the RTI Gigalink Blockset

Limitations with the Gigalink Blockset

When implementing Gigalink connections, note the following limitations:

- All processor boards must be equipped with Gigalink modules.
- Consistency of data transfer is ensured by the swinging buffer protocol in blocking or non-blocking mode. For details, refer to Swinging Buffer Protocol on page 118.

  Blocking communication should be combined with the synchronization of timer tasks. This use case is only supported by SCALEXIO to DS1006/DS1007 connections.
- Signals must be scalar or vectorial and of `double` data type. Otherwise an error message appears during model initialization.
- At most 1024 signals of `double` data type can be transmitted via a Gigalink channel.
- Each combination of Gigalink number and channel number can be used for a block only once. Otherwise an error message appears during the build process.
- If Gigalink_Send and Gigalink_Receive blocks and RTI-MP blocks for interprocessor communication use the same Gigalink number, they cannot use the same channel numbers. The RTI Gigalink Blockset and RTI-MP do not check the channel numbers and the related Gigalink number. You must ensure that the channel numbers used for RTI-MP interprocessor communication are not the same channel numbers used by the RTI Gigalink Blockset Send and Receive blocks.
- The RTI Gigalink Blockset does not support MicroLabBox.

Related topics

Basics

Interprocessor Communication Using Gigalink Blocks..... 138

HowTos

How to Implement Interprocessor Communication Using Gigalink Blocks..... 141

Limitations with Model Referencing

Introduction

There are important limitations on the use of model referencing, which are also relevant when you use protected models. The most important of these are summarized below. For a detailed list of model referencing limitations, refer to the Simulink and Simulink Coder documentation.

The limitations with model referencing are divided into the following subgroups:

- General limitations with model referencing
- Limitations applying to referenced models

General limitations with model referencing

The following limitations apply both to top-level and referenced models:

- Solver settings

  Only one solver is used for all models in a model referencing hierarchy. When a referenced model uses other solver settings than the top-level model, its solver settings are ignored and the top-level model's solver settings are used instead.
- Configuration parameters

  Some configuration parameters of a top-level model and its referenced models must be identical. The most important ones are the following:
  - **Application lifespan (days)** on the **Math and Data Types** page.
  - **Code Generation**
    - **System target file** on the **General** page
    - **Code Replacement library** on the **Interface** page

  In addition, model referencing requires that the **Include only Simulink.Parameter and Simulink.Signal objects with global storage class** option is identical between all models in a model referencing hierarchy.
- Variable description file options

  The variable description file (TRC file) options **Include states** and **Include Derivatives** on the **RTI variable description file options** page are not supported for model referencing. If you select one of these options, a message is displayed during the build process that they are ignored.
- Goto/From blocks

  Goto/From blocks cannot cross model reference boundaries.
- Asynchronous function-call input to a Model block

  It is possible to connect a function-call signal to an inport of a **Model** block using the **Asynchronous Task Specification** block. This feature can be used with RTI Interrupt blocks. However, the triggered function-call subsystem in the referenced model must not contain any blocks that depend on absolute time. For a list of blocks that depend on absolute time, refer to *Inheriting Sample Times* in the *Simulink User's Guide*.
- Wrong search path when using MATLAB workers

  If you use the Parallel Computing Toolbox from MathWorks®, which allows the parallel build of referenced models on a multicore PC, you can create MATLAB workers by executing the `parpool` command. These MATLAB workers may use wrong search paths if you switch RTI platform using the **`rti<xxxx>`** command. This results in unpredictable errors during the build process.

  To avoid this, exit the MATLAB session after an RTI platform switch and restart it.

Limitations applying to referenced models

The following limitations are specific for referenced models:

- RTI blocks

  RTI blocks are not supported in referenced models. This also includes RTI Interrupt Blocks and blocks from the RTI Blocksets. The TRC Exclusion block from RTI's Extras library is the only RTI block that can be used in a referenced model.

  The use of unsupported RTI blocks in referenced models is checked during a model reference build. If one or more unsupported blocks are found, an error message is displayed and the build process is aborted.

  With RTI-MP, it is not possible to use IPI, IPC and other RTI-MP blocks in a referenced model. A Model block can therefore be assigned to one CPU only.
- *Simulink.Bus* objects

  A referenced model must use *Simulink.Bus* objects to specify any buses that have inputs and outputs to the parent model. Each *Simulink.Bus* object must be defined in a workspace which is visible to both the referenced model and its parent model.
- User code files

  User code files (USR.C files) are only supported for top-level models. If a user code file exists for referenced models, for example, because you have converted former RTI models to referenced models, it is ignored.
- User makefiles

  Changes in the user makefile (USR.MK file) of a referenced model are not automatically detected during successive build processes of the top-level model. To ensure that a referenced model is rebuild after changes in its related user makefile, the user makefile needs to be listed in the **Model dependencies** option in the **Model Referencing** dialog. For details, refer to Model Referencing Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮).
- Model workspace parameters of referenced models

  Parameters defined in the model workspace of referenced models can be accessed via the **Model Parameters** group of the referenced model. Block-level references to these parameters can be accessed via the referenced model group of the respective model.

Limitations applying to protected models

In Simulink, protected models can contain other protected referenced models. Nested protected models are not supported.

Related topics

Basics

Limitations with RTI-MP 276

Limitations for Custom Code

Introduction

If you include custom code in your models, the following additional limitations apply.

Simulink Coder/Stateflow Custom Code

You have to specify custom C code using the Custom Code settings of Code Generation:

RTI does not support the Defines setting in the Additional build information frame. Specified values for Defines will be ignored. Instead of, use the Compiler options setting on the RTI general build options page.

For RTI-MP, you must use the Use Simulink Coder Custom Code settings checkbox on the CPU pages of the Multiprocessor Setup dialog to configure the CPUs for which the Simulink Coder custom code settings are used.

General limitations

The following limitations apply to custom C code (User-Code, S-Functions, etc.) and custom M code used in MATLAB Function blocks.

Unsupported functions Code generation for MATLAB functions that need access to PC hardware, for example, file I/O or network access, is not supported for RTI targets.

Limitations common to S-functions and User-Code

Access to internal variables RTI generates all Simulink variables into the `<[sub]model>.trc` variable description file. You can access them during simulation, for example, via ControlDesk. However, RTI cannot include the internal variables of S-Function blocks and User-Code in this file. If you want to access them via ControlDesk, you have to declare them as global and create a

user variable description file (`<[sub]model>_usr.trc`). Refer to User Variable Description File (USR.TRC File) (RTI and RTI-MP Implementation Reference 🕮).

I/O function calls There are no error checks regarding the correct use of the I/O function calls; for example, to check if the same I/O channel is accessed twice.

Limitations concerning S-functions

M-coded S-functions Do not use any M-coded S-function for real-time applications, as the C compilers always need C code. Keep in mind that even some of the original Simulink blocks use M-coded S-functions.

Reuse of S-functions If your model contains more than one instance of a specific S-function that uses global variables, unexpected side-effects may occur in the real-time application.

Generated S-function With Simulink Coder, it is possible to generate code from a Simulink model and then hook the generated C code to the **Generated S-Function** block. As a result you can reduce the build times because the code contained in the Generated S-Function block does not need to be regenerated or recompiled. Such an S-function cannot contain any RTI- or RTI-MP-specific block (for example, I/O).

After you have built a model using the Generated S-Function, Simulink Coder opens a new model and inserts a correctly configured Generated S-Function block.

For details, refer to the Simulink user documentation.

Limitations concerning User-Code

User-Code driven by an interrupt block You cannot use the `<model>_usr.c` User-Code file to add custom code to the tasks driven by interrupt blocks. Instead, use an S-function or Simulink Coder's Custom Code block in a subsystem that is triggered by the desired interrupt block.

Limitations Regarding Simulink Variables in the TRC File

Block output signals

In some cases, the Simulink Coder does not generate separate variables for block output signals. To make sure that the Simulink Coder generates a separate variable for an output signal, you can designate any signal in a model as a test point.

No TRC file entries for variables with keyword names

You cannot use keyword names for global variables, such as a Simulink.Parameter or a Simulink.Signal object with *ExportedGlobal* as the storage class. Do not use the following names:

_application	default	init	scaleback
_author	DEPRECATED	offs	struct
_description	desc	origin	type
_gendate	endgroup	OUTPUT	unit
_genname	endstruct	PARAM	value
_genversion	enum	range	xml
_model	enums	READONLY	
addr	flags	refelem	
alias	group	refgroup	
bitmask	HIDDEN	refvar	
block	increment	scale	

Note

The TRC keywords are case-sensitive. While a global parameter with the name type conflicts with the keyword, a parameter name Type can be used without limitations regarding the TRC file.

No TRC file entries for string signals

Simulink signals support the String data type. A string signal can be used for modeling, but it does not appear in the generated TRC file, and can therefore not be accessed, e.g., for experimenting in ControlDesk.

TRC file entries for INT64 and UINT64 variables

The 64-bit integer data types can be used in an RTI model. However, to enable the generation of variables of 64-bit integer data type into the variable description file, you must set the **Support long long** option in the **Hardware Implementation** dialog. Refer to Hardware Implementation Dialog (Model Configuration Parameters Dialogs) (RTI and RTI-MP Implementation Reference 🕮).

No TRC file entries for structured variables containing not supported data types

TRC file entries for structured variables, such as non-virtual buses or structured tunable parameters, are not generated into the variable description if they contain variables with unsupported data types, e.g., string variables.

Separate TRC file entries for parameters with dependencies

You can define a Simulink parameter with a dependency to another Simulink parameter. For example, you can define the Simulink parameter myParamPlusOne with the value `myParam + 1`. However, due to a Simulink Coder limitation, parameter dependencies are not propagated to the generated code.

Handling of Selector block and the Bus Selector block

The outports of the **Bus Selector** block, and other virtual blocks that are connected to it are not generated into the variable description file, if the inport of the **Bus Selector** block is connected to a non-virtual bus.

To access a signal of a bus, do the following:

- Access the signal directly from the bus.

  Or

- Connect a non-virtual block to the signal, for example, a **Signal Conversion** block with the **Output** property set to `Signal Copy`. Its outport is available in the variable description file.

If the inport of the **Bus Selector** block is connected via virtual bus, the outports are generated into the variable description file.

The same limitation applies to the **Selector** block. If its inport is connected to a contiguous array, the outports are not generated into the variable description file. To solve the problem, directly access the input signal or connect a non-virtual block to the signal, for example, a **Signal Conversion** block, see above.

Incorrect entries for Simulink.Bus objects

If you use Simulink.Bus objects with the Imported data scope, you must make sure that the data type definition for the Simulink.Bus object matches the one in your custom header file exactly. Otherwise, accessing variables on the real-time hardware can lead to incorrect results.

Limitations in Building Models

Introduction

The following limitations apply to the build process.

Path and file names

For path and file names the following limitations apply:

- Path and file names containing `"#"` characters are not supported.
- Path and file names containing `"$"` characters can lead to unexpected results.
- Path and file names containing `"&"` characters are not supported.

These characters are reserved characters of the make utility.

Quotation marks

The " character (quotation mark) leads to syntactically incorrect variable description files because it is interpreted as a string delimiter. Therefore, it must not be used in any of the following strings:

- Mask prompt names
- Simulink block names
- Signal labels
- Simulink block descriptions
- Simulink.Parameter or Simulink.Signal Description and DocUnits properties

Dot notation	The '.' character (dot) might lead to an error in the software that uses the TRC file, since the dot is interpreted as a structure delimiter. Therefore it must not be used in any of the following strings: ▪ Simulink block names ▪ Signal labels
Rapid Accelerator	The rapid accelerator is not supported by RTI.

Miscellaneous Limitations

Introduction	There are the following miscellaneous limitations you have to consider.
Model Advisor messages	With MATLAB's Model Advisor, you can analyze your model to configure Simulink and Simulink Coder appropriately. Since the Model Advisor is a tool from MathWorks, it cannot take RTI specifics into account. The Model Advisor may therefore issue tips which are unsuitable or inapplicable for RTI. Double-check against the RTI documentation if in doubt.
Using the Upgrade Advisor API	The `upgradeadvisor` function lets you analyze and perform required migration steps for your model when you upgrade to a newer MATLAB version. The third-party blocksets that you use, such as RTI blocksets, should not be upgraded by this function. Do not change the default value of the `SkipBlocksets` argument from `true` to `false`.
Using the Property Inspector	The Property Inspector lets you edit parameters and properties of a model element. The Property Inspector supports only blocks whose parameters are configured by using a standard Simulink mask interface. It does not support most of the RTI blocks that provide their own dialogs.
Dynamic memory allocation	Dynamic memory allocation in MATLAB Function blocks is not supported. Thus, the following configuration settings are unavailable: ▪ Dynamic memory allocation in MATLAB Function blocks ▪ Dynamic memory allocation threshold in MATLAB Function blocks
Error messages concerning S-function upgrades	If you select *error* or *warning* from the S-function upgrades needed list on the Compatibility page of the Diagnostics dialog, an error message or warning may be displayed for RTI blocks. You can ignore these messages.

Tip

It is recommended to select *none* or *warning* from the **S-function upgrades needed** list.

Simulink Coder C++ Language Option is not supported

Simulink Coder has an option for switching the target language between C and C++. For RTI, the target language must be set to C.

If you want to use C++, refer to Using C++ Code in an RTI Application on page 180.

The File generation control option in the Simulink Preferences dialog is not supported

Simulink Coder allows you to specify custom folders for generating code and temporary fragments for simulation. RTI and RTI-MP does not support modifications in the File generation control setting. If you used one of the available options, e.g., `CodeGenFolder`, `CacheFolder`, or `CodeGenFolderStructure`, the RTI build process stops with an error message.

If you have specified the `Code generation folder` via command line, for example, by using `Simulink.fileGenControl('set', 'CacheFolder', cacheFolderPath, 'CodeGenFolder', codeGenFolderPath)`, the RTI build process is not able to recognize a conflict, and no RTI error message will occur. However, specifying these custom folders via command line is also not supported and might lead to erroneous behavior.

The MATLAB Distributed Computing Server (MDCS) software is not supported

When you work with referenced models, the MATLAB Distributed Computing Server (MDCS) software can be used for parallel builds distributed across remote workers. RTI does not support the MDCS software.

Read topology option disabled

To enable the **Read topology** option, you have to use the **`Check Gigalink Topology`** (ControlDesk Platform Management 🕮) command in the ControlDesk's Platform Manager. Note, that for some platforms, e.g., DS1007, the Gigalink topology check is not possible in ControlDesk.

Optional blocksets are not disabled

Not all RTI blocksets can disable their access when you selected an unsupported RTI platform.

For example, the **Time-Triggered Blockset** can be opened with RTI1104 selected as the current platform although RTI1104 is not supported by this blockset.

Troubleshooting

Experiencing problems

When implementing an RTI or RTI-MP model, when building and downloading the real-time application or when running the simulation, you might experience problems. If any RTI-related problem occurs, follow the instructions provided by the error message.

The error messages are listed in alphabetical order. Problems without a message are placed first. Terms in angle brackets <...> refer to specific parts of your model, for example, a specific block, subsystem or CPU.

Getting further support

Support Knowledge Base If the information in this section does not help you to solve the problem, check the Support Knowledge Base on our website. See http://www.dspace.com/go/kb.

dSPACE Support If self-help does not help you to solve the problem, contact dSPACE Support and provide information about your dSPACE environment and the problems you have. It is recommended to use the support request form provided on the website at http://www.dspace.com/go/supportrequest. However, you can also send an e-mail or phone us.

Where to go from here

Information in this section

Implementing the Model....290
Automatic Program Building....293
Checking the Model....295
Generating the Code....296
Compiling and Linking the Code....297
Using Model Referencing....299
Using Subsystem Referencing....302

Initializing the Application.....302
Running the Application.....304
Running ControlDesk.....308

Information in other sections

How to Start the Build and Download.....228
Debugging an Application.....234

Implementing the Model

Introduction

These problems and error messages might occur when you implement your Simulink model.

RTI-MP error message

Source block of block <name> is not valid.

An IPI block has to be directly connected to a hardware or software interrupt block. There must be no block in between.

- Ensure that there is no other block inserted in the interrupt signal line, for example, a Function-Call Split block.

RTI-MP error message

<subsys> is not a triggered subsystem

The outport of an IPI block has to be directly connected to the trigger port of a function-call subsystem.

- Ensure that the outport of the IPI block is connected to a trigger port whose trigger type is configured as **function-call**.
- Ensure that there is no Function-Call Split block between the IPI block and the connected function-call subsystem.

RTI Block displays

Bad Link

This appears if a block comes from an old RTI or an RTI that is different from the one you are currently using.

The RTI library uses Simulink library links. If you include blocks from the RTI library in your application, they function as references to the original blocks of the library. Therefore, any library updates automatically update your application

as well. If the changes to a new block are too significant in comparison to the block you are using in your application, this message is shown.

- Replace the old block manually with the new one from the library.

RTI-MP error message

Error in MP network topology!

CPU <cpu> is not a member of the registered MP system.

The following CPUs are registered with ControlDesk or have been specified in the MP Topology setup

<…>

The Simulink multiprocessor model contains one or more CPUs that have not been specified in the Multiprocessor Topology Setup dialog.

- Rename the CPU <cpu> in the Multiprocessor Setup dialog or adapt the settings in the Multiprocessor Topology Setup dialog.

RTI-MP error message

Hardware configuration

Allocation of <swinging buffer | virtual shared memory> connection from <cpu1> to <cpu2> failed.

The number of available swinging buffer channels/virtual shared memory channels is not sufficient for the model configuration (each Gigalink connection provides eight swinging buffer channels and seven virtual shared memory channels per direction).

- Change your Simulink model so that it does not use more than eight swinging buffer channels and seven virtual shared memory channels between <cpu1> and <cpu2> (per Gigalink connection and direction). As an alternative, you can add further Gigalink connections to your hardware to match the requirements of your Simulink model.

RTI-MP error message

Hardware configuration

Missing gigalink connection from <cpu1> to <cpu2>.

An IPC channel has been specified between <cpu1> and <cpu2>, but the CPUs are not directly connected via Gigalinks.

- Install all the required Gigalink connections.

RTI-MP error message

Incompatible Step Size Multiples

The destination Step Size Multiple <n> of connection <cpu1 to cpu2 on ch2> must be a multiple of the Step Size Multiple <m> of destination CPU <cpu2>!

All communication channels require that their source and destination step sizes are compatible with the step size of the corresponding source and destination CPUs.

- For each communication channel of your model ensure that its source and destination step sizes are compatible with the step sizes of the corresponding CPUs. All of these step sizes have to be entered as integer multiples of the multiprocessor system's basic step size. You have to check that:
  - The source step size of the channel is an integer multiple of the source CPU's step size,
  - The destination step size of the channel is an integer multiple of the destination CPU's step size.

RTI-MP error message

IPC Connector block <name> must not be located in block diagram-depth>2

IPC Connector block <name> must not be located in subsystem <subsys>

All IPC blocks must be placed either on the root level of the model or on the top level of a subsystem that is driven by an interrupt block or assigned to the background task.

- Place the IPC blocks of your model only on the root level of the model or on the top level of triggered subsystems.

RTI-MP error message

No Interrupt block found for IPI Block <name>

No triggered subsystem found for IPI Block <name>

An IPI block is always required between a hardware or software interrupt block and the corresponding function-call subsystem. The trigger type of this subsystem must be set to function-call.

- Ensure that an IPI block is included in each connection between the hardware or software interrupt block and its corresponding function-call subsystem.

RTI-MP error message

Source and Destination CPU of line between blocks <blockX> and <blockY> are not equal

All blocks of the multiprocessor model inherit their CPU identity from the IPC blocks they are connected to. If your model contains a connection between blocks that are assigned to different CPUs, their CPU identity is ambiguous.

- Ensure that the CPU identity of all of the blocks in your model is unambiguous.

RTI-MP error message

Incompatible Step Size Multiples

The source Step Size Multiple <n> of connection <cpu1 to cpu2 on ch2> must be a multiple of the Step Size Multiple <m> of source CPU <cpu1>!

All communication channels require that their source and destination step sizes are compatible with the step sizes of the corresponding source and destination CPUs.

- For each communication channel of your model ensure that its source and destination step sizes are compatible with the step sizes of the corresponding CPUs. All of these step sizes have to be entered as integer multiples of the multiprocessor system's basic step size. You have to check that:
  - The source step size of the channel is an integer multiple of the source CPU's step size,
  - The destination step size of the channel is an integer multiple of the destination CPU's step size.

Accuracy of the simulation step size

The simulation step size during the real-time simulation and the fixed step size selected in the model might differ, depending on the resolution of the timer device that provides the simulation step size. This difference is a discretization error resulting from the approximation of the fixed step size by an integral multiple of the timer resolution. The discretization error is always smaller than the timer resolution and can be minimized by selecting a fixed step size that is an integral multiple of the timer resolution.

Consider the following example:

- Timer resolution = 0.015 µs
- Fixed step size = 25 µs

In this case the simulation step size is given by:

floor (25 µs / 0.015 µs) * 0.015 µs = 24.99 µs.

Related topics

Basics

Automatic Program Building..... 293

Automatic Program Building

Instructions

During automatic program building, RTI uses a number of tools that might issue error messages. Most of these error messages are self-explanatory. Therefore you will find instructions only for:

- Error situations that might be misleading or do not always point to the reason for the problem,
- Irregular program behavior.

The instructions are structured according to the steps RTI carries out when you initiate the process of building and downloading real-time code.

Screen output

The following screen output of MATLAB is taken from a successful, error-free build of a model (`model.slx`). Thus, when you build a model and encounter a problem, you can find the step where the problem occurred by comparing your screen output with the corresponding section of this example. RTI-MP's build process for a single submodel is very similar (the lines enclosed in brackets `[]` refer to building a model for a multiprocessor system). RTI-MP does not download the MP application immediately.

Model check

```
-----------------------------------------------------------------
Starting build procedure with RTI x.y (RTI<xxxx>, <DATE>)
Model: "<model>" (<WorkingDirectory>\<model>.slx)
-----------------------------------------------------------------
*** Using configuration set: "Configuration"
*** Working directory: "<WorkingDirectory>"
[*** Copying Blocks to new model "<submodel>" ...]
```

Refer to Checking the Model on page 295.

File generation

```
[*** Generating communication functions for application "<submodel>" ...]
[*** Generating real-time simulation engine for application "<submodel>" ...]
*** Optional User System Description File <model>_usr.sdf not available
*** Initializing code generation
### Starting Simulink Coder build procedure for model: <model>
### Generating code into build directory: .\<model>_rti<xxxx>
### Invoking Target Language Compiler on <model>.rtw
*** Generating Variable Description File <model>.trc
*** Optional User Variable Description File <model>_usr.trc not available
*** Generating template: User-Code File <model>_usr.c
*** Generating template: User Makefile <model>_usr.mk
### Creating project marker file: rtw_proj.tmw
### Creating <model>.mk from d:\dspace\matlab\rti<xxxx>\m\rti<xxxx>.tmf
```

Refer to Generating the Code on page 296.

Compiler, linker, loader

```
### Building <model>: dsmake -f <model>.mk
BUILDING APPLICATION (Single Timer Task Mode)
WORK  DIRECTORY "<WorkDirectory>"
BUILD DIRECTORY "<WorkDirectory>\<model>_rti<xxxx>"
TARGET COMPILER "<CompilerPath>"
COMPILING  <model>.c
[...]
USING LIBRARY "<RCPHILPath>\MATLAB\RTI<xxxx>\C\Lib\rtwlib_rxx_ds<xxxx>.lib"
LINKING APPLICATION ...
LINKING FINISHED
[...]
MAKE PROCESS SUCCEEDED
[...]
LOADING APPLICATION "<model>.sdf" ...
[...]
LOADING FINISHED
```

Refer to Compiling and Linking the Code on page 297.

Executing the real-time application

```
### Successful completion of Simulink Coder build procedure for model: <model>
*** Finished RTI build procedure for model <model>
```

Refer to:

- Initializing the Application on page 302
- Running the Application on page 304
- Running ControlDesk on page 308

Unspecific errors

If the build procedure stops with unspecific errors, you should firstly execute the cleanup procedure by using **`rti_build2`** or **`rtimp_build2`** with the `CleanUpAll` value and then restart the build procedure.

Related topics

References

Implementing the Model..... 290

Checking the Model

Introduction

The following errors might occur when RTI checks your Simulink model. They appear either in the MATLAB Command Window or in a MATLAB or Simulink message box.

Error message

Could not check licenses

The *Real-Time Interface* and *Real-Time Interface Multiprocessor* licenses cannot be checked due to the problem stated in the error message.

- Check your licenses in the dSPACE Installation Manager, refer to How to Show Available Licenses and Their Attributes (Working with CodeMeter Licensing Technology 🕮).
- Check the license attributes, such as the expiration time of your licenses.
- If you work with a license server, check the connection to it.

Error message

The block 'MainMdl/Timer Interrupt/S-Function1' must specify asynchronous task priority of 10 to match the priority specified at input port 1 of function-call model block 'MainMdl/Model'.

You have specified the task priority of an Asynchronous Task Specification block with an absolute value (with 10 in this example). This is not supported by RTI and RTI-MP.

- Set the Task priority parameter of the Asynchronous Task Specification block to `[]` (empty value).

Error message

The time source choice for S-function 'MainMdl/Timer Interrupt/S-Function1' is not valid. The function-call destination 'MainMdl/Model' input port 1 needs absolute time and requires that this option be set to `SS_TIMESOURCE_BASERATE`.

The asynchronously triggered function-call subsystem which is triggered in the referenced model contains a block that depends on absolute time. This is not supported by RTI and RTI-MP.

- Remove the related block from the subsystem.
- Refer to *Inheriting Sample Times* in the *Simulink User's Guide*. There you find a list of blocks that depend on absolute time.

RTI-MP warning

Warning: Unable to honor user-specified block priorities. Block '<block1>' (pri=10000) has to execute before block '<block2>' (pri=-10000) to satisfy data dependencies.

Either the IPC blocks <block1> and <block2> form a direct feedthrough or you specified illogical block priorities. As a result, the block priorities are ignored, and the IPC Input block is calculated before the IPC Output block.

- Try to avoid direct feedthrough. You should also avoid specifying block priorities manually. See Model Evaluation on page 120 for details.

Generating the Code

Introduction

The following errors might occur while RTI generates real-time code from your Simulink model. They appear either in the MATLAB Command Window or in a MATLAB or Simulink message box.

Error message

Error using ==> rtwgen

--> Invalid "function-call" connection

(The message might be longer.)

The Trigger block in a subsystem triggered by a hardware or software interrupt is not set to the function-call type.

The Trigger block in a subsystem triggered by a Simulink signal is set to the function-call type.

1. Set the **Trigger** blocks of subsystems triggered by hardware or software interrupts to the **function-call** type.
2. Set the **Trigger** blocks of subsystems triggered by Simulink signals to one of the following types: **rising**, **falling**, or **either**.

Error message

RTI Build Error

Error in MAT to C conversion.

MATLAB:I18n:InconsistentLocale - The system locale setting, English (United States)_United States.1252, is different from the user locale setting, German (Germany)_Germany.1252.

(or similar)

The Window's user locale and the Window's system locale do not have the same value.

- To get information on how to view and change user locale settings, refer to http://www.mathworks.com/matlabcentral/answers/96166.

Compiling and Linking the Code

Introduction

The following errors might occur while RTI compiles and links the real-time code. They appear in either the MATLAB or the DOS window.

Compiler message (Microtec PowerPC compiler (version 3.x)

Caution: routine <model> contains <n> Basic Blocks and <n> Expressions. Normally, aggressive register-allocation would be disabled for this routine, to improve compile-time. But it is not possible to disable Inter-Procedural Register Allocation (IRA) after part of the compilation is complete. It is strongly suggested to re-compile with IRA disabled by, for example, not using the -Ox option.

During the compilation of complex models, the Microtec PowerPC compiler (version 3.x) issues this warning and recommends to disable aggressive register-allocation, to reduce the optimization level and to start the compilation again.

- Disable the aggressive register-allocation by omitting the optimization option -Ox.

  You can do so as follows:

  1. Open the Code Generation dialog of your model's configuration set in the Model Explorer.
  2. Select the **RTI general build options** page and change the settings according to your needs.

  For details, refer to Code Generation Dialog (Model Configuration Parameters Dialogs).
- It may be useful to reduce the optimization level to -Ot -Oi -Or or to use the standard optimization -O.

  You can reduce the optimization level also on the **RTI general build options** page.
- The compilation time can be also improved by creating atomic subsystems which generate less complex functions.

For details on compiler options, refer to the following manuals:

Board	Processor Type	Compiler Manual
DS1104 MicroAutoBox II	PowerPC	*C/C++ Compiler User's Guide by Microtec*
DS1007 MicroLabBox	PowerPC	http://gcc.gnu.org/onlinedocs/gcc-4.8.3/gcc
DS1006	x86	http://gcc.gnu.org/onlinedocs/

Error message

<cmd-line> deprecated option "-O0"

Other options may be, for example:

-O1, -O2, -O3, -O4, -O5, -OXMnnn.

These optimization options are only valid for the Microtec PowerPC compiler (version 2.0). The Microtec PowerPC compiler (version 3.x) uses different optimization options.

1. Open the Code Generation dialog of your model's configuration set in the Model Explorer.
2. Select the **RTI general build options** page and change the settings according to your needs.

For details on compiler options, refer to the following manuals:

Board	Processor Type	Compiler Manual
DS1104 MicroAutoBox II	PowerPC	*C/C++ Compiler User's Guide by Microtec*
DS1007 MicroLabBox	PowerPC	http://gcc.gnu.org/onlinedocs/gcc-4.8.3/gcc
DS1006	x86	http://gcc.gnu.org/onlinedocs/

Error message

"model_usr.c", line 39: [F108] 'u' undefined

(or similar)

The User-Code of your model contains pointer variables that access root-level Inport or Outport blocks. Due to internal Simulink Coder changes imposed by data typing, this feature is no longer supported by RTI.

- Use **Data Store** blocks instead of pointer variables as described in How to Include User-Code in Simulink Models on page 174.

Error message

OPUS MAKE: Don't know how to make <sfunname>.<…>. Stop

(This appears while the make utility is running.)

Your model contains user-written S-functions, but the make utility cannot find the corresponding C-coded source file.

Your model contains Simulink blocks consisting of M-coded S-functions.

1. Place the C-coded source files of your S-functions in the same working folder as the model, or specify the S-function folder in the user makefile. To specify the S-function folder in the user makefile, add the name of the folder to the entry

   ```
   # Directories for S-Functions
   SFCN_DIR =
   ```

   You can find the user makefile template `<model>_usr.mk` in the working folder of the model.
2. Do not use M-coded S-functions in Simulink models that you want to use for building real-time programs.

Using Model Referencing

Introduction

The following error messages might occur when using model referencing.

For further error messages concerning model referencing, refer to Checking the Model on page 295.

Error message

RTI Build Error

Model <RefModelName> is used as a referenced model. It contains RTI blocks that are not supported in this case:

<RefModelName>/<RTIBlockName>

With the model referencing feature, all RTI blocks need to be placed in the top-level model and cannot be used in the referenced model. The TRC exclusion block from RTI's Extras Library is the only RTI block that can be used in a referenced model.

(or similar)

Your referenced model contains at least one RTI block that is not supported for model referencing. The RTI build process is therefore aborted.

- Remove the unsupported RTI block which is listed in the message from your referenced model.

Error message

??? Error using ==> rtwgen

Cannot reference model <RefModelName> multiple times. The S-Function block <RefModelName>/<RTIBlockName> does not support code reuse. The block (a) should have a TLC implementation (b) should set the option

''SS_OPTION_USE_TLC_WITH_ACCELERATOR' and (c) should set the option 'SS_OPTION_WORKS_WITH_CODE_REUSE'.

Consider updating the "Total number of instances allowed per top model" parameter, in the 'Model Referencing' page of the Configuration Parameters dialog, to "One".

(or similar)

Your referenced model allows multiple references to itself and contains unsupported RTI blocks. During the generation of the SIM target, an error message is displayed indicating S-functions which do not support code reuse. This feature does not agree with the "multiple" setting for the **Total number of instances allowed per top model** option in the **Model Referencing** dialog of the **Model Configuration Parameters** dialog. To avoid these errors, try one of the following solutions:

- Set the **Total number of instances allowed per top model** option in the **Model Referencing** dialog of the **Model Configuration Parameters** dialog to "one".
- Adapt your S-functions to support code reuse.
- Remove unsupported RTI blocks from your referenced model.

Error message

??? Error using ==> rtwgen

Cannot generate model reference target. Block <RefModelName>/<RTIInterrupt....> in model <...> has one of the following options: SS_OPTION_ASYNCHRONOUS, SS_OPTION_ASYNCHRONOUS_CUSTOM, SS_OPTION_ASYNCHRONOUS_INTERRUPT or SS_OPTION_FORCE_NONINLINED_FCNCALL. These options are not supported for model reference target.

(or similar)

Your referenced model contains an RTI Interrupt block which is not supported for model referencing. Thus an error message is issued during the generation of the SIM target.

- Remove the RTI Interrupt block from your referenced model.

Error message

Error occurred when setting a bus element name to signal name <...>. Output port 1 of block <...> is the signal source. The actual error message is: Invalid Name <...> specified; Name must be a valid identifier that starts with an alphabetic or '_' character, followed by alpha numeric or '_' characters. In addition, Name may not start with 'sl_padding'.

(or similar)

You are trying to convert a nonvirtual subsystem to a Model block and your model contains bus elements with invalid signal names. To be converted, signal names must follow the rules given in the error message.

- Rename your signals.

Error message

??? --> Goto/From connections cannot cross nonvirtual subsystem boundaries. The only exception is when a goto is connected to a state output port.

- *Invalid connection starts with < ... >.*
- *Invalid connection ends with < ... >.*

(or similar)

You are trying to convert a nonvirtual subsystem to a Model block and your model contains Goto/From connections that pass nonvirtual subsystem boundaries. This is not supported by Simulink.

- Adapt your model so that it does not contain Goto/From connections which pass nonvirtual subsystem boundaries.

Error message

??? Error using ==> Simulink.

SubSystem.convertToModelReference>handle_diagnostic_l at 249

The first input must be an atomic or function-call subsystem block. Cannot convert a conditionally executed subsystem to a model.

(or similar)

You are trying to convert a subsystem to a Model block that is neither atomic nor a function-call subsystem. This is not supported by Simulink.

- Adapt the subsystem you want to convert to a Model block.

Error message

??? Error using ==> RTW.makertw.setup_config_set_for_model_reference at 162

Error encountered when creating the model reference RTW target for <...> : Error using ==> RTW.makertw.setup_config_set_for_model_reference at 143

The following Configuration parameter option(s) do not match between <...> and <...> models:

- *Code Generation/RTI simulation options*

  AssertionMode (OFF != WARN)

  AssertionMode is mapped to 'Assertion mode:'

(or similar)

Some configuration parameter options are not identical between your top-level model and your referenced model(s). This is not supported by Simulink Coder.

- Adapt the configuration parameter options which do not match.

Refer to Simulating and Building Models Containing Model Blocks on page 160.

Using Subsystem Referencing

Introduction

The following error messages might occur when using subsystem referencing.

Error message

RTI Build Error

Model <ModelName> uses referenced subystems. Referenced Subsystem contains RTI blocks that are not supported in this case:

<ModelName>/<RefSubsystemName>/<RTIBlockName>

With the subsystem referencing feature, all RTI blocks need to be placed in the model root or subsystem and cannot be used in the referenced subsystem. The TRC exclusion block from RTI's Extras Library is the only RTI block that can be used in a referenced subsystem.

(or similar)

Your referenced subsystem contains at least one RTI block that is not supported for subsystem referencing. The RTI build process is therefore aborted.

- Remove the unsupported RTI block which is listed in the message from your referenced subsystem.

Initializing the Application

Introduction

The following errors can appear in the log file or Log Viewer.

If an error occurs at this stage, RTI exits the application. Hence, both the real-time application and the background task terminate. The error might result from hardware or from the initialization of the User-Code or an S-function. You cannot restart your application, but have to download it again after you have solved the problem.

Error message

Application terminated

(The application stopped executing right after downloading; meaning that the real-time application might not have finished its initialization correctly.)

During the initialization of an S-function or User-Code an `exit()` command was encountered.

Due to programming errors, the real-time program crashed during the initialization phase.

- Check the S-functions and User-Code of the model for `exit()` commands.
- Try to run your application without S-functions and User-Code to detect whether possible programming errors might be causing the problem. If this is the case, fix these programming errors.

Error message

Memory allocation failed

(Or a similar message.)

The real-time processor has run out of dynamic memory. This situation might occur when you use S-functions that allocate large amounts of dynamic memory.

- Check if your Simulink model uses such S-functions. If this is the case, try to reduce the amount of the allocated dynamic memory.

Error message

ds<XXXX>_init(): Invalid PHS-Bus address 0x00000000 (201)

This error message with address 0x00 typically appears when the application contains blocks of the DS<XXXX> board, but the processor board is not connected to a DS<XXXX> board via the PHS bus.

Method 1

To initialize the application

1. Check the PHS bus connection of your processor board and of the connected I/O boards.

Tip

The Platform Manager of ControlDesk displays the I/O boards found for the processor board.

Method 2

To initialize the application

1. Check whether the correct board numbers were used for the I/O blocks in the RTI model, for example, an RTI block specifies board number 2 (DS2004_B2), and only one DS2004 board is connected to the PHS-bus. Refer to PHS-Bus Address and Board Number on page 35.
2. If you installed several processor boards, check that you are downloading to the right one. Refer to How to Activate a Specific Platform Support on page 22.
3. Check that you are using the correct platform connection mode (bus or net). Refer to How to Register a Platform (ControlDesk Platform Management 🕮).

Running the Application

Introduction

The following errors and messages can occur while the dSPACE system runs your real-time application. Errors and messages appear in the log file or Log Viewer.

If your application stops executing, check whether there are any messages. If an error occurs in this stage, RTI switches the simulation to the STOP mode. Hence, the real-time application terminates, but the background task is still active. The error might result from your application or from the `usr_output()`, `usr_sample_input()` or `usr_input()` sections of your User-Code or S-function.

No message

[...]

(The application has stopped.)

The real-time program hung due to a statement that corrupted the real-time program in an S-function or User-Code of your model.

- If your model uses S-functions or User-Code, try to run it without these parts to detect whether programming errors might be causing the problem.

Exception message

Alignment Error (PowerPC processor)

(The application has stopped.)

The processor can read or write only aligned data. 64-bit data types can be accessed only at addresses on an 8-byte boundary (ending with 0H or 8H), 32 bit data types at addresses on a 4-byte boundary (ending with 0H, 4H, 8H, CH). If an attempt is made to access data at other addresses, this exception is raised.

- Debug your application and find the block or function that is responsible for this exception.

  Refer to How to Trace an Exception on page 237.

Exception message

Data Access Error (PowerPC processor)

(The application has stopped.)

The processor cannot read data from or write data to the specified address. An attempt was made to access a memory location where there is physically no memory present.

- Debug your application and find the block or function that is responsible for this exception.

  Refer to How to Trace an Exception on page 237.

Exception message

Instruction Access Error (PowerPC processor)

(The application has stopped.)

The processor cannot fetch the instruction at the specified address. An attempt was made to access a memory location where memory is either not present or protected.

- Debug your application and find the block or function that is responsible for this exception.

  Refer to How to Trace an Exception on page 237.

Exception message

Page fault exception (x86 processor)

(The application has stopped.)

The processor cannot access the specified address. An attempt was made to access a memory location where memory is either not present or protected.

- Debug your application and find the block or function that is responsible for this exception.

Refer to How to Trace an Exception on page 237.

Message

Model has requested to stop simulation

(The application has stopped.)

The Stop block has stopped the simulation due to an input signal other than zero.

- Check the signal source for the **Stop** block.

Exception message

Program Error (PowerPC processor)

(The application has stopped.)

This exception is raised if one or more of the exception conditions defined for the PowerPC architecture occur.

The exception conditions are:

- Illegal instruction

  This exception occurs when the execution of an instruction with an illegal opcode is attempted.
- Privileged instruction

  This exception occurs when the execution of a privileged instruction is attempted and the MSR register user privilege bit is set.
- Trap

  This exception occurs when any of the conditions specified in a trap instruction is met.
- Debug your application and find the block or function that is responsible for this exception.

  Refer to How to Trace an Exception on page 237.

Message

Simulation Stop time reached

(The application has stopped.)

The stop time for the simulation has been reached.

- Specify a larger value or `inf` for the stop time.

Error message

Task Overrun: Occurred in task "<task name>", execution step <number>

(The application has stopped.)

An overrun situation has occurred, because the scheduler attempted to start a task that was still executing. This might result from a sample time that was too small.

- Choose either bigger sample times for your timer tasks or a bigger base sample time.
- Choose a different execution mode (single timer task mode or multiple timer task mode) for your model.
- Try a different distribution of priorities over your tasks.
- Try a different overrun behavior.
- Try to configure to increase the first simulation steps by using the Configure Initial Step Size Dialog for an RTI model or the Configure Initial Step Size Dialog (Multiprocessor Setup Dialog) for an RTI-MP model.

  For details, see:

  - Configure Initial Step Size Dialog (RTI and RTI-MP Implementation Reference 🕮)
  - Configure Initial Step Size Dialog (Multiprocessor Setup Dialog) (RTI and RTI-MP Implementation Reference 🕮)
- If you still encounter task overruns, you can use a more powerful dSPACE board, or migrate the model to a multiprocessor system. If you already use a multiprocessor system, you can extend it by adding additional processor boards.

Error message

SimStruct error: <Simulink S-function message>

Example SimStruct error: Error encountered in S-function MEX file.

The simulation engine has encountered an error message that was set by an S-function.

Error messages

RTI: I/O board signaled an error on PHS bus I/O error line

RTI: Error on PHS bus I/O error line has been cleared

The PHS bus I/O error line was activated by a specific board. The boards react differently to the error line activation. For detailed information, refer to the description of the electrical characteristics in the board-specific chapters of the PHS Bus System Hardware Reference 🕮.

Error message

ds1006 - RTLIB: <FUNCTION>: I/O board at PHS offset < ADDRESS> is not initialized! (-715)

If you use a multicore DS1006 Processor Board, a check is made on whether all I/O boards have been initialized. Normally, board initialization is included in the RTI I/O blocks, but there are some exceptions. Some RTI Interrupt blocks do not automatically start board initialization. If there is only an RTI Interrupt block in your model, and no other RTI I/O block, the I/O board may be not initialized and the above error message occurs.

There are two ways to add the required initialization code to your model:

- If you want to manage the modification within the model, you can add a **System Initialize** block from Simulink Coder's **custcode** library to add hand-written code to your real-time application.

  Double-click the block to open its dialog. Enter the `#include` statement for the I/O board used in the **System Initialize Function Declaration Code** edit field and the corresponding initialization function in the **System Initialize Function Execution Code** edit field.

  For example, if a DS4001 I/O board (registered as ds4001_1) was not initialized, enter the following code:

  ```
  #include <ds4001.h>
  ```

  and

  ```
  ds4001_init(DS4001_1_BASE);
  ```

- If you do not want to change the model, you can use an external file for user code. When you build your model for the first time, you will find the user-code template file `<[sub]model>_usr.c` in your current working directory. Edit this file to add the required initialization code to your model.

  For example, if a DS4001 I/O board (registered as ds4001_1) was not initialized, enter the following code:

  ```
  ...
  /* Insert #include directives for header files here. */
  #if defined(_DS1006)
  # include <ds4001.h>
  #endif
  ...
  static void usr_initialize(void)
  {
  #if defined(_DS1006)
    /* dSPACE I/O Board DS4001 #1 */
    ds4001_init(DS4001_1_BASE);
  #endif
  }
  ...
  ```

  For further information on user code, refer to Implementing User-Code on page 171.

Running ControlDesk

Introduction

The following messages might occur while you are working with your real-time application. These messages are issued by ControlDesk and can be found in the log file.

No message

[...]

(The update of ControlDesk's instruments is very slow.)

The tasks of the real-time program are using almost 100% of the processor's capacity.

The Simulink model contains a Background block connected to a subsystem that is computationally intensive.

- Free more processor time for the background task by increasing the step size of the model.
- Use the Background block only for subsystems that are not computationally intensive.

No message

[...]

(There are no block parameters available below the model root node in ControlDesk's **Variables** controlbar.)

You set the **Default parameter behavior** option to **Inlined** in the **Code Generation - Optimization** dialog of the model's configuration set and no other global parameters were generated into the TRC file.

- Set the **Default parameter behavior** option to **Tunable** and rebuild the model.

No message

[...]

(The block outputs of some blocks observed with ControlDesk do not change.)

You selected the **Conditional input branch execution** checkbox in the **Simulation Target** dialog of the model's configuration set in the Model Explorer. The observed blocks drive a Switch block (or another block with conditional inputs) and their outputs are not necessary for the current model computation.

Note that you can observe the same behavior with Simulink simulation.

- Clear this checkbox and rebuild the model.

Error message

Cannot find MAP file

(This appears when you attempt to load the `<model>.sdf` file in ControlDesk.)

There is no `<model>.map` file available because the linker could not build the `<model>.obj` file.

- Build the model again to get a `<model>.map` file.

Message

Trace capture in progress

(This appears after you started the data capture, but the data acquisition does not continue.)

The real-time application has terminated (STOP or PAUSE mode).

The host service code is located in a task that is currently idle.

- Use ControlDesk to switch the simulation into RUN mode via the `simState` variable.
- Either enable the execution of the task containing the **Data Capture** block, or place the block in another task.

A

accessing
  RTI blocks 23
acquiring
  data 210
activating
  platform support 22
adding
  custom C code 163
  default CPUs 136
  I/O blocks 38
  multiprocessor support 108
  ports
    User-Code 171
  task handling 41
addresses
  PHS-bus 35
    S-functions 168
algebraic loops 122
applications
  building 219
  debugging 234
assertion handling 265
Asynchronous Task Specification block 156
asynchronous tasks 57
atomic subsystems 246
  reducing
    file sizes 249
    function sizes 248
automating
  RTI build process 230

B

background tasks 81
blocks
  bundling 247
  execution order 256
  model verification 265
  multiplexed I/O 36
  names 33
  priorities 258
  scalar I/O 36
board
  libraries 24
  numbers 33
branching interrupts 67
buffer mode 125
building
  default options 26
  models 219
    optimizing 246
    overview 222
bundling
  blocks 247

C

C code 163
  accessing variables 196
  comparing 186
  S-functions 164
    C methods 166
    MATLAB_MEX_FILE 167
    PHS-bus based system 168
    rebuilding 230
  User-Code 171
    adding ports 171
    interprocessor communication 173
capturing
  data 213
channels
  communication 115
  I/O block 33
code format 220
code generation target (RTW target) 160
combining
  interrupt sources 94
commands
  dsstartup.m 25
  rti_mdlcleanup 31
  rtimpdiag 234
Common Program Data folder 11
communication 115
  channels 115
  connections
    GigaLink 116
  sample time transitions 124
compiling
  models 219
configuration reference 31
configuring
  RTI-MP models 111
connections
  GigaLink 116
control design 15
conventions 268
  I/O 33
converter numbers 33
CPUs
  default 109
  master 111
  optional 109
currentTime
  precision problems 212
  time-stamping 271
custom C code 163
  accessing variables 196
  execution order 186
  rti_build2 230
  rtimp_build2 230
customizing
  MATLAB start-up 25

D

data
  acquisition 210
    modifying 213
  consistency
    tasks driven by interrupt blocks 60
  type
    restrictions 269
  types 37
data mode
  buffer 125
  unbuffer 125
deadlock situations 122
debugging 234
default
  build options 26
  CPUs 136
development steps 15
diagnostic tool 234
direct feedthrough 122
Documents folder 11
downloading
  models 219
dsstartup.m 25

E

enabled subsystems 246
errors
  reaction on run-time errors 215
examples
  FFT 53
  I/O block names 34
  throttle control 16
exceptions 236
Excluding
  subsystems 250
execution modes 54
execution order
  blocks 256
    bundling 247
  custom C code 186
  priorities 86
  RTI-MP 145
  tasks 44

F

feedthrough
  direct 122
FFT example 53
files
  reduced size
    TRC file 250
  reducing sizes 249

G

GigaLink connections 116
group numbers 33

H

handcrafting
  C-coded S-functions 164
  User-Code 171
handling 236
  assertions 265
  exceptions 236
  tasks 41
hardware interrupt blocks 62

hiding model parts
  subsystem omission tag (DsVdOmit) 251
  TRC Exclusion block 250
HIL 16

I

I/O blocks 36
  adding 38
  conventions 33
  unused channels 38
idle 44
implementing
  interprocessor communication 132
inlined S-functions 165
interprocessor communication 115
  channels 115
  Gigalink blocks 137
  sample time transitions 124
  User-Code 173
interprocessor interrupts 143
interrupt numbers 33
interrupts
  asynchronous 57
  branching 67
  interprocessor 143
  tasks driven by interrupt blocks 57
    combining 94
    hardware 62
    software 65
    subscheduling 96
    timer 64
  timer tasks 48

K

kernel 42

L

libraries
  board 24
limitations 270
  custom C code 282
  data types 269
  MATLAB Function block 282
  RTI-MP 276
  Stateflow 275
  task transition 74
Local Program Data folder 11

M

makefile
  user makefile 243
master CPU 111
MATLAB
  customizing start-up 25
MATLAB worker 278
MATLAB_MEX_FILE 167
message module
  sending messages 197
migrating models
  without RTI 30
model dependency viewer 153
model referencing 149
  basics 150
  configuring tasks 156
  creating a model reference from an atomic subsystem 155
  creating model references 154
  executing referenced models in interrupt-driven tasks 157
  multiple asynchronous function calls 156
  parent model 150
  referenced model 150
  simulating and building models containing Model blocks 160
  top-level model 150
model verification blocks 265
models
  building 219
  downloading 219
  evaluation 120
  multiprocessor distribution 114
module
  numbers 33
  types 33
multiple timer task mode 51
  data consistency 70
multiplexed I/O blocks 36
multiprocessor model 108
  optional CPU 109
  using submodels 109
multiprocessor systems 108

N

naming conventions 33
network topology 113
noninlined S-functions 165
numbers
  communication channels 133

O

optimizing
  build process 246
  RTI-MP model distribution 114
  simulation 246
options 226
  default 26
overrun situations 45
overview
  build processes 222
  development steps 15
  execution order
    blocks 256
    custom C code 186
    RTI-MP 145
    tasks 44
  naming conventions 33
  PHS-bus address 35

P

parallel builds 161
Parallel Computing Toolbox 278
parameter tuning
  State-Space blocks 259
  Transfer Fcn blocks 259
  Zero-Pole blocks 259
parent model 150
parpool 278
PHS-bus address 35
PHS-bus based system
  S-functions 168
platform support 22
port numbers 33
precision problems
  currentTime 212
preempting tasks 51
priorities 43
  blocks 256
  tasks 86
    RTI-MP 145
program building 219
protected data transfer 73
protocols
  swinging buffer 118
  virtual shared memory 119

R

rate transition 70
  alternative 74
  block options 72
  protected data transfer 73
  unprotected data transfer 73
rate-monotonic scheduling 42
RCP 15
ready 44
real-time kernel 42
rebuilding
  S-functions 230
reducing
  file sizes 248, 249
  function sizes 248
referenced
  models 149
referenced configuration set 31
referenced subsystem
  error messages 302
  limitations 275
restrictions 270
  data types 269
  RTI-MP 276
  Stateflow 275
RTI build process
  automating 230
RTI Gigalink Blockset 137
rti_mdlcleanup 31
RTI-MP
  communication
    channels 115
    sample time transitions 124

configuring models 111
diagnostic tool 234
execution order 145
network topology 113
rtimpdiag 234
RUN/STOP mechanism 205
running 44
run-time errors 215

S

sample times
offsets 51
transitions 70
scalar I/O blocks 36
scheduler 42
selecting
platform support 22
setting the simulation state 208
S-functions 164
C methods 166
MATLAB_MEX_FILE 167
PHS-bus system 168
rebuilding 230
shared memory 119
shared utilities directory 111
simulation
control 205
optimization 246
start/stop 205
time 271
simulation state
setting 208
simulation target (SIM target) 160
single timer task mode 50
software interrupt blocks 65
start/stop mechanism 205
starting
platform support 22
RTI 21
simulation 205
start-up
customizing MATLAB 25
state
simState 205
simulation 205
Stateflow
limitations 275
stopping
simulation 205
subscheduling
tasks 96
subsystems
atomic 246
enabled 246
excluding 250
trigger types 57
triggered 246
virtual 246
swinging buffer protocol 118
switching time 43
systems
multiprocessor 108

T

task overrun 90
task transition 70
demo model 74
limitations 74
task transitions
invalid 72
TaskConfiguration API
using 92
tasks 41
asynchronous 57
background 81
combining interrupts 94
driven by interrupt blocks 57
execution order 44
RTI-MP 145
multiple timer task mode 51
data consistency 70
preempting 51
priorities 86
single timer task mode 50
states 44
subscheduling 96
switching time 43
timer 48
timetable 99
time-triggered 99
transitions 60
templates
RTI-MP standard topology 113
User-Code 177
throttle control example 16
time
stamping 210
timetable 99
triggered 99
timer
interrupt blocks 64
tasks 48
timetable task 99
time-triggered task 99
topology
RTI-MP system 113
TRC file
reducing size 250
triggered subsystems 246
troubleshooting 289
tuning parameters
State-Space blocks 259
Transfer Fcn blocks 259
Zero-Pole blocks 259
turnaround times 45
types
data 37
triggered subsystems 57

U

unbuffer mode 125
unprotected data transfer 73
unused I/O channels 38
user makefile 243
User-Code 171
adding ports 171
in Simulink 174
interprocessor communication 173
templates 177

V

variable description
configuration settings 225
consuming products 224
data source 225
dSPACE configuration settings 225
generating products 224
Simulink configuration settings 225
variables
currentTime
precision problems 212
time-stamping 271
custom C code 196
vectorized I/O blocks 36
virtual
subsystems 246
virtual shared memory protocol 119

W

wrapper S-functions 165