RTI FPGA Programming Blockset

Guide

For RTI FPGA Programming Blockset 3.11

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.

© 2009 - 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

Content

This guide provides basic information about the RTI FPGA Programming Blockset, including the entire software environment and the supported hardware. You will also learn how to apply the main features of the blockset.

Audience profile

It is assumed that you have good knowledge in:

- Applying generally accepted FPGA design rules to ensure a stable and reliable FPGA application.
- The architectural structure of FPGAs (CLB architecture, slice flip-flops, memory resources, DSP resources, clocking resources) with a verifiable experience on digital designs (structural mapping, tool-flow knowledge, synthesis options, timing analysis).
- Modeling with Simulink®.
- Modeling with the Xilinx® System Generator Blockset.
- Using the Xilinx® design tools for simulation and debugging.

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.

Symbol	Description
⍰	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.

Introduction to the RTI FPGA Programming Blockset

Basic feature of the blockset

The RTI FPGA Programming Blockset is a Simulink® blockset that allows you to program an FPGA in a dSPACE system.

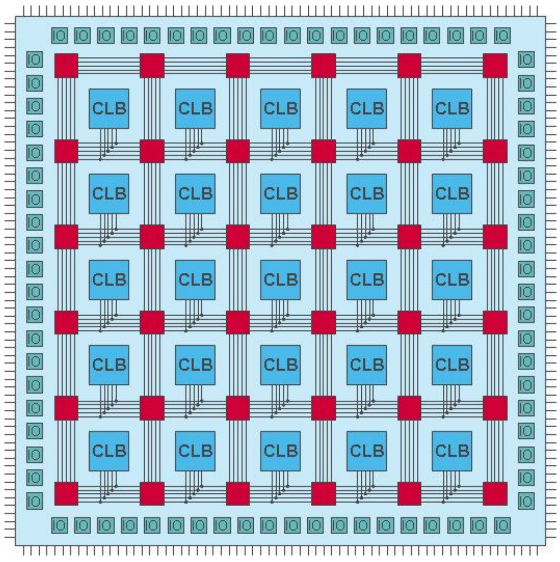
Where to go from here

Information in this section

FPGA Basics

FPGA architecture

FPGA stands for Field Programmable Gate Array. It is used in digital technology to allow modifications in a circuit's functionality without replacing hardware. A specific circuit is implemented as a specific configuration of the internal logic cells. The configurable logic blocks (CLBs) are connected with each other via switch boxes. The external connection is realized by I/O pins.



Reasons for using FPGAs

The main advantages of an FPGA are its flexibility and high speed for signal processing:

- Flexibility

  You can implement any kind of circuit, from a simple counter to an entire microprocessor.

  You can reconfigure your implementation in the development phase without hardware changes.

- Speed

  The specific implementations are usually concentrated on one functionality without any overhead.

  The bit architecture is often more efficiently than the word architecture with a fix data width.

  Execution can be done in parallel which results in a high throughput.

  Control loops (input of data, result calculation, output of data) can usually be performed at higher overall sample rates on an FPGA than on non-FPGA platforms.

Fields of application

All of the above-mentioned features of an FPGA are very useful for function prototyping, which can be performed with dSPACE hardware and software.

FPGAs are used, if you have a task that cannot be solved by standard implementations. They are used for tasks that require high performance, for example, for complex signal processing, or if you want to move some of your model's functionality to a separate application.

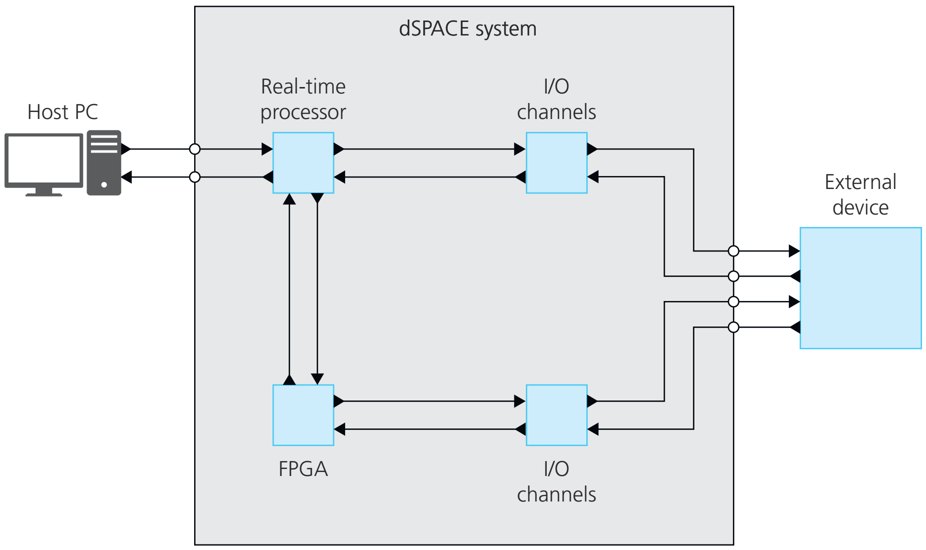
Basics on the System Architecture

Introduction

The system architecture determines the signals your FPGA applications can read, process, and update. It shows the data flow to download applications and to analyze data.

Components of a dSPACE system and their connections

The following illustration shows you the components and the signal connections of a dSPACE system, such as a MicroAutoBox III.



Description of the elements shown in the illustration:

- Host PC

  With the host PC, you download the processor and FPGA application. You can use dSPACE software installed on the host PC to experiment with your real-time application.
- Real-time processor

  The real-time processor executes your processor application. It initiates sending and receiving data to/from the FPGA via a board-specific bus. The real-time processor can also be connected to its own I/O channels to provide I/O functionality independently of the FPGA.
- FPGA

  The FPGA processes your FPGA application (FPGA logic implementation). The FPGA is connected to its own I/O channels to process I/O signals independently of the real-time processor. If the real-time processor initiates data exchange, the FPGA sends and receives data to/from a board-specific bus.

- I/O channels

  To access the external I/O signals of the external device, the signals must be converted so that they meet the requirements of the connected components. This is done, for example, by filters, signal amplifiers or analog-to-digital converters.
- External device

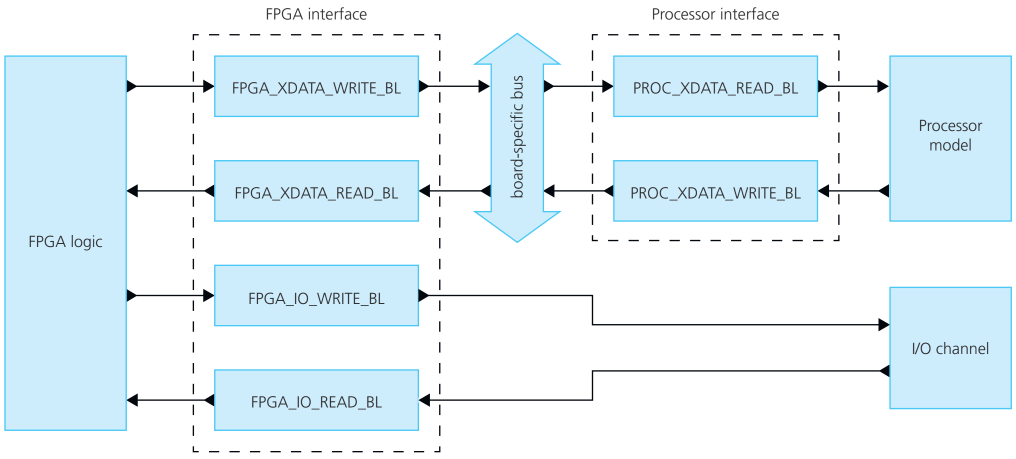
  Device that can be connected to the dSPACE system, such as an ECU.

Components of the RTI FPGA Programming Blockset

Introduction

The RTI FPGA Programming Blockset is a Simulink blockset for integrating an FPGA application into a dSPACE system. It provides RTI blocks for implementing the interface between the FPGA and the I/O channels of the dSPACE I/O board on which the FPGA is mounted, and the interface between the FPGA and the real-time processor. Furthermore, script functions let you execute functions via command line or M file.

Overview of the blockset



FPGA interface

The RTI blocks of the FPGA Interface library are used to model the interface of an FPGA application. The FPGA Interface blocks are used in the FPGA model. The blocks let you configure the interface between the FPGA and the I/O channels of the FPGA board that provides access to external I/O signals. The blocks also provide access to data storage that is used for exchanging data with the real-time processor.

The data exchange and all other communication between the FPGA board and the processor board runs via the board-specific bus:

- Intermodule bus when the MicroAutoBox II/III is used.
- Local bus when MicroLabBox is used.
- IOCNET when SCALEXIO is used.
- PHS bus when a PHS-bus-based system is used.

Processor interface

Processor interface when using a MicroAutoBox II, a MicroLabBox, or a PHS-bus-based system The RTI blocks of the Processor Interface library are used to implement processor communication with the FPGA via the board-specific bus. These blocks in the processor model are the counterparts of the related FPGA_XDATA blocks in the FPGA model. They access the data storage that you configured in the FPGA model. The setup block of the Processor Interface library lets you automatically generate the interface blocks for the processor model.

Processor interface when using a MicroAutoBox III or a SCALEXIO system When you use a MicroAutoBox III or a SCALEXIO system, the Processor Interface library is not required. Instead, you implement the data exchange between the FPGA model and the processor model with generated interface blocks of the Model Interface Package for Simulink.

Available script functions

The RTI FPGA Programming Blockset comes with script functions for executing functions via command line or M file. You can use them, for example, to execute scheduled build processes at night. For more information on the available script functions, refer to Introduction to the Script Interface of the RTI FPGA Programming Blockset (RTI FPGA Programming Blockset Script Interface Reference 🕮).

Related topics

Basics

Basics on Exchanging Data Between Processor and FPGA

Introduction

Only the processor application initiates a data exchange between the real-time processor and the FPGA. To be able to exchange data, the FPGA and processor interfaces provide different data storages.

Provided data storage

You can choose the data storage that you want to use for the data exchange with the following access types:

- Register

  Registers are implemented as flip-flops to exchange scalar values.
- Register group

  Register groups are implemented as flip-flops to exchange a data package with synchronized elements.
- Buffer

  Buffers are implemented in the FPGA RAM to exchange data packages with a buffer size of up to 32,768 elements.

For board-specific details on, such as data width or data format, refer to Exchanging Data With the Processor Model on page 57.

Details on the access types

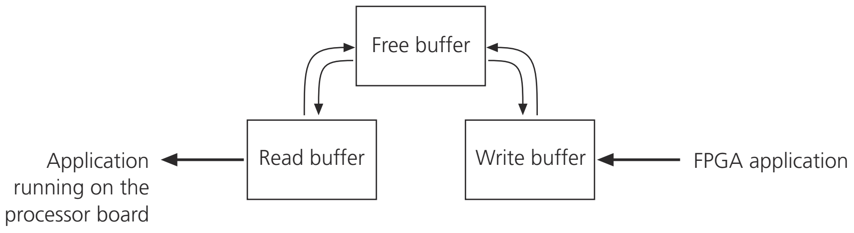
Register access Register access lets you access a scalar value in the register. The data is identified by the specified channel number. The values are transmitted element by element.

Register group access You can group registers to a register group via a common **Register Group ID**. All the values that belong to the same **Register Group ID** are synchronously updated in the FPGA subsystem.

For read access, the registers of a register group are read from the board-specific bus sequentially and then provided to the FPGA application simultaneously. For write access, the registers of a register group are sampled simultaneously in the FPGA application. These values form a consistent data group that is written to the board-specific bus.

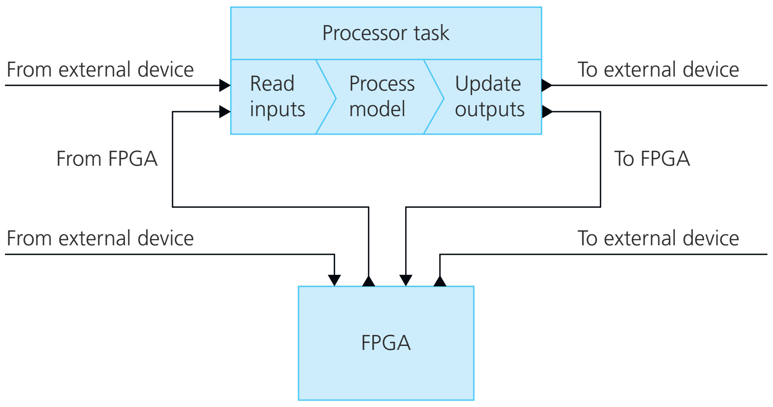
Buffer access Buffer access lets you access a vector value in the data buffer. One specific value of the data is identified by the specified channel number and the position within the buffer.

Data exchange is implemented via a FIFO buffer that works as a swinging buffer. This means that there are two separate buffers for reading and writing, and one buffer that switches between reading and writing. Only the pointer has to be changed to switch the buffer so that no buffer has to be copied from one position to another.



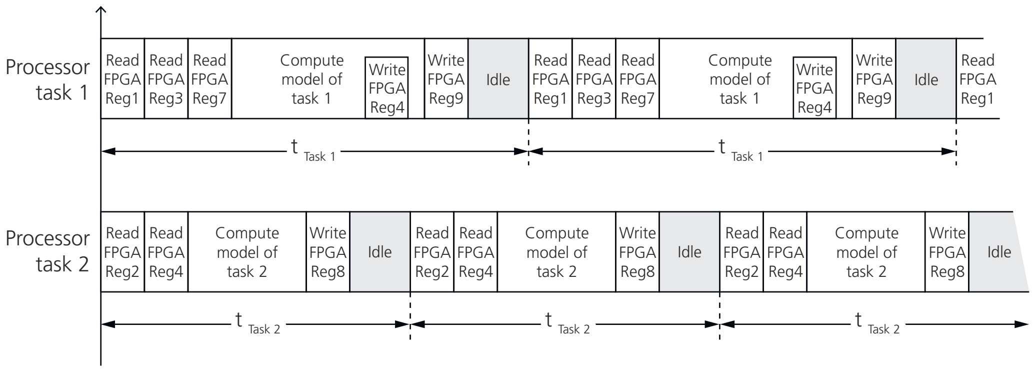
Processing the data exchange

Only the processor application can initiate data exchange between the real-time processor and the FPGA. The following illustration shows you the main steps that a processor task performs.



The FPGA executes its operations in parallel because the FPGA application is a logic implementation: i.e., the execution time of an FPGA is much faster than the execution time of the real-time processor.

Several processor tasks with different task periods can request read/write access to the FPGA at different points in time. These read/write requests are executed by the FPGA in parallel. The following illustration shows you the read/write requests of a real-time application with two processor tasks.



Related topics

Basics

Software Tools for Working With the RTI FPGA Programming Blockset

Introduction

To work with the RTI FPGA Programming Blockset, several software tools are necessary.

Tools from MathWorks®

MATLAB®, Simulink® and Simulink® Coder™ are used for modeling the application, which can be simulated in Simulink and executed on the real-time hardware.

For compatibility information, refer to Supported MATLAB Releases (New Features and Migration 🕮).

Tools from Xilinx

Xilinx provides several tools for designing applications for Xilinx FPGAs. The Vivado® Design Suite covers all the aspects of designing FPGAs. Working with the RTI FPGA Programming Blockset requires the following products of the Vivado Design Suite:

Xilinx System Generator for DSP Blockset (XSG) The Xilinx System Generator for DSP Blockset is a Simulink block library to graphically model FPGA applications. It contains blocks for standard functions, for example, for adding two inputs or implementing a FIR filter, but also special blocks to include VHDL code and logic circuits developed with the Xilinx Embedded Development Kit (EDK).

Xilinx Vivado The Xilinx Vivado software provides a logic design environment for Xilinx FPGAs. It contains tools and wizards, for example, for I/O assignment, power analysis, timing-driven design closure, and HDL simulation. With the Xilinx Vivado software, you can build the FPGA application according to the implemented FPGA model.

Currently you cannot order the Xilinx software from dSPACE. Install the Xilinx software before or after installing dSPACE software and follow the installation instructions from the Xilinx manuals.

Supported software versions

The following table shows you the supported operating systems and MATLAB versions for a specific Xilinx design tools version.

Xilinx Design Tools Version	MATLAB Version[1]	Operating System
Vivado 2020.2[2]	▪ MATLAB R2019b ▪ MATLAB R2020a ▪ MATLAB R2020b	Windows operating system that is supported by the RCP and HIL software of the current Release. For a list of supported operating systems, refer to Operating System (New Features and Migration 🕮). The listed Windows Server 2016 edition is not officially supported by Xilinx, but tested by dSPACE.

[1] The Processor Interface sublibrary of the RTI FPGA Programming Blockset also supports MATLAB R2021a.

[2] The Vivado HL WebPACK Editions of the Xilinx design tools also support the DS2655 (7K160) and DS6601 FPGA base boards. A separate license for the Xilinx System Generator for DSP is required for modeling FPGA applications with the RTI FPGA Programming Blockset.

dSPACE Blocksets

While the Simulink model for the FPGA application is implemented by using the Xilinx System Generator Blockset and the FPGA Interface blocks of the RTI FPGA

Programming Blockset only, the Simulink model for the processor application consists of the following blocksets:

- MicroAutoBox II, MicroLabBox, or PHS-bus-based system:
  Processor Interface blocks of the RTI FPGA Programming Blockset.
- MicroAutoBox III or SCALEXIO:
  Model port blocks of the Model Interface Package for Simulink.
- Blocks provided by a Simulink block library or a dSPACE RTI block library.

In this way you can enlarge the processor model with any function, further I/O or bus communication.

ControlDesk and ConfigurationDesk

ControlDesk is dSPACE's software tool for experimenting with a dSPACE system. It can be used to register the connected hardware, to manage the simulation platform, to download the real-time application (processor and FPGA application) and to control the experiment. The great variety of instruments allows you to access and visualize variables of the processor application in a comfortable way.

ConfigurationDesk is dSPACE's software tool for registering and managing a MicroAutoBox III or a SCALEXIO system. It is used to configure the signal chain from the ECU to the behavior model and building the real-time application. For experimenting with a MicroAutoBox III or a SCALEXIO real-time application, you have to use ControlDesk.

Related topics

References

Operating System (New Features and Migration 🕮)

Hardware Supported by the RTI FPGA Programming Blockset

Introduction

In contrast to other RTI blocksets, the RTI FPGA Programming Blockset is not board-specific. Here is an overview of the supported hardware.

MicroLabBox

Internally, MicroLabBox consists of the two boards DS1202 and DS1302. The DS1302 board provides the FPGA capabilities.

MicroLabBox's I/O FPGA module provides channels for analog input and output, digital input and output, and bus communication. The DS1302 board of MicroLabBox provides an FPGA unit with a Xilinx FPGA that can be programmed by using Xilinx Vivado and the RTI FPGA Programming Blockset.

An FPGA application is automatically integrated into a MicroLabBox application when building.

Note

dSPACE also provides RTI blocksets for MicroLabBox. To use blocks from the RTI FPGA Programming Blockset and the RTI blocksets in the same model, you have to use the flexible I/O framework for MicroLabBox. Refer to Features of the MicroLabBox Flexible I/O Framework on page 39.

The following table shows the main features of MicroLabBox's I/O FPGA:

Feature	**Description**
Programmable FPGA	▪ Xilinx® Kintex®-7 XC7K325T ▪ 326,080 logic cells ▪ 50,950 slices ▪ 4,000 kbit distributed RAM (max.) ▪ 840 DSP slices ▪ 16,020 kbit block RAM ▪ 100 MHz hardware clock frequency ▪ Multiple clock domains: Up to ten clock domains can be used to process FPGA subsystems with individual clock periods.
Analog input	▪ ADC (Class 1): 24 parallel A/D converters with 16-bit resolution and a sample rate of 1 MS/s. ▪ ADC (Class 2): 8 parallel A/D converters with 16-bit resolution and a sample rate of 10 MS/s.
Analog output	DAC (Class 1): 16 parallel D/A converters with 16-bit resolution.
Digital I/O	▪ Digital InOut (Class 1): 48 digital bidirectional channels, single-ended. ▪ Digital InOut (Class 2): 8 digital bidirectional channels, differential.
Resolver interface	2 resolver interfaces that support resolvers with an excited single coil.
Serial interface	2 UART (RS232) and 2 UART (RS422/485) interfaces. The UART (RS422/485) function supports full-duplex mode and half-duplex mode.
Feedback elements	▪ Status In: State of the initialization sequence that is started after programming the FPGA. ▪ LED Out: 4 customizable LEDs. ▪ Proc App Status: Execution state of the loaded executable application.

For further information on the hardware, refer to General Data (MicroLabBox Hardware Installation and Configuration 📖).

For further information on the software features, refer to RTI Block Settings for the DS1202 FPGA I/O Type 1 (RTI FPGA Programming Blockset - FPGA Interface Reference 📖).

MicroAutoBox II/III

The MicroAutoBox II/III consists of a board with the real-time processor and at least one I/O board. An FPGA base board provides the FPGA and the interfaces for adding different I/O modules.

DS1514 FPGA Base Board The DS1514 FPGA Base Board provides an FPGA. The following table shows the main features of the DS1514 FPGA Base Board.

Feature	Description
Programmable FPGA	▪ Xilinx Kintex®-7 XC7K325T ▪ 326,080 logic cells ▪ 50,950 slices ▪ 4,000 kbit distributed RAM (max.) ▪ 840 DSP slices ▪ 80 MHz hardware clock frequency. ▪ Multiple clock domains: Up to ten clock domains can be used to process FPGA subsystems with individual clock periods.
I/O interfaces	I/O interfaces are provided by I/O modules that are installed to the FPGA base board. Refer to I/O modules available for MicroAutoBox II/III on page 23.
Feedback elements	▪ Status In: State of the initialization sequence that is started after programming the FPGA. ▪ Temperature: Sensor to measure the FPGA's die temperature. ▪ LED Out: 1 FPGA Status LED.

For further information on the hardware, refer to the following data sheets:

- MicroAutoBox II 1401/1511/1514: General Data (MicroAutoBox II Hardware Reference 📖)
- MicroAutoBox II 1401/1513/1514: General Data (MicroAutoBox II Hardware Reference 📖)
- MicroAutoBox III: DS1514 FPGA Base Board Data Sheet (MicroAutoBox III Hardware Installation and Configuration 📖)

I/O modules available for MicroAutoBox II/III

I/O modules provide the I/O interface for the DS1514 FPGA Base Board. The following table shows the main features of the supported I/O modules:

Feature	DS1552 Multi-I/O Module	DS1552B1 Multi-I/O Module	DS1554 Engine Control I/O Module
Analog input	▪ ADC (Type A), Analog In 10[1]: 8 parallel A/D converters with 16-bit resolution and a conversion	▪ ADC (Type A), Analog In 11[1]: 8 parallel A/D converters with 16-bit resolution and a conversion	ADC (Type A), Analog In 14[1]: 14 parallel A/D converters with 16-bit

Feature	DS1552 Multi-I/O Module	DS1552B1 Multi-I/O Module	DS1554 Engine Control I/O Module
	time of 1 µs, conversion can be triggered. Input voltage range: 0 V ... +5 V ▪ ADC (Type B), Analog In 12[1]: 16 parallel A/D converters with 16-bit resolution and a conversion time of 5 µs. Input voltage range: -10 V ... +10 V.	time of 1 µs, conversion can be triggered. Input voltage range: -10 V ... +10 V ▪ ADC (Type B), Analog In 12[1]: 16 parallel A/D converters with 16-bit resolution and a conversion time of 5 µs. Input voltage range: -10 V ... +10 V.	resolution and a conversion time of 1 µs. Input voltage range: -10 V ... +10 V
Analog output	ADC (Type B), Analog Out 13[1]: 16 parallel A/D converters with 16-bit resolution and a conversion time of 5 µs. Input voltage range: -10 V ... +10 V		-
Digital I/O	▪ Digital In (Type A), Digital In 5[1]: 16 digital input channels. ▪ Digital In (Type B)/Digital Out (Type B), Digital InOut 6[1]: 8 digital bidirectional channels. ▪ Digital Out (Type A), Digital Out 5 [1]: 16 digital output channels.		▪ Digital In (Type B)/Digital Out (Type B), Digital InOut 8[1]: 8 digital bidirectional channels. ▪ Digital Out (Type A), Digital Out 7: 40 digital output channels.
Digital Crank/Cam input	3 digital inputs to read digital camshaft and crankshaft sensors.		5 digital inputs to read digital camshaft and crankshaft sensors.
Inductive zero voltage detector	1 digital input to read an inductive zero voltage detector.		
Knock signal input	-		4 analog inputs to read knock sensors.
Serial interface	2 UART (RS232/422/485) interfaces. RS422/485 supports full-duplex mode and half-duplex mode. UART 1 can be used without modification. To use UART 2, your DS1552 has to be modified by dSPACE.		-
Sensor supply	1 adjustable supply voltage in the voltage range 2 V ... 20 V.		1 supply voltage providing 5 V.

[1] The channel name depends on the framework.

Note

dSPACE also provides an RTI blockset for the DS1552 Multi-I/O Module. You cannot use blocks from the RTI FPGA Programming Blockset and the RTI DS1552 I/O Extension Blockset in the same model.

For further information on the hardware, refer to the following data sheets.

- MicroAutoBox II:
  - Data Sheet DS1552 Multi-I/O Module (MicroAutoBox II Hardware Reference 📖)
  - Data Sheet DS1554 Engine Control I/O Module (MicroAutoBox II Hardware Reference 📖)
- MicroAutoBox III:
  - DS1552 Multi-I/O Module Data Sheet (MicroAutoBox III Hardware Installation and Configuration 📖)
  - DS1554 Engine Control I/O Module Data Sheet (MicroAutoBox III Hardware Installation and Configuration 📖)

For more information on the software features, refer to the following framework references.

- MicroAutoBox II
  - RTI Block Settings for the FPGA1401Tp1 with Multi-I/O Module Frameworks (RTI FPGA Programming Blockset - FPGA Interface Reference 📖)
  - RTI Block Settings for the FPGA1401Tp1 with Engine Control I/O Module Framework (RTI FPGA Programming Blockset - FPGA Interface Reference 📖)
- MicroAutoBox III
  - RTI Block Settings for the FPGA1403Tp1 with Multi-I/O Module Frameworks (RTI FPGA Programming Blockset - FPGA Interface Reference 📖)
  - RTI Block Settings for the FPGA1403Tp1 with Engine Control I/O Module Framework (RTI FPGA Programming Blockset - FPGA Interface Reference 📖)

SCALEXIO system

A SCALEXIO system consists of SCALEXIO processing hardware and a selection of I/O boards. The SCALEXIO processing hardware, such as a SCALEXIO Real-Time PC, provides the computation power of the system.

I/O boards are available for the various requirements, for example, for accessing analog input and output signals, digital input and output signals, and bus communication.

With the SCALEXIO FPGA base boards, you can integrate an FPGA application into a SCALEXIO system. The board provides an FPGA that you can program using the Xilinx System Generator Blockset and the RTI FPGA Programming Blockset.

The following table shows the main features of the supported SCALEXIO FPGA base boards.

Feature	**DS2655 (7K160)**	**DS2655 (7K410)**	**DS6601**	**DS6602**
Programmable FPGA	XILINX® KINTEX®-7-160T ▪ 162,240 logic cells ▪ 600 DSP slices ▪ 2,188 kbit maximum distributed RAM	XILINX® KINTEX®-7-410T ▪ 406,720 logic cells ▪ 1,540 DSP slices ▪ 5,663 kbit maximum distributed RAM	XILINX® KINTEX® UltraScale™ KU035 ▪ 444,343 system logic cells ▪ 1,700 DSP slices	XILINX® KINTEX® UltraScale+™ KU15P ▪ 1,143,450 system logic cells ▪ 1,968 DSP slices

Feature	DS2655 (7K160)	DS2655 (7K410)	DS6601	DS6602
	▪ 11,700 kbit total block RAM ▪ 125 MHz hardware clock frequency ▪ Multiple clock domains: Up to ten clock domains can be used to process FPGA subsystems with individual clock periods.	▪ 28,620 kbit total block RAM ▪ 125 MHz hardware clock frequency ▪ Multiple clock domains: Up to ten clock domains can be used to process FPGA subsystems with individual clock periods.	▪ 5,908 kbit maximum distributed RAM ▪ 19,000 kbit total block RAM ▪ 125 MHz hardware clock frequency ▪ Multiple clock domains: Up to ten clock domains can be used to process FPGA subsystems with individual clock periods.	▪ 9,800 kbit maximum distributed RAM ▪ 34,600 kbit total block RAM ▪ 36,000 kbit UltraRAM ▪ 125 MHz hardware clock frequency ▪ Multiple clock domains: Up to ten clock domains can be used to process FPGA subsystems with individual clock periods.
I/O interfaces	I/O interfaces are provided by up to 5 I/O modules that are installed to the SCALEXIO FPGA base board. Refer to I/O modules available for SCALEXIO FPGA base board on page 27.			
Angular processing units	▪ APU Master: You can write angle-based time base values to the IOCNET bus with up to 6 angular processing units of the SCALEXIO FPGA base board. Other boards connected to the APU bus as APU slaves can read the corresponding time base value. ▪ APU Slave: You can read angle-based time base values from the IOCNET with up to 6 angular processing units of the SCALEXIO FPGA base board. Another board connected to the IOCNET is specified as the APU master and writes the time base value.			
MGT interface	-		1 Multi-Gigabit Transceiver (MGT) interface with 4 bidirectional lanes that can be used for communication with external devices or with other DS6601/DS6602 FPGA Base Boards. To support MGT communication, an MGT module must be installed to the FPGA base board.	
Inter-FPGA communication	Direct data exchange with other SCALEXIO FPGA base boards via inter-FPGA communication bus.			
Interrupt lines	8		16	
Feedback elements	▪ Status In: State of the initialization sequence that is started after programming the FPGA. ▪ CN App Status: Execution state of the loaded executable application on the computation node. ▪ IOCNET Global Time: Time that is specified for all modules connected to IOCNET. ▪ LED Out: 1 FPGA Status LED.			

For further information on the hardware, refer to Hardware for FPGA Applications (SCALEXIO Hardware Installation and Configuration 🕮).

For further information on the software features, refer to the following:

- RTI Block Settings for the DS2655 FPGA Base Board Framework (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮)
- RTI Block Settings for the DS6601 FPGA Base Board Framework (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮)

- RTI Block Settings for the DS6602 FPGA Base Board Framework (RTI FPGA Programming Blockset - FPGA Interface Reference 📖)

I/O modules available for SCALEXIO FPGA base board

I/O modules provide the I/O interface for the SCALEXIO FPGA base boards DS2655, DS6601, and DS6602. The following table shows the main features of the supported I/O modules:

Feature	DS2655M1	DS2655M2	DS6651
Analog input	Analog In: 5 analog input channels with 14-bit resolution and 4 MS/s sample rate.	-	A total of 6 analog input channels with 16-bit resolution and 5 MS/s sample rate: ▪ 4 Analog In channels ▪ 2 Analog In-L channels with a switchable load. All channels can be triggered by different trigger sources.
Analog output	Analog Out: 5 analog output channels with 14-bit resolution and 7.8125 MS/s update rate.	-	A total of 6 analog output channels with 16-bit resolution and 10.417 MS/s update rate: ▪ 4 Analog Out channels ▪ 2 Analog Out-T channels that can output the voltage signal via a transformer.
Digital I/O	10 digital I/O channels that can be used as follows: ▪ Digital In ▪ Digital Out ▪ Digital InOut (bidirectional): An external signal is only available if the direction of a channel is set to In, otherwise the input signal is consistent with the output signal.	32 versatile digital I/O channels that can handle bit-wise data or serial communication. The main features are I/O functions that use the digital channels to implement a specific I/O functionality. ▪ Digital In: Up to 32 digital input functions that provide bit-wise access. ▪ Digital Out: Up to 32 digital output functions that provide bit-wise access. ▪ Digital Out-Z: Up to 16 digital output functions that provide bit-wise access and a high-impedance output state (tristate). ▪ RS232 Rx: Up to 8 serial functions that receive data values from RS232 networks. ▪ RS232 Tx: Up to 8 serial functions that transmit data values to RS232 networks. ▪ RS485 Rx: Up to 8 serial functions that receive data	16 versatile digital I/O channels that can handle bit-wise data or serial communication. The main features are I/O functions that use the digital channels to implement a specific I/O functionality. ▪ Digital In: Up to 16 digital input functions that provide bit-wise access. ▪ Digital In/Out-Z: Up to 4 digital I/O functions that provide bit-wise access and a high-impedance output state (tristate). ▪ Digital Out: Up to 16 digital output functions that provide bit-wise access. ▪ Digital Out-Z: Up to 8 digital output functions that provide bit-wise access and a high-impedance output state (tristate). ▪ RS485 Tx: Up to 8 serial functions that transmit data values to RS485 networks in simplex mode. ▪ RS485 Rx: Up to 8 serial functions that receive data values from RS485 networks in simplex mode. ▪ RS485 Rx/Tx: Up to 4 serial functions that exchange data values with RS485 networks in half-duplex mode.

Feature	DS2655M1	DS2655M2	DS6651
		values from RS485 networks in simplex mode. ▪ RS485 Rx/Tx: Up to 8 serial functions that exchange data values with RS485 networks in half-duplex mode. ▪ RS485 Tx: Up to 8 serial functions that transmit data values to RS485 networks in simplex mode.	

For further information on the hardware, refer to the following:

- Data Sheet of the DS2655M1 Multi-I/O Module (SCALEXIO Hardware Installation and Configuration 🕮)
- Data Sheet of the DS2655M2 Digital I/O Module (SCALEXIO Hardware Installation and Configuration 🕮)
- Data Sheet of the DS6651 Multi-I/O Module (SCALEXIO Hardware Installation and Configuration 🕮)

For further information on the software features, refer to the following:

- RTI Block Settings for the DS2655M2 I/O Module Framework (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮)
- RTI Block Settings for the DS2655M2 I/O Module Framework (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮)
- RTI Block Settings for the DS6651 Multi-I/O Module Framework (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮)

PHS-bus-based system

A PHS-bus-based system consists of at least one processor board and a selection of I/O boards. You can use a DS1006 Processor Board or DS1007 PPC Processor Board as the processor board that provides the computation power of the system.

I/O boards are available for the various requirements, for example, for accessing analog input and output signals, digital input and output signals, and bus communication.

With the DS5203 FPGA Board, you can integrate an FPGA application into a PHS-bus-based system. The board provides a Xilinx FPGA that you can program using the Xilinx System Generator Blockset and the RTI FPGA Programming Blockset.

The following table shows the main features of the supported FPGA boards.

Feature	DS5203 (7K325)	DS5203 (7K410)
Programmable FPGA	Xilinx Kintex®-7 XC7K325T ▪ approx. 326 k logic cells: ▪ 50950 Kintex-7 slices	Xilinx Kintex® XC7K410T ▪ approx. 407 k logic cells: ▪ 63550 Kintex-7 slices

Feature	DS5203 (7K325)	DS5203 (7K410)
	▪ 840 DSP slices ▪ 4000 kbit distributed RAM ▪ 16020 kbit block RAM ▪ 100 MHz hardware clock frequency	▪ 1540 DSP slices ▪ 5663 kbit distributed RAM ▪ 28620 kbit block RAM ▪ 100 MHz hardware clock frequency
Analog input	ADC: 6 analog input channels with 14-bit resolution and 10 MS/s sample rate.	
Analog output	DAC: 6 analog output channels with 14-bit resolution and 10 MS/s update rate.	
Digital I/O	▪ Digital In: 16 digital bidirectional channels. ▪ Digital Out: 16 digital bidirectional channels (also used for Digital In).	
Further I/O interfaces	Further I/O interfaces can be provided by an I/O modules that are installed to the DS5203 FPGA Board. Refer to I/O modules available for DS5203 FPGA Board on page 29.	
Angular processing units	▪ APU Master: The angular processing unit writes the angle-based time base value to the APU bus. Other boards connected to the APU bus as APU slaves can read the time base value. ▪ APU Slave: The angular processing unit reads the angle-based time base value from the APU bus. Another board connected to the APU bus is specified as the APU master and writes the time base value.	
Inter-FPGA communication	Direct data exchange with another DS5203 FPGA Board via inter-FPGA communication bus.	
Feedback elements	▪ Status In: State of the initialization sequence that is started after programming the FPGA. ▪ LED Out: 1 FPGA Status LED.	

DS5203 (SX95) and DS5203 (LX50) Due to the introduction of Xilinx Vivado, the blockset support of the DS5203 (SX95) and DS5203 (LX50) FPGA Boards is limited to building the processor interface.The RTI FPGA Programming Blockset as of version 3.0 does not support the modeling and building of new FPGA applications.

For details, refer to Features of the Processor Interface of the RTI FPGA Programming Blockset (RTI FPGA Programming Blockset - Processor Interface Reference 📖).

I/O modules available for DS5203 FPGA Board

The DS5203M1 Multi-I/O Module is used to extend the I/O capability of the DS5203 FPGA Board.

Feature	Description
Analog input	ADC (M1): 6 analog input channels with 14-bit resolution and 10 MS/s sampling rate.

Feature	Description
Analog output	DAC (M1): 6 analog output channels with 14-bit resolution and 10 MS/s update rate.
Digital I/O	▪ Digital In (M1): 16 digital bidirectional channels. ▪ Digital Out (M1): 16 digital bidirectional channels (also used for Digital In).
Sensor supply	1 adjustable supply voltage in the voltage range 2 V ... 20 V.

For further information on the hardware, refer to DS5203 FPGA Board (PHS Bus System Hardware Reference 📖).

For further information on the software features, refer to RTI Block Settings for the DS5203 with Multi-I/O Module (DS5203M1) Frameworks (RTI FPGA Programming Blockset - FPGA Interface Reference 📖).

Using different dSPACE hardware

Note

Using an FPGA model on different dSPACE hardware requires some model modifications, refer to Migrating to Different FPGA Hardware on page 198.

Related topics

Basics

Software Tools for Working With the RTI FPGA Programming Blockset.. 19

Modeling Aspects and Implementation Workflow

Introduction

Introduces to the features of the RTI FPGA Programming Blockset and how you implement an FPGA application.

Where to go from here

Information in this section

Features of the RTI FPGA Programming Blockset

Introduction

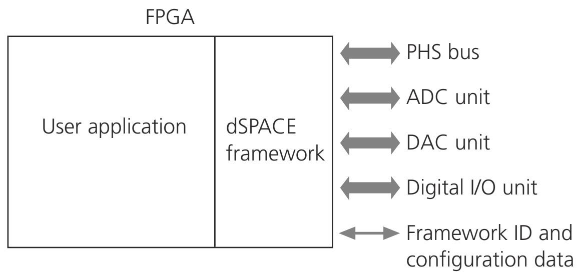
The RTI FPGA Programming Blockset provides dynamically configured dialog settings and an interface to some development features of the Xilinx System Generator Blockset.

Framework-specific settings

You can specify the hardware to be used in the block dialog of the FPGA_SETUP_BL block. The hardware, for example, a DS5203 FPGA Board or a plugged piggyback module, is represented by a framework INI file. This file describes all the relevant settings to be loaded dynamically to the RTI blocks of the FPGA Interface blocks for board communication via PHS bus, intermodule bus, or IOCNET, I/O access and interrupt handling. This allows you to create and configure the FPGA interface and the processor interface without a connection to the hardware.

There can be different framework INI files for the same hardware.

The first time you select a framework after installing the RTI FPGA Programming Blockset, it is loaded to the modeling environment. This enables also the FPGA to provide some basic functionality. For example, the DS5203 (7K325) with onboard I/O framework provides register-based and buffer-based access to the PHS bus. It also provides the number of channels available for external I/O access.



Function-specific settings

The dialogs of the RTI blocks have several pages. The Parameters and the Description pages are empty until you select a function by specifying the access type for board communication or specifying the I/O type for an I/O access on the Unit page of a block dialog. These pages are dynamically filled with text and settings according to the selected function.

These dynamic settings are also defined in the framework.

Flexible signal handling

To model data exchange via the external I/O pins, you must only add the RTI block for a specific access direction (read or write) to the model. The signal type (analog or digital) can be configured in the block dialog. The availability of the signal types depends on the selected framework.

Interaction with the Xilinx System Generator

If you implemented the required functionality of the FPGA application by using the Xilinx System Generator Blockset only, you have direct access to all its features for configuring, debugging and building the application. If you want to integrate the FPGA model in a dSPACE system, certain blocks of the Xilinx Blockset must be replaced by blocks from the RTI FPGA Programming Blockset. Most of the above-mentioned features are now available indirectly by using the RTI blocks, for example, the timing analysis for debugging purposes.

Modeling Aspects

Introduction

Describes general modeling aspects that have no particular context.

One subsystem for one FPGA board

The FPGA application which you implement for one FPGA board must be encapsulated in a Simulink subsystem. If you want to run FPGA applications on several FPGA boards, you must provide the same number of subsystems.

Each subsystem must contain an **FPGA_SETUP_BL** block.

For the MicroAutoBox II, MicroLabBox, and PHS-bus-based systems, the assignment of an FPGA board to a subsystem is done in the **PROC_SETUP_BL** block, which you must add to the processor model. For the MicroAutoBox III and SCALEXIO systems, the assignment of a subsystem to an FPGA board is done in ConfigurationDesk.

Name of the FPGA application

You can specify the name of the FPGA application via the dialog of the **FPGA_SETUP_BL** block. The name is used as a description for the built FPGA application and can be displayed in your experiment software, such as ControlDesk. For MicroAutoBox III and SCALEXIO systems, the FPGA application name is used as the custom function block type in ConfigurationDesk.

By default, the name of the subsystem is automatically used as the first part of the application name when you build the FPGA application.

Tip

If you use the same FPGA application name for variants of the FPGA model for the MicroAutoBox III or SCALEXIO systems, all the variants have the same custom function block type. This helps you reuse FPGA variants in ConfigurationDesk, because you can exchange the FPGA application without changing the custom function block in the signal chain.

Implementing data exchange between FPGA application and processor application

Application data that you want to process in the processor model or other subsystems of your Simulink model must be exchanged via the board-specific bus.

You implement the data exchange between the processor application and the FPGA application by generating processor interface blocks which uses the blocks from the RTI FPGA Programming Blockset's Processor Interface library.

Note

If the FPGA application supports SCALEXIO multicore processor applications, implement the FPGA interface blocks to the subsystems of your FPGA model in a way that a subsystem always exchange data with the same processor model. Refer to Aspects on FPGA Applications Supporting Multicore Processor Applications on page 101.

Synchronizing FPGA data

If different data paths have different length, the data is valid at different points in time and the outputs are asynchronous. This might cause an unexpected behavior of your application.

Ensure that the data paths are synchronized. You can do this, for example, by using the following methods:

- Add Xilinx Register or Delay blocks to extend the data paths to the same length.
- Downsample the sampling rate of all the FPGA Interface blocks in the subsystem to wait a certain time for data synchronization. For details on setting the sampling rate, refer to Parameters Page (FPGA_SETUP_BL) (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮).

Modeling FPGA tasks with individual clock periods

You can set multiple clock domains inside one FPGA model to adapt the clock period of specific parts of your FPGA application to your requirements. PHS-bus-based systems do not support multiple clock domains.

For instructions, refer to How to Use Multiple Clock Domains for FPGA Modeling on page 48.

Features to access the FPGA application via experiment software

You can enable access to the FPGA application with your experiment software when the FPGA application runs on the platform. The following platforms support the tracing of FPGA signals, tuning of FPGA constants, or accessing intervention points for testing and scaling the I/O signals:

Platform	FPGA Signal Tracing	Tunable FPGA Constants	FPGA Test Access	FPGA Scaling		
				Scaling Analog I/O Signals	Inverting Digital I/O Signals	Inverting RS232/485 Signals
SCALEXIO	✓[1)]	✓	✓	✓	✓	✓
MicroLabBox	✓	✓	✓	✓	✓	–

Platform	FPGA Signal Tracing	Tunable FPGA Constants	FPGA Test Access	FPGA Scaling		
				Scaling Analog I/O Signals	Inverting Digital I/O Signals	Inverting RS232/485 Signals
MicroAutoBox II	✓	✓	✓	✓	✓	–
MicroAutoBox III	✓	✓	✓	✓	✓	–
PHS-bus-based system	–	–	–	–	–	–

[1] ✓ = Supported, – = Not supported

For more information, refer to Accessing FPGA Applications with your Experiment Software on page 167.

Modifying models for a MicroAutoBox III or a SCALEXIO system

If you use a MicroAutoBox III or a SCALEXIO system, you should modify the FPGA model or the processor model only in the entire model. If you export the FPGA build results and the processor model to a ConfigurationDesk project, the project will automatically be updated. You cannot update the entire model to updates that are done outside of it. Refer to How to Export Build Results and Processor Models to ConfigurationDesk Projects on page 146.

If you do not use the export functionality, you have to update the model modifications in ConfigurationDesk.

Related topics

Basics

Typical Workflow

Introduction

When you use the RTI FPGA Programming Blockset, there is a typical workflow which you can follow.

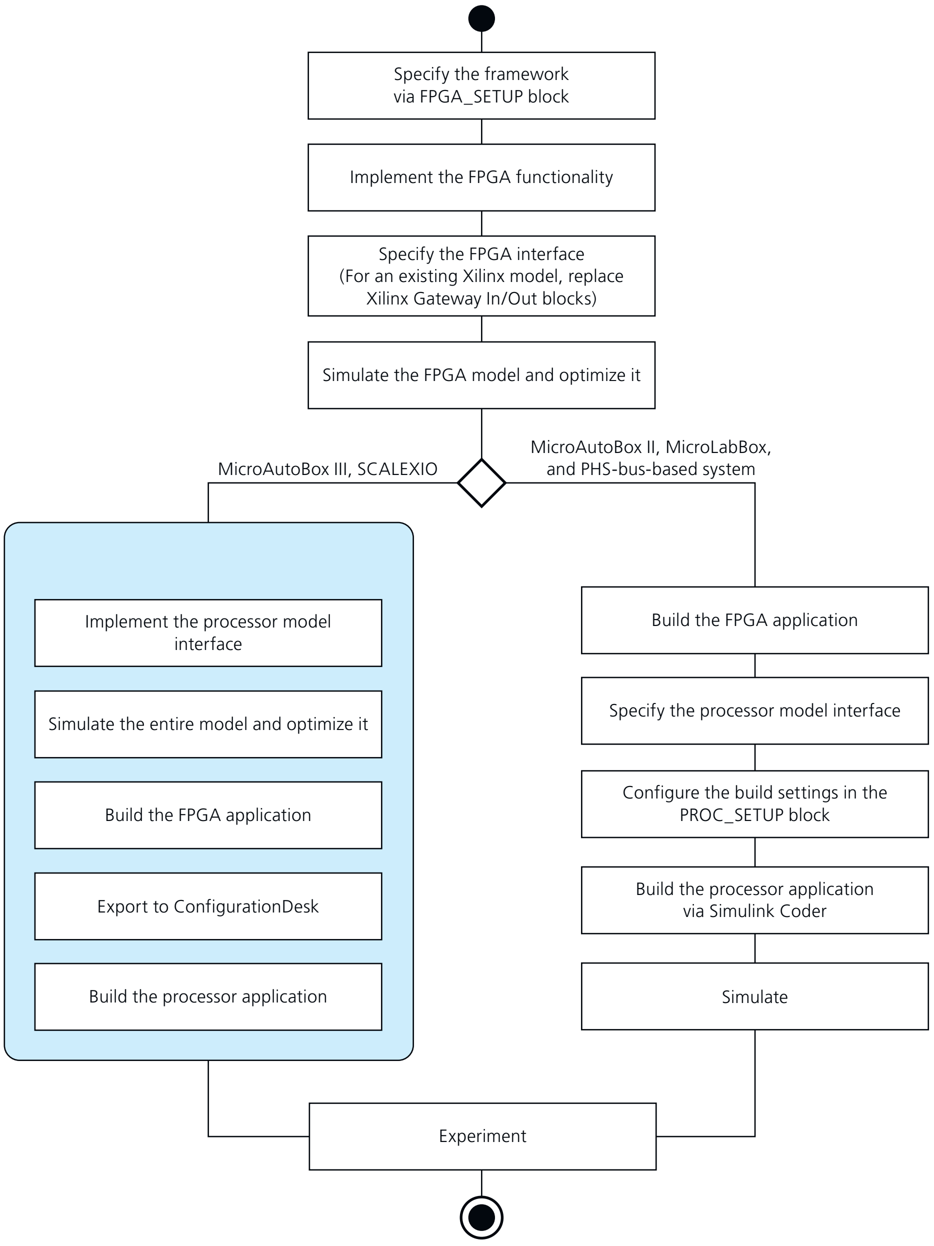
Graph of the workflow

The following image shows the workflow that branches for SCALEXIO and non-SCALEXIO platforms for integrating the built FPGA application in the processor application.

For a description of the working steps, refer to:

- Workflow when using a MicroAutoBox II, MicroLabBox, or PHS-bus-based system on page 37

  or
- Workflow when using a MicroAutoBox III or SCALEXIO system on page 38


Specify the framework via FPGA_SETUP block
Implement the FPGA functionality
Specify the FPGA interface (For an existing Xilinx model, replace Xilinx Gateway In/Out blocks)
Simulate the FPGA model and optimize it
MicroAutoBox III, SCALEXIO
MicroAutoBox II, MicroLabBox, and PHS-bus-based system
Implement the processor model interface
Simulate the entire model and optimize it
Build the FPGA application
Export to ConfigurationDesk
Build the processor application
Build the FPGA application
Specify the processor model interface
Configure the build settings in the PROC_SETUP block
Build the processor application via Simulink Coder
Simulate
Experiment

Workflow when using a MicroAutoBox II, MicroLabBox, or PHS-bus-based system

It is assumed that you start with a new empty Simulink model.

1. Create a subsystem in the Simulink model.

   There must be always a root model that is executed on the processor board and one FPGA subsystem for each FPGA board that you want to use for your processor application.
2. Add the required Xilinx blocks to the subsystem to implement the desired FPGA functionality.

   Do not use the **Gateway In** and the **Gateway Out** blocks in the subsystem. If you use an existing Xilinx model, you have to replace the **Gateway In** and **Gateway Out** blocks with blocks from the RTI FPGA Programming Blockset.
3. Specify the interface of the FPGA

   You must first add the **FPGA_SETUP_BL** block to the FPGA subsystem and specify general settings such as the framework to be used with the block's dialog. Then you can add the required blocks from the RTI FPGA Programming Blockset to the FPGA subsystem to implement read/write access to external I/O and to exchange data with the processor model. For further information, refer to Modeling External I/O Access on page 52 and Modeling Processor Model Access on page 57.
4. Simulate the behavior of the FPGA subsystem.

   You can use the simulation ports of the **FPGA Interface** blocks to stimulate the FPGA subsystem or to view the outputs in an offline simulation. You can also execute a timing analysis using the Xilinx Timing Analyzer to check whether required timing constraints can be met. For further information, refer to Simulating an FPGA Subsystem on page 122.
5. Build the FPGA application from the FPGA subsystem.

   You can start the build process for the FPGA subsystem in the **FPGA_SETUP_BL** block. The Xilinx System Generator and the Vivado compiler are used to generate the FPGA application. For further information, refer to How to Build FPGA Applications (MicroAutoBox II, MicroLabBox, PHS-Bus-Based System) on page 128.
6. Add a **PROC_SETUP_BL** block to the processor model and use it to create the interface blocks required for communication with the FPGA subsystem. For further information, refer to Implementing the Processor Interface to the Processor Model on page 113.
7. Add further blocks to the processor model to implement the required functionality on the processor side.
8. Build the processor application by using the **PROC_SETUP_BL** block.

   You can integrate the programming of the FPGA into the processor application or generate a separate burn application to integrate the programming. For further information, refer to How to Build Single-Core Processor Applications (MicroAutoBox II, MicroLabBox, PHS-Bus-Based System) on page 132 and How to Create Burn Applications on page 138.
9. Simulate the entire model in Simulink simulation mode or execute it on the real-time hardware for experimenting. For further information, refer to Simulating a Processor Model on page 123 and Experimenting with an FPGA Application on page 163.

Workflow when using a MicroAutoBox III or SCALEXIO system

It is assumed that you start with a new empty Simulink model.

1. Create at least two subsystems in the Simulink model:
   - A subsystem for the processor model that is executed on the processor board.
   - One subsystem for each FPGA board that you want to use with your processor model.
2. Add the required Xilinx blocks to the FPGA subsystem to implement the FPGA functionality.

   Do not use the **Gateway In** and the **Gateway Out** blocks in the subsystem. If you use an existing Xilinx model, you have to replace the **Gateway In** and **Gateway Out** blocks with blocks from the RTI FPGA Programming Blockset.
3. Specify the interface of the FPGA.

   You must firstly add the **FPGA_SETUP_BL** block to the FPGA subsystem and specify general settings such as the framework and the I/O modules to be used with the block's dialog. Then you can add the required blocks from the RTI FPGA Programming Blockset to the FPGA subsystem to implement read/write access to external I/O and to exchange data with the behavior model. For further information, refer to Modeling External I/O Access on page 52 and Modeling Processor Model Access on page 57.
4. Simulate the behavior of the FPGA subsystem.

   You can use the simulation ports of the **FPGA Interface** blocks to stimulate the FPGA subsystem or to view the outputs in an offline simulation. You can also execute a timing analysis by using the Xilinx Timing Analyzer to check whether required timing constraints can be met. For further information, refer to Simulating an FPGA Subsystem on page 122.
5. Generate the corresponding processor interface blocks for the processor model and add them to the subsystem of the processor model. For further information, refer to Implementing the Processor Interface to the Processor Model on page 113.
6. Add further blocks to the processor model to implement the required functionality on the processor side.
7. Simulate the entire model in Simulink simulation mode. For further information, refer to Simulating a Processor Model on page 123.
8. Build the FPGA application from the FPGA subsystem.

   You can start the build process for the FPGA subsystem in the **FPGA_SETUP_BL** block. The Xilinx System Generator and the Vivado compiler are used to generate the FPGA application. For further information, refer to How to Build FPGA Applications (MicroAutoBox III, SCALEXIO) on page 140.
9. Export the FPGA application and the processor model to ConfigurationDesk. For further information, refer to How to Export Build Results and Processor Models to ConfigurationDesk Projects on page 146.
10. Configure the signal properties in your instantiated FPGA function, for example, the electrical characteristics. For further information, refer to Configuring Function Blocks (ConfigurationDesk Real-Time Implementation Guide 🕮).

11. Build the processor application using ConfigurationDesk. For further information, refer to How to Build Processor Applications (MicroAutoBox III, SCALEXIO) on page 149.
12. Execute the entire model on the real-time hardware for experimenting. For further information, refer to Experimenting with an FPGA Application on page 163.

Related topics

Basics

Features of the MicroLabBox Flexible I/O Framework

Introduction

The **DS1202 FPGA IO Type 1 (Flexible I/O)** framework lets you model real-time applications for MicroLabBox together with the RTI1202 and the RTI Electric Motor (EMC) blocksets.

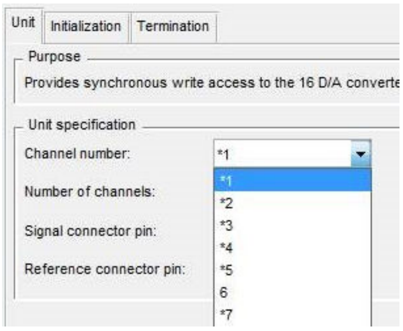
An I/O channel of MicroLabBox is either used by the standard I/O features of MicroLabBox or by a custom FPGA application. The standard I/O features of MicroLabBox let you access the I/O interface with the RTI1202 and the EMC blocksets. These features are automatically implemented into the FPGA if the real-time application that you load to the hardware does not contain a custom FPGA application. Custom FPGA applications that are built with the **DS1202 FPGA I/O Type 1** framework do not provide the standard I/O features for remaining I/O channels. Therefore, you cannot access the remaining I/O channels with the RTI1202 and the EMC blocksets.

Features of the DS1202 FPGA I/O Type 1 (Flexible I/O) framework

The framework provides the following features:

- During the build process of the custom FPGA application, the standard I/O features for remaining I/O channels are added. The standard I/O features let you use the remaining I/O channels with the RTI1202 and the EMC blocksets.
- The framework provides the same blocks as the **DS1202 FPGA I/O Type 1** framework.
- The framework is compatible with the **DS1202 FPGA I/O Type 1** framework. You can switch between the **DS1202 FPGA I/O Type 1 (Flexible I/O)** and the **DS1202 FPGA I/O Type 1** frameworks. The compatibility lets you save time when you model and test the the FPGA application. Refer to Workflow when Using MicroLabBox Flexible I/O Framework on page 40.

- I/O channels that are used by the DS1202 FPGA I/O Type 1 (Flexible I/O) framework are marked within the RTI1202 and the EMC blocksets with an asterisk.



Purpose

Provides synchronous write access to the 16 D/A converte

Unit specification

Channel number: *1

Number of channels:

Signal connector pin:

Reference connector pin:

*1
*2
*3
*4
*5
6
*7

You must not access the same channel with the DS1202 FPGA I/O Type 1 (Flexible I/O) framework and the RTI1202 or EMC blocksets.

Note

The DS1202 Serial Interface blocks (rti1202serlib) do not mark I/O channels that are used by the custom FPGA application with an asterisk.

- If you use DS1202 Serial Interface blocks, make sure that the custom FPGA application does not use the same communication channels.

Related topics

Basics

Overview of the RTI Electric Motor Control Blockset (RTI Electric Motor Control Blockset Reference 🕮)

References

Overview of RTI1202 (MicroLabBox RTI Reference 🕮)

Workflow when Using MicroLabBox Flexible I/O Framework

Introduction

Building FPGA applications with the DS1202 FPGA I/O Type 1 framework is substantially faster than building FPGA applications with the DS1202 FPGA I/O Type 1 (Flexible I/O) framework. The implementation of the standard I/O features, which let you use the remaining I/O channels with the RTI1202 and the EMC blocksets, takes additional FPGA resources and more build time.

The compatibility of the frameworks lets you use the DS1202 FPGA I/O Type 1 framework for modeling and testing and the DS1202 FPGA I/O Type 1 (Flexible I/O) framework for building the final FPGA application including the standard I/O features for the remaining I/O channels.

Recommended workflow

1. Model, test, and build the FPGA application with the **DS1202 FPGA I/O Type 1** framework until you have the final version.
2. Open the **FPGA SETUP BL** dialog and select the **DS1202 FPGA I/O Type 1 (Flexible I/O)** framework.

   A **Framework Incompatibilities** confirmation prompt and a **Framework Upgrade** confirmation prompt are displayed. Confirm both prompts.
3. Build the final FPGA application. Refer to How to Build FPGA Applications (MicroAutoBox II, MicroLabBox, PHS-Bus-Based System) on page 128.
4. Generate the processor interface for the processor model. Refer to How to Generate a Processor Interface on page 114.
5. Use the RTI1202 and the EMC blocksets to access the remaining I/O channels. Refer to MicroLabBox RTI Reference 🕮 and RTI Electric Motor Control Blockset Reference 🕮.
6. Build the processor application. Refer to How to Build Single-Core Processor Applications (MicroAutoBox II, MicroLabBox, PHS-Bus-Based System) on page 132.

   The build process makes sure that no I/O channel is accessed multiple times, for example, by the FPGA application and the processor application.
7. Download the application to MicroLabBox.

Modifying the FPGA I/O interface

The **DS1202 FPGA I/O Type 1 (Flexible I/O)** framework cannot display whether an I/O channel is already used by a processor application. If you modify the FPGA I/O interface, you must ensure that no I/O channel is used multiple times.

Related topics

Basics

Features of the MicroLabBox Flexible I/O Framework.. 39

Modeling the FPGA Application

Introduction

The RTI FPGA Programming Blockset is a Simulink blockset for using an FPGA model created with the Xilinx System Generator blockset on a dSPACE FPGA board. It provides the interfaces to the I/O channels and to the processor application.

Where to go from here

Information in this section

Modeling FPGA Functionality

Introduction

Aspects on modeling the FPGA functionality with the Xilinx blockset.

Where to go from here

Information in this section

Implementing the FPGA Functionality

Modeling the functionality

Note

Modeling the FPGA functionality has an significant impact on the resulting timing performance, utilization grade, and the overall FPGA system reliability. Therefore, consider the generally accepted FPGA design rules to ensure a stable and reliable FPGA application.

You model the FPGA functionality with blocks of the Xilinx blockset. However, the FPGA interface to the board-specific bus and I/O channels is provided by the RTI FPGA Programming Blockset. Do not use the Gateway In and Gateway Out blocks of the Xilinx blockset. For more information on the provided Xilinx blocks, refer to the Xilinx user documentation.

Most platforms provide multiple clock domains for modeling specific parts of your FPGA design with an individual clock period. For instructions, refer to How to Use Multiple Clock Domains for FPGA Modeling on page 48.

Reusing existing Xilinx FPGA models

You can adapt existing Xilinx models to use them on the dSPACE hardware. Refer to Adapting Xilinx FPGA Models on page 194.

Note on modeling with Xilinx Black Box blocks

If you use Xilinx Black Box blocks to incorporate hardware description language models, the files of the incorporated models must be located in special folders before you start the FPGA build process. The following table lists the folders that you can use.

Folder	Supported Files of Incorporated Models
Model root folder of the FPGA model	All files with the following file extensions are supported during the build process: `.vhdl`, `.vhd`, `.v`, `.dcp`, `.ncg`, `.ncd`, `.edit`, `.edf`, `.m`, and `.slx`. The files must not be located in a subfolder
`/Includes` subfolder in the model root folder	All file types are supported during the build process. You can also use subfolders. Before you start the build process, the `Includes` subfolder and optional subfolders must be added to the MATLAB search paths as relative paths. For example: `addpath('.\Includes')`.

The special folders are mandatory for the build process, because the FPGA Programming Blockset automatically copy all files to the build folder before the build starts. If the files are outside the given folders, they are not available for the build process.

Related topics

Basics

Notes on Using Multi-Cycle Paths

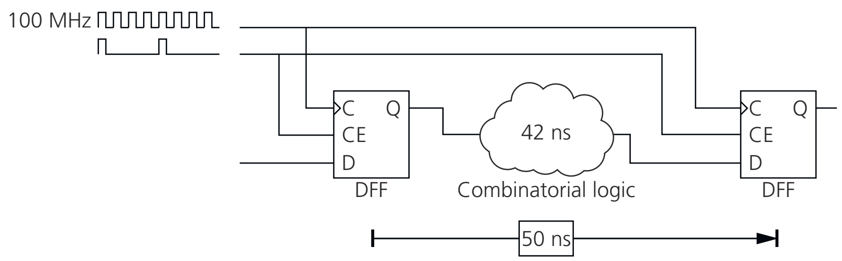
DSPs can cause timing problems in down-sampled paths

Adding a DSP with a latency greater than z^{-0} to a down-sampled path can result in a timing problem.

Note

In FPGA logic, timing is always measured between two flip-flops. This means that if a signal does not make its way through combinatorial logic (i.e., all operations with latency 0) between two flip-flops in the FPGA clock period, the timing is violated and the FPGA build is aborted.

Mechanism of down-sampled paths The following illustration shows the normal mechanism of down-sampled paths. As an example, the combinatorial logic has a latency of 42 ns. The signal path is down-sampled from 100 MHz to 20 MHz by adding a Xilinx Down Sample block at the beginning of the path and an Up Sample block at the end. The added Xilinx blocks are based on D-flip-flops.

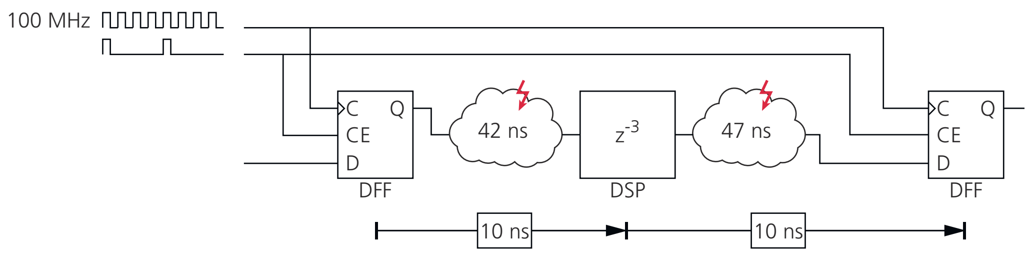


DSP problem The normal mechanism of simply increasing block latency for DSPs does not work in down-sample paths due to its mechanisms:

- An n-time down-sample path enables each element on its path using the chip enable (CE) port only at every n-th clock cycle.
- Additionally, a timing constraint is generated that allows n-times the time between two flip-flops on the down-sample path for all elements with a CE port.

DSPs do not have a CE port, so they cannot support the normal mechanism. This means that DSPs cannot transfer the timing constraint to their flip-flops.

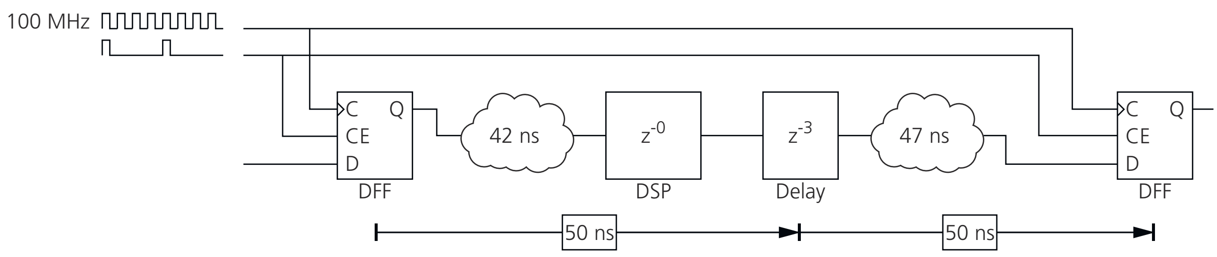
The following illustration shows the problem. In the example, a DSP is added to the down-sampled path.



Remedy

Take the flip-flops out of the DSP into the down-sample path, because the flip-flops themselves have a CE-port. This can be done by a modeling trick: Add a Xilinx **Delay** block after the DSP and Vivado automatically moves the flip-flops behind the time-critical object within the DSP. The latency of the **Delay** block must match the latency of the DSP.

The following illustration shows the modified DSP in the down-sample path. The timing constraints are now effective between flip-flops.



How to Use Multiple Clock Domains for FPGA Modeling

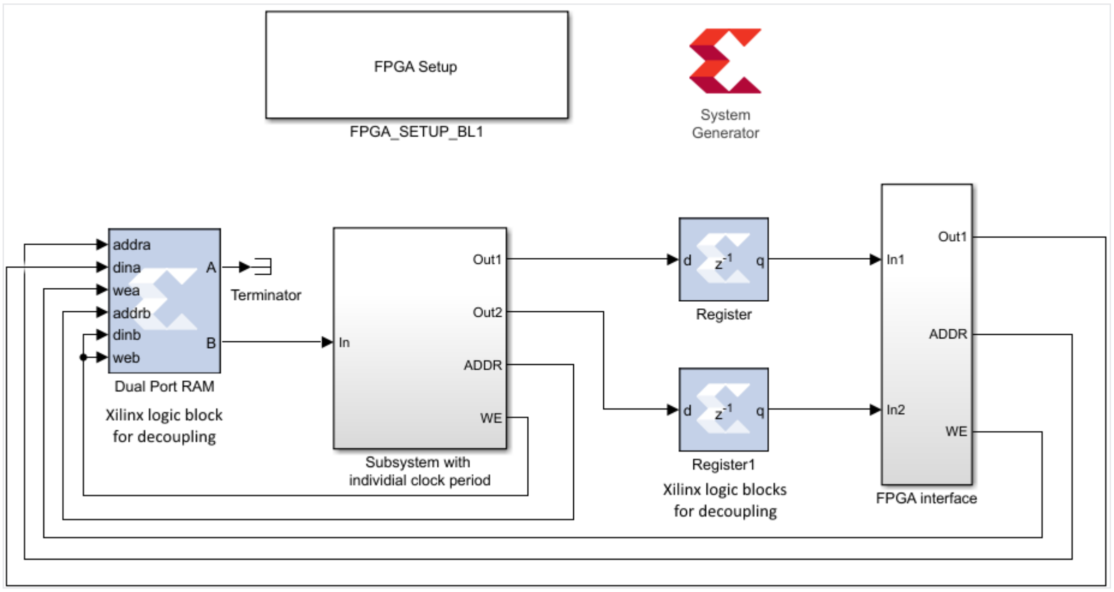
Objective

You can use multiple clock domains for modeling parts of your FPGA design with individual clock periods.

Basics

When you use multiple clock domains, the FPGA design must be partitioned into subsystems, where each subsystem has an individual clock period. Up to 10 subsystems can be used.

The subsystems are decoupled via Xilinx logic blocks. The following logic blocks can be used: FIFO block, Dual Port RAM block, Register block, or Black Box block. As a result, only the FPGA_Setup_BL block and the logic blocks for decoupling can be added to the top level of your FPGA model as shown in the following illustration.



Note

If you add a Register block to decouple the subsystems, the Xilinx System Generator generates 4 registers. The first register is driven by the clock period of the input signal. The other registers are driven by the clock period of the output signal.

Customizing the update rate of digital I/O The update rate of digital I/O functions depend on the FPGA clock period. Digital I/O functions of a SCALEXIO system or a MicroAutoBox II/III can be used with an individual clock period to customize the update rate.

A higher update rate increases the time resolution to generate or sample a digital signal. A higher update rate does not affect the minimum pulse duration or frequency of the digital channel.

If you implement a UART communication, you can adapt the update rate to the require baud rate.

Maximum update rate of digital I/O The update rate (FPGA clock frequency) can theoretically be increased up to the maximum frequency of the clock buffers (BUFG) and IO buffers (BUFIO) of the FPGA. However, this theoretical value cannot be implemented. The following table gives an indication of configurable update rates.

Platform	I/O Board/Module	Recommended Update Rate
SCALEXIO	DS2655M1	Max. 400 MHz
	DS2655M2	Input: Max. 250 MHz$^{1)}$ Output: Max. 400 MHz
	DS6651	Input: Max. 250 MHz$^{1)}$ Output: Max. 400 MHz Bidirectional: Max. 400 MHz$^{1)}$
MicroAutoBox II/III	DS1552	Max. 400 MHz
	DS1554	Max. 400 MHz

1) Maximum time resolution of the input channel is 4 ns.

Periods for offline simulation For offline simulation, you can also specify the individual offline simulation periods. For more information on the clock period range, refer to Subsystem Clocks Page (FPGA_SETUP_BL) (RTI FPGA Programming Blockset - FPGA Interface Reference 📖).

More information on using multiple clock domains For more information on the hardware design for using multiple independent clocks, refer to https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug897-vivado-sysgen-user.pdf.

Limitations

- All platforms except PHS-bus-based systems support multiple clock domains.
- Only the following I/O blocks of the RTIFPGA Programming Blockset can be used with individual clock periods:
  - DS2655M1: All digital I/O blocks, except Digital InOut
  - DS2655M2: All I/O blocks, incl. UART
  - DS6651: All digital I/O blocks, including the Trigger I/O block.
  - DS1552: All digital I/O blocks, incl. UART
  - DS1554: All digital I/O blocks

  All other blocks must be used with the base rate of the FPGA base board, such as blocks for analog I/O and processor communication.

Preconditions

The FPGA_SETUP_BL block is added to the FPGA model and a suitable framework is selected.

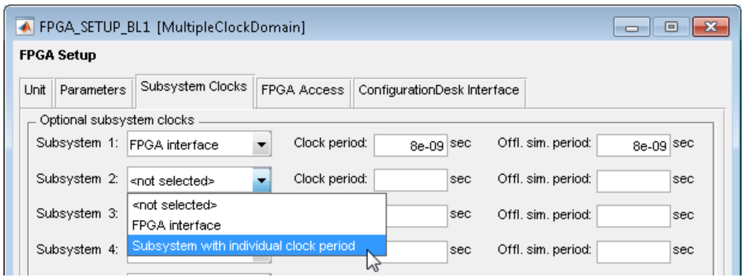
Method

To use an individual clock period for an FPGA subsystem

1. In the FPGA model, create the subsystem to be driven by individual clock periods and subsystems for the I/O functions of the RTI FPGA Programming Blockset.
2. To pass data to or from the subsystem, connect one of the following Xilinx logic blocks to the inports and outports of the subsystem:
   - FIFO block
   - Dual Port RAM block
   - Register block

     If you add a Register block to decouple subsystems with different clock periods, the Xilinx System Generator generates 4 registers for decoupling. The first register is driven by the clock period of the input signal, the other registers are driven by the clock period of the output signal.
   - Black Box block

   The logic blocks decouple the data paths with different clock periods.
3. On the top level of the FPGA model, double-click the FPGA_SETUP_BL block and open the Subsystem Clocks page.
4. On the Subsystem Clocks page, select a subsystem of the FPGA model.



FPGA_SETUP_BL1 [MultipleClockDomain]

FPGA Setup

Unit | Parameters | Subsystem Clocks | FPGA Access | ConfigurationDesk Interface

Optional subsystem clocks

Subsystem 1:	FPGA interface	Clock period:	8e-09	sec	Offl. sim. period:	8e-09 sec
Subsystem 2:	<not selected>	Clock period:		sec	Offl. sim. period:	sec
Subsystem 3:				sec	Offl. sim. period:	sec
Subsystem 4:				sec	Offl. sim. period:	sec

5. Specify the clock and simulation period for the selected subsystem:
   - Set the clock period of subsystems with I/O functions of the RTI FPGA Programming Blockset to the FPGA clock period. The FPGA clock period is displayed on the Parameters page of the FPGA_SETUP_BL block.
   - The value for the simulation period must be greater than or equal to the clock period of the selected subsystem.

   After entering the clock period, the dialog recalculates all clock periods for a clock setup with a minimum error for all clock periods. After recalculation, the dialog immediately updates the clock period's values to the calculated values in double precision.
6. Repeat steps 4 ... 5 until you specified the clock period for all subsystems.
7. Verify all recalculated clock periods.

Result

Subsystems of the FPGA model are driven with individual clock periods.

Related topics

Basics

Notes on Using Multi-Cycle Paths.. 46

References

Subsystem Clocks Page (FPGA_SETUP_BL) (RTI FPGA Programming Blockset - FPGA Interface Reference 📖)

Modeling External I/O Access

Introduction

The RTI FPGA Programming Blockset provides RTI blocks for reading from input channels and writing to output channels of the FPGA board.

Where to go from here

Information in this section

Reading From Input Channels

Introduction

The RTI FPGA Programming Blockset provides an RTI block for accessing input channels of the FPGA board.

Type and number of input channels

The FPGA_IO_READ_BL block provides read access to an input channel of the hardware used. You can select different types of input channels depending on the framework for the FPGA board or piggyback module.

The channels are represented by their functionality defined in the selected framework. For example, if you have specified the DS5203 (7K325) with onboard I/O framework, the Digital In channel list entry represents a digital input channel and the ADC channel list entry represents an analog input channel.

The number of available input channels also depends on the specified framework.

The channel list only provides input channels that are not already assigned to a block.

Note

An FPGA_IO_READ_BL block can only be connected to one input channel.

I/O mapping for the input channels

For the mapping between the input channels and the I/O connector pins, refer to the hardware-specific description in the RTI FPGA Programming Blockset - FPGA Interface Reference 📖. For example, for the I/O mapping of the input channels defined by the DS5203 (7K325) with onboard I/O framework, refer to Parameters Page (FPGA_IO_READ_BL) (RTI FPGA Programming Blockset - FPGA Interface Reference 📖).

Function-specific settings of the FPGA_IO_READ_BL block

After you have selected an input channel, the function-specific settings are loaded on the Parameters page where you can configure them. For detailed information, refer to the RTI FPGA Programming Blockset - FPGA Interface Reference 📖.

Tip

You can copy & paste previously configured interface blocks of the RTI FPGA Programming Blockset. The blockset automatically analyzes the FPGA model and reassigns new hardware resources.
You can speed up the copy & paste process for the current session by deactivating the automatic reassignment of new hardware resources. In the MATLAB Command Window, type the following two lines:

```
global G_RTIFPGA_COPYBLOCK_CALLBACK_ENABLE
G_RTIFPGA_COPYBLOCK_CALLBACK_ENABLE = false
```

Related topics

Basics

Modeling External I/O Access.. 52

Writing To Output Channels

Introduction

The RTI FPGA Programming Blockset provides an RTI block for accessing output channels of the FPGA board.

Type and number of output channels

The FPGA_IO_WRITE_BL block is providing write access to an output channel of the hardware used. Depending on the framework for the FPGA board or piggyback module, you can select different types of output channels.

The channels are represented by their functionality defined in the selected framework. For example, if you have specified the DS5203 (7K325) with onboard I/O framework, the Digital Out channel list entry represents a digital output channel and the DAC channel list entry represents an analog output channel.

The number of available output channels also depends on the specified framework.

The channel list provides only output channels that are not already assigned to a block.

Note

An FPGA_IO_WRITE_BL block can be connected to only one output channel.

I/O mapping for the output channels

For the mapping between the output channels and the I/O connector pins, refer to the hardware-specific description in the RTI FPGA Programming Blockset - FPGA Interface Reference. For example, for the I/O mapping of the output channels defined by the **DS5203 (7K325) with onboard I/O** framework, refer to Parameters Page (FPGA_IO_WRITE_BL) (RTI FPGA Programming Blockset - FPGA Interface Reference).

Function-specific settings of the FPGA_IO_WRITE_BL block

After you have selected an output channel, the function-specific settings are loaded to the Parameters page, where you can configure them. For detailed information, refer to the RTI FPGA Programming Blockset - FPGA Interface Reference.

Tip

You can copy & paste previously configured interface blocks of the RTI FPGA Programming Blockset. The blockset automatically analyzes the FPGA model and reassigns new hardware resources.
You can speed up the copy & paste process for the current session by deactivating the automatic reassignment of new hardware resources. In the MATLAB Command Window, type the following two lines:

```
global G_RTIFPGA_COPYBLOCK_CALLBACK_ENABLE
G_RTIFPGA_COPYBLOCK_CALLBACK_ENABLE = false
```

Related topics

Basics

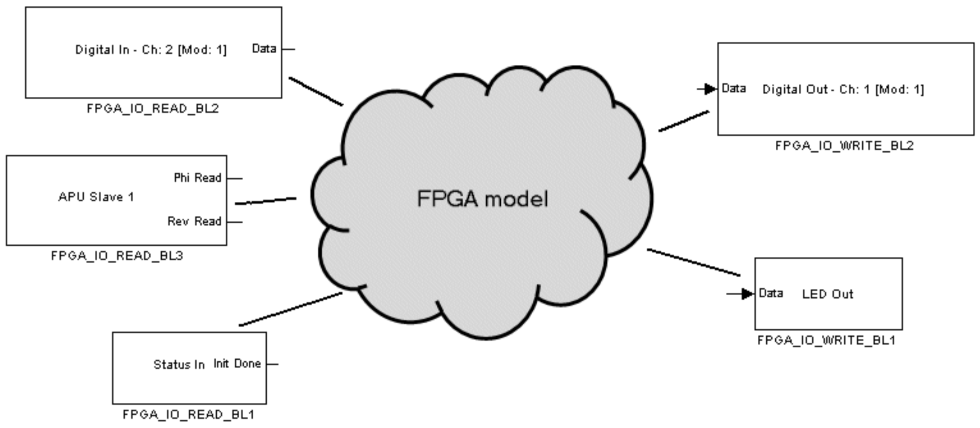
Effects of Modeling External I/O Access

Introduction

Read and write accesses to external devices depend on the platform.

Demo model

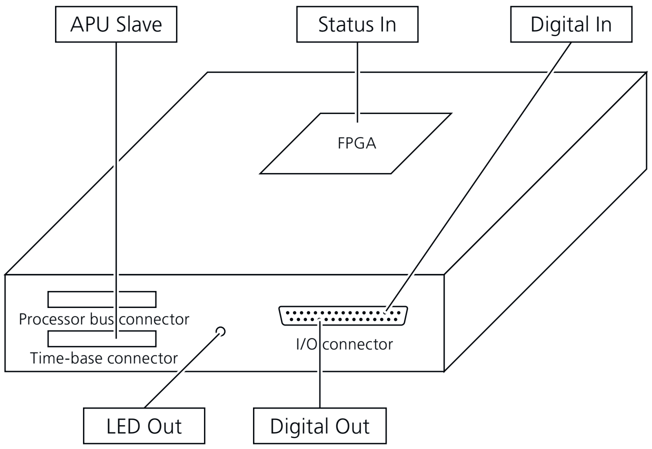
The demo model below contains some FPGA_IO_READ_BL and FPGA_IO_WRITE_BL blocks configured with different I/O functions to show how they will be connected to the hardware.



Effects in a MicroAutoBox II, MicroLabBox, or PHS-bus-based system

The FPGA_IO_READ_BL and FPGA_IO_WRITE_BL blocks provide different types of I/O functions for the MicroAutoBox II, MicroLabBox, and PHS-bus-based system:

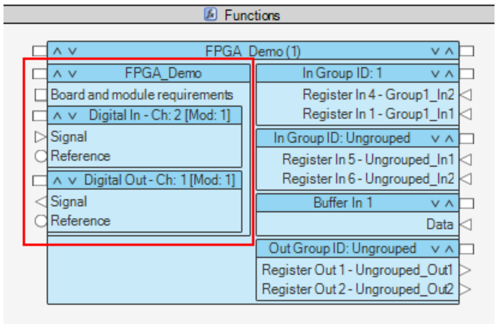
- I/O functions without external connection, for example, the **Status In** function, can only be handled within the FPGA subsystem.
- I/O functions with external connection provide their signals via the board's I/O connector.



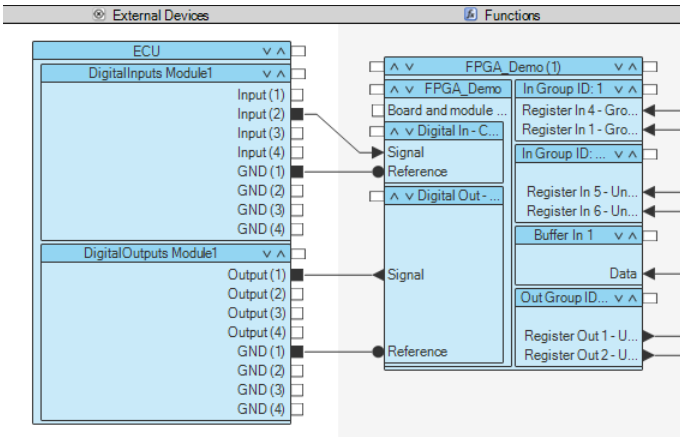
Effects in a MicroAutoBox III or SCALEXIO system

The FPGA_IO_READ_BL and FPGA_IO_WRITE_BL blocks provide different types of I/O functions for the MicroAutoBox III or SCALEXIO systems:

- I/O functions without external connection, for example, Status In, LED Out or APU Slave, are not represented in the FPGA custom function in ConfigurationDesk and can only be handled within the FPGA subsystem.
- I/O functions with external connection provide their signals as signal ports in ConfigurationDesk.


Functions
FPGA_Demo (1)
FPGA_Demo
Board and module requirements
Digital In - Ch: 2 [Mod: 1]
Signal
Reference
Digital Out - Ch: 1 [Mod: 1]
Signal
Reference
In Group ID: 1
Register In 4 - Group1_In2
Register In 1 - Group1_In1
In Group ID: Ungrouped
Register In 5 - Ungrouped_In1
Register In 6 - Ungrouped_In2
Buffer In 1
Data
Out Group ID: Ungrouped
Register Out 1 - Ungrouped_Out1
Register Out 2 - Ungrouped_Out2

The signal ports of the I/O functions have to be connected to external devices. For further information on hardware resource assignment, refer to Assigning Hardware Resources to Function Blocks (ConfigurationDesk Real-Time Implementation Guide 🕮).



Related topics

Basics

Modeling Processor Model Access

Introduction

You have to implement the interface to the board-specific bus to exchange data with the processor application and to trigger interrupts.

Where to go from here

Information in this section

Exchanging Data With the Processor Model

Introduction

The RTI FPGA Programming Blockset provides blocks for exchanging data between the FPGA subsystem and the processor model.

Basics on the processor communication using MicroLabBox

The data is exchanged via the communication line between MicroLabBox's base board and the I/O FPGA module mounted on the DS1302 board. For MicroLabBox, communication runs via the local bus.

The 32-bit parallel data bus enables a complete 32-bit word to be transferred in a single operation. For transferring a 64-bit word, two operations are internally required.

The data is transmitted via the local bus to specific data storage areas. The implementation details of these storage areas are defined in the framework.

Data Storage Area	Data Width	Data Storage Size	Data Format
Register	32 bit	256 32-bit registers and 256 64-bit registers	Can be specified to fixed-point or floating-point format. ▪ Fixed-point format: Signed or unsigned data format with adjustable binary point position.
	64 bit		
Buffer	32 bit	32 32-bit buffers and 32 64-bit buffers Each buffer with up to 32,768 elements	
	64 bit		

Data Storage Area	Data Width	Data Storage Size	Data Format
			All 64-bit fixed-point data types are converted to double. Therefore, the fixed-point resolution of fixed-point data types is restricted to 53 bits. ▪ 32-bit floating-point format: Single precision (IEEE 754 standard) data format with a fraction width of 24. ▪ 64-bit floating-point format: Double precision (IEEE 754 standard) data format with a fraction width of 53.

Basics on the processor communication using MicroAutoBox II/III

The data is exchanged via the communication line between the real-time processor of the MicroAutoBox II/III and the FPGA module mounted on the DS1514 I/O board. The communication runs via the intermodule bus of the MicroAutoBox II/IIII.

The 16-bit parallel data bus enables a complete 16-bit word to be transferred in a single operation.

The data is transmitted via the intermodule bus to specific data storage areas. The implementation details of these storage areas are defined in the framework.

Data Storage Area	Data Width	Data Storage Size	Data Format
Register	32 bit	128 32-bit registers and 128 64-bit registers	Can be specified to fixed-point or floating-point format. ▪ Fixed-point format: Signed or unsigned data format with adjustable binary point position All 64-bit fixed-point data types are converted to double. Therefore, the fixed-point resolution of fixed-point data types is restricted to 53 bits. ▪ 32-bit floating-point format: Single precision (IEEE 754 standard) data format with a fraction width of 24 ▪ 64-bit floating-point format: Double precision (IEEE 754 standard) data format with a fraction width of 53
	64 bit		
Buffer	32 bit	32 32-bit buffers and 32 64-bit buffers Each buffer with up to 32,768 elements	
	64 bit		

Basics on the processor communication using a SCALEXIO system

The internal communication between the real-time processor and a SCALEXIO FPGA base board runs via IOCNET (I/O carrier network). IOCNET lets you connect more than 100 I/O nodes and even place the parts of your SCALEXIO system long distances apart. IOCNET is dSPACE's proprietary protocol that gives you real-time performance plus time and angular clocks. You can also use IOCNET ports to connect to PHS-bus based simulator systems via Gigalink.

The data is transmitted via IOCNET to specific data storage areas. The implementation details of these storage areas are defined in the framework.

Data Storage Area	Data Width	Data Storage Size	Data Format
Register	32 bit	256 32-bit registers and 256 64-bit registers	Can be specified to fixed-point or floating-point format. ▪ Fixed-point format: Signed or unsigned data format with adjustable binary point position. All 64-bit fixed-point data types are converted to double. Therefore, the fixed-point resolution of fixed-point data types is restricted to 53 bits. ▪ 32-bit floating-point format: Single precision (IEEE 754 standard) data format with a fraction width of 24. ▪ 64-bit floating-point format: Double precision (IEEE 754 standard) data format with a fraction width of 53.
	64 bit		
Buffer	32 bit	32 32-bit buffers and 32 64-bit buffers Each buffer with up to 32,768 elements	
	64 bit		

Register access type If you have to exchange certain data and high performance is not required, it is recommended to use the register access type.

Note

Any specified register that does not belong to a group is automatically collected in one single group called *Ungrouped* in the FPGA custom function built for ConfigurationDesk. There is one group for register input data and one group for register output data. Independently of the groups specified in your FPGA model, the ungrouped registers are automatically grouped in the Simulink model tasks in ConfigurationDesk to increase the performance of IOCNET transfers.

Note

You can access a register group by one task only.

Buffer access type If you have to exchange a lot of data with high performance, it is recommended to use the buffer access type. However, to access a specific data item within the buffer, you have to address it explicitly.

Basics on the processor communication using a PHS-bus-based system

The data is exchanged via the communication line between the processor board and the FPGA board. Communication in a PHS-bus-based system runs via the PHS bus. The 32-bit parallel data bus enables a complete 32-bit word to be transferred in a single operation. You can connect up to 16 I/O boards to the PHS bus.

The data is transmitted via the PHS bus to specific data storage areas. The implementation details of these storage areas are defined in the framework.

Data Storage Area	Data Width	Data Storage Size	Data Format
Register	32 bit	128 32-bit registers and 128 64-bit registers	Can be specified to fixed-point or floating-point format. ▪ Fixed-point format: Signed or unsigned data format with adjustable binary point position All 64-bit fixed-point data types are converted to double. Therefore, the fixed-point resolution of fixed-point data types is restricted to 53 bits. ▪ 32-bit floating-point format: Single precision (IEEE 754 standard) data format with a fraction width of 24 ▪ 64-bit floating-point format: Double precision (IEEE 754 standard) data format with a fraction width of 53
	64 bit		
Buffer	32 bit	32 32-bit buffers and 32 64-bit buffers Each buffer with up to 32,768 elements	
	64 bit		

Read/write blocks in the FPGA subsystem

You can use the FPGA_XDATA_READ_BL block from the FPGA Interface library to read data from the processor bus, and the FPGA_XDATA_WRITE_BL block to write data to the board-specific bus.

The values in the FPGA subsystem are processed as fixed-point values. The data exchange with the processor model requires data type conversion between fixed-point and floating-point values. Automatic data conversion is implemented in these blocks. The fixed-point format to be used can be set in the blocks' dialogs.

Tip

You can copy & paste previously configured interface blocks of the RTI FPGA Programming Blockset. The blockset automatically analyzes the FPGA model and reassigns new hardware resources.
You can speed up the copy & paste process for the current session by deactivating the automatic reassignment of new hardware resources. In the MATLAB Command Window, type the following two lines:

```
global G_RTIFPGA_COPYBLOCK_CALLBACK_ENABLE
G_RTIFPGA_COPYBLOCK_CALLBACK_ENABLE = false
```

Read/write blocks in the processor model

The Processor Interface library of the RTI FPGA Programming Blockset provides the PROC_XDATA_READ_BL and PROC_XDATA_WRITE_BL blocks to read data from and write data to the board-specific bus. These blocks must be configured as counterparts to the related blocks in the FPGA subsystem. The simplest way to create and configure these blocks in the processor model is to use the **`Generate`** command.

For instructions, refer to How to Generate a Processor Interface on page 114.

Related topics

Basics

How to Trigger Interrupt-Driven Processor Tasks

Objective

You can use an output signal from the FPGA application to trigger an asynchronous task in the processor model.

Basics

As with the other interface blocks from the RTI FPGA Programming Blockset, you need an interrupt block in the FPGA subsystem and its counterpart in the processor model to trigger a function-call subsystem by a value coming from the FPGA.

Method

To trigger an interrupt-driven processor task

1 Add an FPGA_INT_BL block to the FPGA subsystem.
2 Double-click the block to open its dialog.
3 Specify a channel number and enter a descriptive channel name.
4 Connect the Int inport with the signal that you want to use for triggering a task in the processor model.

Result

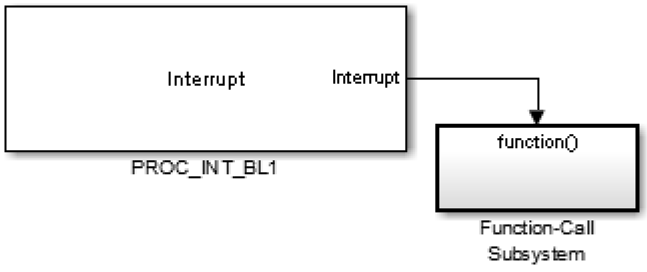
When the FPGA_INT_BL block in the FPGA subsystem receives the specified data, it is transferred to the processor model.

Next steps

After you implemented the processor interface, you connect the processor interface block to a function-call subsystem in the processor model. For instructions on implementing the processor interface, refer to Implementing the Processor Interface to the Processor Model on page 113.

Examples

The following example shows the resulting processor model implementation of an interrupt-driven task.



Related topics

Basics

Modeling Processor Model Access.. 57

Using Simulink Buses for Modeling the Processor Communication

Introduction

The SCALEXIO and the MicroAutoBox III frameworks support Simulink buses to implement the communication between the real-time processor and the FPGA application.

Where to go from here

Information in this section

How to Use Simulink Buses for Modeling the Processor Communication

Objective

Modeling the processor interface with Simulink buses.

Platforms supporting data exchange via Simulink buses

The following platforms support the use of Simulink buses for modeling the data exchange with the processor interface:

- The MicroAutoBox III
- SCALEXIO

Methods

Depending on the data direction, refer to one of the following methods:

- To write data to the processor model, refer to Method 1.
- To read data from the processor model, refer to Method 2.

Method 1

To use a Simulink bus to write data values to the processor application

1. Add a Buffer64 Out block to the FPGA model.
2. Open the Parameters page of the block dialog and select Enable bus transfer mode.
3. Connect the Simulink bus to the Data inport.
4. Press `Ctrl` + `D` to update the model.
5. On the Parameters page, click Analyze bus topology of input.

   The RTI FPGA Programming Blockset analyzes the connected Simulink bus and configures the Data inport.

Method 2

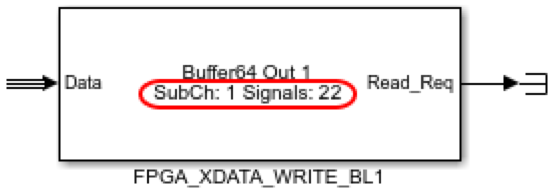
To use a Simulink bus to read data values from the processor application

1. Add a Buffer64 In block to the FPGA model.
2. Open the Parameters page of the block dialog and select Enable bus transfer mode.
3. If the Simulink bus is not already implemented in the FPGA model, implement a temporary Simulink bus:
   - Add a Simulink Bus Creator block to the FPGA model.
   - Connect FPGA blocks to the input with matching data types, for example, Xilinx Counter blocks.
4. Press `Ctrl` + `D` to update the model.
5. Select the Simulink bus to be copied by selecting a Bus Creator block, subsystem inport block, or subsystem outport block.
6. On the Parameters page, click Copy bus topology from gcb.

   The RTI FPGA Programming Blockset analyzes the selected Simulink bus and configures the Data outport.
7. You can delete the temporary Simulink bus from step 3.

Result

You modeled the processor communication with Simulink buses. The buffer block displays the used subchannel and the number of signals that must be connected to the Data port. The following example shows a Buffer64 Out block.



Related topics

Basics

Exchanging Data With the Processor Model.. 57
Notes on Handling Simulink Buses.. 67

References

AnalyzeFPGAXDATAWriteBus (RTI FPGA Programming Blockset Script Interface Reference 🕮)
CopyFPGAXDATAReadBus (RTI FPGA Programming Blockset Script Interface Reference 🕮)
GetFPGAXDATABusSettings (RTI FPGA Programming Blockset Script Interface Reference 🕮)
ResetFPGAXDATABus (RTI FPGA Programming Blockset Script Interface Reference 🕮)
SetFPGAXDATABusSettings (RTI FPGA Programming Blockset Script Interface Reference 🕮)

How to Configure the Bus Data Transmission Method

Objective

Configuring the method for writing data to the processor application when using the bus transfer mode.

Basics

To transmit data from the FPGA to the processor, the processor application makes a read request. The currentness of the transmitted data depends on the selected bus data transmission method:

- Synchronous to Read_Req method

  Select this method to transmit data that is captured synchronously to the processor task.

  The FPGA application writes data to the swinging buffer when the processor application makes a read request. After the data is written to the buffer, the buffer swings and sends the data to the processor application.
- Free running method

  Select this method if the transmission time is crucial.

  The FPGA application continuously writes data to the swinging buffer. A read request of the processor application immediately transmits the last complete data set of the swinging buffer to the processor application.

For more information on the swinging buffer, refer to Basics on Exchanging Data Between Processor and FPGA on page 17.

Limitation

The selected bus data transmission method applies to all subchannels of a **Buffer64 Out** channel. To use different bus data transmission methods, you must use different **Buffer64 Out** channels.

For more information on subchannels, refer to Using Subchannels for Data Exchange on page 66.

Method

To configure the bus data transmission method

1 Select the **Buffer64 Out** block of subchannel 1.

Subchannel 1 is the default subchannel.

2 Open the **Parameters** page of the block dialog and select a bus data transmission method.

3 Click **Apply** or **OK**.

Result

You configured the bus data transmission method. If you use subchannels, the selected method applies to all subchannels of the selected **Buffer64 Out** channel.

Related topics

Basics

HowTos

Using Subchannels for Data Exchange

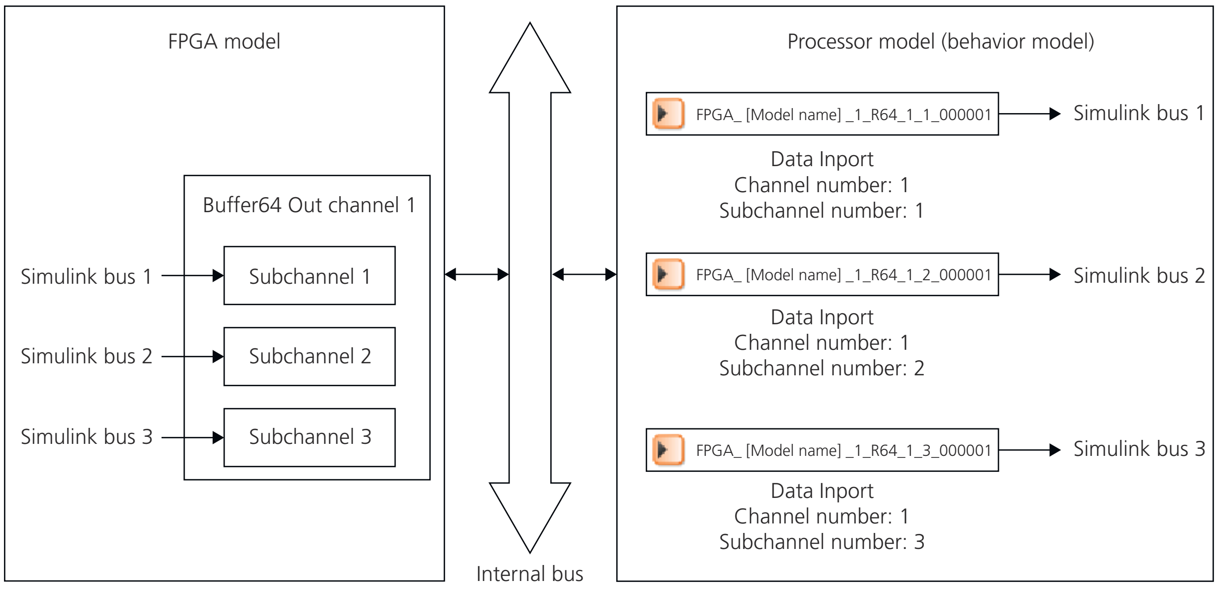
Introduction

The SCALEXIO and the MicroAutoBox III frameworks support subchannels to implement the communication between a task of the processor application and the FPGA application.

With subchannels, you can use a single buffer to exchange data via several Simulink buses. This gives you the flexibility of registers with the performance of buffers with the support of Simulink buses.

Implementing subchannels

After you enabled the bus transfer mode for a **Buffer64 Out/Buffer64 In** function, you can set the buffer to a channel that is in use and select a different subchannel. You have to add a **Buffer64 Out/Buffer64 In** function for each subchannel.



Observe the following requirements if you use subchannels:

- Up to 256 subchannels can be used for each channel. Subchannel number 1 of a buffer channel must always be used, the other subchannels can be used in any order.
- In the processor model, all subchannels of the same buffer channel must be implemented to the same application process.

For using the bus transfer mode, refer to How to Use Simulink Buses for Modeling the Processor Communication on page 63.

Related topics

HowTos

How to Use Simulink Buses for Modeling the Processor Communication................................. 63

Notes on Handling Simulink Buses

Using Simulink data dictionaries

The RTI FPGA Programming Blockset can use a data dictionary instead using the base workspace for the bus elements.

Using models that are linked to data dictionaries If the model is already linked to a data dictionary, the FPGA Programming Blockset automatically uses this data dictionary for the bus elements.

Migrating to data dictionaries To use a data dictionary instead the base workspace, create a data dictionary, link it to the model, and import the bus elements from the base workspace into this data dictionary.

Alternatively, you can create a data dictionary, link it to the model, close the model, and reopen it. The RTI FPGA Programming Blockset recreates the Simulink bus objects each time you open the model and automatically saves the bus elements in the linked data dictionary instead in the base workspace.

More information on data dictionaries For more information on Simulink data dictionaries, refer to the Simulink help.

Observing bus elements

The Simulink bus editor lets you observe Simulink bus elements.

To use the bus editor, open the Simulink **Model Explorer**, select a bus element, and click **Launch Bus Editor**.

For more information on the bus editor, refer to the Simulink help.

Related topics

HowTos

How to Use Simulink Buses for Modeling the Processor Communication................................ 63

Modeling DDR4 RAM Access

Introduction

The RTI FPGA Programming Blockset provides RTI blocks to implement a communication interface for the DDR4 RAM of the DS6602.

Where to go from here

Information in this section

Accessing the DDR4 RAM of the DS6602

Introduction

The DS6602 FPGA Base Board provides a 4 GB DDR4 RAM that can be used by the FPGA application.

The RAM interface always handles 512 bits at once. Therefore, the FPGA application can read/write 16 x 32 bits data or 8 x 64 bits data within one memory access.

You can select different I/O types to access the DDR4 RAM:

- **DDR4 32 Mode 1** and **DDR4 64 Mode 1** to read/write 32/64-bit values. These I/O types use the memory access mode 1.
- **DDR4 32 Mode 2** and **DDR4 64 Mode 2** to read/write 32/64-bit values. These I/O types use the memory access mode 2.

Tip

The DDR4 RAM processes the data values as raw data. To use certain data types, such as single, you have to add Xilinx **Reinterpret** blocks to the FPGA model.

Memory access modes

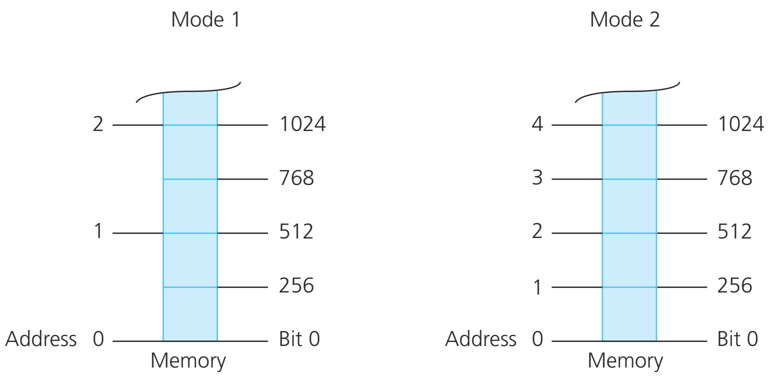
The RTI FPGA Programming Blockset provides two memory access modes:

- Mode 1

  Mode 1 addresses one memory area of 512 bits, each with read/write access.
- Mode 2

  Mode 2 addresses two memory areas of 256 bits, each with read/write access.

The following example shows the memory areas addressed by the different access modes.



Note

The memory areas are addressed differently by the different access modes.

Optimizing read/write access

To decrease the read/write time, use consecutive addresses for read/write accesses. A random address access approximately triples the read/write time:

Read access	Linear addressing	7.37 GB/s
	Random addressing	2.58 GB/s
Write access	Linear addressing	6.77 GB/s
	Random addressing	2.29 GB/s

Related topics

Basics

Exchanging Data With the Processor Model... 57

References

Parameters Page (FPGA_IO_WRITE_BL) (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮)

Initializing the DDR4 RAM of the DS6602 for Offline Simulation

Introduction

You can provide a workspace variable with initial values for the RAM memory to simulate the FPGA application with defined DDR4 RAM data values.

Generating an initialization variable

The initialization variable is a MATLAB workspace variable. You can use one of the following MATLAB scripts to generate an initialization variable.

32-bit values The following examples generate an initialization variable for 32-bit values (DDR4 32 Mode 1 and DDR4 32 Mode 2).

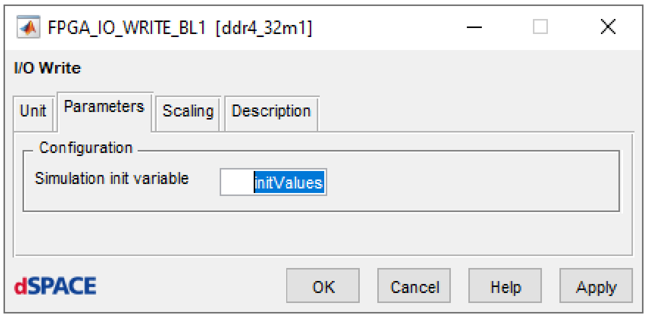
```
% generate sample 32 bit values (4GB - 1 Word) 1..2^30-1 (2^30*4 Byte = 4 GB)
% For initialization with datatype double values, use:   single(<values>)
% For initialization with datatype uint32/UFix_32_0, use: uint32(<values>)

% Use less than 2^30 values for faster experimenting.

% Small example for datatype single:
>> initValues = single((1:10)/3.1)
initValues =
  1×10 single row vector
    0.3226    0.6452    0.9677    1.2903    1.6129    1.9355    2.2581    2.5806    2.9032    3.2258

% Small example for datatype uint32/UFix_32_0:
>> initValues = uint32((1:10)/3.1)
initValues =
  1×10 uint32 row vector
   0   1   1   1   2   2   2   3   3   3
```

64-bit values The following examples generate an initialization variable for 64-bit values (DDR4 64 Mode 1 and DDR4 64 Mode 2).

```
% generate sample 64 bit values (4GB - 1 Word) 1..2^29-1 (2^29*8 Byte = 4 GB)
% For initialization with datatype double values, use:   double(<values>)
% For initialization with datatype uint64/UFix64_0, use: fi(<values>, 0, 64, 0)

% Use less than 2^29 values for faster experimenting.

% Small example for datatype double:
>> initValues = double((1:10)/3.1)
initValues =
    0.3226    0.6452    0.9677    1.2903    1.6129    1.9355    2.2581    2.5806    2.9032    3.2258

% Small example for datatype uint64/UFix_64_0:
>> initValues = fi((1:10)/3.1, 0, 64, 0)
initValues =
     0     1     1     1     2     2     2     3     3     3
          DataTypeMode: Fixed-point: binary point scaling
            Signedness: Unsigned
            WordLength: 64
        FractionLength: 0
```

Initializing the RAM with the variable

To initialize the RAM at the beginning of the offline simulation, you have to enter the variable name to the Simulation init variable parameter on the Parameters page of the used DDR4 block.



Configuration

Simulation init variable: initValues

Related topics

References

Parameters Page (FPGA_IO_WRITE_BL) (RTI FPGA Programming Blockset - FPGA Interface Reference 📖)

Initializing the DDR4 RAM of the DS6602 for Real-Time Simulation

Introduction

You can provide a file with initial values for the RAM memory to start the FPGA application with defined DDR4 RAM data values. The processing hardware writes the initial values to the FPGA base board during the initialization phase of the SCALEXIO system.

Preconditions

The following preconditions must be fulfilled:

- One of the following SCALEXIO Processing Unit variants must be installed:
  - A SCALEXIO Processing Unit of the HCP product line with a Real-Time PC Version 2.0 or later.
  - A SCALEXIO Processing Unit of the HPP product line.

  For more information on the SCALEXIO Processing Unit variants, refer to Variants of the SCALEXIO Processing Unit (SCALEXIO Hardware Installation and Configuration 📖).
- A SCALEXIO SSD must be installed to save the initialization file.

Generating an initialization file

The initialization file is a binary file. You can use the following MATLAB script to generate an initialization file with MATLAB.

```
% generate sample 32 bit values (4GB - 1 Word) 1..2^30-1 (2^30*4 Byte = 4 GB)
% generate sample 64 bit values (4GB - 1 Word) 1..2^29-1 (2^29*8 Byte = 4 GB)
ddr4_4gb = 1:2^30-1;
% open, write, close file
fid = fopen('ddr4_4gb.bin', 'w');
fwrite(fid, ddr4_4gb, 'uint32');
```

```
% or as different data type
% fwrite(fid, ddr4_4gb, 'single');
% fwrite(fid, ddr4_4gb, 'double');
% 64 bit -> use only half as many elements
…
fclose(fid)
```

Note

You can use any data type to initialize the DDR4 RAM.

Providing the initialization file to the SCALEXIO system

For information on providing the initialization file, refer to How to Initialize the DDR4 RAM of the DS6602 (ConfigurationDesk I/O Function Implementation Guide 🕮).

Related topics

HowTos

How to Initialize the DDR4 RAM of the DS6602 (ConfigurationDesk I/O Function Implementation Guide 🕮)

Modeling UART Communication

Introduction

The RTI FPGA Programming Blockset provides RTI blocks to implement an UART communication interface. A UART demo project for SCALEXIO systems provides a model block that simplifies the implementation of UART interfaces.

Where to go from here

Information in this section

Information in other sections

UART interface blocks for MicroLabBox:

Parameters Page (FPGA_IO_WRITE_BL) (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮)
To specify relevant settings for the selected I/O function.

UART interface blocks for SCALEXIO (DS2655M2 Digital I/O Module):

Parameters Page (FPGA_IO_READ_BL) (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮)
To specify relevant settings for the selected I/O function.

Parameters Page (FPGA_IO_WRITE_BL) (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮)
To specify relevant settings for the selected I/O function.

UART interface blocks for the MicroAutoBox II (DS1552 Multi-I/O Module):

Parameters Page (FPGA_IO_WRITE_BL) (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮)
To specify relevant settings for the selected I/O function.

UART interface blocks for the MicroAutoBox III (DS1552 Multi-I/O Module):

Parameters Page (FPGA_IO_WRITE_BL) (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮)
To specify relevant settings for the selected I/O function.

Using the UART Demo Model for SCALEXIO Systems

Supported platforms

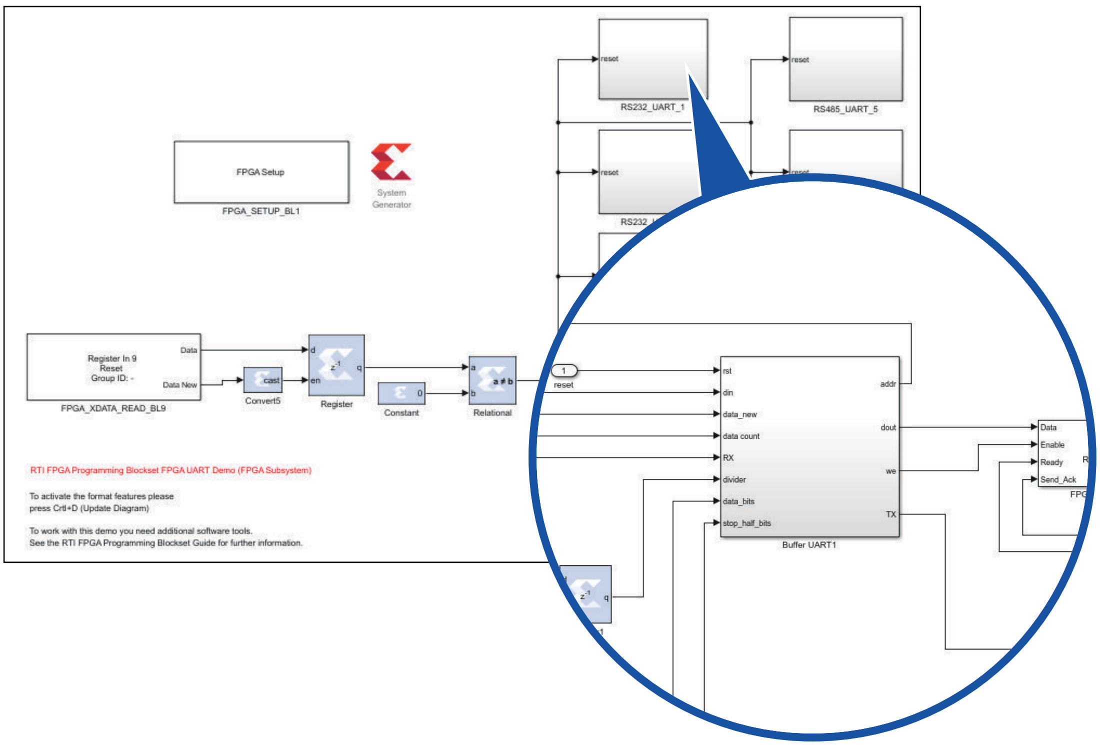
The demo model supports only SCALEXIO systems.

Accessing the demo

Open the block library and select Demos – DemoFPGAuart – DS2655_XC7K160T.

Demo overview

The following illustration shows the FPGA model with the subsystem of a RS232_UART model block.



RTI FPGA Programming Blockset FPGA UART Demo (FPGA Subsystem)

To activate the format features please press Crtl+D (Update Diagram)

To work with this demo you need additional software tools. See the RTI FPGA Programming Blockset Guide for further information.

The demo model provides eight UART communication interfaces: Four RS232 interfaces and four RS485 interfaces. The main component for all interfaces is the Buffer UART model block that resides in every UART subsystem.

Using the demo

You can use the Buffer UART model block in your FPGA application to implement a UART communication interface in your project.

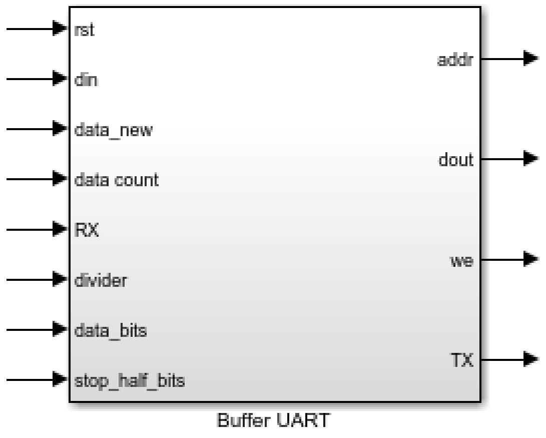
The Buffer UART model block provides the following features:

- Configures the UART frames.
- Configures the baud rate.
- Controls the data direction for RS485 communication.

- Controls the sequential read of a **Buffer In** block to send the elements of the buffer with UART frames.
- Controls the sequential write to a **Buffer Out** block to send received UART frames to the processor application.

Description of the Buffer UART model block

Appearance of the block:



I/O characteristics The following table describes the ports of the block:

Port	Description
Input	
rst	Resets the model block. Value range: ▪ 0: No reset ▪ 1: Reset
din	Receives new data value to be sent with an UART frame. Data type: UFix_32_0 Value range: Depends on the specified number of data bits of the UART frame.
data_new	Indicates a change in the buffer status of a connected **Buffer In** interface block. Data type: UFix_1_0
data count	Specifies the current number of elements in the buffer of a connected **Buffer In** interface block. Data type: UFix_16_0
RX	Receives the UART frame as a sequence of bits. Data type: UFix_1_0
divider	Specifies the clock divider value to set the baud rate of the UART interface. You can calculate the value for the clock divider to set the baud rate as follows: $\mathit{Clock\ divider} = \frac{1}{\mathit{Baud\ rate} \cdot 32\ \mathit{ns}} - 1$ For values to set common baud rates, refer to Settings for common baud rates on page 77. Data type: Fix_32_0 The value range depends on the maximum baud rate of the UART interface hardware: ▪ RS232: 125 ... n_{max} ▪ RS485: 1 ... n_{max}

Port	Description
data_bits	Specifies the number of data bits. The data bits include optional parity bits and exclude the start bit of the UART frame. Data type: Fix_32_0 Value range: 5 ... 9
stop_half_bits	Specifies the number of stop half-bits of the UART frame. The number value specifies the number of half bits. For example: 2 specifies 1 stop bit in the frame. Data type: Fix_32_0 Value range: 1 ... 5
Output	
addr	Outputs the buffer address of the next element in a **Buffer In** interface block to be sent with the next frame. Data type: UFix_16_0
dout	Outputs the data value of the received UART frame at the **RX** port. Data type: UFix_32_0 Value range: Depends on the specified number of data bits of the UART frame.
we	Specifies that the current value of the **dout** port is valid to control a connected **Buffer Out** interface block. Data type: UFix_1_0
TX	Outputs the UART frame as a sequence of bits. Data type: UFix_1_0

Settings for common baud rates

The following table gives you clock divider values for the **divider** port to set the UART interface to a common baud rate:

Desired Baud Rate	Clock Divider Value
9600	3254
19200	1627
38400	813
57600	542
115200	270
1000000	30

Modeling Inter-FPGA Communication

Introduction

An inter-FPGA communication bus lets you exchange data directly between FPGA boards.

Where to go from here

Information in this section

Introduction to Inter-FPGA Communication

Overview of Inter-FPGA Communication

Platforms supporting inter-FPGA communication

The following platforms support inter-FPGA communication:

- SCALEXIO systems with SCALEXIO FPGA base boards DS2655, DS6601, or DS6602.

  SCALEXIO provides different types of inter-FPGA communication. Refer to Overview of inter-FPGA communication between SCALEXIO boards on page 79.
- PHS-bus-based systems with DS5203 FPGA Boards.

  The DS5203 FPGA Board provides inter-FPGA connectors on the board to establish an inter-FPGA communication bus. For more details, refer to Board Overview (PHS Bus System Hardware Reference 📖).

Overview of inter-FPGA communication between SCALEXIO boards

The following table shows the different types of inter-FPGA communication between SCALEXIO FPGA base boards.

	Inter-FPGA via I/O Module Slots	**Inter-FPGA via MGT Module**	**Inter-FPGA via IOCNET**
Topology	1:1	1:1	1:n
FPGA stack[1] location	Next to each other	No restriction	▪ Within the same IOCNET segment[2] ▪ Other I/O boards must not be connected to this IOCNET segment.
Number of direct connections	2	4	Not directly limited
Data rate	Max. 582.3 Mbit/s with default values	Max. 10.3125 Gbit/s with **Aurora 64b66b 128 Bit** block	▪ 1.25 Gbit/s IOCNET: Approx. 800 Mbit/s ▪ 2.5 Gbit/s IOCNET: Typ. 1 Gbit/s, max. 1.6 Gbit/s[3]
Latency	72 ns … 96 ns with default values	Typ. 384 ns, max. 472 ns for single words	Typ. 1.0 µs per network hop, including onboard hops
Supported FPGA base boards	▪ DS2655 ▪ DS6601 ▪ DS6602	▪ DS6601 ▪ DS6602	▪ DS2655 ▪ DS6601 ▪ DS6602
Required accessory	SCLX_INT_FPGA_CAB1	DS6601_MGT1 or DS6602_MGT1	–

[1] FPGA base board with installed I/O modules.
[2] Refer to Implementing Inter-FPGA Communication via IOCNET on page 80.
[3] Theoretical maximum

Implementing the inter-FPGA interface

To implement an inter-FPGA communication bus, you have to implement the functionality to the FPGA model.

Depending on the used hardware, refer to one of the following topics:

- Implementing Inter-FPGA Communication via IOCNET on page 80
- Implementing Inter-FPGA Communication via MGT Modules on page 83
- Implementing Inter-FPGA Communication via I/O Module Slots on page 85
- Implementing Inter-FPGA Communication Between DS5203 FPGA Boards on page 96

Modeling Inter-FPGA Communication via IOCNET

Where to go from here

Information in this section

Implementing Inter-FPGA Communication via IOCNET

Introduction

IOCNET can be used to transfer data directly between FPGA boards of a SCALEXIO system.

Basic interface characteristics

You have to consider the following interface characteristics if you implement inter-FPGA communication via IOCNET.

- Up to 32 channels to transmit 32-bit values and up to 32 channels to transmit 64-bit values can be implemented.
- The data rate depends on the IOCNET data rate:
  - 1.25 Gbit/s IOCNET:

    About 800 Mbit/s
  - 2.5 Gbit/s IOCNET:

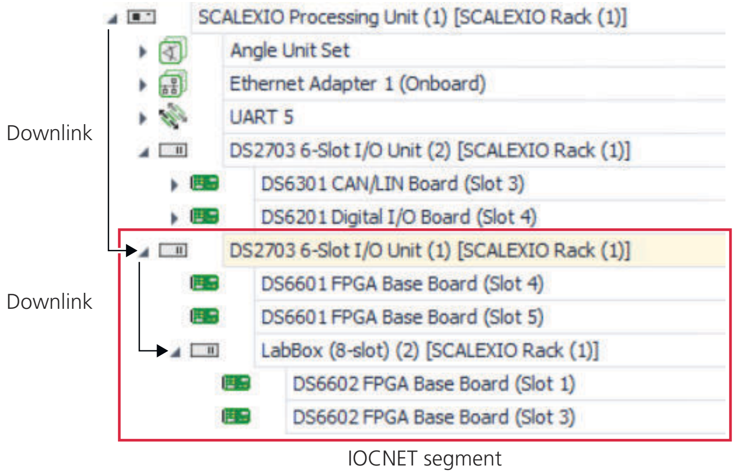
    Typ. 1 Gbit/s, max. 1.6 Gbit/s theoretical
- The latency is typ. 1 µs per network hop. Refer to Calculating the Latency (IOCNET) on page 82.
- Each 32-bit channel can address up to 1024 data values.
- Each 64-bit channel can address up to 512 data values.
- The data type is a raw data type: UFix_32_0 or UFix_64_0.

  You can transfer any data type with a matching bit width via inter-FPGA over IOCNET. Use the Xilinx **Reinterpret** block to change your data type to UFix_32_0 or UFix_64_0 and vice versa. Reinterpreting data types does not cost any hardware or latency.

Restrictions on the IOCNET topology

Inter-FPGA via IOCNET can only be used for FPGA base boards if the following requirements on the IOCNET topology are fulfilled:

- The boards are installed to the same IOCNET segment. An IOCNET segment are the IOCNET nodes (I/O unit/LabBox/AutoBox) that are linked in serial to the processing hardware. The following illustrations shows an example.



SCALEXIO Processing Unit (1) [SCALEXIO Rack (1)]
- Angle Unit Set
- Ethernet Adapter 1 (Onboard)
- UART 5
- DS2703 6-Slot I/O Unit (2) [SCALEXIO Rack (1)]
  - DS6301 CAN/LIN Board (Slot 3)
  - DS6201 Digital I/O Board (Slot 4)
- DS2703 6-Slot I/O Unit (1) [SCALEXIO Rack (1)]
  - DS6601 FPGA Base Board (Slot 4)
  - DS6601 FPGA Base Board (Slot 5)
  - LabBox (8-slot) (2) [SCALEXIO Rack (1)]
    - DS6602 FPGA Base Board (Slot 1)
    - DS6602 FPGA Base Board (Slot 3)

Tip

In ConfigurationDesk, switch the Platform Manager to the network view via the context menu.

- Other I/O boards must not be connected to the used IOCNET segment.

Adding the inter-FPGA communication functionality to the FPGA application

The frameworks of the FPGA base boards provide inter-FPGA blocks to add the inter-FPGA communication functionality to the FPGA application.

For more information, refer to the following topics:

- RTI Block Settings for the DS2655 FPGA Base Board Framework (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮)
- RTI Block Settings for the DS6601 FPGA Base Board Framework (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮)
- RTI Block Settings for the DS6602 FPGA Base Board Framework (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮)

Specifying the inter-FPGA connections

After you added the FPGA application to the signal chain in ConfigurationDesk, you can reference the Inter-FPGA In blocks to the Inter-FPGA Out blocks to specify the communication bus.

For more information, refer to Configuring the Basic Functionality (FPGA) (ConfigurationDesk I/O Function Implementation Guide 🕮).

Related topics

Basics

Calculating the Latency (IOCNET)

Latency calculation

The latency depends on the network hops between the FPGA base boards. A network hop occurs when a data packet is passed from one network connection to the next. Each hop takes about 1.0 µs.

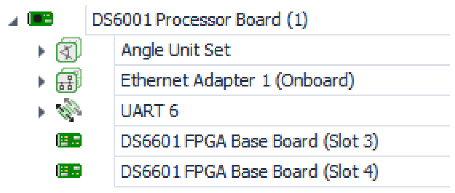
To calculate the latency, you have to count the IOCNET routers between the FPGA base boards, including the onboard routers.

Tip

In ConfigurationDesk, switch the **Platform Manager** to the network view via the context menu.

Example #1

The following illustration shows a topology with 3 network hops between the FPGA base boards.



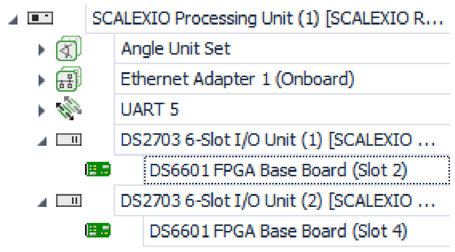
DS6001 Processor Board (1)
- Angle Unit Set
- Ethernet Adapter 1 (Onboard)
- UART 6
- DS6601 FPGA Base Board (Slot 3)
- DS6601 FPGA Base Board (Slot 4)

These are the network hops:
- Onboard IOCNET router of the DS6601 FPGA Base Board (Slot 3)
- Onboard IOCNET router of the DS6001 Processor Board
- Onboard IOCNET router of the DS6601 FPGA Base Board (Slot 4)

```
Latency = 3 hops · 1.0 µs/hop (typ.) ≈ 3 µs
```

Example #2

The following illustration shows a topology with 5 network hops between the FPGA base boards.


SCALEXIO Processing Unit (1) [SCALEXIO R...
Angle Unit Set
Ethernet Adapter 1 (Onboard)
UART 5
DS2703 6-Slot I/O Unit (1) [SCALEXIO ...
DS6601 FPGA Base Board (Slot 2)
DS2703 6-Slot I/O Unit (2) [SCALEXIO ...
DS6601 FPGA Base Board (Slot 4)

These are the network hops:

- Onboard IOCNET router of the DS6601 FPGA Base Board (Slot 2)
- IOCNET router of the DS2703 6-Slot I/O Unit (1)
- IOCNET router of the Processing Unit
- IOCNET router of the DS2703 6-Slot I/O Unit (2)
- Onboard IOCNET router of the DS6601 FPGA Base Board (Slot 4)

```
Latency = 5 hops · 1.0 µs/hop (typ.) = 5.0 µs
```

Modeling Inter-FPGA Communication via MGT Modules

Implementing Inter-FPGA Communication via MGT Modules

Introduction

Multi-gigabit transceiver (MGT) modules can be used to transfer data directly between FPGA boards of a SCALEXIO system.

Basic interface characteristics

You have to consider the following interface characteristics if you implement inter-FPGA communication via MGT modules.

- MGT modules are optional modules that must be installed to the FPGA base board.
- Up to 4 channels to transmit data values to up to 4 FPGA base boards.
- The maximum data rate is 10.3125 Gbit/s if you use an **Aurora 64b66b 128 Bit** block.
- The typical latency is 384 ns for single words. The maximum latency is 472 ns.
- Each 64-bit channel can address up to 512 data values.
- The data type is a raw data type: UFix_32_0 or UFix_64_0.

  You can transfer any data type with a matching bit width via inter-FPGA over IOCNET. Use the Xilinx **Reinterpret** block to change your data type to

UFix_32_0 or UFix_64_0 and vice versa. Reinterpreting data types does not cost any hardware or latency.

Adding the inter-FPGA communication functionality to the FPGA application

The *DS660X_MGT* framework provides I/O functions to transmit and receive data via an installed MGT module using the Aurora protocol, but you can also use customized protocols.

Preventing additional latencies for data streams Latency can increase if the transmission must pause to prevent the RX-FIFO buffer from overflowing due to clock drift between the different FPGA boards. When data is sent with every FPGA clock cycle, a pause can happen despite the high precision clocks of the DS6202 FPGA Base Boards. The latency is about 5.4 µs for 64-bit data transmission and 10.4 µs for 128-bit data transmission.

To transfer data with minimal latency, pause the transmission of data at least every 16.666 FPGA clock cycles for one clock cycle. This can be implemented with a 14-bit counter that pauses the transmission every 16.384 clock cycles, for example.

Further information For more information, refer to the following topics:

- Parameters Page (FPGA_IO_READ_BL) (RTI FPGA Programming Blockset - FPGA Interface Reference 📖)
- Parameters Page (FPGA_IO_WRITE_BL) (RTI FPGA Programming Blockset - FPGA Interface Reference 📖)
- Modeling MGT Communication Using a Customized Protocol on page 106

Related topics

Basics

Modeling Inter-FPGA Communication via I/O Module Slots

Where to go from here

Information in this section

Implementing Inter-FPGA Communication via I/O Module Slots

Introduction

With the Inter-FPGA Interface framework, the I/O module slots of the SCALEXIO FPGA base boards can be used to establish an inter-FPGA communication bus.

Avoiding damage to the board

NOTICE

The improper assembly of inter-FPGA communication buses will damage the FPGA boards

For inter-FPGA communication buses, special inter-FPGA communication cables must be used. Other cables, such as the cables used for connecting the I/O modules, will damage the FPGA boards. Furthermore, special rules for attaching the FPGA boards must be observed to ensure proper bus communication.

- Use the **SCLX_INT_FPGA_CAB1** inter-FPGA cables and observe the enclosed documentation for assembling.
- Do not connect FPGA boards via inter-FPGA cables if the FPGA boards are connected to different processors via IOCNET.

Basic interface characteristics

You have to consider the following interface characteristics if you implement inter-FPGA communication via I/O module slots.

- The inter-FPGA communication bus is a point-to-point one-way communication. It is implemented directly between the connected FPGA boards without using IOCNET.
- An inter-FPGA communication cable can only connect FPGA base boards that are installed next to each other. Therefore, up to two FPGA base boards can be directly connected to an FPGA base board. For an example, refer to Inter-FPGA communication between multiple SCALEXIO FPGA base boards on page 88.
- The maximum data rate is 582.3 Mbit/s with default values. Refer to Calculating the Data Rate and Latency (SCALEXIO) on page 94.

- The latency is 72 ns … 96 ns with default values. Refer to Calculating the Data Rate and Latency (SCALEXIO) on page 94.
- The maximum data width for inter-FPGA communication with bus synchronization is 27 bits.

  The interface provides a 28-bit parallel data bus. One data bit of each subbus is reserved for synchronization purposes.

  In expert mode, the *Inter-FPGA Interface* framework provides inter-FPGA communication without bus synchronization. In this mode, the maximum data width for inter-FPGA communication is 28 bits.
- You can configure up to eight subbuses for each inter-FPGA communication bus.
- Inter-FPGA communication between different types of SCALEXIO FPGA boards is supported. For example: A DS2655 FPGA Base Board can be connected to a DS6601 FPGA Board and a DS6602 FPGA Base Board.

Adding the inter-FPGA communication functionality to the FPGA application

The Inter-FPGA Interface framework provides the functionality to implement the inter-FPGA communication. You must select the framework on the **Unit** page of the **FPGA_SETUP_BL** block. With the selection you also specify the I/O module slot that supports the inter-FPGA communication.

For instructions, refer to How to Specify I/O Module Slots of SCALEXIO FPGA Base Boards as Inter-FPGA Interfaces on page 89.

Specifying the inter-FPGA communication

The **I-FPGA In** and **I-FPGA Out** I/O function blocks let you specify the inter-FPGA communication.

The function blocks' dialogs let you enable an expert mode. In expert mode, you can implement an inter-FPGA communication bus without bus synchronization or you can change the default values for clock, bit length, and filter depth. For robust communication, it is not recommended to activate expert mode if you do not have enough experience with configuring buses and knowledge of checking whether the configured transmission is correct with regard to the observed signal integrity at the applicable temperature range.

For more information on the inter-FPGA I/O functions, refer to RTI Block Settings for the Inter-FPGA Interface Framework (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮).

Configuring subbuses

Eight communication channels let you use up to eight subbuses. If you configure subbuses, you can access a specific subbus by specifying a bit range with the related start bits and end bits in the **Parameters** pages of the inter-FPGA I/O function block dialogs. The bit ranges of the subbuses must not overlap. One bit has to be reserved for synchronization purposes for each configured subbus. The maximum data width of a synchronized subbus is therefore
`Endbit - Startbit`.

You cannot use two I/O function blocks that use the same communication channel within the same FPGA application. For example, you cannot use an

I-FPGA In 1 function block if there is already an I-FPGA Out 1 function block, but you can add an I-FPGA In 2 block.

Note

If you send and receive data with the same inter-FPGA interface, you have to consider limitations on the bit ranges for the subbuses. Refer to How to Determine the Bit Ranges for Inter-FPGA Subbuses Between SCALEXIO FPGA Base Boards on page 91.

General configuration aspects

NOTICE

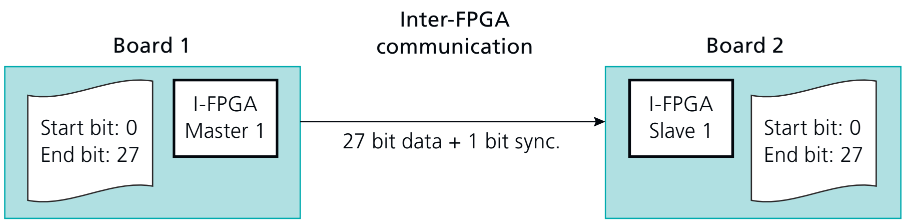
An incorrect configuration might damage the electrical interface.
If you configure both ends of an inter-FPGA connection bus to write on the bus, the connection results in a short circuit. This short circuit might damage the electrical interface of the used I/O module slots. In multiprocessor applications, an incorrect configuration cannot be detected automatically to beware hardware damage.

- Make sure that you implement for each I-FPGA Out block an I-FPGA In block as counterpart on the other FPGA board.
- Make sure that the inter-FPGA I/O blocks of the same subbus uses the same Startbit and Endbit to send or receive data.
- Do not use inter-FPGA communication in a multiprocessor application to transmit data between FPGA boards that are connected to different processors via IOCNET.

Application examples

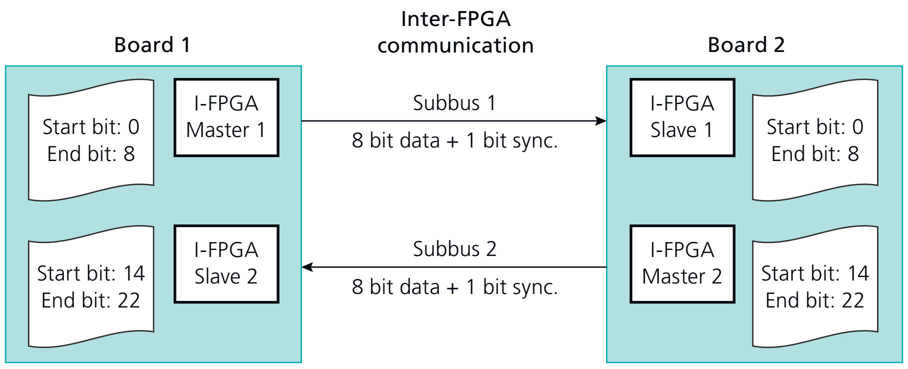
The hardware connections, the implemented model, and the block settings for the bus configuration have to match to get the required data transfer.

Inter-FPGA communication in one direction without a subbus In the following example with two FPGA base boards, the FPGA application that runs on board 1 uses the entire data width of the bus to send data to board 2.

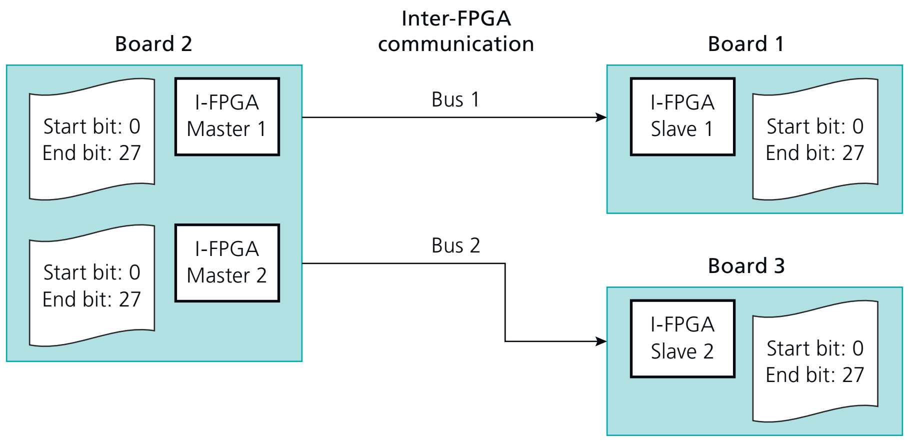


Inter-FPGA communication in both directions with two subbuses In the following example, the FPGA applications can exchange data with a data width of 8 bits in both directions. At the inports and outports of an I-FPGA Block, the position of the LSB is always 0. The offset configured by the start bit for an Inter-FPGA subbus does only apply to the arrangement of bits on the Inter-FPGA bus.

The master and the slave which are communicating have to be configured with the same communication settings. For example, it is not allowed to specify the slave settings to receive only a subrange of the transmitted data.

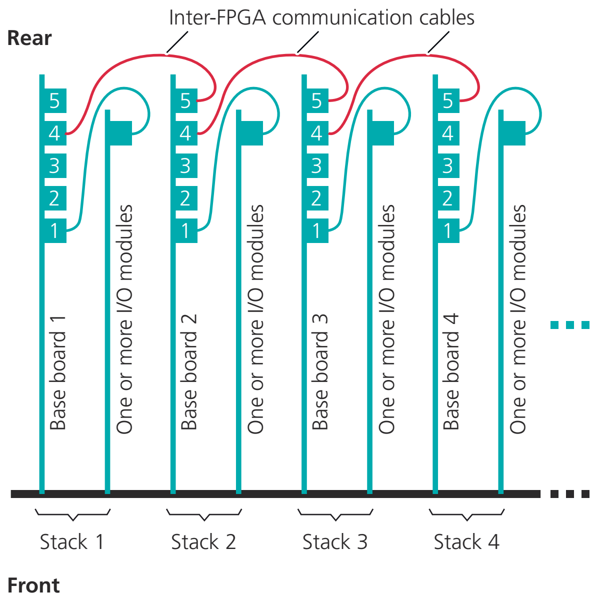


Inter-FPGA communication between multiple SCALEXIO FPGA base boards In the following example with three SCALEXIO FPGA base boards, the FPGA application of board 2 can directly send data to board 1 and board 3. The FPGA application that runs on board 2 uses two inter-FPGA interfaces, one interface for each connected board.



More than three boards can be connected via inter-FPGA communication cables, but only the boards that are located next to each other can be connected with an inter-FPGA communication cable.

The following illustration shows the assembly of four FPGA stacks (FPGA base board with I/O modules) that are used for inter-FPGA communication.



Related topics

Basics

HowTos

How to Specify I/O Module Slots of SCALEXIO FPGA Base Boards as Inter-FPGA Interfaces

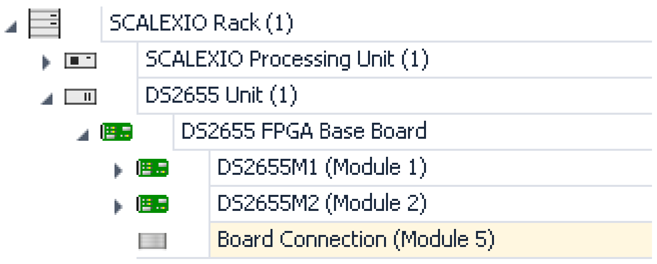
Objective

You have to specify the inter-FPGA connector only if you use the inter-FPGA communication on SCALEXIO FPGA base boards.

Method

To specify I/O module slots of a SCALEXIO FPGA base board as inter-FPGA interface

1 Via the Platform Manager, for example of ConfigurationDesk, check the I/O module slots that are used for inter-FPGA communication.



SCALEXIO Rack (1)
SCALEXIO Processing Unit (1)
DS2655 Unit (1)
DS2655 FPGA Base Board
DS2655M1 (Module 1)
DS2655M2 (Module 2)
Board Connection (Module 5)

2 On the dialog of the FPGA_SETUP_BL block, make sure that a SCALEXIO FPGA base board framework is selected.

3 Select Inter-FPGA Interface for I/O module <slot number> for the I/O module slot that is connected to another FPGA board for inter-FPGA communication.

4 Repeat step 3 to specify other I/O module slots as inter-FPGA interface.

Result

You specified the I/O module slots that support inter-FPGA communication. Now, the I/O function blocks to implement the inter-FPGA communication are available.

Tip

On the Parameters page of the FPGA SETUP BL block enter a description of the implemented inter-FPGA connection.
When you add the FPGA model INI file to ConfigurationDesk, the FPGA description property of the FPGA custom function block displays the description. The description helps you to identify the correct hardware resources for the FPGA custom function block.

Related topics

Basics

Modeling Inter-FPGA Communication.......... 78

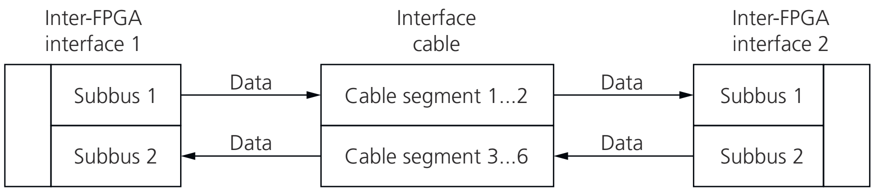
How to Determine the Bit Ranges for Inter-FPGA Subbuses Between SCALEXIO FPGA Base Boards

Objective

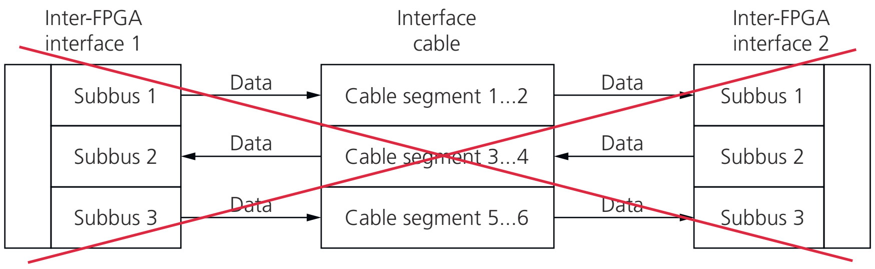
When you implement subbuses to send and receive data via an inter-FPGA communication bus between SCALEXIO FPGA base boards, you must observe limitations for the bit ranges of subbuses.

Limitations

For a robust communication, the wires of the interface cable are grouped to six cable segments. Within a cable segment, data can only be sent in one direction. That means, you cannot use the same cable segment to send and receive data. Furthermore, the interface cable supports only two cable segment ranges that transmit data as shown in the example below.



The following example shows an unsupported subbus configuration, because the subbuses use three cable segment ranges.



Note

If you use the inter-FPGA communication bus only in one direction, you can specify the bit ranges of subbuses without limitations. You must consider the limitations only if you send and receive data with the same inter-FPGA interface.

Bit ranges supported by the cable segments

The following table shows which cable segment supports which bit range:

Cable Segment	Bit range
1	0 ... 5
2	6 ... 7
3	8 ... 13
4	14 ... 19

Cable Segment	Bit range
5	20 ... 21
6	22 ... 27

Method

To determine the bit ranges for inter-FPGA subbuses between SCALEXIO FPGA base boards

1. Choose one continuous range of cable segments that can be used to send data and one continuous range that can be used to receive data. The ranges must not overlap. For examples, refer to Configuration examples on page 92.
2. Identify the bit range that is supported by the chosen cable segment range for sending data. This bit range can be used by subbuses to send data.

   For example: The cable segments 1 ... 3 support the bit range 0 ... 13.
3. Use the identified bit range to determine the bit ranges of subbuses to send data. You can determine multiple subbuses, as shown by Example 2 on page 93.
4. Identify the bit range that is supported by the chosen cable segment range for receiving data. This bit range can be used by subbuses to receive data.
5. Use the identified bit range to determine the bit ranges of subbuses to receive data.

Result

You determined the bit ranges of the subbuses.

Next step

Add the I-FPGA In and I-FPGA Out function blocks to the FPGA model to implement the inter-FPGA communication and configure their start and end bits with the determined bit ranges.

Configuration examples

The examples help you configure the bit ranges of your subbuses and show you possible configuration errors.

Example 1 The following table shows you an example for bit ranges of two subbuses.

Bus Direction	Chosen Cable Segments		Configured Subbuses		
	Segment Range	Resulting Bit range for Subbuses	I/O Function Block	Start Bit	End Bit
Sending data	1 ... 3	0 ... 13	I-FPGA Out	0	8
Receiving data	4 ... 6	14 ... 27	I-FPGA In	14	22

Example 2 The following table shows you an example for bit ranges of four subbuses.

Bus Direction	Chosen Cable Segments		Configured Subbuses		
	Segment Range	Resulting Bit range for Subbuses	I/O Function Block	Start Bit	End Bit
Sending data	1 … 2	0 … 7	I-FPGA Out	0	4
			I-FPGA Out	5	6
Receiving data	3 … 6	8 … 27	I-FPGA In	8	16
			I-FPGA In	19	27

Example 3 (unsupported configuration) The following table shows you an example of an unsupported configuration of two subbuses.

Configured Subbuses			Bus Direction	Affected Cable Segments	Resulting Conflict
I/O Function Block	Start Bit	End Bit			
I-FPGA Out	0	8	Sending data	1 … 3	Cable segment 3 is used in both directions, but one cable segment can be used either to send data or to receive data. Remedy: Use the bit range 14 … 22 for the I-FPGA In function block.
I-FPGA In	9	17	Receiving data	3 … 4	

Example 4 (unsupported configuration) The following table shows you an example of an unsupported configuration of three subbuses.

Configured Subbuses			Bus Direction	Affected Cable Segments	Resulting Conflict
I/O Function Block	Start Bit	End Bit			
I-FPGA Out	0	4	Sending data	1	More than two segment ranges are affected by the subbuses. Remedy: Use cable segment 6 for the I-FPGA In function block and cable segment 3 for the I-FPGA Out function block. Thus cable segment range 1 … 3 is used to send data and cable segment 6 is used to receive data.
I-FPGA In	8	12	Receiving data	3	
I-FPGA Out	22	26	Sending data	6	

Related topics

References

RTI Block Settings for the Inter-FPGA Interface Framework (RTI FPGA Programming Blockset - FPGA Interface Reference 📖)

Calculating the Data Rate and Latency (SCALEXIO)

Calculating data rate

The data rate of an inter-FPGA (sub)bus depends on the block settings for the clock, the bit length and the data width.

$\mathtt{DataRate_{Max} = I\text{-}FPGA\_Clock \cdot (DataWidth - 1) / BitLength}$

Resulting data rate for data transfer with bus synchronization The maximum data rate with the default values is 562.5 Mbit/s for the entire bus or 20.83 Mbit/s per bit.

$\mathtt{DataRate_{Max,Bus} = 125\ MHz \cdot (28 - 1)\ Bit / 6}$

$\mathtt{DataRate_{Max,Bit} = 125\ MHz \cdot 1\ Bit / 6}$

Resulting data rate for data transfer without bus synchronization The maximum data rate with the default values is 582.3 Mbit/s for the entire bus or 20.83 Mbit/s per bit.

$\mathtt{DataRate_{Max,Bus} = 125\ MHz \cdot 28\ Bit / 6}$

$\mathtt{DataRate_{Max,Bit} = 125\ MHz \cdot 1\ Bit / 6}$

Calculating latency

The latency consists of a constant value and dependencies on the clock rate of the inter-FPGA bus, filter depth, and on whether jitters and spikes occur.

General formula for the latency:

$\mathtt{Latency = T_{I\text{-}FPGA\ Out} + T_{inter\text{-}FPGA\ Master} + T_{inter\text{-}FPGA\ Slave} + T_{I\text{-}FPGA\ In}}$

The formula use the following parameters:

$\mathtt{T_{I\text{-}FPGA\ Out}}$: Latency of the I-FPGA Out FPGA function.

$\mathtt{T_{I\text{-}FPGA\ Out} = 8\ ns}$

$\mathtt{T_{inter\text{-}FPGA\ Master}}$: Latency of the inter-FPGA component to write data to the inter-FPGA bus.

$\mathtt{T_{inter\text{-}FPGA\ Master} = ClockPeriod_{inter\text{-}FPGA} \cdot 2\ ClockCycles}$

$\mathtt{T_{inter\text{-}FPGA\ Slave}}$: Latency of the inter-FPGA component to read data from the inter-FPGA bus.

$\mathtt{T_{inter\text{-}FPGA\ Slave}}$ depends on the filter depth, and jitter and spikes on the bus lines:

- Filter depth = 0, no jitter and spikes on the bus lines:
  $\mathtt{T_{inter\text{-}FPGA\ Slave} = ClockPeriod_{inter\text{-}FPGA} \cdot 4\ ClockCycles}$
- Filter depth = 0 and jitter occurs:
  $\mathtt{T_{inter\text{-}FPGA\ Slave} = ClockPeriod_{inter\text{-}FPGA} \cdot (4 + 1)\ ClockCycles}$
- Filter depth >0, no jitter and spikes on the bus lines:
  $\mathtt{T_{inter\text{-}FPGA\ Slave} = ClockPeriod_{inter\text{-}FPGA} \cdot (3\ ClockCycles + FilterDepth)}$

- Filter depth >0 and jitter occurs:
  $T_{\text{inter-FPGA Slave}} = \text{ClockPeriod}_{\text{inter-FPGA}} \cdot (3\ \text{ClockCycles} + (\text{FilterDepth} + 1))$
- Filter depth >0 and jitter and spikes on the bus lines:
  $T_{\text{inter-FPGA Slave}} = \text{ClockPeriod}_{\text{inter-FPGA}} \cdot (3\ \text{ClockCycles} + (\text{FilterDepth} + 1 + [0 \dots \text{FilterDepth}]))$
  With the range `[0 … FilterDepth]` as the latency caused by spikes.

$T_{\text{I-FPGA In}}$: Latency of the I-FPGA In FPGA function.
$T_{\text{I-FPGA In}} = 8\ \text{ns}$

Latency calculation examples

The following examples display the resulting latency for different applications.

Calculation example for an inter-FPGA bus in standard mode The clock rate of the inter-FPGA bus is 125 MHz in standard mode and the filter depth is set to 2:

- $\text{Latency} = 56\ \text{ns} + \text{FilterDepth} \cdot 8\ \text{ns}$
  - If jitter occurs: $\text{Latency} = 56\ \text{ns} + (\text{FilterDepth} + 1) \cdot 8\ \text{ns}$
  - If spikes occur: $\text{Latency} = 56\ \text{ns} + (\text{FilterDepth} + [0 \dots \text{FilterDepth}]) \cdot 8\ \text{ns}$
  - Maximal: $\text{Latency} = 64\ \text{ns} + (2 \cdot \text{FilterDepth}) \cdot 8\ \text{ns}$

With the default values, the latency is in the range 72 ns … 96 ns.

$\text{Latency}_{\text{Min}} = 56\ \text{ns} + 2 \cdot 8\ \text{ns} = 72\ \text{ns}$

$\text{Latency}_{\text{Max}} = 64\ \text{ns} + (2 \cdot 2) \cdot 8\ \text{ns} = 96\ \text{ns}$

Calculation example for an inter-FPGA bus in expert mode

Note

Use the expert mode only if you have enough experience of configuring buses and knowledge of checking the correctness of the configured transmission with regard to the observed signal integrity at the applicable temperature range.
The default values for the bit length, clock, and filter depth have been tested by dSPACE.

The clock rate of the inter-FPGA bus is set to 250 MHz.

- If the filter depth is set to 0:
  $\text{Latency} = 40\ \text{ns}$
  - If jitter occurs: $\text{Latency} = 44\ \text{ns}$
- If the filter depth is set to >0:
  $\text{Latency} = 36\ \text{ns} + \text{FilterDepth} \cdot 4\ \text{ns}$
  - If jitter occurs: $\text{Latency} = 36\ \text{ns} + (\text{FilterDepth} + 1) \cdot 4\ \text{ns}$
  - If spikes occur: $\text{Latency} = 36\ \text{ns} + (\text{FilterDepth} + [0 \dots \text{FilterDepth}]) \cdot 4\ \text{ns}$
  - Maximal: $\text{Latency} = 40\ \text{ns} + (2 \cdot \text{FilterDepth}) \cdot 4\ \text{ns}$

With the default values and a 250 MHz clock rate, the latency is in the range 44 ns … 56 ns.

$\text{Latency}_{\text{Min}} = 36\ \text{ns} + 2 \cdot 4\ \text{ns} = 44\ \text{ns}$

$\text{Latency}_{\text{Max}} = 40\ \text{ns} + (2 \cdot 2) \cdot 4\ \text{ns} = 56\ \text{ns}$

Modeling Inter-FPGA Communication in a PHS-Bus-Based System

Where to go from here

Information in this section

Implementing Inter-FPGA Communication Between DS5203 FPGA Boards

Modeling inter-FPGA communication

The DS5203 FPGA Board provides two inter-FPGA communication connectors. Together, the two connectors provide a 32-bit parallel data bus. The *DS5203 with onboard I/O* frameworks provides the I/O function blocks to implement inter-FPGA communication.

You have to consider the following inter-FPGA interface characteristics:

- The inter-FPGA communication bus is a point-to-point one-way communication. It is implemented directly between the connected FPGA boards without using the PHS bus.
- The maximum data width is 31 bits.

  At least one bit of the 32-bit parallel data bus must be used for bus synchronization.
- The maximum data rate is 516.67 Mbit/s with default values. Refer to Calculating the Data rate and Latency (PHS-Bus-Based System) on page 98.
- The latency is 90 ns … 120 ns with default values. Refer to Calculating the Data rate and Latency (PHS-Bus-Based System) on page 98.

- Inter-FPGA communication between more than two DS5203 FPGA Boards is not supported, because both inter-FPGA connectors must be connected to the other FPGA board.

  For a board overview, refer to DS5203 Components (PHS Bus System Hardware Reference 🕮).

Configuring subbuses Eight communication channels let you use up to eight subbuses. If you configure subbuses, you can access a specific subbus by specifying the related start and end bits in the I/O function blocks' **Parameters** pages. The maximum data width of a subbus is `Endbit - Startbit`. You have to make sure that the sending I/O function uses the same bit range as the receiving I/O function.

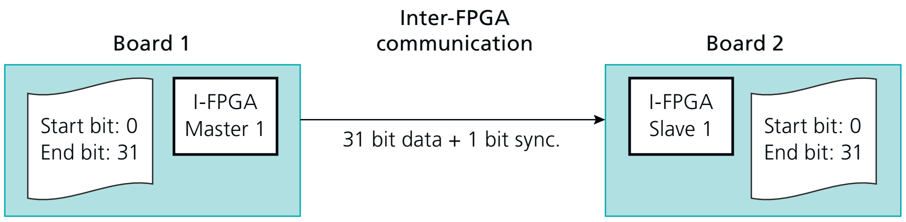
Within one model, you can add either an **I-FPGA Master <channel>** block or an **I-FPGA Slave <channel>** block for the same communication channel. For example, you cannot add an **I-FPGA Slave 1** block to the model if there is already an **I-FPGA Master 1** block, but you can add an **I-FPGA Slave 2** block.

Bus settings For a robust communication, it is recommended to use the default values for the clock, bit length, and filter depth. You should change these values only if you have enough experience of configuring buses and knowledge of checking the correctness of the configured transmission with regard to the observed signal integrity at the applicable temperature range.

Application examples

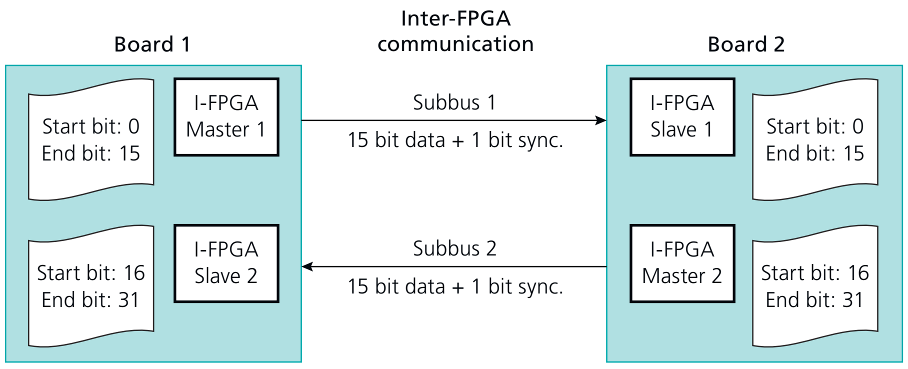
The hardware connections, the implemented model, and the block settings for the bus configuration have to match to get the required data transfer.

Inter-FPGA communication in one direction without a subbus In the following example with two DS5203 FPGA Boards, the FPGA application that runs on board 1 uses the entire data width of the bus to send data to board 2.



Inter-FPGA communication in both directions with two subbuses In the following example with two DS5203 FPGA Boards, the FPGA applications can exchange data with a data width of 15 bits in both directions. At the inports and outports of an I-FPGA Block, the position of the LSB is always 0. The offset configured by the start bit for an Inter-FPGA subbus does only apply to the arrangement of bits on the Inter-FPGA bus.

The master and the slave which are communicating have to be configured with the same communication settings. For example, it is not allowed to specify the slave settings to receive only a subrange of the transmitted data.



Related topics

Basics

Overview of Inter-FPGA Communication.. 78

References

RTI Block Settings for the DS5203 with onboard I/O Frameworks (RTI FPGA Programming Blockset - FPGA Interface Reference 📖)

Calculating the Data rate and Latency (PHS-Bus-Based System)

Calculating data rate

The data rate of an inter-FPGA (sub)bus depends on the block settings for the clock, the bit length and the data width.

$\mathtt{DataRate}_{\mathtt{Max}} = \mathtt{I\text{-}FPGA\_Clock} \cdot (\mathtt{DataWidth} - 1) / \mathtt{BitLength}$

With the default values, the maximum data rate is 516.67 Mbit/s for the entire bus or 16.67 Mbit/s per bit.

$\mathtt{DataRate}_{\mathtt{Max}} = \mathtt{100\ MHz} \cdot (\mathtt{32} - \mathtt{1})\ \mathtt{Bit} / \mathtt{6}$

Calculating latency

The latency consists of a constant value and dependencies on the clock rate of the inter-FPGA bus, filter depth, and on whether jitters and spikes occur.

General formula for the latency:
$\texttt{Latency} = T_{\texttt{I-FPGA Out}} + T_{\texttt{inter-FPGA Master}} + T_{\texttt{inter-FPGA Slave}} + T_{\texttt{I-FPGA In}}$

The formula use the following parameters:

$T_{\texttt{I-FPGA Out}}$:	Latency of the **I-FPGA Master** FPGA function. $T_{\texttt{I-FPGA Out}} = \texttt{10 ns}$
$T_{\texttt{inter-FPGA Master}}$:	Latency of the inter-FPGA component to write data to the inter-FPGA bus. $T_{\texttt{inter-FPGA Master}} = \texttt{ClockPeriod}_{\texttt{inter-FPGA}} \cdot \texttt{2 ClockCycles}$
$T_{\texttt{inter-FPGA Slave}}$:	Latency of the inter-FPGA component to read data from the inter-FPGA bus. $T_{\texttt{inter-FPGA Slave}}$ depends on the filter depth, and jitter and spikes on the bus lines: ▪ Filter depth = 0, no jitter and spikes on the bus lines: $T_{\texttt{inter-FPGA Slave}} = \texttt{ClockPeriod}_{\texttt{inter-FPGA}} \cdot \texttt{4 ClockCycles}$ ▪ Filter depth = 0 and jitter occurs: $T_{\texttt{inter-FPGA Slave}} = \texttt{ClockPeriod}_{\texttt{inter-FPGA}} \cdot \texttt{(4 + 1) ClockCycles}$ ▪ Filter depth >0, no jitter and spikes on the bus lines: $T_{\texttt{inter-FPGA Slave}} = \texttt{ClockPeriod}_{\texttt{inter-FPGA}} \cdot \texttt{(3 ClockCycles + FilterDepth)}$ ▪ Filter depth >0 and jitter occurs: $T_{\texttt{inter-FPGA Slave}} = \texttt{ClockPeriod}_{\texttt{inter-FPGA}} \cdot \texttt{(3 ClockCycles + (FilterDepth + 1))}$ ▪ Filter depth >0 and jitter and spikes on the bus lines: $T_{\texttt{inter-FPGA Slave}} = \texttt{ClockPeriod}_{\texttt{inter-FPGA}} \cdot \texttt{(3 ClockCycles + (FilterDepth + 1 + [0 … FilterDepth]))}$ With the range `[0 … FilterDepth]` as the latency caused by spikes.
$T_{\texttt{I-FPGA In}}$:	Latency of the **I-FPGA Slave** FPGA function. $T_{\texttt{I-FPGA In}} = \texttt{10 ns}$

Latency calculation examples

The following examples display the resulting latency for different applications.

Calculation example for an inter-FPGA bus in standard mode The clock rate of the inter-FPGA bus is 100 MHz in standard mode and the filter depth is set to 2:

- `Latency = 70 ns + FilterDepth · 10 ns`
  - If jitter occurs: `Latency = 70 ns + (FilterDepth + 1) · 10 ns`
  - If spikes occur: `Latency = 70 ns + (FilterDepth + [0 … FilterDepth]) · 10 ns`
  - Maximal: `Latency = 80 ns + (2 · FilterDepth) · 10 ns`

With the default values, the latency is in the range 90 ns … 120 ns.

$\mathtt{Latency}_{\mathtt{Min}}$ `= 70 ns + 2 · 10 ns = 90 ns`

$\mathtt{Latency}_{\mathtt{Max}}$ `= 80 ns + (2 · 2)· 10 ns = 120 ns`

Calculation example for an inter-FPGA bus in expert mode

Note

Use the expert mode only if you have enough experience of configuring buses and knowledge of checking the correctness of the configured transmission with regard to the observed signal integrity at the applicable temperature range.
The default values for the bit length, clock, and filter depth have been tested by dSPACE.

The clock rate of the inter-FPGA bus is set to 200 MHz.

- If the filter depth is set to 0:

  `Latency = 50 ns`

  - If jitter occurs: `Latency = 55 ns`
- If the filter depth is set to >0:

  `Latency = 45 ns + FilterDepth · 5 ns`

  - If jitter occurs: `Latency = 45 ns + (FilterDepth + 1) · 5 ns`
  - If spikes occur: `Latency = 45 ns + (FilterDepth + [0 … FilterDepth]) · 5 ns`
  - Maximal: `Latency = 50 ns + (2 · FilterDepth) · 5 ns`

With the default values and a 200 MHz clock rate, the latency is in the range 55 ns … 70 ns.

$\mathtt{Latency}_{\mathtt{Min}}$ `= 45 ns + 2 · 5 ns = 55 ns`

$\mathtt{Latency}_{\mathtt{Max}}$ `= 50 ns + (2 · 2)· 5 ns = 70 ns`

Modeling FPGA Applications Supporting Multicore Processor Applications

Introduction

The RTI FPGA Programming Blockset lets you model FPGA applications that support multicore processor applications. There are some aspects you have to consider when you model an FPGA application with multicore support.

Where to go from here

Information in this section

Aspects on FPGA Applications Supporting Multicore Processor Applications

Platforms for FPGA application with multicore support

FPGA applications of MicroLabBox, MicroAutoBox III and SCALEXIO systems can support multicore processor applications (multicore real-time applications).

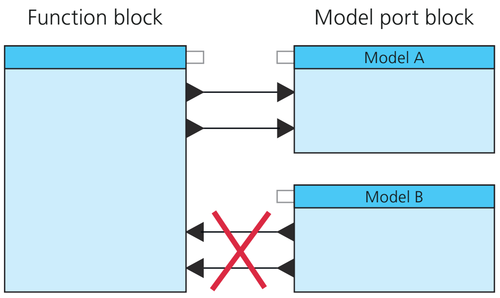
FPGA applications of DS5203 FPGA Boards (PHS-bus-based systems) support only single-core processor applications. But each processor core in a PHS-bus-based system can access its own FPGA board.

Multicore support for MicroLabBox

There are no special aspects you have to consider when you are modeling an FPGA application with multicore support. Keep in mind, that the assignment to the processor interface must be unique. You cannot access the same FPGA interface block from different processor interface blocks.

Multicore support for MicroAutoBox III and SCALEXIO systems

ConfigurationDesk lets you implement and build processor applications for MicroAutoBox III and SCALEXIO systems. In ConfigurationDesk, model port blocks represent the interfaces of processor models and FPGA function blocks represent the FPGA application. Model port blocks let you connect FPGA function blocks with the processor model, but model port blocks that reside in different processor models must not be mapped to the same function block as shown below.

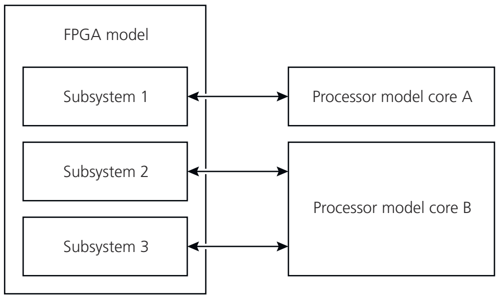


However, a multicore processor application is built from different processor models (behavior models in ConfigurationDesk), so there must be one processor model for each processor core. Therefore, the FPGA application can be represented by several FPGA custom function blocks. This lets you use one FPGA application by model port blocks that reside in different processor models.

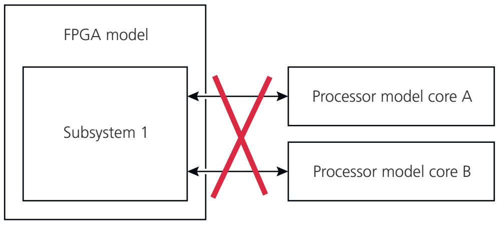
Specifying the representation of FPGA applications The FPGA custom function blocks are defined by XML files during the build process of the FPGA. To specify which parts of the FPGA model are represented by one FPGA custom function, you can select subsystems of the FPGA model.

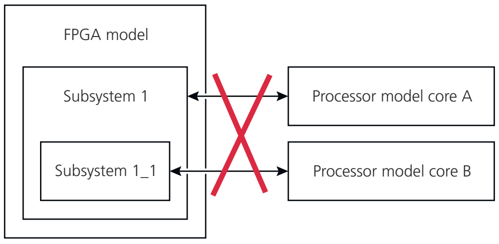
For instructions on selecting the subsystems so that the build process generates several FPGA custom functions, refer to How to Select Subsystems for Multicore Support (MicroAutoBox III, SCALEXIO) on page 103.

Modeling FPGA applications with multicore support You cannot access one subsystem and its subelements of the FPGA model from several processor models, but several subsystems of the FPGA model can access the same processor model. Therefore, implement subsystems to your FPGA in the way that the FPGA interface of one subsystem provides the access for only one processor model. The following illustration shows this modeling rule.



The following illustrations show FPGA interfaces that are not supported in ConfigurationDesk.





Related topics

Basics

Modeling FPGA Applications Supporting Multicore Processor Applications.............................. 101

References

GetMCSubsystems (RTI FPGA Programming Blockset Script Interface Reference 🕮)
SetMCSubsystems (RTI FPGA Programming Blockset Script Interface Reference 🕮)

How to Select Subsystems for Multicore Support (MicroAutoBox III, SCALEXIO)

Objective

To support a multicore processor application of a MicroAutoBox III or a SCALEXIO system with your FPGA application, you have to select the subsystems of the FPGA model that are connected to the different processor cores.

Aspects on modeling multicore support

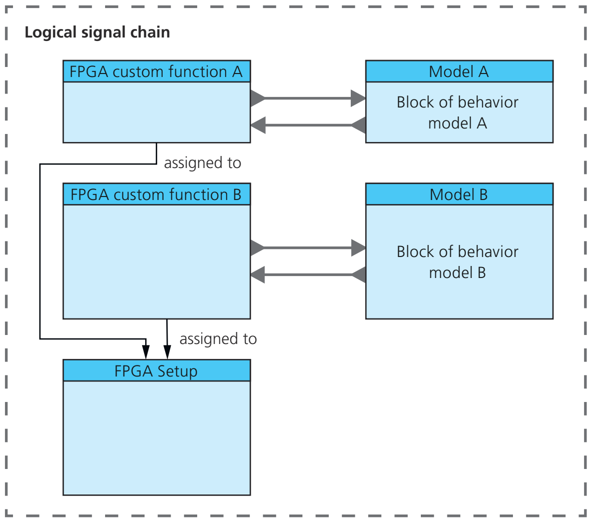
Refer to Aspects on FPGA Applications Supporting Multicore Processor Applications on page 101.

FPGA application with multicore support in ConfigurationDesk

To support multicore processor applications, at least three function blocks represent the entire FPGA application in ConfigurationDesk. The separation of the FPGA application into one **FPGA Setup** block (**<application name>_Setup**) and at least two FPGA custom function blocks let you use one FPGA application by several processor models (behavior models in ConfigurationDesk).

In ConfigurationDesk, the **FPGA Setup** function block lets you configure and initialize the access to the FPGA base board. At least two FPGA custom function blocks provide the interfaces to map processor models and external devices.

Representation in ConfigurationDesk The following illustration shows you an FPGA application in ConfigurationDesk that is mapped to the processor models of the different processor cores (behavior models) and assigned to the FPGA Setup block.



Precondition

The FPGA interface is implemented to the FPGA model.

Method

To select subsystems for multicore support

1 On the top level of the FPGA model, double-click the FPGA_SETUP_BL block and open the ConfigurationDesk Interface page.

2 In the tree view, click the subsystem that provide blocks of the FPGA interface to a processor model (FPGA_XDATA_WRITE_BL blocks, FPGA_XDATA_READ_BL blocks, and FPGA_INT_BL blocks).

   You select the subsystem and all subelements. The selected subelements are not click-able.

3 Repeat step 2 until you selected all other subsystems providing blocks of an FPGA interface to a processor model.

Result

Analyze Multicore Assignment displays the number of function block types that will be defined by the build process. The tree view grouped the subsystems according to the FPGA custom function block types they belong to. The top level of the FPGA model and all unselected subsystems belong to one FPGA custom function block.

The following example shows you which FPGA custom function block type represents which part of the FPGA model.

Tree-View	Resulting Function Block Types	Represented Subsystems
□ DS2655_Multicore □ ├──Subsystem1 ■ ├──Subsystem2 ■ ├──Subsystem3 │ └──Subsystem3_1 │ └──Subsystem3_1_1 ■ ├──Subsystem4 □ └──Subsystem5	DS2655_Multicore	▪ DS2655_Multicore ▪ Subsystem1 ▪ Subsystem5
	DS2655_Multicore_Subsystem2	▪ Subsystem2
	DS2655_Multicore_Subsystem3	▪ Subsystem3 ▪ Subsystem3_1 ▪ Subsystem3_1_1
	DS2655_Multicore_Subsystem4	▪ Subsystem4
	DS2655_Multicore_Setup	—

Related topics

Basics

Modeling FPGA Applications Supporting Multicore Processor Applications............................ 101

Modeling MGT Communication Using a Customized Protocol

Introduction

A DemoFPGAAurora8b10b demo project (Aurora 8b10b demo) for SCALEXIO systems shows how to implement a customized protocol.

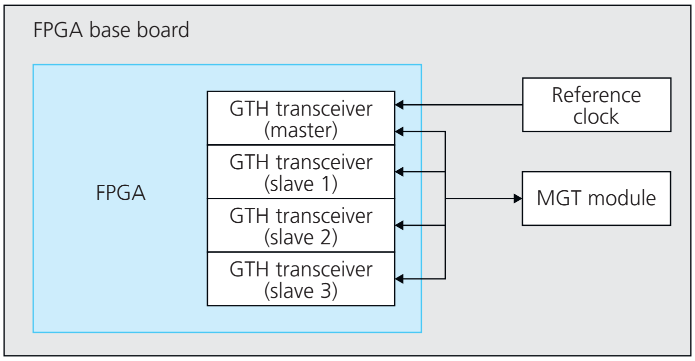
Where to go from here

Information in this section

Basic Structure of the MGT Interface

Block diagram of the interface

The DS6601 and DS6602 FPGA base boards provide a connector to insert an optical adapter for MGT (MGT module). The following diagram shows the basic components of the FPGA base boards to support the MGT communication bus.


FPGA base board
FPGA
GTH transceiver (master)
GTH transceiver (slave 1)
GTH transceiver (slave 2)
GTH transceiver (slave 3)
Reference clock
MGT module

Components description

The following table shows the description of the components.

Component	Description
FPGA	The FPGA processes the build FPGA application.
GTH transceiver	The GTH transceivers are configurable transceivers that are integrated with the logic resources of the FPGA. The GTH transceivers support line rates from 500 Mbit/s … 16.375 Gbit/s. The master

Component	Description
	transceiver receives the differential reference clock signal and provide it to the slave transceivers. Master or slave has no influence on the MGT communication. The GTH transceivers can be configured by the FPGA application to support different protocols, line rates, etc. For details on the GTH transceiver, refer to https://www.xilinx.com/support/documentation/user_guides/ug576-ultrascale-gth-transceivers.pdf.
Reference clock	This clock provides the configured reference frequency for the GTH transceivers.
MGT module	The MGT module is an optical adapter that can be connected to the FPGA base board. The MGT module provides four channels for communication. The order number of the adapter for MGT are as follows: ▪ DS6601_MGT1 for the DS6601 FPGA Base Board. ▪ DS6602_MGT1 for the DS6602 FPGA Base Board.

Related topics

Basics

Overview of the DS6601 FPGA Base Board (SCALEXIO Hardware Installation and Configuration 📖)
Overview of the DS6602 FPGA Base Board (SCALEXIO Hardware Installation and Configuration 📖)

Customized Protocol Demo

Supported platforms

MGT communication is supported by SCALEXIO systems with a DS6601 or DS6602 FPGA Base Board with an installed MGT module.

Accessing the demo

Open the block library and select Demos – DemoFPGAAurora8b10b – DS6601_XCKU035 or DS6602_XCKU15P.

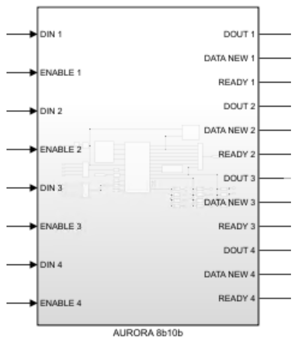
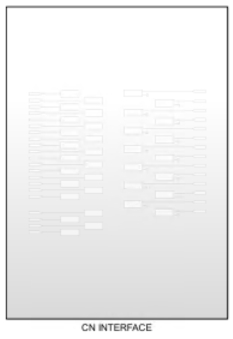
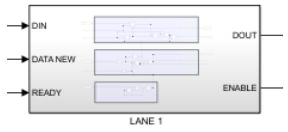
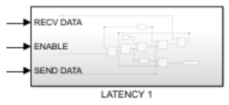
Use case

The Aurora8b10b demo uses a customized transfer protocol to send data values in a loop via the MGT interface. The customized transfer protocol is the Aurora 8b10b protocol for data transmission.

The demo measures the latency, lets you inject errors and provides status information about the communication bus. Backup projects for ConfigurationDesk and ConrolDesk let you directly experiment with the demo.

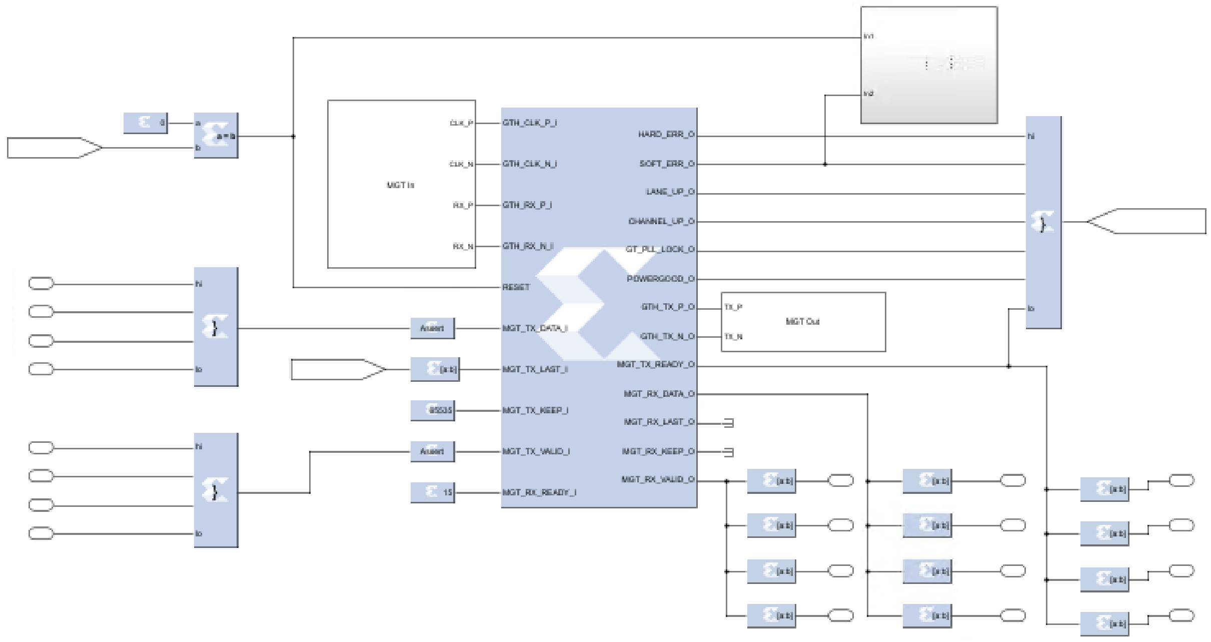
Demo overview

The following table shows the top level of the Simulink model with the subsystems.

Subsystem	Description
AURORA 8b10b  DIN 1 ENABLE 1 DIN 2 ENABLE 2 DIN 3 ENABLE 3 DIN 4 ENABLE 4 DOUT 1 DATA NEW 1 READY 1 DOUT 2 DATA NEW 2 READY 2 DOUT 3 DATA NEW 3 READY 3 DOUT 4 DATA NEW 4 READY 4 AURORA 8b10b	Provides the interface and configuration of the GTH transceiver to support a customized protocol. For more information, refer to AURORA 8b10b subsystem on page 109.
CN interface 	Provides the interface to the processor model. The subsystem writes the following status information of the protocol lanes to the processor model: ▪ RX_COUNTER: Number of received data values. ▪ TX_COUNTER: Number of transmit data values. ▪ LATENCY: Measured latencies. ▪ ERROR_COUNTER: Number of detected hard errors. ▪ SOFT_ERROR: Number of detected soft errors. ▪ AURORA_FLAGS: Bit-coded protocol information. The subsystem reads the following values from the processor model: ▪ RESETN: Flag to enable the GTH transceiver. ▪ ENABLE_TX: Flags to enable the different protocol lanes. ▪ SKIP: Flags to skip the data transmission for the different protocol lanes. The flags can be used to control the fill level of the FIFO buffers. ▪ ERROR_INJECTION: Flags to inject errors in the data transmission for the different protocol lanes. ▪ TX_LAST: Flags to indicate the last data value to be transmitted for the different protocol lanes.
Lane 1 … 4 	Controls the data transfer. The subsystem provides the following features: ▪ Generates the data values and detects transmission errors. The data values are the output of an incrementing counter. ▪ Enables and resets the counter. ▪ Delays the transmission of data values.
Latency 1 … 4 	Calculates the latency to send and receive data values over an optical loop.

AURORA 8b10b subsystem

The following illustration shows the components of the AURORA 8b10b subsystem.



Block	Description
Xilinx Black Box	Instantiates the GTH transceivers and defines the used transfer protocol. The MGT module provides four channels for communication. Each channel is connected to one GTH transceiver. To customize the protocol, the **Black Box** must be changed.
MGT In	Provides information about the connection between the GTH transceivers and the MGT module for data reception and let you specify the reference clock frequency. **MGT In** is a block of the RTI FPGA Programming Blockset. Refer to Parameters Page (FPGA_IO_READ_BL) (RTI FPGA Programming Blockset - FPGA Interface Reference 📖).
MGT Out	Provides information about the connection between the GTH transceivers and the MGT module. **MGT Out** is a block of the RTI FPGA Programming Blockset. Refer to Parameters Page (FPGA_IO_WRITE_BL) (RTI FPGA Programming Blockset - FPGA Interface Reference 📖).

Using a customized protocol

You can use the **IP Catalog** of **Vivado** to customize a predefined protocol. The IP Catalog provides a list of IP cores that can be customized. For more information, refer to https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug896-vivado-ip.pdf.

The **IP Catalog** lets you generate a net list that you can integrate in a **Black Box**. You can also include optional features like FIFOs for the inports and outports, or a reset logic for the GTH transceiver.

Required settings

The following parameters depend on the FPGA base board and must be set to a specific value in the VHDL file:

- The low power mode of the equalizer must be enabled.
  `gt_rxlpmen => (others => '1')`
- The differential swing voltage must be set to board-specific value:
  - DS6601: 460 mV_{pp} (peak-to-peak voltage)
    `gt_txdiffctrl => "0100"`
  - DS6602: 498 mV_{pp}
    `gt_txdiffctrl => "01010"`

Provided demo files

The table shows the files that are provided with the demo. You can use these files to experiment with the demo, to change the demo, or to specify a new Black Box.

File Name	Description
Files for Experimenting	
`CD_DemoFPGAAurora8b10b.ZIP`	Provides a ControlDesk project to experiment with the demo.
`CFG_DemoFPGAAurora8b10b.ZIP`	Provides a ConfigurationDesk project including a build real-time application.
Files for Changing the Demo	
`DemoFPGAAurora8b10b.slx`	Provides the FPGA model.
`aurora_8b10b_top_config_ds6601.m` `aurora_8b10b_top_config_ds6602.m`	Provides the configuration for the Black Box. The file name must match the FPGA base board used.
`aurora_8b10b_top.vhd`	Provides the top level design entry for the Black Box.
`aurora_8b10b_top_sim.vhd`	Provides a dummy design for the Black Box for offline simulation. The outports of the Black Box are set to 0 during offline simulation to avoid error messages concerning the data types.
`aurora_8b10b_master.edn`	Provides the Aurora master net list. The master net list is used to instantiate the GTH master transceiver.
`aurora_8b10b_slave.edn`	Provides the Aurora slave net list. The slave net list is used to instantiate the three GTH slave transceivers.
`axis_fifo.edn`	Provides the net list of the added FIFOs. FIFOs are implemented to the Black Box for buffering. Four FIFOs for reception and four FIFOs for transmission are used in total.

Related topics

Basics

Notes on Implementing Customized Protocols

File ending of net lists

The file ending of net list files must be `.edn` if you use net lists for the Xilinx Black Box.

Offline simulation of the Black Box

During offline simulation, the data types of the Xilinx Black Box outports can lead to error messages.

To avoid the error messages, you can provide a dummy design with the outports set to 0 for offline simulation. The `aurora_8b10b_top_sim.vhd` file of the Aurora8b10b demo is an example for such a dummy design.

Aurora-specific transmission

The last frame of a Aurora transmission must be flagged, for example, with the TX_LAST flag in the Aurora8b10b demo.

The receiver outputs a frame when it receives a new frame. If the last frame is not flagged, the frame remains in the memory of the receiver.

Related topics

Basics

Customized Protocol Demo..107

Implementing the Processor Interface to the Processor Model

Introduction

The processor interface is the counterpart of the FPGA interface to exchange data between the processor application and the FPGA application.

Where to go from here

Information in this section

Information in other sections

How to Generate a Processor Interface

Objective

The RTI blocks in the processor model for exchanging data with the FPGA application can be generated with all the relevant settings

Basics

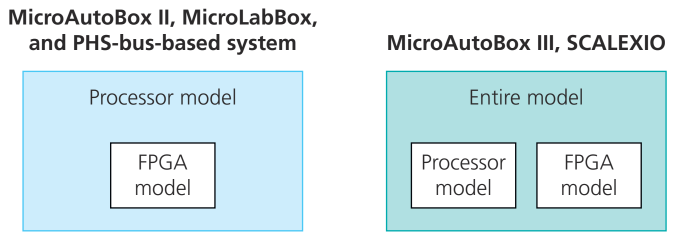
Usually, you implement the FPGA subsystem first, and then the interface to the processor model. This means that all information required for the data exchange is already specified in the FPGA interface blocks. It therefore makes sense to generate the corresponding preconfigured interface blocks automatically instead of adding them to the processor model and configuring them manually.

The information that is required to generate the processor interface is contained in the FPGA subsystem and the FPGA model INI file that is available as build result after the build process (see How to Build FPGA Applications (MicroAutoBox II, MicroLabBox, PHS-Bus-Based System) on page 128). The processor interface can therefore be generated from either the FPGA subsystem or the FPGA model INI file.

Preconditions

The following preconditions must be fulfilled:

- The FPGA_XDATA_READ_BL, FPGA_XDATA_WRITE_BL and FPGA_INT_BL blocks in the FPGA subsystem should be completely configured before generating the corresponding blocks for the processor model.
- The processor model must not be a part of the FPGA model as shown below.


MicroAutoBox II, MicroLabBox, and PHS-bus-based system
Processor model
FPGA model
MicroAutoBox III, SCALEXIO
Entire model
Processor model
FPGA model

Possible methods

The method to generate the processor interface depends on the platform used:

- To generate a processor interface for a MicroAutoBox II, a MicroLabBox, or a PHS-bus-based system, refer to Method 1.
- To generate a processor interface for a MicroAutoBox III or a SCALEXIO system, refer to Method 2.

Method 1

To generate a processor interface for MicroAutoBox II, MicroLabBox, or a PHS-bus-based system

1 Add a PROC_SETUP_BL block to the processor model.

2 Double-click the block to open its dialog.

3. Specify the number of FPGA boards that are available in your PHS-bus-based system to enable the required number of columns in the specification list below.
4. Select the FPGA subsystem whose model interface you want to generate.

   If you already built the FPGA application, you can also specify the FPGA model INI file after you have added the file to the FPGA model INI files list on the Advanced page.
5. On the Interface page, click Generate to generate the RTI blocks required for the processor interface.

Method 2

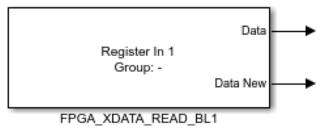
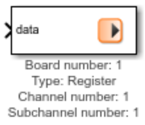
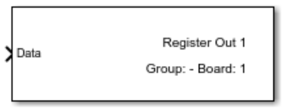
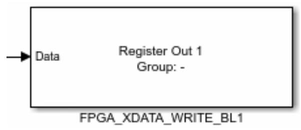
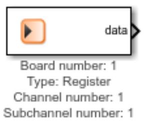
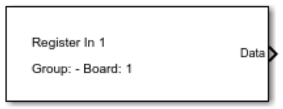
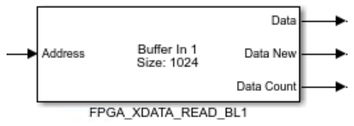
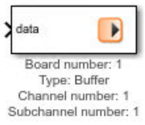
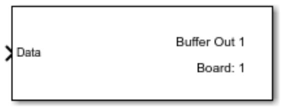
To generate a processor interface for a MicroAutoBox III or a SCALEXIO system

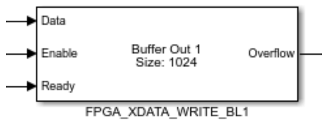
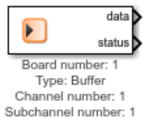
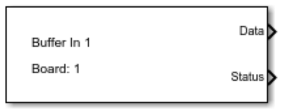
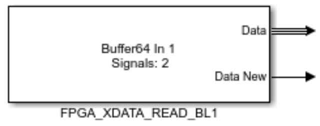
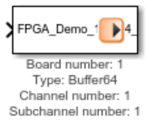
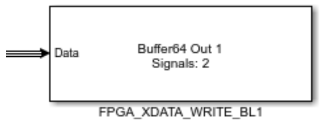
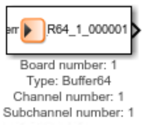
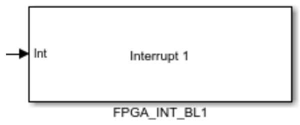
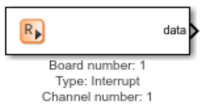
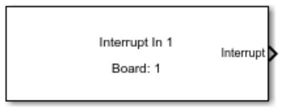
1. In the FPGA model, double-click the FPGA_SETUP_BL block to open its dialog.
2. On the ConfigurationDesk Interface page, click Generate to generate the RTI blocks required for the processor interface.

Result

You have created a new model containing RTI blocks to implement the processor interface to the processor model. The blocks are configured with all the relevant settings of their counterparts in the FPGA subsystem.

The generated blocks depend on the used platform.

FPGA Interface Block	**Generated Processor Interface Block**	
	MicroAutoBox III, SCALEXIO	**MicroLabBox, MicroAutoBox II, PHS-bus-based system**
Register		
 Data Register In 1 Group: - Data New FPGA_XDATA_READ_BL1	 data Board number: 1 Type: Register Channel number: 1 Subchannel number: 1	
	 data Board number: 1 Type: Register Channel number: 1 Subchannel number: 1	
Buffer		
 Address Buffer In 1 Size: 1024 Data Data New Data Count FPGA_XDATA_READ_BL1	 data Board number: 1 Type: Buffer Channel number: 1 Subchannel number: 1	

FPGA Interface Block	Generated Processor Interface Block	
	MicroAutoBox III, SCALEXIO	MicroLabBox, MicroAutoBox II, PHS-bus-based system
 Data Enable Ready Buffer Out 1 Size: 1024 Overflow FPGA_XDATA_WRITE_BL1	 data status Board number: 1 Type: Buffer Channel number: 1 Subchannel number: 1	
 Buffer64 In 1 Signals: 2 Data Data New FPGA_XDATA_READ_BL1	 FPGA_Demo_ Board number: 1 Type: Buffer64 Channel number: 1 Subchannel number: 1	–
 Data Buffer64 Out 1 Signals: 2 FPGA_XDATA_WRITE_BL1	 R64_1_000001 Board number: 1 Type: Buffer64 Channel number: 1 Subchannel number: 1	–
Interrupt		
	 data Board number: 1 Type: Interrupt Channel number: 1	

The MicroAutoBox III and SCALEXIO use the model port blocks of the Model Interface Package for Simulink.

The MicroLabBox, MicroAutoBox II, and PHS-bus-based system use processor interface blocks of the Processor Interface sublibrary of the RTI FPGA Programming Blockset.

Next steps

Copy the generated blocks to your processor model and connect them to the other Simulink blocks. You must not reconfigure these blocks manually. The settings for data type conversion (floating-point to fixed-point and vice versa) are implicit and can only be modified in the appropriate FPGA blocks.

After you copy the generated processor model interface blocks to the processor model, you can close the generated interface model without saving.

Related topics

Basics

Exchanging Data With the Processor Model.. 57

Model Interface Blockset (Model Interface Package for Simulink Reference 📖)

HowTos

How to Trigger Interrupt-Driven Processor Tasks.. 61

References

GenerateProcessorInterface (RTI FPGA Programming Blockset Script Interface Reference 📖)
GenerateProcInterfaceBlocks (RTI FPGA Programming Blockset Script Interface Reference 📖)
Processor Interface Blocks (MicroAutoBox III, SCALEXIO) (RTI FPGA Programming Blockset - Processor Interface Reference 📖)
Processor Interface RTI Blocks (MicroAutoBox II, MicroLabBox, PHS-Bus-Based System) (RTI FPGA Programming Blockset - Processor Interface Reference 📖)

Modeling Aspects for Multicore Processor Applications

Multicore processor applications for MicroLabBox

The RTI-MP Blockset allows you to assign parts of your processor model to different CPU cores. MicroLabBox provides two cores, so the processor model is separated into one master model and one slave model. You must implement the slave model as a subsystem of the master model. The inports and outports of the slave subsystem are connected with interprocessor communication blocks in the master model.

Processor model containing the FPGA model Either the master or the slave model must contain the FPGA model INI file with the FPGA bitstream. However, both models must contain a PROC_SETUP_BL block when you build the processor application.

Implementing the processor interface After you generated the processor interface, you move the interface blocks to the master and slave models as needed. For instructions on generating the processor interface, refer to How to Generate a Processor Interface on page 114.

For more information on modeling multicore processor applications, refer to Distributing the Model for MP Systems (RTI and RTI-MP Implementation Guide 📖).

Multicore processor applications for PHS-bus-based systems

The RTI-MP Blockset allows you to assign parts of your processor model to different processor cores. The processor model is separated into one master model and multiple slave models.

One FPGA application can support only one individual processor model, either the master model or one of the slave models. However, each individual model can access its own FPGA application. Implementing of the master and slave models is done in the same way as for multiprocessor applications. Refer to Modeling Aspects for RTI Multiprocessor Applications on page 118.

Multicore processor applications for MicroAutoBox III and SCALEXIO systems

You can implement an application where several single processor models can be linked to ConfigurationDesk. With this, you can build a multicore processor application (multicore real-time application) which can be downloaded to dSPACE real-time hardware to execute the models in parallel on single cores of the processor.

Modeling the processor models You implement the different processor models as subsystems to the entire model. Then, the different processor models can be separated before the build process starts. Refer to Workflow for Creating a Multicore Real-Time Application Using One Overall Behavior Model (ConfigurationDesk Real-Time Implementation Guide 📖).

Implementing the processor interface After you generate processor interface blocks to implement the processor interface, you move the blocks to the processor models as needed. Keep in mind that processor interface blocks residing in different processor models must not be mapped to the same function block. For more information, refer to Aspects on FPGA Applications Supporting Multicore Processor Applications on page 101.

Related topics

Basics

Details on MP Systems (RTI and RTI-MP Implementation Guide 📖)
Modeling Executable Applications and Tasks (ConfigurationDesk Real-Time Implementation Guide 📖)

Modeling Aspects for RTI Multiprocessor Applications

Introduction

There are some points to note when using the RTI FPGA Programming Blockset for a PHS-bus-based multiprocessor system.

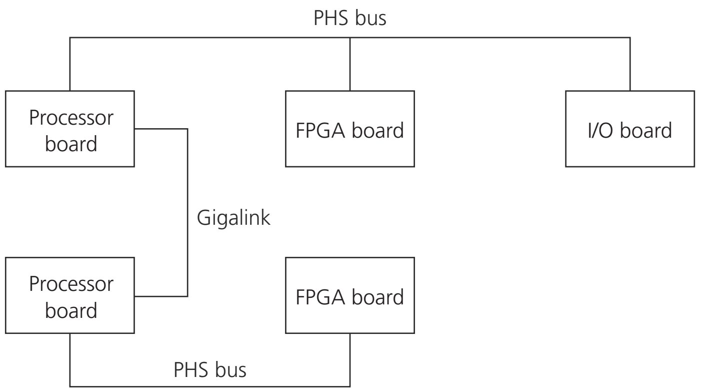
Basics on Models for Multiprocessor Systems

If your application requires very high computing power, you can use a dSPACE multiprocessor system consisting of two or more DS1006 or DS1007 boards. The RTI-MP Blockset allows you to assign parts of your model to different CPUs. The model is separated into one master model and several slave models which are connected to each other via blocks for interprocessor communication. For further information, refer to RTI and RTI-MP Implementation Guide 📖.

Topology of a multiprocessor system

In a multiprocessor system the processor boards of each PHS-bus-based system communicate with each other via interprocessor communication. The interprocessor communication is performed via Gigalink. Gigalink is a dSPACE-specific fiber-optic connection that allows bidirectional serial data transmission at high speed (1.25 Gbit/s).

The following illustration shows a typical multiprocessor topology.



Gigalink One Gigalink Module is required on each processor board to perform interprocessor communication. Each Gigalink module provides 4 Gigalinks to connect other processor boards. You can use any network topology for Gigalink.

PHS bus Each processor board is connected to its own PHS bus. Processor boards must not be connected to other processor boards via PHS bus.

Implementing a master MP model

The master MP model is implemented as described above. It contains the blocks from the **Processor Interface** library and a subsystem containing the FPGA model with the corresponding **FPGA Interface** blocks.

In addition to a processor model for a single-processor system, the processor model contains blocks for interprocessor communication, for example, IPC blocks. These blocks form the interface between a master model and a slave model.

Implementing a slave MP model

A slave MP model must be implemented as a subsystem. Its inports and outports are connected with interprocessor communication blocks in the master model. To include FPGA functionality for the FPGA board that is connected to the slave processor board, you must provide a further FPGA subsystem in the slave MP subsystem. This FPGA subsystem includes the **FPGA Interface** blocks, the **FPGA_SETUP_BL** block, and the Xilinx blocks. The slave MP model must contain the related blocks for the processor interface including a **PROC_SETUP_BL** block on its top level.

Related topics

HowTos

How to Build RTI-MP Applications for PHS-Bus-Based Systems.. 136

Updating the Processor Interface in a PHS-Bus-Based System

Assigning another FPGA board to the FPGA application

If you already implemented the processor interface in the processor model, and you want to assign the FPGA application to another FPGA board, you must use the **`Adapt`** command on the Interface page of the block dialog to set the new board number in the related FPGA subsystem. You can also check compatibility with the current FPGA subsystem by using the **`Adapt`** command. This will create a copy of the processor model that you can modify manually according to the displayed messages.

Note

If you use the **`Adapt`** command, the processor model must contain only processor interface blocks concerning one FPGA subsystem. Otherwise the FPGA board number will be replaced in any processor interface block used in the entire model.

Related topics

HowTos

How to Generate a Processor Interface........ 114

Simulating and Debugging Processor and FPGA Applications

Introduction

You can run the processor model and its FPGA subsystems in different simulation modes.

Where to go from here

Information in this section

Running Processor Models and FPGA Subsystems in Simulink Simulation Mode

Introduction

You can run the processor model and its FPGA subsystems in Simulink simulation mode (offline simulation) to test the model behavior.

Where to go from here

Information in this section

Simulating an FPGA Subsystem

Introduction

Before you execute the FPGA application on the real-time hardware, you should test its behavior in a simulated I/O environment.

Providing ports for simulation data

In a Simulink simulation (also called *offline simulation*), the implemented FPGA subsystem runs without connection to the external hardware. The simulation is based on the Simulink model and not on the generated model code. The input signals and output signals of the model interface must be simulated within the model. For this purpose, the **FPGA Interface** blocks provide simulation ports that you can enable for simulation. The simulation ports and their connected model components are ignored when you build the FPGA application.

The enabled simulation ports of the **FPGA_IO_READ_BL** and **FPGA_IO_WRITE_BL** blocks can be connected to an I/O model that simulates the expected real input and output signals applied to the I/O pins of the FPGA board.

The enabled simulation ports of the **FPGA_XDATA_READ_BL** and **FPGA_XDATA_WRITE_BL** blocks can be connected to Simulink Source and Sink blocks to simulate the FPGA model independently from the processor model. There is no need to implement the corresponding processor interface blocks for simulation.

Specifying the offline simulation period

In the Parameters page of the FPGA_SETUP_BL block, you can specify the offline simulation period. This is the step size that will be used for the simulation. If you have specified *fixed step size*, the offline simulation period must be an integer multiple of the fixed step size specified in the Simulink Configuration Parameters. Officially, the Xilinx System Generator supports only *variable step size*.

Example If you have specified a fixed step size of 1 µs in the Simulink Configuration Parameters, the offline simulation period can be greater than or equal to 1 µs, for example 1 ms.

Note

There are some Xilinx blocks, for example, the Constant and Counter blocks, for which you can specify a sampling rate in their block dialogs.

If there is a difference between the specified offline simulation period and the block's explicit period, the block period is calculated by:

```
HWBlockPeriod = SimulinkBlockPeriod · HWClockPeriod /
                OfflineSimulationPeriod
```

For example, if you have specified the explicit period of a Counter block with 10 µs and the offline simulation period with 1 µs, the hardware block period is 100 ns for a hardware clock period of 10 ns.

Starting the simulation

You can start the simulation in the model window of the FPGA subsystem by clicking the Start simulation toolbar button, choosing Simulation - Run from the menu bar or using the `Ctrl+T` shortcut.

The simulation is executed for the specified simulation time or until you stop it.

Related topics

Basics

Simulating a Processor Model

Introduction

Before you execute the processor application on the real-time hardware, you should test its behavior in a simulated environment.

Simulating a MicroAutoBox II, MicroLabBox, or PHS-bus-based system

If you simulate the processor model of a MicroAutoBox II, a MicroLabBox, or a PHS-bus-based system, you can do this in different model modes.

Simulating in FPGA-Build / Offline simulation model mode The processor model and its FPGA subsystems are simulated (see also Simulating an FPGA Subsystem on page 122).

You can simulate the interaction between the processor model and the FPGA model in offline mode without modifying the models. The simulation data is delivered by the specified simulation ports or, if these ports are not available, by the corresponding interface blocks in the model.

Simulating in Processor-Build model mode The processor model only contains the blocks from the processor model. The FPGA subsystems are removed from the model. The simulation can therefore be used only to test the behavior of the processor model, and there is no simulated data exchange with the FPGA subsystems.

Simulating a MicroAutoBox III or a SCALEXIO systems

You can simulate the interaction between the processor model and the FPGA model in offline mode without modifying the models. The simulation data is delivered by the specified simulation ports or, if these ports are not available, by the corresponding interface blocks in the model.

Starting the simulation

You can start the simulation in the model window of the processor model by clicking the Start simulation toolbar button, choosing Simulation - Run from the menu bar or using the `Ctrl+T` shortcut.

The simulation is executed for the specified simulation time or until you stop it.

Related topics

Basics

Simulating and Debugging Processor and FPGA Applications.. 121

Debugging Processor Models and FPGA Subsystems

Introduction

There are a lot of standard tools in Simulink that help you find mistakes in your processor model, for example, unconnected signals or incorrect data type conversions. The Xilinx Design Suite provides special tools for analyzing FPGA systems.

Analyzing FPGA Models

Introduction

Xilinx provides tools for timing analysis, resource utilization analysis, and HDL simulation.

Timing analysis

If the build process for your FPGA subsystem has detected timing problems, use the timing analysis tool from the Xilinx System Generator.

Timing analysis of the entire FPGA application The build process automatically starts a timing analysis of the entire FPGA application including the framework for the used platform. In the MATLAB Command Window, the build process outputs a link to the results of the analysis. Click the link to open the results in Xilinx's Timing Analyzer or select the FPGA model action Show Last Timing Report on the Parameters page of the FPGA_SETUP_BL block.

For building the FPGA application, refer to Building FPGA and Processor Applications on page 127.

Timing analysis of the custom FPGA model You can start the timing analysis on the Parameters page of the FPGA_SETUP_BL block in your FPGA subsystem after selecting it from the list of FPGA model actions. This timing analysis considers only the custom FPGA model without the framework of the platform that is automatically added to your FPGA model during the build process.

The result of the analysis is displayed in Xilinx's Timing Analyzer and saved to the `<ModelName>_timing` folder in the working folder. To show the last report, select the FPGA model action Show Last Timing Report on the Parameters page of the FPGA_SETUP_BL block.

Resource utilization analysis

If the build process for your FPGA subsystem has detected FPGA resource problems, use the resource utilization analysis tool from the Xilinx System Generator.

Resource analysis of the entire FPGA application The build process automatically starts a resource analysis of the entire FPGA application including the framework for the used platform. In the MATLAB Command Window, the build process outputs a link to the results of the analysis. Click the link to open the results in Xilinx's Resource Analyzer or select the FPGA model action Show Last Resource Utilization Report on the Parameters page of the FPGA_SETUP_BL block.

Resource analysis of the custom FPGA model You can start the resource analysis on the Parameters page of the FPGA_SETUP_BL block in your FPGA subsystem after selecting it from the list of FPGA model actions. This resource analysis considers only the custom FPGA model without the framework of the platform that is automatically added to your FPGA model during the build process.

The result of the analysis is displayed in Xilinx's Resource Analyzer and saved to the `<ModelName>_resource` folder in the working folder. To show the last report, select the FPGA model action Show Last Timing Report on the Parameters page of the FPGA_SETUP_BL block.

HDL simulation

If you completed a Simulink simulation and are satisfied with the behavior of the model, the next step is to compare the simulation data from the FPGA model with the simulation data from generated HDL code. To do so, you can start the HDL Simulation on the Parameters page of the FPGA_SETUP_BL block in your FPGA subsystem after selecting it from the list of FPGA model actions.

Note

The HDL simulation only starts if the simulation time is not infinite.

When the simulation has finished, you can look at a textual report. The results are also stored in the `HDLAnalysis` folder in the working folder.

Further information

For more information on timing analysis, resource utilization analysis, and HDL simulation, refer to the Xilinx user documentation.

Related topics

Basics

Simulating and Debugging Processor and FPGA Applications.. 121

Building FPGA and Processor Applications

Introduction

If the behavior of your model in Simulink simulation fulfills your requirements, you can generate the model code and build the executable files from the processor model and the FPGA subsystems.

Where to go from here

Information in this section

Building the Applications (MicroAutoBox II, MicroLabBox, PHS-Bus-Based System)

Introduction

The RTI FPGA Programming Blockset manages the build process of the FPGA application and the processor application for a MicroAutoBox II, a MicroLabBox, or a PHS-bus-based system.

Where to go from here

Information in this section

How to Build FPGA Applications (MicroAutoBox II, MicroLabBox, PHS-Bus-Based System)

Objective

Before you can execute the FPGA model on the real-time hardware, you must build an application from its generated code.

Basics

The code generation and the build process are managed by the RTI FPGA Programming Blockset. It uses the Xilinx System Generator to generate VHDL code and starts the Xilinx implementation process, including synthesis, mapping, routing and generating the bitstream file.

Even if no errors occurred during modeling, simulating, and building, your FPGA application is not guaranteed to work correctly. Some combination of factors might prohibit the execution of your FPGA application, e.g., combinations of the operating temperature, the power of the entire hardware system, technological limitations of the FPGA, the sample rate, etc. Some FPGA applications do not start at all, others start correctly but fail during operation.

Due to the complexity of FPGA applications and their dependency on the used FPGA hardware, the Xilinx design tools cannot ensure that all possible design problems are detected during the analysis and simulation of the FPGA model.

To solve the problem, you have to modify the FPGA model and make sure that you consider generally accepted FPGA design rules to ensure a stable and reliable FPGA application (refer to Audience profile on page 9). For more information, contact dSPACE Support.

Using the script interface for building an FPGA application

The RTI FPGA Programming Blockset provides a script interface that you can use to start the FPGA build process in the MATLAB Command Window.

The syntax for the FPGA build is:

```
[listOfAppIDs, listOfiniFileRefs, pModelPath, errorcode] =
 rtifpga_scriptinterface('FPGABuild',<SimulinkHandle>)
```

The script considers all subsystems that are contained in the model/subsystem which is specified by the Simulink handle.

Preconditions

- Your platform is a PHS-bus-based system, MicroAutoBox II, or MicroLabBox.
- The FPGA subsystem that you want to build must be implemented correctly, for example, the **FPGA Interface** blocks must be configured correctly.
- If you use Xilinx **Black Box** blocks to incorporate hardware description language models, the files of the incorporated models are located in the model root folder or in an `Includes` subfolder in the model root folder. The `Includes` subfolder and optional subfolders must be added to the MATLAB search paths as relative paths. For more information, refer to Implementing the FPGA Functionality on page 45.
- The build process is executed only if the versions of the installed Xilinx tools are suitable for the RTI FPGA Programming Blockset. For suitable software tools, refer to Software Tools for Working With the RTI FPGA Programming Blockset on page 19.
- Switching to the **Processor-Build** model mode is only possible if all the specified FPGA subsystems has been built before.

Note

An FPGA application must be build with a variable-step solver.
If you have specified a fixed-step solver it is automatically switched.

Possible methods

You can build an FPGA application in different ways:

- In the FPGA_SETUP_BL block, you can start the build process for the FPGA subsystem it belongs to. Refer to Method 1.
- Only PHS-bus-based systems and MicroAutoBox: In the PROC_SETUP_BL block, you can start the build process for a specified set of FPGA subsystems. Refer to Method 2.
- In the MATLAB Command Window, you can use the script interface from the RTI FPGA Programming Blockset to build all the available FPGA subsystems from the specified Simulink model handle. Refer to Method 3.
- You can use an FPGA Build Server to build the FPGA application. Refer to Using an FPGA Build Server on page 151.

Method 1

To build an FPGA application via FPGA_SETUP_BL block

1 Double-click the FPGA_SETUP_BL block in the FPGA subsystem to open the block's dialog and switch to its Parameters page.

2 Select FPGA Build as FPGA model action and click Execute to start the build process.

Method 2

To build an FPGA application via PROC_SETUP_BL block

1 Double-click the PROC_SETUP_BL block in the processor model to open the block's dialog.

2 On the Unit page, set the Include option from the FPGA build column for each FPGA subsystem which you want to build an FPGA application for.

The Include option is disabled if:

- You specified an FPGA model INI file.

  The FPGA application was already built.
- The model mode is set to Processor-Build.

  In this mode, you can only build the processor application. To allow the building of FPGA applications, you must switch to the FPGA-Build / Offline Simulation mode on the Model Configuration page.

3 Click Build to start the build processes for the selected FPGA subsystems. The build processes are executed sequentially.

Method 3

To build an FPGA application via script interface

1 Use the script interface and execute the `FPGABuild` script function by entering it directly in the command line of the MATLAB Command Window or referencing an M-file that contains the script function.

Example: The following script will start an FPGA build process for any FPGA subsystems found in the processor model that is called *MyProcModel*.

```
ProcModelHandle = get_param('MyProcModel','handle')
rtifpga_scriptinterface('FPGABuild',ProcModelHandle)
```

Result

The build process opens a temporary model `<FPGAModelName>_rtiFPGAtmp` and closes the temporary model when the FPGA code is generated.

You will find the FPGA model INI file in `<FPGABuildFolder>/<ModelName>_rtiFPGA/ini/<FPGAApplicationName>_<ApplicationID>.ini`. *ApplicationID* is a 56-bit identifier which is uniquely created at each build process.

Steps of the build process Because the build process might take hours, it is helpful to know which steps will be processed:

- Starting model preparation for code generation …
- Updating model …
- Starting System Generator code generation …
- Starting synthesis of System Generator output files …
- Starting FPGA build ...
- Starting XSG Synthesis ...
- Starting Framework Synthesis ...
- Waiting for Framework Synthesis ...
- Starting Implementation ...
- Waiting for Implementation ...
- Starting Timing Report ...
- Starting Utilization Report ...
- Starting Bitgen ...
- Waiting for Bitgen ...
- Starting Post Bitgen ...
- Starting Generate Ini ...
- Waiting for Utilization Report ...
- Waiting for Timing report generation ...
- Starting Post Build ...
- Show Timing Report of complete FPGA
- Show Resource Utilization Report of complete FPGA
- WORK DIRECTORY: <WorkFolderPath>

  BUILD DIRECTORY: <BuildFolderPath>

  RESULT FILE: <FPGAApplicationIniFile>

  ...

  FPGA Build Done

  Elapsed time is <ElapsedTime> seconds.

For debugging the build process and getting detailed design information, for example, the mapping report, you can find the build log files in the `<FPGABuildFolder>/<ModelName>_rtiFPGA/<FPGAApplicationName>_<ApplicationID>/sysgen/` subfolder.

In the next step you must specify how the processor application has to handle the FPGA application. You can program it into the RAM of the FPGA or into the flash of the FPGA board (see How to Create Burn Applications on page 138).

Related topics

Basics

Introduction to the Script Interface of the RTI FPGA Programming Blockset (RTI FPGA Programming Blockset Script Interface Reference 🕮)

How to Build Single-Core Processor Applications (MicroAutoBox II, MicroLabBox, PHS-Bus-Based System)

Objective

To build a single-core processor application that includes and supports your FPGA application.

Basics

The code generation and the build process are managed by the Simulink Coder. For general information, refer to Building and Downloading the Model (RTI and RTI-MP Implementation Guide 🕮).

Note

Before you start the build process for the processor application, you must set the Fixed-step size setting on the Solver page to a value that is the multiple of the offline simulation period specified in the FPGA_SETUP_BL block.

Model separation

The processor model that you want to build must not contain an FPGA subsystem. Otherwise the build process will terminate with an error message.

The RTI FPGA Programming Blockset provides two methods to remove an FPGA subsystem from the processor model only for the time of building the processor application.

Switching the model mode On the Model Configuration page of the PROC_SETUP_BL block, you can choose between the FPGA-Build / Offline Simulation and the Processor-Build model modes. If you switch to the Processor-Build model mode, a copy of the processor model is created before the FPGA subsystems are removed. The copy is saved as `<ProcModelName>_rtiFPGASeparationFile.mdl` in the same folder as the original model. If you switch back to the FPGA-Build / Offline Simulation model mode, the copy is used to restore the model. The copy is not deleted in the file system.

Note

- You cannot switch to the Processor-Build model mode without having built the FPGA application beforehand.
- If you want to give the model to another person, you must provide both models: the model that was saved in the Processor-Build model mode and its separation file. Alternatively, you can switch to the FPGA-Build / Offline Simulation model mode before you transmit the MDL file of the processor model.
- If you save the model in the Processor-Build model mode with a new path or a new name, restoring the original model will not work.

Using the script interface for separating the processor model The script interface of the RTI FPGA Programming Blockset can be used to separate the processor model before you start the build process. It works in the same way as the model mode switch via the PROC_SETUP_BL block dialog.

The syntax for separation is:

```
[errorcode, path] = rtifpga_scriptinterface('Separation',
 <ProcModelHandle>)
```

- errorcode = 0: Operation finished successfully
- errorcode != 0: Operation terminated with an error
- path: Returns the path of the separation file

You can get the handle of the processor model by using:

```
ProcModelHandle = get_param(<ProcessorModelName>,'handle')
```

You can restore the processor model by using `Deseparation` instead of `Separation` as the first argument.

The syntax for deseparation is:

```
[errorcode] = rtifpga_scriptinterface('Deseparation',
 <ProcModelHandle>)
```

- errorcode = 0: Operation finished successfully
- errorcode != 0: Operation terminated with an error

The script interface can also be used for building the FPGA application (see How to Build FPGA Applications (MicroAutoBox II, MicroLabBox, PHS-Bus-Based System) on page 128).

Preconditions

- Your platform is a MicroAutoBox II, MicroLabBox, PHS-bus-based system.
- The FPGA model INI file must be available before you start the build process.
- The working folder must be set to the model path.
- The processor model to be built must be opened in MATLAB.
- If you want to download the processor application after the build process, the real-time hardware must be registered, for example, via the Platform Manager in ControlDesk.

Method

To build a single-core processor application

1 Prepare the processor model for the build process. Either switch the model mode to Processor-Build in the PROC_SETUP_BL block or use the script interface to separate the FPGA subsystems from the processor model. For details, see above.

2 On the Unit page of the PROC_SETUP_BL block, select the required programming option. If you specify not to program the related FPGA application into flash or RAM, the FPGA application must be available in the FPGA board's flash memory. Otherwise, the processor application will terminate after the FPGA board's initialization phase.

> **Tip**
>
> **Using several DS5203 FPGA boards in a PHS-bus-based system**
>
> The boards are numbered in order of their PHS-bus addresses. The board with the lowest PHS-bus address is number 1.

3 Open the Code Generation dialog, specify the build options and check the configuration, for example, the selected target file and the specified fixed step size.

4 Click Build in the Code Generation dialog to start the build process.

Result

If the build process finishes successfully, you will find the build results, such as the processor application file and the variable description file, in the working folder. If you have specified to program the FPGA application into flash or RAM, the flash or the FPGA is automatically programmed when the processor application is loaded. For more information on programming the FPGA, refer to How to Create Burn Applications on page 138.

For example, if you built the *MyDemo* application for a PHS-bus-based system with a DS1007 PPC Processor Board, you will find the executable PPC file in the working folder and a subfolder called `MyDemo_rti1007` containing the generated code and all the intermediate build results.

Related topics

HowTos

How to Build Processor Applications (MicroAutoBox III, SCALEXIO).. 149

References

AddIniFile (RTI FPGA Programming Blockset Script Interface Reference 🕮)
Deseparation (RTI FPGA Programming Blockset Script Interface Reference 🕮)
GetFPGABlocks (RTI FPGA Programming Blockset Script Interface Reference 🕮)
RemoveIniFile (RTI FPGA Programming Blockset Script Interface Reference 🕮)
Separation (RTI FPGA Programming Blockset Script Interface Reference 🕮)

How to Build Multicore Processor Applications for MicroLabBox

Objective

To build a multicore processor application that includes and supports your FPGA application.

Basics

The code generation and the build process are managed by the Simulink Coder. For general information, refer to Building and Downloading the Model (RTI and RTI-MP Implementation Guide 📖).

Note

Before you start the build process for the processor application, you must set the Fixed-step size setting on the Solver page to a value that is the multiple of the offline simulation period specified in the FPGA_SETUP_BL block.

Effects on FPGA signal tracing

The processor model that programs the FPGA triggers the data acquisition of FPGA signals with the task period of the subsystem in which the PROC_SETUP_BL block is located. You can specify which processor model programs the FPGA when you configure the build process.

Preconditions

- The FPGA model INI file must be available before you start the build process.
- The master model and the slave model must contain the PROC_SETUP_BL block.
- The working folder must be set to the model path.
- The processor model to be built must be opened in MATLAB.
- If you want to download the processor application after the build process, the real-time hardware must be registered, for example, via the Platform Manager in ControlDesk.

Method

To build a multicore processor application for MicroLabBox

1. Prepare the processor model including the FPGA subsystem for the build process. Either switch the model mode to Processor-Build in the PROC_SETUP_BL block or use the script interface to separate the FPGA subsystems from the processor model. For more information, refer to Model separation on page 132.
2. Open the PROC_SETUP_BL block dialog of the processor model that you want to program the FPGA. This could be the PROC_SETUP_BL block dialog of the master model or of the slave model.
3. On the Unit page, select the FPGA model INI file. If the file is not selectable, add the FPGA model INI file to the Advanced page.
4. Select the programming option into ram or into flash and click OK.

5 Open the **PROC_SETUP_BL** block dialog of the other processor model.

6 On the **Unit** page, select the FPGA model INI file. If the file is not selectable, add the FPGA model INI file to the **Advanced** page.

7 Clear the programming option (select - - -) and click **OK**.

8 Start the build process with the **Multiprocessor Setup** dialog. For more information, refer to Basics on the Build and Download (RTI and RTI-MP Implementation Guide 🕮).

Result

If the build process finishes successfully, you will find the build results, such as the processor application file and the variable description file, in the working folder. The FPGA is automatically programmed when the processor application is loaded.

Related topics

Basics

Details on MP Systems (RTI and RTI-MP Implementation Guide 🕮)
Modeling Aspects for Multicore Processor Applications..117

How to Build RTI-MP Applications for PHS-Bus-Based Systems

Objective

To build a multiprocessor or multicore application that includes and supports your FPGA applications in a PHS-bus-based system.

Basics

In a PHS-bus-based multiprocessor or multicore system, each FPGA board can be accessed only by one processor board/core. Therefore each processor application has its own FPGA application.

The code generation and the build process are managed by the Simulink Coder. For general information, refer to Building and Downloading the Model (RTI and RTI-MP Implementation Guide 🕮).

Note

Before you start the build process for the processor application, you must set the **Fixed-step size** setting on the **Solver** page to a value that is the multiple of the offline simulation period specified in the **FPGA_SETUP_BL** block.

Preconditions

- The FPGA model INI files for each processor application must be available before you start the build process.

- The master and the slave models must contain the PROC_SETUP_BL block.
- The working folder must be set to the model path.
- The master processor model to be built must be opened in MATLAB.
- If you want to download the multiprocessor application after the build process, the real-time hardware must be registered, for example, via the Platform Manager in ControlDesk.

Method

To build a RTI-MP application for a PHS-bus-based system

1. Prepare the master and slave processor models including the FPGA subsystems for the build process. Either switch the model mode to Processor-Build in the PROC_SETUP_BL blocks or use the script interface to separate the FPGA subsystems from the processor models. For more information, refer to Model separation on page 132.
2. Open the PROC_SETUP_BL block dialog of a master or slave processor model.
3. On the Unit page, select the FPGA model INI file for the current processor model. If the file is not selectable, add the FPGA model INI file to the Advanced page.
4. Select the programming option into ram or into flash and click OK.
5. Repeat step 2 ... 4 for all other processor models.
6. Start the build process with the Multiprocessor Setup dialog. For more information, refer to Basics on the Build and Download (RTI and RTI-MP Implementation Guide 🕮).

Result

If the build process finishes successfully, you will find the build results, such as the processor application file and the variable description file, in the working folder. The FPGA is automatically programmed when the processor applications are loaded.

Related topics

Basics

Basics on the Build and Download (RTI and RTI-MP Implementation Guide 🕮)
Details on MP Systems (RTI and RTI-MP Implementation Guide 🕮)
Modeling Aspects for RTI Multiprocessor Applications.. 118

How to Create Burn Applications

Objective

You can explicitly program the FPGA with a burn application.

Note

This instruction is relevant only for a MicroAutoBox II or a PHS-bus-based system.

Basics

A burn application is used to load an FPGA application to the FPGA. Usually, the FPGA programming is included in the processor application and there is no need to use a burn application. But in some cases, it is useful to program the FPGA explicitly.

Note

If you load a processor application that was built without specifying a programming option for the real-time hardware, and there is no application running on the FPGA, the processor application is terminated after the initialization phase of the FPGA board.

Programming into flash If you use the flash memory of the FPGA board, you must execute the burn application only once. Each time you start the FPGA board, the FPGA application is loaded to the FPGA automatically. In this case, the processor application should be built without specifying a programming option. This gives you the shortest startup time for your system. If you want to load a new FPGA application to the flash, you must execute the burn application to replace the flash contents and restart the system to program the FPGA from the flash with the new application.

Note

When programming into flash, you must consider the Autoboot features of your hardware.

- DS5203 FPGA Board
  The board's Autoboot jumper must be set accordingly, refer to DS5203 FPGA Board (PHS Bus System Hardware Reference 📖).
- MicroAutoBox II
  The board's Autoboot feature can be enabled and disabled via the MicroAutoBox Configuration Tool (`<RCP_HIL_InstallationPath>/Exe/DS1401_ConfigGUI.exe`) or via RTLib, refer to fpga_tp1_enable_autoboot (MicroAutoBox II RTLib Reference 📖) and fpga_tp1_disable_autoboot (MicroAutoBox II RTLib Reference 📖).

Programming into RAM If you use the RAM memory of the FPGA board, the FPGA must be programmed every time you start the system. It is therefore recommended to build the processor application with the RAM programming option to program the FPGA application automatically when loading the processor application. A burn application for the RAM is useful if you want to test the FPGA application only. If you program into the RAM memory, any running FPGA application is immediately stopped and replaced with the new FPGA application, unlike flash programming, where you must restart the board to activate the new FPGA application.

Preconditions

- Your platform is a PHS-bus-based system or MicroAutoBox II.
- The model must be in the **Processor-Build** model mode.

Method

To create a burn application

1. Double-click the PROC_SETUP_BL block in the processor model to open its dialog.
2. On the **Unit** page, choose a programming option for each of the specified FPGA applications.
3. Click **Create burn application** to start the build process for the burn application. The Simulink Coder is used to manage the build process.

Result

If the build process succeeded, the build result can be found in `<ModelName>_rtiFPGA/burnapplication`. The executable is not automatically downloaded to the processor board. The intermediate build results can be found in `<ModelName>_rtiFPGA/burnapplication/<ModelName>_burnapplication_rti<XXXX>`.

For example, if you built the burn application for the *MyDemo* model, and the system target is a PHS-bus-based system with a DS1007 PPC Processor Board, you will find the executable PPC file in `MyDemo_rtiFPGA/burnapplication` and the intermediate build results in `MyDemo_rtiFPGA/burnapplication/MyDemo_burnapplication_rti1007`.

For information how to load a burn application, refer to Experimenting with an FPGA Application on page 163.

Related topics

Basics

Experimenting with an FPGA Application.. 163

References

CreateBurnApplication (RTI FPGA Programming Blockset Script Interface Reference 🕮)

Building the Applications (MicroAutoBox III, SCALEXIO)

Introduction

The RTI FPGA Programming Blockset manages the build process of the FPGA application for a MicroAutoBox III or a SCALEXIO system. To build the real-time application, you have to use ConfigurationDesk.

Where to go from here

Information in this section

How to Build FPGA Applications (MicroAutoBox III, SCALEXIO)

Objective

Before you can execute the FPGA model on a MicroAutoBox III or SCALEXIO system, you must build an application from its generated code.

Basics

The code generation and the build process are managed by the RTI FPGA Programming Blockset. It uses the Xilinx System Generator to generate VHDL code and starts the Xilinx implementation process, including synthesis, mapping, routing and generating the bitstream file.

Even if no errors occurred during modeling, simulating, and building, your FPGA application is not guaranteed to work correctly. Some combination of factors might prohibit the execution of your FPGA application, e.g., combinations of the operating temperature, the power of the entire hardware system, technological limitations of the FPGA, the sample rate, etc. Some FPGA applications do not start at all, others start correctly but fail during operation.

Due to the complexity of FPGA applications and their dependency on the used FPGA hardware, the Xilinx design tools cannot ensure that all possible design problems are detected during the analysis and simulation of the FPGA model.

To solve the problem, you have to modify the FPGA model and make sure that you consider generally accepted FPGA design rules to ensure a stable and reliable FPGA application (refer to Audience profile on page 9). For more information, contact dSPACE Support.

Preconditions

- Your platform is a MicroAutoBox III or a SCALEXIO system.
- The FPGA subsystem that you want to build must be implemented correctly, for example, the **FPGA Interface** blocks must be configured correctly.
- If you use Xilinx **Black Box** blocks to incorporate hardware description language models, the files of the incorporated models are located in the model root folder or in an `Includes` subfolder in the model root folder. The `Includes` subfolder and optional subfolders must be added to the MATLAB search paths as relative paths. For more information, refer to Implementing the FPGA Functionality on page 45.
- The build process is executed only if the versions of the installed Xilinx tools are suitable for the RTI FPGA Programming Blockset. For suitable software tools, refer to Software Tools for Working With the RTI FPGA Programming Blockset on page 19.

Note

An FPGA application must be build with a variable-step solver.
If you have specified a fixed-step solver it is automatically switched.

Possible methods

You can build an FPGA application in different ways:

- In the **FPGA_SETUP_BL** block, you can start the build process for the FPGA subsystem it belongs to. Refer to Method .
- You can use an FPGA Build Server to build the FPGA application. Refer to Using an FPGA Build Server on page 151.

Method

To build an FPGA application via FPGA_SETUP_BL block

1. Double-click the **FPGA_SETUP_BL** block in the FPGA subsystem to open the block's dialog and switch to its **Parameters** page.
2. Select **FPGA Build** as FPGA model action and click **Execute** to start the build process.

Result

The build process opens a temporary model `<FPGAModelName>_rtiFPGAtmp` and closes the temporary model when the FPGA application is built.

You will find the FPGA model INI file according to the settings on the **Parameters** page of the **FPGA_SETUP_BL** block in

`<FPGABuildFolder>/<ModelName>_rtiFPGA/ini/<FPGAApplicationName>_<ApplicationID>.ini`. *ApplicationID* is a 56-bit identifier which is uniquely created at each build process.

Steps of the build process Because the build process might take hours, it is helpful to know which steps will be processed:

- Starting model preparation for code generation …
- Updating Model …
- Starting System Generator code generation …
- Starting generation of Custom IO Function ...
- Generating XML file for Custom IO Function ...
- Generating driver code for Custom IO Function ...
- Generating XML file for Custom IO Function ...
- Generating driver code for Custom IO Function ...
- Custom function code created: <CustomFunctionPath>
- Starting synthesis of System Generator output files …
- Starting FPGA build ...
- Starting XSG Synthesis ...
- Starting Framework Synthesis ...
- Waiting for Framework Synthesis ...
- Waiting for XSG Synthesis ...
- Starting Implementation ...
- Waiting for Implementation ...
- Starting Timing Report ...
- Starting Utilization Report ...
- Starting Bitgen ...
- Waiting for Bitgen ...
- Starting Post Bitgen ...
- Starting Generate Ini ...
- Waiting for Utilization Report ...
- Waiting for Timing report generation ...
- Starting Post Build ...
- Show Timing Report of complete FPGA
- Show Resource Utilization Report of complete FPGA
- WORK DIRECTORY: <WorkFolderPath>
  BUILD DIRECTORY: <BuildFolderPath>
  RESULT FILE: <FPGAApplicationIniFile>
  ...
  FPGA Build Done
  Elapsed time is <ElapsedTime>.

For debugging the build process and getting detailed design information, for example, the mapping report, you can find the build log files in the `<FPGABuildFolder>/<ModelName>_rtiFPGA/<FPGAApplicationName>_<ApplicationID>/sysgen/` subfolder.

Port-specific user functions in ConfigurationDesk After you generate the FPGA application, the `<FPGABuildFolder>/<ModelName>_rtiFPGA/customio_<ApplicationID>` folder contains templates for user functions that can be used in ConfigurationDesk to trigger port-specific actions, for example conversion of values or other computations.

- <FPGAApplicationName>_cfusr.h

  Provides header information and defines for enabling the port-specific functions.
- <FPGAApplicationName>_cfusr.cpp

  Provides port-specific function templates.

For each port that you have implemented in your FPGA model, three defines are available in `<FPGAApplicationName>_cfusr.h`:

```
// #define <Port>_JUMPOUT1_ENABLED
// #define <Port>_JUMPOUT2_ENABLED
// #define <Port>_STDCODE_DISABLED
```

With the `JUMPOUT1` define, you activate the user-specific port function `usercode1` in `<FPGAApplicationName>_cfusr.cpp` that is to be executed *before* the standard driver code.

With the `JUMPOUT2` define, you activate the user-specific port function `usercode2` in `<FPGAApplicationName>_cfusr.cpp` that is to be executed *after* the standard driver code.

With the `STDCODE` define, you disable the standard driver code.

Example of defines and functions prepared for the `Register_In_1` port:

```
…
/* -------- Register_In_1 ---------- */
// #define Register_In_1_JUMPOUT1_ENABLED
// #define Register_In_1_JUMPOUT2_ENABLED
// #define Register_In_1_STDCODE_DISABLED
void usercode1_Register_In_1(
   ucf_ApplicationNameStruct_T* pBlockInstance, dsfloat*pValue);
void usercode2_Register_In_1(
   ucf_ApplicationNameStruct_T* pBlockInstance, dsfloat pValue);
…
```

Note

If you have specified custom code in these files, be careful that you do not overwrite them with the newly generated files containing the default function templates. You have to merge their contents manually.

Related topics

Basics

How to Prepare the Processor Models for Separating (MicroAutoBox III, SCALEXIO)

Objective

You have to prepare the processor models in the entire model so that the framework can automatically separate the processor models from other parts of the entire model, such as the FPGA model.

Required preparations

Before the framework can export the processor models, you have to specify the following:

- Specify the processor models in the entire model.

  In a ConfigurationDesk project, the processor interfaces are a part of the behavior model. Other parts of the entire model, such as the FPGA application or stimulus signals of external I/O signals, are not part of the behavior model. Therefore, the framework separates the processor models from the entire model. To do this, you have to specify the processor models in the entire model.
- Specify a name and a folder for the separated processor models.

  In ConfigurationDesk, the separated processor models are implemented outside ConfigurationDesk in a Simulink model. When the framework exports the processor models to ConfigurationDesk, it implements the separated processor models that are saved at the specified location to ConfigurationDesk.

Separating multicore processor models

For more information on separating multicore processor models, refer to Workflow for Creating a Multicore Real-Time Application Using One Overall Behavior Model (ConfigurationDesk Real-Time Implementation Guide 📖).

Precondition

Your platform is a MicroAutoBox III or a SCALEXIO system.

Method

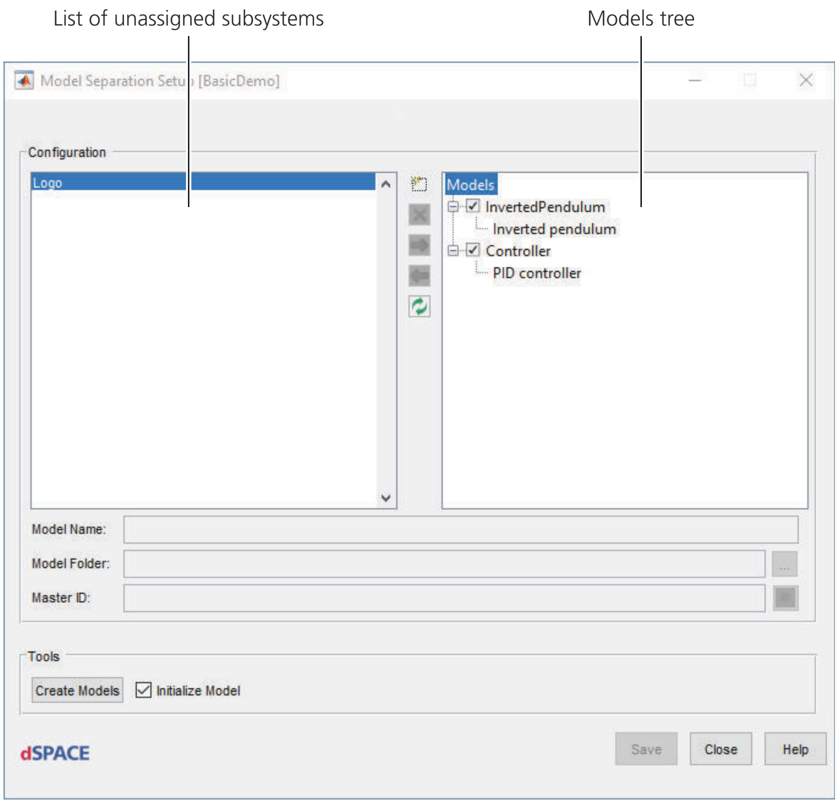
To prepare the processor model for separating

1 On the top-level of the entire model, add the Model Separation Setup block from the dSPACE Model Separation Block Library.

2 Open the block dialog.

The **Model Separation Setup** block analyzes the entire model and displays all top-level subsystems that can be separated as models.

3 Click ⊞ to add a new model entry in the **Models** tree.



4 In the **Model name** edit field, enter a name for the Simulink model that includes a processor model that must be separated.

5 In the **Model folder** edit field, enter a folder name or path for the separated processor model to be saved when the framework exports the processor model.

If you do not enter a folder or path, the framework saves the separated processor model to the folder of the entire model.

6 From the **Unassigned subsystems** list, select the subsystems of the processor model and click ⇐ to assign the subsystems to the model entry.

7 If the entire model includes more than one processor model to be separated, repeat steps 3 ... 6.

8 Click OK to close the dialog.

Result

Now the model is prepared for separating. The framework is able to process the following steps when you export the processor model:

- To separate the processor models from the entire model.
- To save the separated processor models as Simulink models with the specified name to the specified folder.

Next step

Now you can export the FPGA build results and the processor models to ConfigurationDesk. Refer to How to Export Build Results and Processor Models to ConfigurationDesk Projects on page 146.

Related topics

Basics

How to Export Build Results and Processor Models to ConfigurationDesk Projects

Objective

You have to export the processor model and the FPGA application to ConfigurationDesk before you can build the real-time application.

Steps of the export process

The framework mainly performs the following steps when it exports the build results and the processor model to ConfigurationDesk:

- Opens ConfigurationDesk.
- Exports the displayed build result of the current FPGA application as a custom function block type to a ConfigurationDesk project.

  If the entire model contains other subsystems with FPGA models for other FPGA boards, the framework also exports the build results of these FPGA models.
- Adds instances of the added custom function block types to the signal chain.

The framework performs the following steps only if the processor model can be separated and the processor interface is implemented:

- Separates and saves the processor models according to the settings of the **Model Separation Setup** block.

In ConfigurationDesk, the processor models are implemented outside ConfigurationDesk in Simulink models.
- Exports the model interface of the processor models to ConfigurationDesk.
- Maps the function ports of the added custom functions to the model ports of the processor model (behavior model).

Preconditions

- Your platform is a MicroAutoBox III or a SCALEXIO system.
- The processor interface is implemented. Missing parts of the processor interface cannot be automatically mapped in ConfigurationDesk. Refer to How to Generate a Processor Interface on page 114.
- The Model Separation Setup block is added to the entire model and configured. Refer to How to Prepare the Processor Models for Separating (MicroAutoBox III, SCALEXIO) on page 144.
- The FPGA application is built. Refer to How to Build FPGA Applications (MicroAutoBox III, SCALEXIO) on page 140.

Possible methods

You can export the build results and the processor model to a new or a recent ConfigurationDesk project.
- To export to a new project, refer to Method 1.
- To export to a recent project, refer to Method 2.

Method 1

To export the build result and the processor model to a new ConfigurationDesk project

1. Double-click the FPGA_SETUP_BL block in the FPGA subsystem to open the block's dialog and switch to its ConfigurationDesk Interface page.
2. On Name of (new) project, enter a name for a new ConfigurationDesk project. If the project already exists, the framework exports the build results and the processor model to the existing project.
3. Click Export to new project.

Method 2

To export the build result and the processor model to a recent ConfigurationDesk project

1. Double-click the FPGA_SETUP_BL block in the FPGA subsystem to open the block's dialog and switch to its ConfigurationDesk Interface page.
2. On Select recent project, select a project for exporting.

   Tip

   Click Refresh list to import the list of recent projects from ConfigurationDesk. If needed, the framework starts ConfigurationDesk to refresh the list.

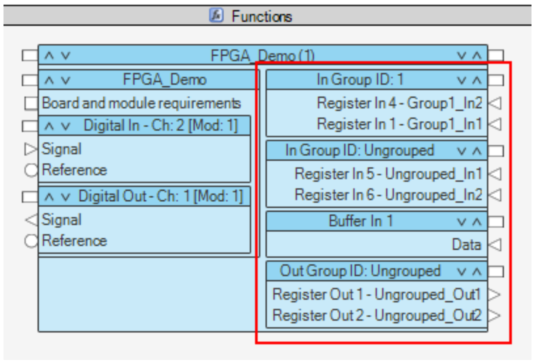
3. Click Export to recent project.

Result

You exported all build FPGA applications of the entire model and the processor model to ConfigurationDesk. In ConfigurationDesk, the processor model is a part of the behavior model.

The FPGA_XDATA_READ and FPGA_XDATA_WRITE blocks are represented by functions in the FPGA function block.

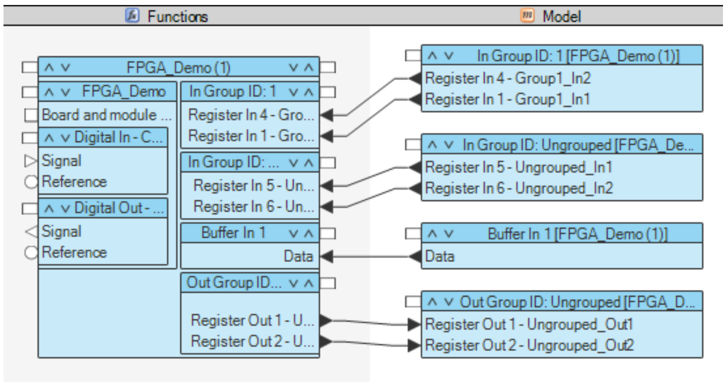
- A **Buffer In** block is represented by a **Buffer In** function.
- A **Buffer Out** block is represented by a **Buffer Out** function.
- All **Register In** blocks for which you have not specified a group are represented by one **In Group ID: Ungrouped** function.
- All **Register Out** blocks for which you have not specified a group are represented by one **Out Group ID: Ungrouped** function.
- All **Register In** blocks for which you have specified a group are represented by one group-specific **In Group ID: <GroupNumber>** function.
- Not contained in the figure: All **Register Out** blocks for which you have specified a group are represented by one group-specific **Out Group ID: <GroupNumber>** function.


Functions
FPGA_Demo (1)
FPGA_Demo
Board and module requirements
Digital In - Ch: 2 [Mod: 1]
Signal
Reference
Digital Out - Ch: 1 [Mod: 1]
Signal
Reference
In Group ID: 1
Register In 4 - Group1_In2
Register In 1 - Group1_In1
In Group ID: Ungrouped
Register In 5 - Ungrouped_In1
Register In 6 - Ungrouped_In2
Buffer In 1
Data
Out Group ID: Ungrouped
Register Out 1 - Ungrouped_Out1
Register Out 2 - Ungrouped_Out2

If the processor interface is implemented, the model port mapping is done automatically. Otherwise you have to do this manually.

Tip

- In ConfigurationDesk, click to rearrange the blocks of a signal chain so that as many mapping lines as possible are horizontal.
- To map the ports of Simulink buses, drag & drop the model port group to the FPGA custom function.



Note

You can access a register group by one task only.

Next step

You can map the device ports of the external devices to the signal ports of the FPGA function blocks, assign the hardware resources, enable the angular processing unit, and specify the task priority of interrupts. Refer to Handling FPGA Custom Function Blocks in ConfigurationDesk (ConfigurationDesk I/O Function Implementation Guide 🕮).

Related topics

Basics

Building FPGA and Processor Applications..127

References

ExportToNewProject (RTI FPGA Programming Blockset Script Interface Reference 🕮)
ExportToRecentProject (RTI FPGA Programming Blockset Script Interface Reference 🕮)

How to Build Processor Applications (MicroAutoBox III, SCALEXIO)

Objective

Before you can execute the behavior model including the processor model on the real-time hardware, you must build an application from its generated code.

Basics

The code generation and the build process are managed by ConfigurationDesk.

For general information, refer to Building Real-Time Applications (ConfigurationDesk Real-Time Implementation Guide 🕮).

Note

Before you start the build process for the processor application, you must set the **Fixed-step size** setting on the **Solver** page to a value, which is the multiple of the offline simulation period specified in the **FPGA_SETUP_BL** block. Check also the compatibility of the solver settings used for Xilinx-based offline simulation, for example, the solver type.

Preconditions

- Your platform is a MicroAutoBox III or a SCALEXIO system.
- The working folder must be set to the model path.
- If you want to download the real-time application after the build process, the platform must be registered via the Platform Manager in ConfigurationDesk.

Method

To build a processor application when using a MicroAutoBox III or a SCALEXIO system

1. Prepare the behavior model for the build process.
2. Only if you do not export the build results to ConfigurationDesk via the **FPGA_SETUP_BL** block:
   - In ConfigurationDesk, execute the `Analyze Model (Complete Analysis)` command, to make all the required **Runnable Functions** (used for FPGA_interrupts) available to the behavior model.
   - Execute the `Create Preconfigured Application Process` command to initialize the application process.
3. In ConfigurationDesk, configure the build options.
4. Click **Build** in ConfigurationDesk's Build Manager to start the build process.

Result

If the build process finished successfully, you will find the build results, such as the processor application file and the variable description file, in ConfigurationDesk's *Build Results* folder. The FPGA application is automatically programmed to the RAM of the FPGA when the processor application is loaded.

For example, if you built the *MyDemo* application for a MicroAutoBox III or a SCALEXIO system, you will find the executable RTA file called `MyDemo.rta` in the *Build Results* folder.

Related topics

Basics

Building Real-Time Applications (ConfigurationDesk Real-Time Implementation Guide 🕮)

Using an FPGA Build Server

Introduction

The dSPACE FPGA Build Server can take over the build process from the RTI FPGA Programming Blockset.

Where to go from here

Information in this section

Basics on the FPGA Build Server

Introduction

The dSPACE FPGA Build Server can execute the FPGA build that was started with the RTI FPGA Programming Blockset.

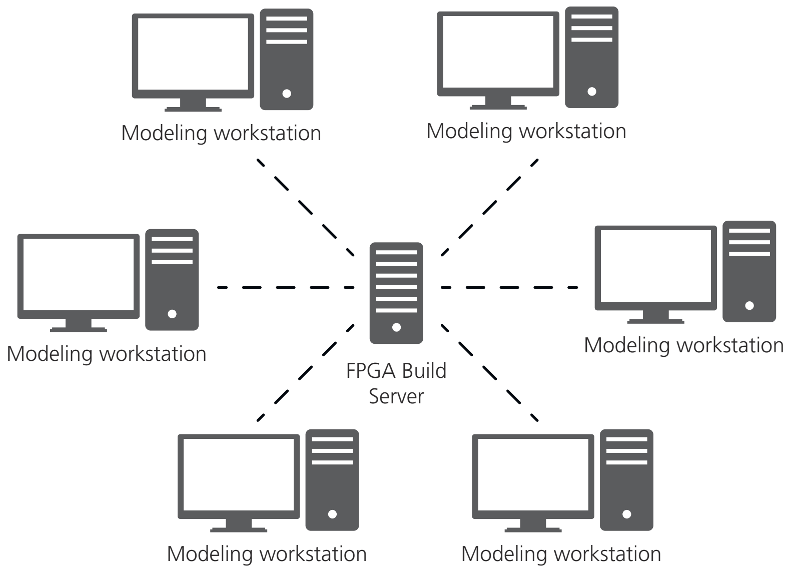
Use cases

Main use cases for the FPGA Build Server:

- To continue modeling after you started the build process.

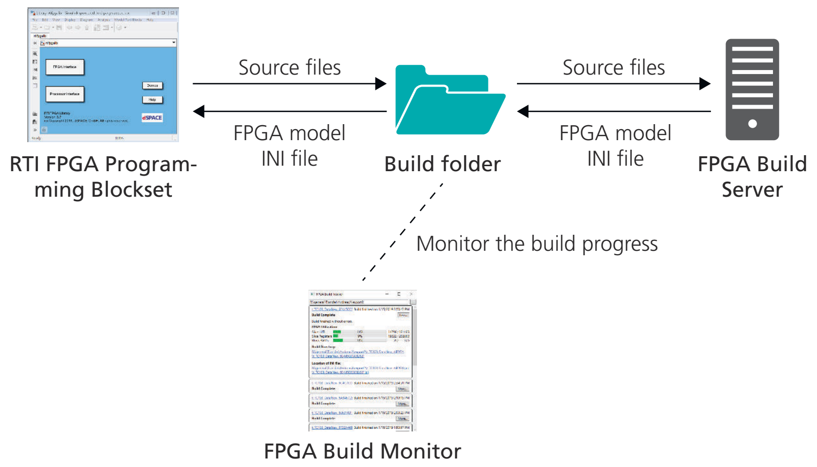
  The FPGA Build Server can run on the same PC as the RTI FPGA Programming Blockset.
- To provide high computing capacity to build FPGA applications faster for several modeling workstations.

One computer with optimized computing capacity builds the FPGA application for several modeling workstations.



Principle of operation

The following illustration shows the principle of operation:



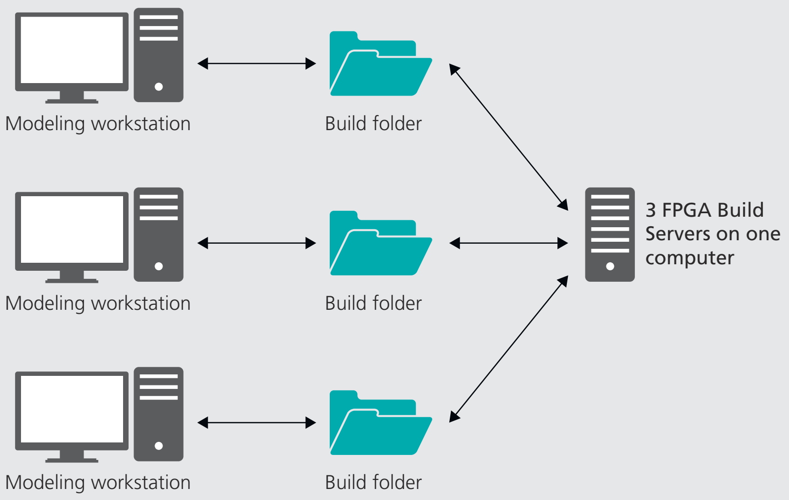
A common build folder is used to exchange the files to build the FPGA application and the resulting FPGA model INI file.

- The RTI FPGA Programming Blockset provides the source files for building the FPGA application.
- The FPGA Build Server builds the FPGA application and provides the FPGA model INI file.

  The FPGA Build Server can build multiple FPGA applications at the same time.
- The FPGA Build Monitor displays the build progress of all builds that are executed in the common build folder.

Tip

If one computer is used as the build server for several modeling workstations, you can use a separate build folder for each modeling workstation.
To use separate build folders, install multiple FPGA Build Servers on the computer.



Related topics

HowTos

How to Prepare the Computer for the FPGA Build Server

Objective

To install the software and exchange folder required to run the FPGA Build Server.

Required software

The following software is required to run the FPGA Build Server.

- Windows operating system that is supported by the RCP and HIL software of the current Release.

  For a list of supported operating systems, refer to Operating System (New Features and Migration 🕮). The listed Windows Server 2016 edition is not officially supported by Xilinx, but tested by dSPACE.

- The Vivado version that is used for FPGA modeling. Refer to Software Tools for Working With the RTI FPGA Programming Blockset on page 19.
- Python 3.6
- Microsoft .NET Framework 4.7.2

Precondition

You have access to the source media of the required software.

Method

To prepare the computer for the FPGA Build Server

1 Install the required software.

> **Tip**
>
> If the RTI FPGA Programming Blockset is installed on the same computer, the required software is already installed.

2 Create an exchange folder as a common build folder. The folder must meet the following requirements:

- The build server and the modeling workstations have read/write access to exchange the files for the build process.
- The path to the build folder is the same for the FPGA Build Server and for the modeling workstations.

  The same path is required, because absolute paths are used in the FPGA models.
- The path length of the build folder is short to avoid errors during the build process.

  During the build process, the path length can exceed the maximum path length specified by Windows. This might lead to error messages during file system operations. For more information, refer to Xilinx Answer Record 52787 (http://www.xilinx.com/support/answers.html).

For information on creating an exchange folder with read/write access, refer to the documentation of the operating system.

Result

You prepared the computer for running the FPGA Build Server.

Related topics

Basics

Basics on the FPGA Build Server...151

How to Install and Start the FPGA Build Server

Objective

The FPGA Build Server must be configured and run before you start the build process in the FPGA_SETUP_BL block.

Precondition

The computer for the FPGA Build Server must be prepared. Refer to How to Prepare the Computer for the FPGA Build Server on page 153.

Method

To install and start the FPGA Build Server

1 Copy the complete `FPGABuildServer` installation folder to the computer that will run the FPGA Build Server. The installation folder is located in the following folder:

```
<RCP_HIL_InstallationPath>\MATLAB\RTIFPGA\RTIFPGA\bin\
```

2 Use an ASCII text editor to open the `FpgaBuildServerConfig.json` configuration file:

```
<ServerFolder>\FPGABuildServer\FpgaBuildServerConfig.json
```

3 Enter the path of the common build folder. Use one of the following characters to separate the folders of the path.

- Use / to enter the path:

```
"BuildDirectory" : "<root>/.../<CommonBuildFolder>"
```

- Use \\ to enter the path:

```
"BuildDirectory" : "<root>\\...\\<CommonBuildFolder>"
```

The other configuration parameters are optionally. Refer to FPGA Build Server Configuration File Reference on page 162.

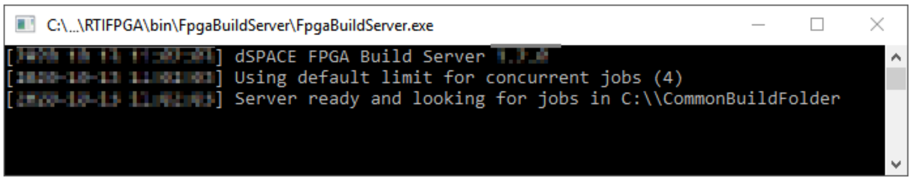
4 Save the configuration file.

5 Execute the `FpgaBuildServer.exe` batch file to start the FPGA Build Server. Location of the batch file:

```
<ServerFolder>\FPGABuildServer\FPGABuildServer\
```

Result

You configured and started the FPGA Build Server.



```
[[redacted]] dSPACE FPGA Build Server [redacted]
[[redacted]] Using default limit for concurrent jobs (4)
[[redacted]] Server ready and looking for jobs in C:\\CommonBuildFolder
```

Tip

- To make sure that the server is automatically available after a restart, configure Windows to start the FPGA Build Server when Windows starts. For more information, refer to https://support.microsoft.com/en-us/help/4026268/windows-10-change-startup-apps.
- To use multiple build folders, you can run multiple FPGA Build Servers in parallel: Repeat step 1 ... 5 with other installation and build folders to use more than one FPGA Build Server.

Related topics

Basics

HowTos

References

How to Build FPGA Applications via a Server

Objective

To provide the files for the FPGA build process.

Basics

You start the build process with the FPGA_SETUP_BL block of the FPGA model. The build process is automatically started and managed by the FPGA Build Server. The FPGA Build Server starts the Xilinx implementation process, including synthesizing, mapping, routing and generating the bitstream file.

Even if no errors occurred during modeling, simulating, and building, your FPGA application is not guaranteed to work correctly. Some combination of factors might prohibit the execution of your FPGA application, e.g., combinations of the operating temperature, the power of the entire hardware system, technological limitations of the FPGA, the sample rate, etc. Some FPGA applications do not start at all, others start correctly but fail during operation.

Due to the complexity of FPGA applications and their dependency on the used FPGA hardware, the Xilinx design tools cannot ensure that all possible design problems are detected during the analysis and simulation of the FPGA model.

To solve the problem, you have to modify the FPGA model and make sure that you consider generally accepted FPGA design rules to ensure a stable and reliable

FPGA application (refer to Audience profile on page 9). For more information, contact dSPACE Support.

Preconditions

The following preconditions must be fulfilled:

- The FPGA model that you want to build must be correctly implemented. For example, the FPGA Interface blocks must be correctly configured.
- If you use Xilinx Black Box blocks to incorporate hardware description language models, the files of the incorporated models are located in the model root folder or in an `Includes` subfolder in the model root folder. The `Includes` subfolder and optional subfolders must be added to the MATLAB search paths as relative paths. For more information, refer to Implementing the FPGA Functionality on page 45.
- The FPGA_SETUP_BL block has read/write access to the common build folder.
- The FPGA Build Server must be running. Refer to How to Install and Start the FPGA Build Server on page 155.

Method

To build an FPGA application via a server

1 Open the FPGA_SETUP_BL block dialog and switch to the Parameters page.

2 Clear Current directory.

3 In FPGA build directory, enter the path to the common build folder.

Note

The FPGA Build Server must use the same path for the common build folder, because the FPGA model uses absolute paths.

4 In FPGA model action, select Remote FPGA Build.

5 Click Execute to start start the build process.

Result

The RTI FPGA Programming Blockset starts the Xilinx System Generator and a temporary model: `<FPGAModelName>_rtiFPGAtmp`. The Xilinx System Generator generates and synthesizes the files for the build process and saves them to the common build folder. The temporary model automatically closes. You can continue FPGA modeling after the required files for the build process are available and saved to the common build folder.

The FPGA Build Server automatically starts the build process after the server detected the new build job.

Next step

The build process can take a long time. You can use the FPGA Build Monitor to monitor the build progress. Refer to How to Monitor the Build Process on page 158.

Related topics

Basics

HowTos

How to Monitor the Build Process

Objective

To configure and start the FPGA Build Monitor.

Basics

The FPGA Build Monitor monitors build processes that are available in the selected build folder. The FPGA Build Monitor displays status information and the build progress.

For more information, refer to FPGA Build Monitor Reference on page 160.

Possible methods

You can choose between the following methods to monitor the build process.

- To monitor the build process that you started with the FPGA_SETUP_BL block, refer to Method 1.
- To monitor the build processes in a specific build folder, refer to Method 2.

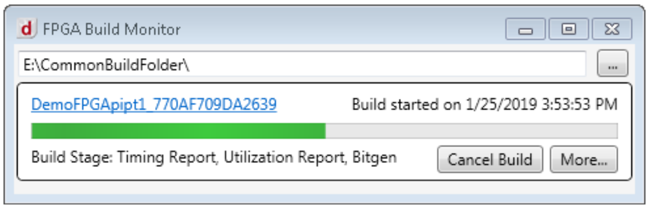
Method 1

To monitor the build process that you have started

1 Open the MATLAB Command Window.

2 Click the `Start FPGA Build Monitor` message that is output after you started the build process with the FPGA_SETUP_BL block.

The FPGA Build Monitor opens and displays the build process that you have started as well as all former build processes of the FPGA model.



E:\CommonBuildFolder\

DemoFPGApipt1_770AF709DA2639

Build started on 1/25/2019 3:53:53 PM

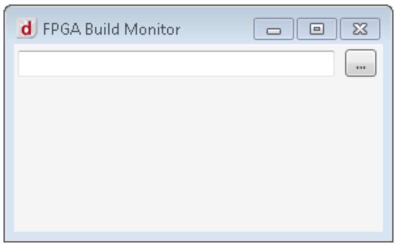
Build Stage: Timing Report, Utilization Report, Bitgen

Method 2

To monitor all build processes in a build folder

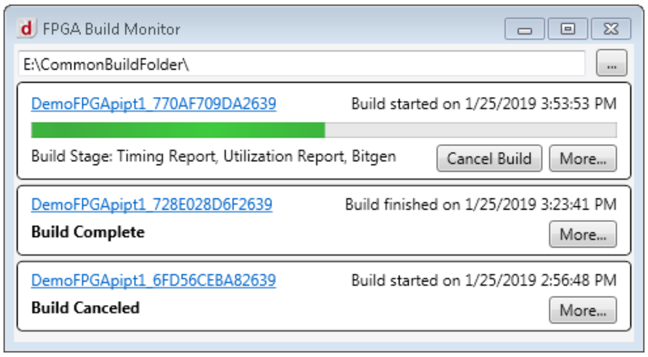
1 On the Start menu, select All Programs - dSPACE RCP and HIL <Release> - dSPACE FPGA Build Monitor.

The FPGA Build Monitor starts.



2 Click the ... Browse button and select the build folder to be monitored.

The FPGA Build Monitor displays all build processes that are available in the selected folder.



E:\CommonBuildFolder\

DemoFPGApipt1_770AF709DA2639 Build started on 1/25/2019 3:53:53 PM
Build Stage: Timing Report, Utilization Report, Bitgen

DemoFPGApipt1_728E028D6F2639 Build finished on 1/25/2019 3:23:41 PM
Build Complete

DemoFPGApipt1_6FD56CEBA82639 Build started on 1/25/2019 2:56:48 PM
Build Canceled

Result

The FPGA Build Monitor displays information on the monitored build processes.

Next step

When the FPGA build is complete, you can build the processor application. Refer to Building the Applications (MicroAutoBox II, MicroLabBox, PHS-Bus-Based System) on page 128 or Building the Applications (MicroAutoBox III, SCALEXIO) on page 140.

Related topics

Basics

Basics on the FPGA Build Server........151

HowTos

How to Build FPGA Applications via a Server........156

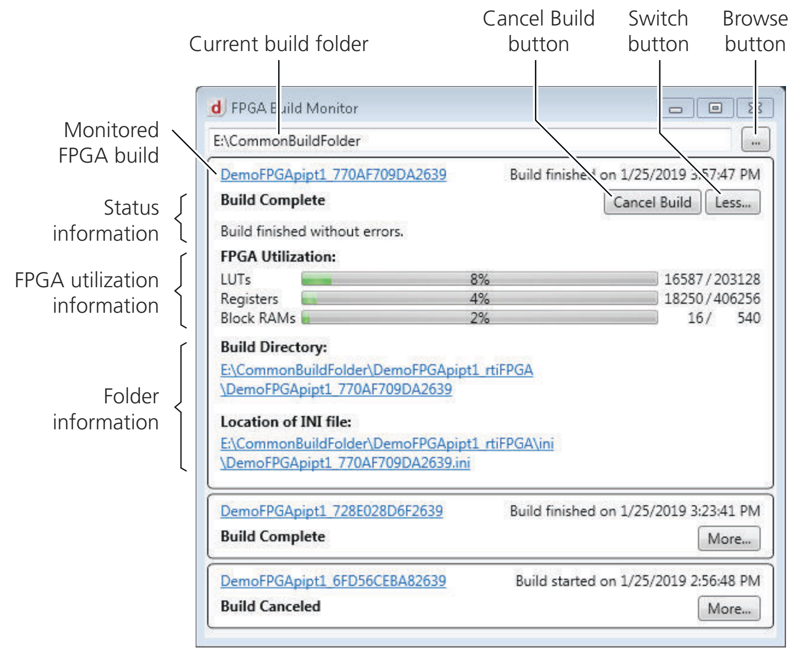
References

FPGA Build Monitor Reference........160

FPGA Build Monitor Reference

Overview

The following illustration shows the elements of the FPGA Build Monitor user interface.



E:\CommonBuildFolder

DemoFPGApipt1_770AF709DA2639 Build finished on 1/25/2019 3:57:47 PM

Build Complete

Build finished without errors.

FPGA Utilization:

LUTs	8%	16587 / 203128
Registers	4%	18250 / 406256
Block RAMs	2%	16 / 540

Build Directory:

E:\CommonBuildFolder\DemoFPGApipt1_rtiFPGA\DemoFPGApipt1_770AF709DA2639

Location of INI file:

E:\CommonBuildFolder\DemoFPGApipt1_rtiFPGA\ini\DemoFPGApipt1_770AF709DA2639.ini

DemoFPGApipt1_728E028D6F2639 Build finished on 1/25/2019 3:23:41 PM

Build Complete

DemoFPGApipt1_6FD56CEBA82639 Build started on 1/25/2019 2:56:48 PM

Build Canceled

Browse button

Lets you open the file explorer to select the build folder to be monitored.

Cancel Build button

Lets you finish the build process before it has been completed.

The Cancel Build button is available only during the build process.

Current build folder

Displays and lets you enter the current build folder.

Folder information

Displays and lets you open the build directory of the selected FPGA build and the resulting FPGA model INI file.

The resulting FPGA model INI file is displayed only for completed FPGA builds.

FPGA utilization information

Displays information on the FPGA utilization after a build is complete:

- LUTs: Number of used Look-up tables (LUTs) in comparison to the available LUTs.
- Registers: Number of used registers in comparison to the available registers.

- Block RAMs: Number of used block RAM in comparison to the available block RAM.

FPGA utilization information is displayed only for completed FPGA builds and after you click **More**.

Monitored FPGA build

Displays information and lets you open the folder of the monitored build. The following information is available:

- Status information
- FPGA utilization information
- Folder information

Each build that is available in the current build folder is displayed with its own table. Running build processes are displayed at the top.

Status information

The following status information can be displayed:

- Build not yet started

  The files for building the FPGA application are available in the common build folder, but the FPGA Build Server has not started building the FPGA application.

  If the build does not start, the FPGA Build Server might not be running or is observing another build folder. Check the settings and restart the server. Refer to How to Install and Start the FPGA Build Server on page 155.
- Build stage with progress bar

  Build stage displays the current build step. If the build is running, the **Cancel Build** button lets you cancel the build process.
- Build complete

  The FPGA model INI file is available. The location of the INI file is displayed after you click **More**.
- Build canceled

  The FPGA build was canceled.
- Build failed with errors

  The build process cannot be finished due to errors.

  The FPGA Build Monitor displays the build step in which the build process stops. The build step is displayed only after you click **More**.

Switch button

Lets you display more or less information.

Related topics

HowTos

How to Monitor the Build Process..158

FPGA Build Server Configuration File Reference

Introduction

Parameters in the `FpgaBuildServerConfig.json` configuration file lets you configure the FPGA Build Server.

File location Location of the configuration file:

```
<ServerFolder>\FPGABuildServer\FpgaBuildServerConfig.json
```

Syntax The following example shows the syntax of the configuration file.

```
{
  "BuildDirectory" : "<root>\\...\\<CommonBuildFolder>",
  "ConcurrentJobs" : 4,
  "LogFile" : "FpgaBuildServer.log"
}
```

BuildDirectory parameter

Lets you enter the path of the common build folder. This parameter must be set.

Use one of the following characters to separate the folders of the path.

- Use / to enter the path:

  ```
  "BuildDirectory" : "<root>/.../<CommonBuildFolder>"
  ```

- Use \\ to enter the path:

  ```
  "BuildDirectory" : "<root>\\...\\<CommonBuildFolder>"
  ```

ConcurrentJobs parameter

Lets you set the maximum number of concurrent build jobs. If the maximum number has been reached, new jobs are not started until old jobs are completed.

If this parameter is not used, the maximum number of concurrent build jobs is set to four.

The `ConcurrentJobs` parameter affects only the number of build jobs and not the number of build threads that are set via the FPGA_SETUP_BL block.

LogFile parameter

Lets you enter a file name of a log file in the build folder to record status information on the build server and build error messages.

The FPGA Build Server adds a new file with the entered name to the build folder if the entered file does not exist.

If this parameter is not used, the FPGA Build Server does not use a log file.

Related topics

HowTos

Running Processor and FPGA Applications on the Real-Time Hardware

Introduction

When you have built the executable applications for the processor and the FPGA board, you can load them to the real-time hardware for experimenting.

Experimenting with an FPGA Application

Introduction

You can observe the behavior of the real-time application and change application-specific parameters by using ControlDesk.

Downloading processor and FPGA application

To run a processor application, the built executable file must only be downloaded to the real-time hardware. The real-time hardware must be registered beforehand, for example, via the Platform Manager in ControlDesk or ConfigurationDesk. Any running application is stopped and overwritten by the newly loaded application. If the processor application contains the programming of the FPGA, the FPGA application is loaded to the FPGA automatically. If it does not, and there is no FPGA application running, you must first download the burn application to program the FPGA and afterwards the processor application.

Note

Before you download the processor application, the flash process started by the burn application must be finished. For more information on burn applications, refer to How to Create Burn Applications on page 138.

Note

Even if no errors occurred during modeling, simulating, and building, your FPGA application is not guaranteed to work correctly. Some combination of factors might prohibit the execution of your FPGA application, e.g., combinations of the operating temperature, the power of the entire hardware system, technological limitations of the FPGA, the sample rate, etc. Some FPGA applications do not start at all, others start correctly but fail during operation.

Due to the complexity of FPGA applications and their dependency on the used FPGA hardware, the Xilinx design tools cannot ensure that all possible design problems are detected during the analysis and simulation of the FPGA model.

To solve the problem, you have to modify the FPGA model and make sure that you consider generally accepted FPGA design rules to ensure a stable and reliable FPGA application (refer to Audience profile on page 9). For more information, contact dSPACE Support.

Required files

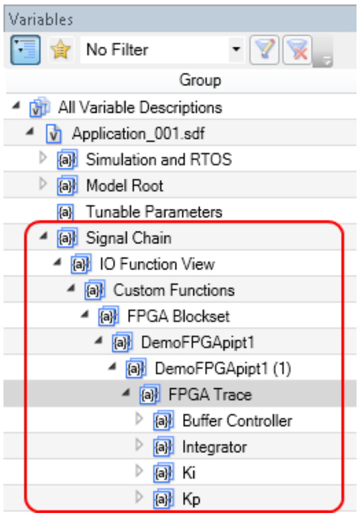
For experimenting, i.e., changing application-specific parameters during run time and displaying the behavior, you need files that are created during the build process, for example, the processor application (.x86, .ppc or .rta), the variable description file (.trc), the system description file (.sdf), and the mapping file (.map).

Note

The parameters and variables that are used in the FPGA application are not included in the variable description file for . If you want to access an FPGA signal of PHS-bus-based systems in the experiment software, you must provide it to the processor model.

Accessing FPGA signals with an experiment software

You use the SDF file to access the FPGA signals and constants with your experiment software, for example, with ControlDesk. In ControlDesk, you can access the signals and constants in the Variables controlbar under `Signal Chain/IO Function View/Custom Functions/FPGA Blockset/<CustomFunctionName>/<CustomFunctionInstance>/FPGA Trace`.


Variables

No Filter

Group

- All Variable Descriptions
  - Application_001.sdf
    - Simulation and RTOS
    - Model Root
    - Tunable Parameters
    - Signal Chain
      - IO Function View
        - Custom Functions
          - FPGA Blockset
            - DemoFPGApipt1
              - DemoFPGApipt1 (1)
                - FPGA Trace
                  - Buffer Controller
                  - Integrator
                  - Ki
                  - Kp

Limitations For limitations, refer to Basics on Tracing FPGA Signals on page 171.

Note

Running applications might stop, if too many FPGA signals are traced

If you trace more than 100 signals with 32-bit values (or 50 signals with 64-bit values) every millisecond with your experiment software, tracing might cause a task overrun that stops the application.

These measures reduce the number of traced FPGA signals per millisecond:

- Disable signal tracing if there is no need to trace FPGA signals.
- Reduce the number of traceable signals. Refer to Basics on Tracing FPGA Signals on page 171.
- Reduce the number of signals that that you trace with the experiment software. Only the values of signals traced with an experiment software are sent to the real-time processor and can cause a task overrun.

Triggering the data acquisition The data acquisition of all traceable FPGA signals is triggered by one trigger source, so all signals of an FPGA application are captured at the same time. It is not possible to capture signals at different points in time.

The experiment software can trace only FPGA signals with integer multiples of the trigger period for tracing FPGA signals.

For the MicroAutoBox III and SCALEXIO systems, you must map the **Trigger** inport of the FPGA custom function block (FPGA Setup block) to the behavior model. This enables the triggering of the data acquisition with the shortest task period of the mapped behavior model. Refer to Basics on Tracing FPGA Signals on page 171.

For MicroAutoBox II and MicroLabBox, the data acquisition is automatically triggered with the task period of the subsystem in which the **FPGA_SETUP_BL** block is located. If you implement a multicore processor application for the MicroLabBox, the data acquisition is triggered with the task period of the subsystem in which the **FPGA_SETUP_BL** block is located that programs the FPGA.

Related documents

- DS100x, DS110x, MicroAutoBox II, MicroLabBox – Software Getting Started 🕮 for a brief overview of experimenting.
- ControlDesk Introduction and Overview 🕮 for information on working with ControlDesk.
- ConfigurationDesk Real-Time Implementation Guide 🕮 for detailed information on working with ConfigurationDesk.

Related topics

Basics

Accessing FPGA Applications with your Experiment Software

Introduction

You can access FPGA signals and constants with your experiment software if it is supported by your platform.

Where to go from here

Information in this section

Accessing Elements of FPGA Applications

Introduction

You can use different methods to access FPGA signals and constants.

Introduction to Access FPGA Applications

Use cases

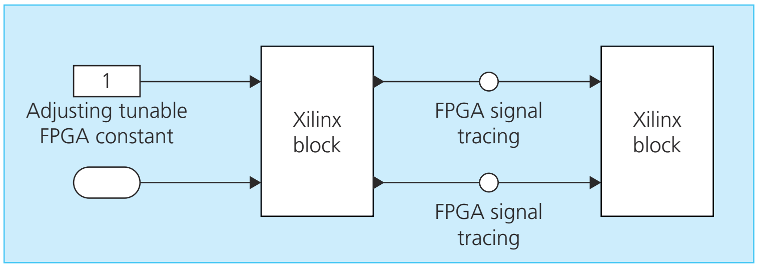
To access an FPGA application with your experiment software is useful in the following cases, for example:

- To trace signals and adjust control parameters for optimization.
- To use the FPGA application with various settings, e.g., you can scale analog I/O signals to support different sensors and actors.
- To debug the application, e.g., you can test the cable harness by switching the I/O interface to defined values.

Possible elements to access

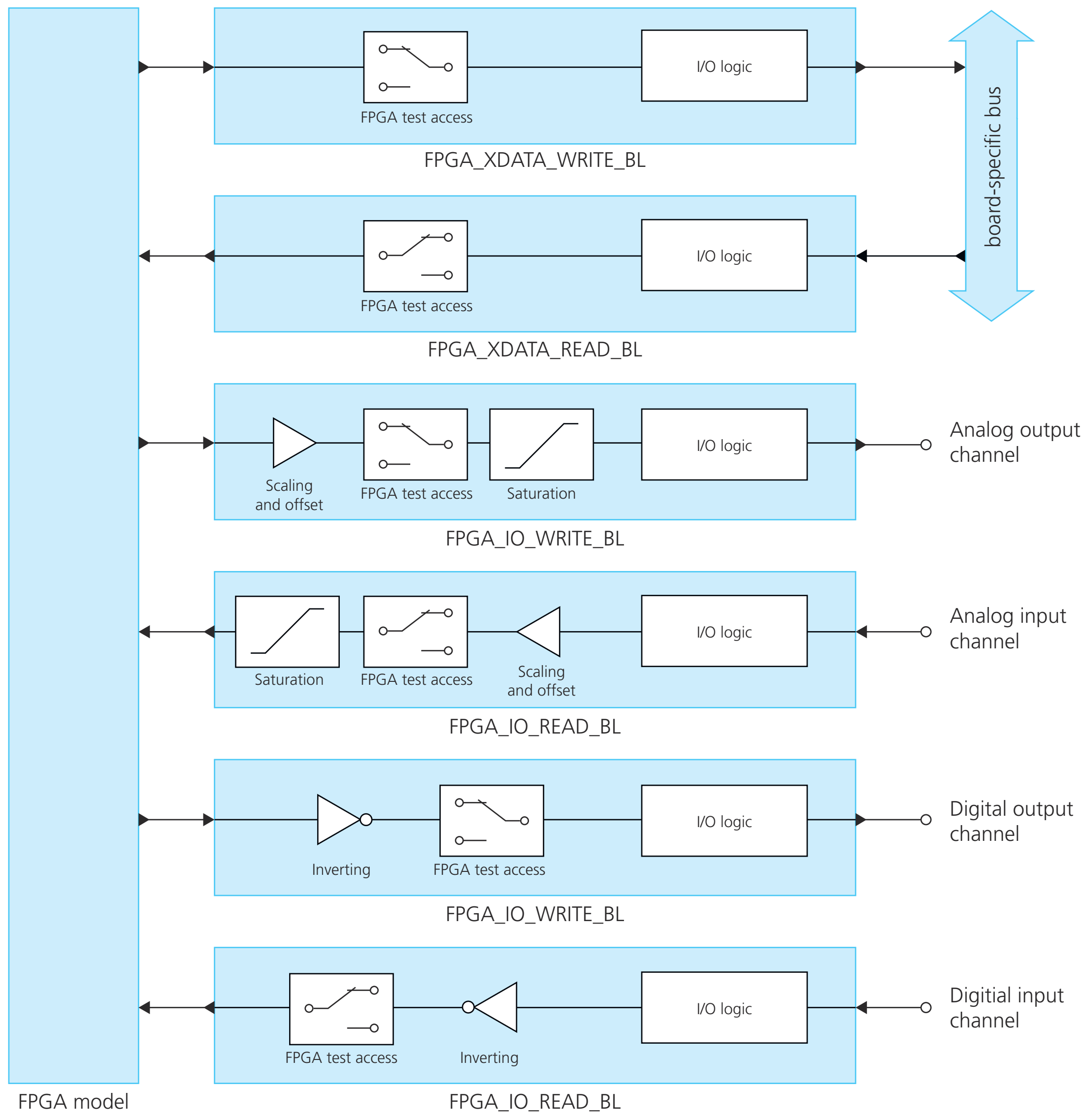
You can access elements of the FPGA logic that includes the FPGA functionality and elements of the FPGA interface.

Accessing elements of the FPGA logic You can access FPGA signals and tunable FPGA signals. The illustration below show you the possible methods to access the elements.



Method	Description	Further Information
FPGA signal tracing	Gives you read access to FPGA signals.	Tracing FPGA Signals on page 171
Adjusting tunable FPGA constants	Gives you read/write access to FPGA constants.	Adjusting Values of FPGA Constants on page 178

Accessing the FPGA interface You can access the signals of the FPGA interface to scale and or substitute them. The illustration below show you the possible methods to access the elements.



Method	Description	Further Information
FPGA test access	Lets you replace signal values.	Enabling the FPGA Test Access and Scaling on page 182
Scaling and offset	Lets you scale and add an offset to analog I/O signals.	
Saturation	Lets you saturate analog I/O signals.	
Inverting	Lets you invert the polarity of digital I/O signals and RS232/485 signals.	

Platform support

The following table gives you an overview of the platforms that support the access of FPGA applications and the methods that are supported.

Platform	FPGA Signal Tracing	Tunable FPGA Constants	FPGA Test Access	FPGA Scaling		
				Scaling Analog I/O Signals	Inverting Digital I/O Signals	Inverting RS232/485 Signals
SCALEXIO	✓[1]	✓	✓	✓	✓	✓
MicroLabBox	✓	✓	✓	✓	✓	–
MicroAutoBox II	✓	✓	✓	✓	✓	–
MicroAutoBox III	✓	✓	✓	✓	✓	–
PHS-bus-based system	–	–	–	–	–	–

[1] ✓ = Supported, – = Not supported

Related topics

Basics

Tracing FPGA Signals

Introduction

You can make FPGA signals traceable to trace FPGA signals with your experiment software.

Where to go from here

Information in this section

Basics on Tracing FPGA Signals

Basics on FPGA tracing

FPGA signals can be traced via FPGA variables that are added to the FPGA application during the build process. These variables provide the read access for the experiment software. The implementation of FPGA variables uses resources of the FPGA (flip-flops). Building the FPGA application with traceable signals also takes more time. Keep in mind that 1000 traceable FPGA signals increase the build time for about 10%, 4000 traceable FPGA signals roughly doubles the build time.

Tracing mechanism To trace an FPGA signal, the values of the FPGA variables must be requested by the experiment software. That is, if the experiment software does not request values, no values are sent to the experiment software. The FPGA application sends requested values of traced FPGA signals via the board-specific bus to the processor application. Then, the processor application sends the values to your host PC. This takes time and limits the number of traceable FPGA signals.

Tracing the signals with the experiment software A system description file (SDF file) containing additional information for tracing variables of the FPGA application and the processor model is generated during the build process of the processor application. You can use the SDF file with your experiment software to access the FPGA variables. For details, refer to Experimenting with an FPGA Application on page 163.

Limitations

There are limitations for tracing signals of FPGA applications:

- Every millisecond, the following number of FPGA signals can be traced:
  - At most 100 signals with up to 32-bit values.
  - At most 50 signals with up to 64-bit values.

  Running applications might stop, if too many FPGA signals are traced.
- Signals with floating-point variables can be traced only if the values are in the single or double precision floating-point formats (IEEE 754 standard). Other floating-point formats are not supported.
- Signals with fixed-point variables can be traced only if the data width is 1 bit ... 64 bit.
- The data acquisition of all traced signals is triggered by one trigger source. So all signals are captured at the same time. It is not possible to capture signals at different points in time.

  The experiment software can trace FPGA signals only with integer multiples of the trigger period from within the behavior model.

  For details on the trigger source, refer to Trigger source for tracing on page 174.
- When the experiment software traces FPGA signals, it takes one task period until the traced signals are provided to the experiment software. If you start a new measurement, the first two measurements are not valid.
- Traced FPGA signals cannot be used for the experiment software as trigger condition for data acquisition or to stimulate variables.
- Real-Time Testing is supported with the following limitations:
  - Support of fixed-point data types up to a data width of 32 bit.
  - No support of fixed-point data types for MicroAutoBox III and SCALEXIO systems.
- Use of API libraries, such as XIL API libraries or Test Automation Python Modules, is not fully supported.

  MicroAutoBox II only: No support of XIL API libraries.
- Flight recording of traced FPGA signals is not supported.
- SCALEXIO systems only: Tracing of FPGA signals is not supported for application processes containing a task configured in ConfigurationDesk as 'No jitter, low latency'.

Selecting signals for tracing

The number of signals that can be traced with your experiment software is limited (refer to Limitations on page 172).

There are two ways to reduce the number of traceable FPGA signals and both can be used at the same time:

- Tracing of selected model parts only.
- Excluding model parts from tracing.

Tracing of selected model parts You can enable FPGA tracing for selected subsystems of the entire FPGA model via the FPGA_SETUP_BL block. For instructions, refer to How to Make FPGA Signals Traceable on page 174.

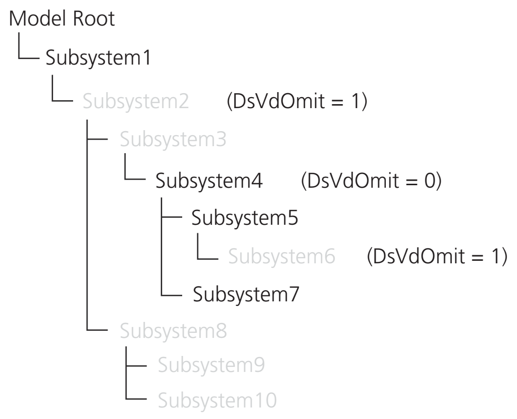
Excluding model parts from tracing You can exclude a subsystem from the TRC file by using the subsystem omission tag (DsVdOmit). You must enter `DsVdOmit=1` in the Tag edit field in the Block Properties dialog of the subsystem.

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 SDF file. Use `set_param(gcb,'Tag', 'DsVdOmit=1')`.
`DsVdOmit` tag is set to 0	The subsystem contents including all blocks beneath this subsystem appear in the generated SDF 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 SDF 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 SDF file.



For instructions, refer to How to Exclude FPGA Signals from Tracing on page 176.

Making FPGA signals traceable

Before you can trace FPGA signals with your experiment software, you must enable the generation of FPGA variables so that the FPGA signals are traceable. You enable the tracing mechanisms via the FPGA_SETUP_BL block. For instructions, refer to How to Make FPGA Signals Traceable on page 174.

Trigger source for tracing

The data acquisition of traced signals is triggered with the task period of the processor application.

If you use a multicore processor application, you can specify which processor application triggers the data acquisition.

MicroAutoBox II If you use a MicroAutoBox II, the processor application triggers the data acquisition.

MicroAutoBox III If you use a MicroAutoBox III, the FPGA custom function block (FPGA Setup block) provides a **Trigger** inport in ConfigurationDesk. When you map this port, you specify the processor application that triggers the data acquisition.

MicroLabBox If you use MicroLabBox, the processor application of the processor core that programs the FPGA triggers the data acquisition. You specify this processor application during the build configuration. Refer to How to Build Multicore Processor Applications for MicroLabBox on page 135.

SCALEXIO If you use SCALEXIO, the FPGA custom function block (FPGA Setup block) provides a **Trigger** inport in ConfigurationDesk. When you map this port, you specify the processor application that triggers the data acquisition.

Related topics

Basics

Accessing the FPGA Application with the Experiment Software (ConfigurationDesk I/O Function Implementation Guide 🕮)

HowTos

References

GetTraceSubsystems (RTI FPGA Programming Blockset Script Interface Reference 🕮)
SetTraceSubsystems (RTI FPGA Programming Blockset Script Interface Reference 🕮)

How to Make FPGA Signals Traceable

Objective

If you want to build an FPGA application that support the tracing of FPGA signals, you must enable the feature before you build the application.

Supported platforms

For platforms that support the tracing of FPGA signals, refer to Platform support on page 170.

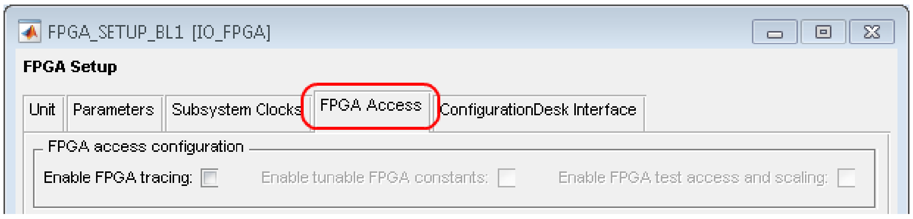
Limitations

For limitations on tracing FPGA signals, refer to Basics on Tracing FPGA Signals on page 171.

Method

To make FPGA signals traceable

1 Double-click the FPGA_SETUP_BL block and open the FPGA Access page.


FPGA_SETUP_BL1 [IO_FPGA]

FPGA Setup

Unit | Parameters | Subsystem Clocks | FPGA Access | ConfigurationDesk Interface

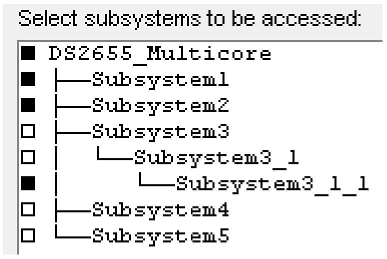
FPGA access configuration

Enable FPGA tracing: ☐ Enable tunable FPGA constants: ☐ Enable FPGA test access and scaling: ☐

2 On the FPGA Access page, select Enable FPGA tracing.

> **Note**
>
> **Running applications might stop, if too many FPGA signals are traced**
>
> If you trace more than 100 signals with 32-bit values (or 50 signals with 64-bit values) every millisecond with your experiment software, tracing might cause a task overrun that stops the application.
>
> These measures reduce the number of traced FPGA signals per millisecond:
>
> - Disable signal tracing if there is no need to trace FPGA signals.
> - Reduce the number of traceable signals. Refer to Basics on Tracing FPGA Signals on page 171.
> - Reduce the number of signals that that you trace with the experiment software. Only the values of signals traced with an experiment software are sent to the real-time processor and can cause a task overrun.

3 In the tree view, select the subsystems to be accessed by clicking the subsystem's name. If you click on a name, you select or clear the system and its subelements.


Select subsystems to be accessed:

```
■ DS2655_Multicore
■ ├──Subsystem1
■ ├──Subsystem2
□ ├──Subsystem3
□ │   └──Subsystem3_1
■ │       └──Subsystem3_1_1
□ ├──Subsystem4
□ └──Subsystem5
```

4 Select the FPGA signals to be traced:

- Select Trace input and output ports of subsystem to specify that incoming and outgoing signals of subsystems inside the selected FPGA subsystems are traceable.

- Select Trace all subsystem internal signals to specify that all signals internally used in the selected FPGA subsystems are traceable.
- Select Trace buses to specify that output signals of Simulink Bus Creator blocks and Bus Selector blocks of the selected FPGA subsystems can be traced directly.

Tip

Analyze lets you update the analysis of the number of signals that will be traceable for the current FPGA model. Analyze model displays the result of the analysis.
For analyzing, the dialog opens a temporary model `<FPGAModelName>_rtiFPGAtmp`. The dialog closes the temporary model at the end of the analysis.

Result

You enabled that the FPGA model INI file includes variables for tracing FPGA signals. The FPGA model INI will be generated during the build process of the FPGA application.

Related topics

Basics

Accessing the FPGA Application with the Experiment Software (ConfigurationDesk I/O Function Implementation Guide 📖)
Basics on Tracing FPGA Signals........ 171

HowTos

How to Exclude FPGA Signals from Tracing........ 176

How to Exclude FPGA Signals from Tracing

Objective

You can reduce the number of traceable signals by excluding parts of the model from tracing.

Supported platforms

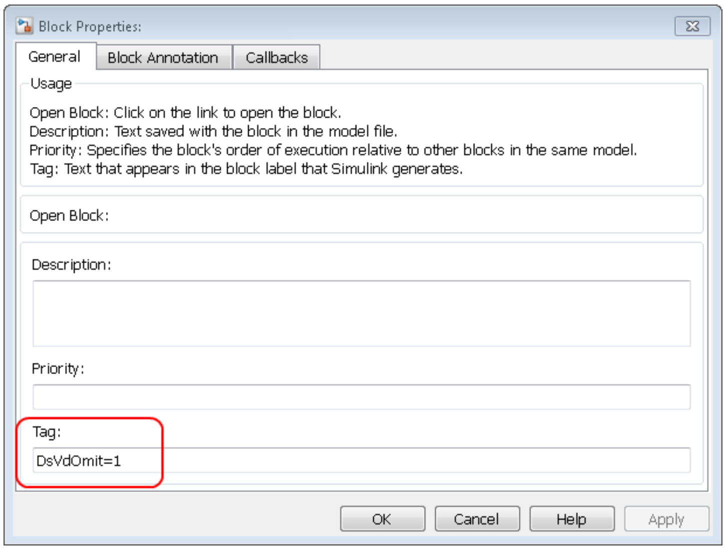
For platforms that support the tracing of FPGA signals, refer to Platform support on page 170.

Method

To exclude FPGA signals from tracing

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.



Usage

Open Block: Click on the link to open the block.
Description: Text saved with the block in the model file.
Priority: Specifies the block's order of execution relative to other blocks in the same model.
Tag: Text that appears in the block label that Simulink generates.

Open Block:

Description:

Priority:

Tag: DsVdOmit=1

For more information on the DsVdOmit tag, refer to Basics on Tracing FPGA Signals on page 171.

Result

You reduced the number of traceable FPGA signals. The signals of the excluded subsystems cannot be traced with an experiment software.

Related topics

Basics

HowTos

Adjusting Values of FPGA Constants

Introduction

You can adjust tunable FPGA constants with your experiment software.

Where to go from here

Information in this section

Basics on Tunable FPGA Constants

Adjusting tunable FPGA constants

If you enable the tracing of FPGA signals, you can also enable that tunable FPGA constants are provided.

Tunable FPGA constants FPGA constants are tunable only if they are added to a subsystem that you selected for FPGA signal tracing. If you enable that tunable FPGA constants are provided, the FPGA constants inside these subsystems are tunable.

For more information on selecting subsystems for tracing, refer to Basics on Tracing FPGA Signals on page 171.

Adjusting mechanism The adjusting mechanism is similar to the tracing mechanism: FPGA constants can be adjusted via FPGA variables that are added to the FPGA application during the build process. These variables provide the read access for the experiment software. The implementation of FPGA variables uses resources of the FPGA (flip-flops).

Adjusting values of FPGA constants with the experiment software A system description file (SDF file) containing additional information for tunable FPGA constants is generated during the build process of the processor application in ConfigurationDesk. You can use the SDF file with your experiment software to access and adjust the tunable FPGA constants. For details, refer to Experimenting with an FPGA Application on page 163.

Limitations

There are limitations for adjusting tunable FPGA constants:

- Floating-point constants can be adjusted only if the values are in the single or double precision floating-point formats (IEEE 754 standard). Other floating-point formats are not supported.
- Fixed-point constants can be adjusted with an experiment software only if the data width is 1 bit ... 64 bit.

  If you use Real-Time Testing for adjusting, fixed-point constants can be adjusted only with the following data types:
- Real-Time Testing is supported with the following limitations:
  - Support of fixed-point data types up to a data width of 32 bit.
  - No support of fixed-point data types for the MicroAutoBox III and SCALEXIO systems.
- The data acquisition of tunable FPGA constants is triggered by the trigger source for the tracing mechanism. The experiment software can change FPGA constants only with integer multiples of the trigger source period.

  For details on the trigger source, refer to Basics on Tracing FPGA Signals on page 171.
- When the experiment software changes FPGA constant values, it takes at least one task period until the new values are provided to the FPGA application.
- Adjusting FPGA constants is not supported for application processes containing a task configured in ConfigurationDesk as 'No jitter, low latency'.
- MicroAutoBox II only: No support of XIL API libraries.

Making FPGA constants tunable

Before you can tune FPGA constants with your experiment software, you must enable the generation of additional FPGA variables to support tunable FPGA constants. You enable this feature via the **FPGA_SETUP_BL** block. For instructions, refer to How to Make FPGA Constants Tunable on page 179.

Related topics

Basics

Accessing the FPGA Application with the Experiment Software (ConfigurationDesk I/O Function Implementation Guide 🕮)

HowTos

How to Make FPGA Constants Tunable..179

How to Make FPGA Constants Tunable

Objective

If you want to build an FPGA application that supports tunable FPGA constants, you must enable the feature before you build the application.

Supported platforms

For platforms that can provide tunable FPGA constants, refer to Platform support on page 170.

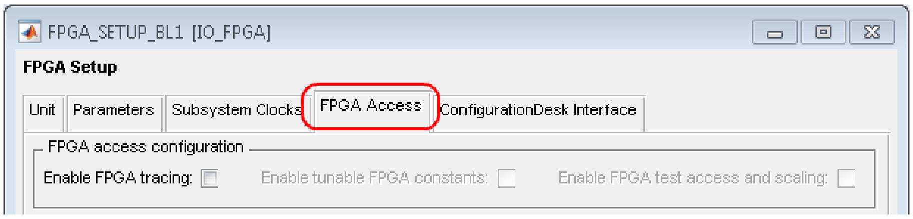
Limitations

For limitations on adjusting tunable FPGA constants, refer to Basics on Tunable FPGA Constants on page 178.

Method

To make FPGA constants tunable

1 Double-click the FPGA_SETUP_BL block and open the FPGA Access page.


FPGA_SETUP_BL1 [IO_FPGA]

FPGA Setup

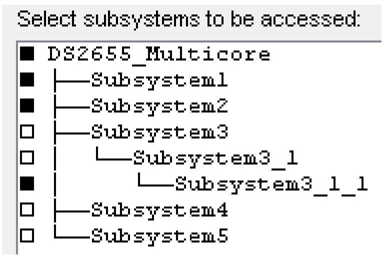
FPGA access configuration

Enable FPGA tracing: ☐ Enable tunable FPGA constants: ☐ Enable FPGA test access and scaling: ☐

2 Select Enable FPGA tracing.

3 Select Enable tunable FPGA constants.

4 In the tree view, select the subsystems including the FPGA constants that you want to be tunable by clicking the subsystem's name. If you click on a name, you select or clear the system and its subelements.


Select subsystems to be accessed:

```
■ DS2655_Multicore
■ ├──Subsystem1
■ ├──Subsystem2
□ ├──Subsystem3
□ │  └──Subsystem3_1
■ │     └──Subsystem3_1_1
□ ├──Subsystem4
□ └──Subsystem5
```

FPGA constants are tunable only if they reside in a subsystem that you selected for FPGA signal tracing. For more information on selecting signals for FPGA signal tracing, refer to Selecting signals for tracing on page 172.

Tip

Analyze lets you update the analysis of the number of constants that will be tunable for the current FPGA model. Analyze model displays the result of the analysis.

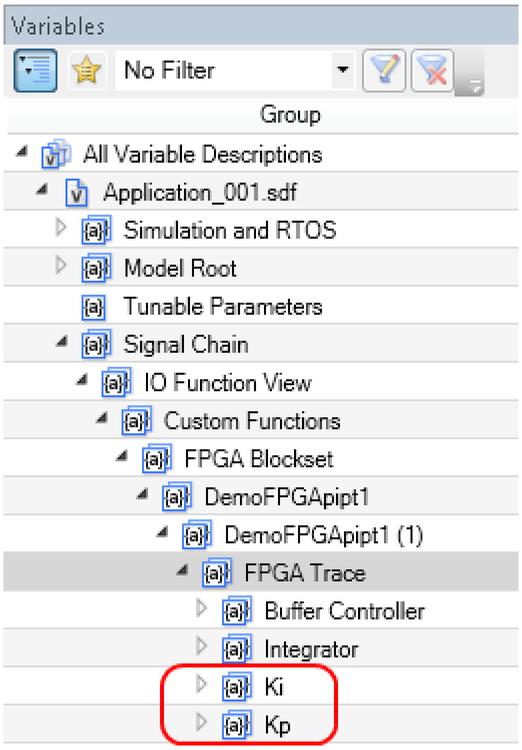
For analyzing, the dialog opens a temporary model `<FPGAModelName>_rtiFPGAtmp`. The dialog closes the temporary model at the end of the analysis.

Result

You enabled that the FPGA model INI file includes variables for tuning FPGA constants. The FPGA model INI will be generated during the build process of the FPGA application.

Example for tunable FPGA constants

The following example shows you the SDF file of a DemoFPGApipt1 demo project in ControlDesk. The FPGA constants Ki and Kp are added to the demo to replace the values that are sent from the processor model. The demo is built with tunable FPGA constants.



```
All Variable Descriptions
  Application_001.sdf
    Simulation and RTOS
    Model Root
    Tunable Parameters
    Signal Chain
      IO Function View
        Custom Functions
          FPGA Blockset
            DemoFPGApipt1
              DemoFPGApipt1 (1)
                FPGA Trace
                  Buffer Controller
                  Integrator
                  Ki
                  Kp
```

Related topics

Basics

Accessing the FPGA Application with the Experiment Software (ConfigurationDesk I/O Function Implementation Guide 📖)
Basics on Tunable FPGA Constants.......... 178
Experimenting with an FPGA Application.......... 163

Enabling the FPGA Test Access and Scaling

Introduction

You can enable the FPGA test access and scaling to substitute or scale interface signals.

Where to go from here

Information in this section

Basics on FPGA Test Access and Scaling

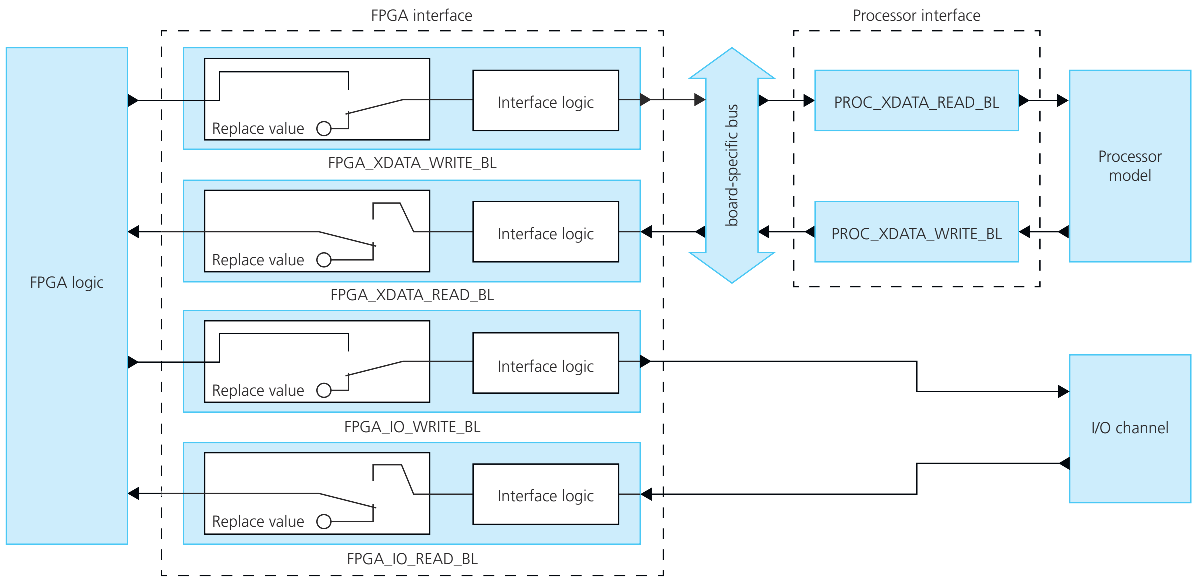
Introduction

FPGA test access and scaling are two methods to access and modify the interface signals of the FPGA application. If you enable FPGA test access and scaling, the build process adds additional FPGA variables to the FPGA application to support the features.

Supported platforms All platforms are supported, except for PHS-bus-based systems.

FPGA test access

An FPGA application with FPGA test access provides intervention points at the FPGA interface. These intervention points let you substitute values that are exchanged with the FPGA interface.

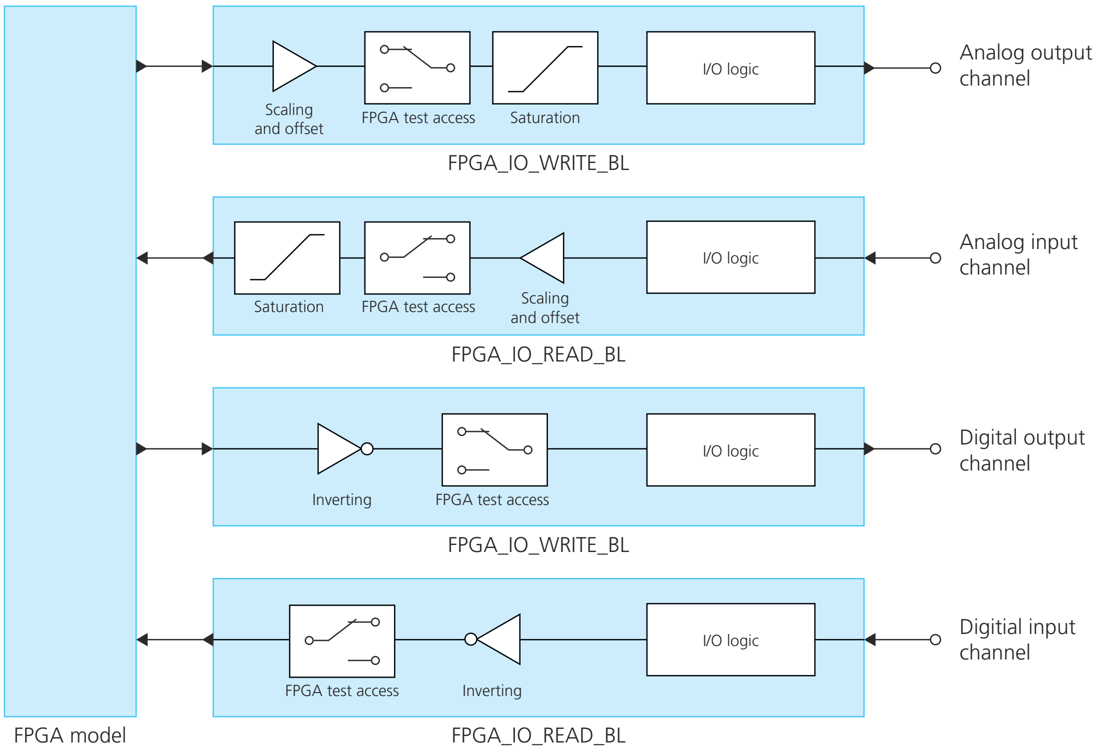


With the experiment software, you can set a value that replaces the original signal. A replace value becomes active when you switch the input or output from the original signal to the replace value.

FPGA Scaling

FPGA scaling lets you modify the signal of the FPGA interface as follows:

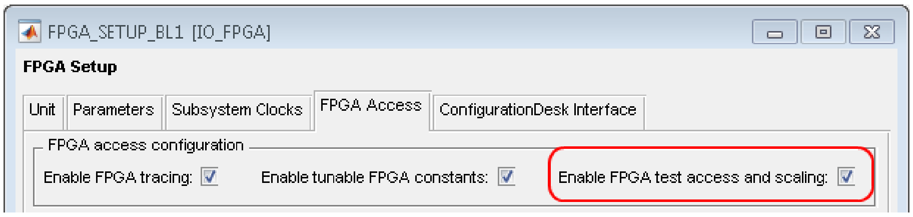
- Scaling of analog signals with a scaling factor.
- Adding signal offsets to analog signals.
- Saturating analog signals.
- Inverting digital I/O signals, including RS232/485 signals.



With the experiment software, you can set variables to scale the I/O signals.

Methods to scale analog signals There are two methods to scale analog signals with different intentions:

- To access and modify the analog signals with your experiment software, you enable FPGA test access and scaling.

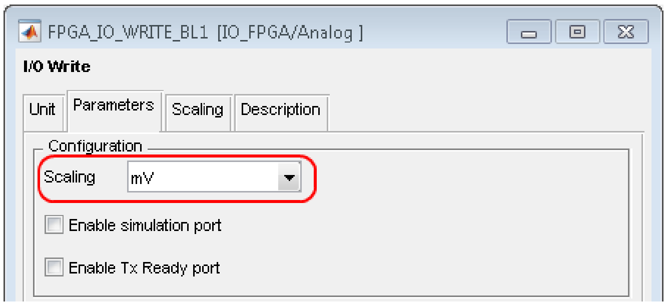

FPGA_SETUP_BL1 [IO_FPGA]

FPGA Setup

FPGA access configuration

Enable FPGA tracing: ✓ Enable tunable FPGA constants: ✓ Enable FPGA test access and scaling: ✓

- To conveniently handle the analog values, you scale the Bit values of the analog converter to the physical unit millivolt.


FPGA_IO_WRITE_BL1 [IO_FPGA/Analog]

I/O Write

Configuration

Scaling: mV

☐ Enable simulation port

☐ Enable Tx Ready port

This scaling is not intended to modify analog signals.

Using the experiment software for access

The following table shows you the types of FPGA variables that are added to the SDF file to support FPGA test access and scaling:

Access Method		Variable	Description
FPGA test access		<FPGASignal> - TA_Switchvalue	Lets you switch between the actual values used as input or output for the FPGA interface and the substitute value. ▪ 0: The actual values are used. ▪ 1: The substitute value is used.
		<FPGASignal> - TA_Replacevalue	Lets you specify the substitute value that can be used instead the actual values.
FPGA scaling	Analog signal modification	<FPGASignal> - Scaling factor	Lets you specify the scaling factor. The scaling factor amplifies the signal before it is saturated or replaced via FPGA test access.
		<FPGASignal> - Scaling offset	Lets you add an offset after the signal is scaled.
		<FPGASignal> - Saturation minimum value	Lets you set the minimum and maximum values to which the signals are saturated.
		<FPGASignal> - Saturation maximum value	
	Digital signal inverting	<FPGASignal> - Invert polarity	Lets you invert the digital I/O signal. ▪ 0: The signal is not inverted: A high-level voltage or a low-high transition represents a 1 and vice versa. ▪ 1: The signal is inverted: A high-level voltage or a low-high transition represents a 0 and vice versa.
	RS232/485 signal inverting	<FPGASignal> - Invert polarity	Lets you invert RS232/485 signals. ▪ 0: The signal is not inverted. ▪ 1: The signal is inverted.

Related topics

Basics

Accessing the FPGA Application with the Experiment Software (ConfigurationDesk I/O Function Implementation Guide)

HowTos

How to Enable FPGA Test Access and Scaling

Objective

If you want to build an FPGA application with FPGA test access and scaling, you must enable the feature before you build the FPGA application.

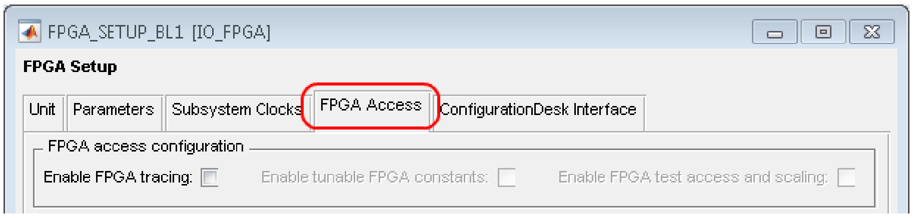
Supported platforms

For platforms that support FPGA test access and scaling, refer to Platform support on page 170.

Method

To enable FPGA test access and scaling

1 Double-click the FPGA_SETUP_BL block and open the FPGA Access page.


FPGA_SETUP_BL1 [IO_FPGA]

FPGA Setup

Unit | Parameters | Subsystem Clocks | FPGA Access | ConfigurationDesk Interface

FPGA access configuration

Enable FPGA tracing: ☐ Enable tunable FPGA constants: ☐ Enable FPGA test access and scaling: ☐

2 Select Enable FPGA tracing.

3 Select Enable tunable FPGA constants.

4 Select Enable FPGA test access and scaling.

Tip

Analyze lets you update the analysis of intervention points for FPGA test access and scaling for the current FPGA model. Analyze model displays the result of the analysis.
For analyzing, the dialog opens a temporary model `<FPGAModelName>_rtiFPGAtmp`. The dialog closes the temporary model at the end of the analysis.

Result

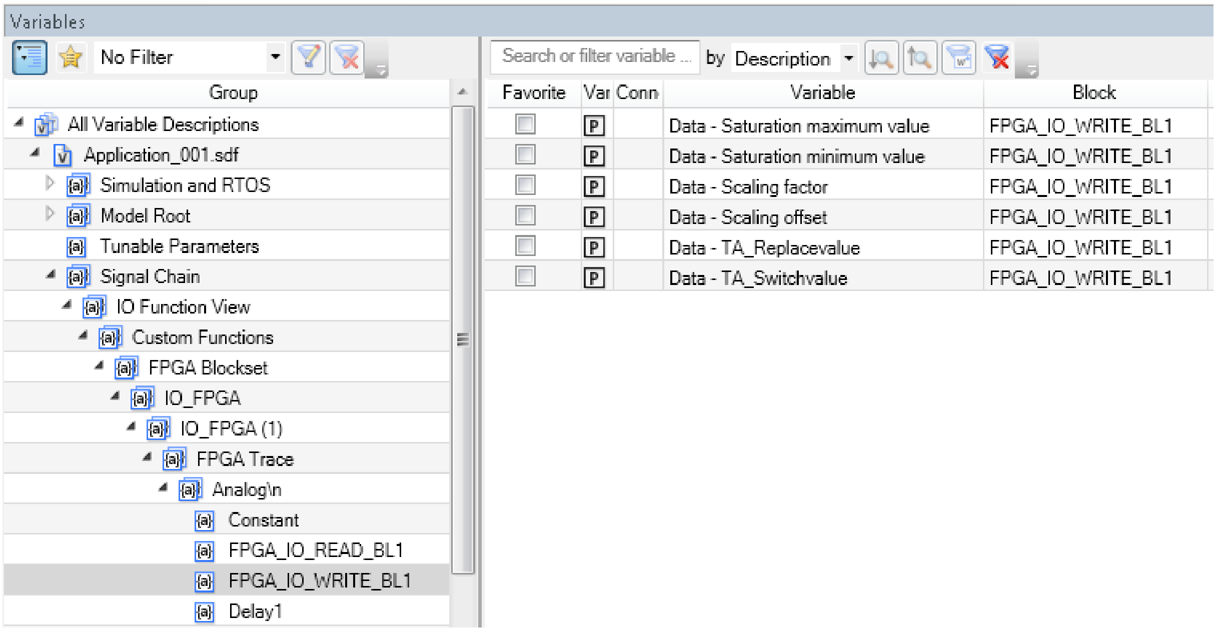
You enabled that the FPGA model INI file includes variables to provide the signal ports and function ports with FPGA test access and scaling. The FPGA model INI will be generated during the build process of the FPGA application.

To access the variables for FPGA test access and scaling, you use the SDF file that is generated when you build the real-time application.

For an overview of the added FPGA variables, refer to Using the experiment software for access on page 185.

Example for FPGA test access and scaling

The following example shows you the added variables for the FPGA_XDATA_READ_BL1 interface block for FPGA test access and scaling in ControlDesk. The example application is built with FPGA test access and scaling.



Group
All Variable Descriptions
Application_001.sdf
Simulation and RTOS
Model Root
Tunable Parameters
Signal Chain
IO Function View
Custom Functions
FPGA Blockset
IO_FPGA
IO_FPGA (1)
FPGA Trace
Analog\n
Constant
FPGA_IO_READ_BL1
FPGA_IO_WRITE_BL1
Delay1

Favorite	Var	Conn	Variable	Block
☐	P		Data - Saturation maximum value	FPGA_IO_WRITE_BL1
☐	P		Data - Saturation minimum value	FPGA_IO_WRITE_BL1
☐	P		Data - Scaling factor	FPGA_IO_WRITE_BL1
☐	P		Data - Scaling offset	FPGA_IO_WRITE_BL1
☐	P		Data - TA_Replacevalue	FPGA_IO_WRITE_BL1
☐	P		Data - TA_Switchvalue	FPGA_IO_WRITE_BL1

Related topics

Basics

Accessing the FPGA Application with the Experiment Software (ConfigurationDesk I/O Function Implementation Guide 📖)
Basics on FPGA Test Access and Scaling..182

How to Change the Data Type of Analog Signals

Objective

To scale and use analog signals with the required precision and value range.

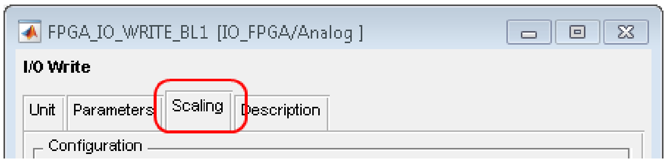
Supported platforms

For platforms that support the scaling of I/O signals, refer to Platform support on page 170.

Method

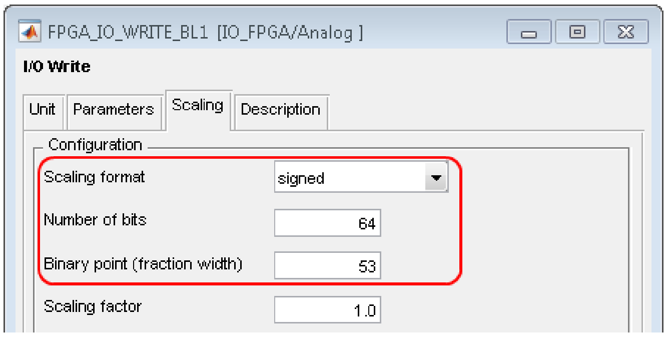
To change the data type of an analog signal

1 Double-click the FPGA I/O interface block (FPGA_IO_WRITE_BL or FPGA_IO_READ_BL) and open the Scaling page.



The **Scaling** page displays the scaling parameters that are supported for the selected I/O function.

2 On the **Scaling** page, specify the format and the bit width of analog signal values.


FPGA_IO_WRITE_BL1 [IO_FPGA/Analog]

I/O Write

Unit | Parameters | Scaling | Description

Configuration

Scaling format	signed
Number of bits	64
Binary point (fraction width)	53
Scaling factor	1.0

For more information on the parameters, refer to FPGA Interface RTI Blocks (RTI FPGA Programming Blockset - FPGA Interface Reference 📖).

Result

You changed the data type that is used by the selected I/O function for analog signal values.

The new data type is used as follows:

- New data type for the following scaling parameters:
  - Scaling factor
  - Scaling offset
  - Saturation minimum value
  - Saturation maximum value
- New data type for the **Data** port of the selected I/O function to support the new precision and value range.

Related topics

Basics

Basics on FPGA Test Access and Scaling 182

How to Specify Scaling Presettings

Objective

To specify scaling presettings for offline simulation and for the build FPGA application.

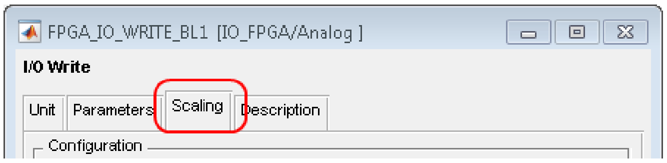
Supported platforms

For platforms that support the scaling of I/O signals, refer to Platform support on page 170.

Method

To specify the scaling presettings

1 Double-click the FPGA I/O interface block (FPGA_IO_WRITE_BL or FPGA_IO_READ_BL) and open the Scaling page.



The Scaling page displays the scaling parameters that are supported for the selected I/O function.

2 On the Scaling page, specify the settings for the scaling parameters.

Note

Scaling of analog Signals
It is not checked whether the specified and displayed values for analog I/O signals are supported by the data type or hardware. The values will be executed with the maximum accuracy and saturated to the minimum and maximum values that are supported by the data type and the hardware.
For specifying the data type, refer to How to Change the Data Type of Analog Signals on page 187.

For more information on the parameters, refer to FPGA Interface RTI Blocks (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮).

Result

You specified the scaling presettings for the selected I/O function. The settings are used for offline simulation and for the build of the FPGA application.

Related topics

Basics

Basics on FPGA Test Access and Scaling..182

References

FPGA Interface RTI Blocks (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮)

Adapting and Updating Existing FPGA Models

Introduction

You can use existing FPGA models to use them in your current project.

Where to go from here

Information in this section

Updating FPGA Models with Settings of FPGA Custom Functions

Introduction

You can update your FPGA model with imported parameter settings of an FPGA custom function in ConfigurationDesk.

How to Update FPGA Models with Imported Parameter Settings

Use case

If you add a FPGA application as FPGA custom function block to ConfigurationDesk, you can import changed settings to update the FPGA model.

Preconditions

The following preconditions must be fulfilled:

- The FPGA application is build and added to ConfigurationDesk as an FPGA custom function block.
- The blocks of the RTI FPGA Programming Blockset are not moved to other I/O channels.

Method

To update FPGA models with imported parameter settings

1. Double-click the FPGA_SETUP_BL block and open the ConfigurationDesk Interface page.
2. From the Select recent project parameter, select the project from which you want to import the parameter settings.

   Tip

   Click Refresh list to import the list of recent projects from ConfigurationDesk. If needed, the framework opens ConfigurationDesk to refresh the list.

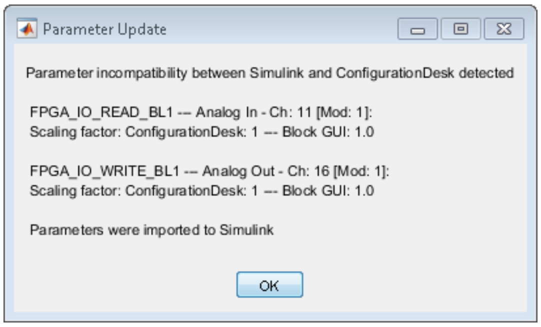
3. Click Import parameters.

   The framework opens the ConfigurationDesk project to import the parameters settings.

   If several instances of the FPGA custom function are instantiated, the Select dialog opens.
4. If the Select dialog opens, select the FPGA custom function from which you want to import the parameter settings and click OK.

Result

The framework analyzes the FPGA custom function and imports the FPGA parameter settings. At the end of the import process, the **FPGA Update** dialog opens. The dialog displays the settings that have been updated.



Parameter Update

```
Parameter incompatibility between Simulink and ConfigurationDesk detected

FPGA_IO_READ_BL1 --- Analog In - Ch: 11 [Mod: 1]:
Scaling factor: ConfigurationDesk: 1 --- Block GUI: 1.0

FPGA_IO_WRITE_BL1 --- Analog Out - Ch: 16 [Mod: 1]:
Scaling factor: ConfigurationDesk: 1 --- Block GUI: 1.0

Parameters were imported to Simulink
```

Related topics

Basics

Accessing the FPGA Application with the Experiment Software (ConfigurationDesk I/O Function Implementation Guide 📖)
Basics on FPGA Test Access and Scaling.......... 182

Adapting Xilinx FPGA Models

Introduction

To integrate an FPGA application into a dSPACE system, you must replace certain Xilinx blocks with blocks from the RTI FPGA Programming Blockset.

Where to go from here

Information in this section

Replacing the Functionality of the Xilinx System Generator Setup Block

Introduction

The Xilinx System Generator setup block remains in the FPGA model, but all the settings for the setup functionality are specified in the FPGA_SETUP_BL block (it is assumed that you do not use Xilinx System Generator blocks in various model hierarchies).

Setup block in the FPGA subsystem

The FPGA_SETUP_BL block assumes control about the RTI blocks from the FPGA Interface library and the Xilinx blocks used in the FPGA model.

Specifying the framework Before you can configure the blocks from the RTI FPGA Programming Blockset in your FPGA subsystem, you must specify the framework to be used. A basic set of FPGA functionality is implemented in the framework INI file. All the related settings for the block dialogs are also defined there.

Executing model actions The FPGA_SETUP_BL block can be used to start several Xilinx tools without leaving your dSPACE environment. The required Xilinx blocks are encapsulated by the dSPACE RTI blocks.

The following FPGA model actions can be triggered:

- FPGA Build

  Starts the XSG code generation followed by the Xilinx FPGA implementation process to build the FPGA application.

- Timing Analysis

  Starts the Timing Analyzer to check whether the timing constraints in your FPGA subsystem are kept.
- HDL Simulation

  Starts the comparative simulation of the Simulink model with the HDL-coded FPGA application.

Replacing Xilinx Gateway Blocks

Introduction

Xilinx Gateway In/Gateway Out blocks in the FPGA model must be replaced by blocks from the FPGA Interface library.

Xilinx Gateway blocks

The Xilinx Gateway blocks provide the interface between the Simulink inports and outports and the signals processed within the FPGA model.

A Gateway In block provides an inport to your FPGA model. It converts Simulink integer, double and fixed-point data types into the Xilinx System Generator fixed-point data type.

A Gateway Out block provides an outport from your FPGA model. It converts the Xilinx System Generator fixed-point data type into the Simulink double data type.

Reading a signal with FPGA Interface blocks

If the FPGA model reads a signal, you usually have a Simulink Inport block followed by a Xilinx Gateway In block that realizes the specified data type conversion.

If the input signal comes from the processor model, you must replace the Xilinx Gateway In block with an FPGA_XDATA_READ_BL block. The Simulink Inport block must also be removed because the counterpart in the processor model (PROC_XDATA_WRITE_BL block) communicates with the FPGA block without a signal line.

The replaced blocks from the FPGA Interface library also implement the necessary data type conversion.

If the input signal comes from an external hardware signal, you must replace the Xilinx Gateway In block with an FPGA_IO_READ_BL block. The Simulink Inport block must also be removed because it is no longer required. The configuration of the block depends on the signal type, for example, on whether the hardware delivers an analog or digital input signal.

Writing a signal with FPGA Interface blocks

If the FPGA model writes a signal, you usually have a Xilinx Gateway Out block that realizes the specified data type conversion, followed by a Simulink Outport.

If the output signal is to be written to the processor model, you must replace the Xilinx **Gateway Out** block with an **FPGA_XDATA_WRITE_BL** block. The Simulink **Outport** must also be removed because the counterpart in the processor model (**PROC_XDATA_READ_BL** block) communicates with the FPGA block without a signal line.

The replaced blocks from the **FPGA Interface** library also implement the necessary data type conversion.

If the output signal is used for an external hardware signal, you must replace the Xilinx **Gateway Out** block with an **FPGA_IO_WRITE_BL** block. The Simulink **Outport** must also be removed because it is no longer required. The configuration of the block depends on the signal type, for example, on whether the hardware expects an analog or digital output signal.

Related topics

Basics

Implementing the Processor Interface to the Processor Model.......113
Modeling External I/O Access.......52
Replacing the Functionality of the Xilinx System Generator Setup Block.......194

Resetting FPGA Models

Resetting the Block Parameters to the Initial Values

Introduction

With the script interface, you can reset the block parameters of the FPGA framework to the initial values.

Resetting to the initial values

The RTI FPGA Programming Blockset provides a script interface that you can use to reset the block parameters of the FPGA framework to the initial values by executing the `FPGAFrameworkUpdate` script function in the MATLAB Command Window.

```
rtifpga_scriptinterface('FPGAFrameworkUpdate',
 <SimulinkHandle>, 'ReInit')
```

The script considers all subsystems that are contained in the model/subsystem which is specified by the Simulink handle. The parameters of the blocks are reset to the initial values.

Example: The following script will reset the FPGA framework for any FPGA subsystems found in the processor model that is called *MyProcModel*. The specified values of the block parameters will be set to the initial values.

```
ProcModelHandle = get_param('MyProcModel','handle')
rtifpga_scriptinterface('FPGAFrameworkUpdate',
 ProcModelHandle,'ReInit')
```

Related topics

References

FPGAFrameworkUpdate (RTI FPGA Programming Blockset Script Interface Reference 🕮)

Migrating FPGA Models

Introduction

You can migrate FPGA models to another dSPACE hardware or update the framework of an FPGA model to the current version.

Where to go from here

Information in this section

Migrating to Different FPGA Hardware

Introduction

The RTI FPGA Programming Blockset supports different dSPACE FPGA boards and hardware platforms. If you want to use an existing FPGA model on another platform, it might be not sufficient to select the hardware-specific framework. Some modifications are required.

Intended use cases

The automatic migration mechanism is intended primarily to migrate from one framework to another framework of the same platform:

- From one SCALEXIO FPGA base board framework to another. For example:
  - From the *DS2655 (7K160) FPGA Base Board* framework to the *DS2655 (7K410) FPGA Base Board* framework.
  - From the *DS2655 (7K160) FPGA Base Board* framework to the *DS6601 (KU035) FPGA Base Board* framework.

Tip

All FPGA models of the *DS2655 (7K160) FPGA Base Board* framework can be migrated to the *DS6601 (KU035) FPGA Base Board/DS6602 (KU15P) FPGA Base Board* frameworks. The DS6601 and DS6602 FPGA base boards are compatible and provide more FPGA resources.
However, you have to use the same I/O modules and to consider the I/O slot assignments.

- From a DS5203 framework to another DS5203 framework. For example, from DS5203 (7K325) to DS5203 (7K410). The FPGA board that is supported by the new framework must have enough FPGA resources to implement the migrated FPGA model.
- From *FPGA1401Tp1 (7K325) with Multi-I/O Module (DS1552)* framework to *FPGA1401Tp1 (7K325) with Multi-I/O Module (DS1552B1)* framework.

The automatic migration also works for the migration from one platform to another, for example from DS5203 to FPGA1401Tp1, but their I/O functions have no or only little compatible compliances.

Migrating FPGA models

Note

- Before you start a migration, you should make a backup of your model.
- To build and run a migrated FPGA model, the new FPGA board must have sufficient FPGA resources.

If you switch to another framework, the framework migration process is started:

1. A dialog opens that lists all the blocks in your model that might be incompatible.
2. Then, a second dialog opens in which you have to confirm the migration.
3. If the migration is running, the compatible blocks in the model are replaced and parameterized again.

   Differently parameterized or renamed blocks will not be replaced.
4. If the migration is finished, a dialog opens that lists the incompatible blocks that you have to replace manually.

The parameterization of the found incompatible blocks is empty. You are not able to reparameterize the blocks concerning the selected framework. While you have not replaced the incompatible blocks, you can switch back to the framework used for the configuration to reset the block parameters to their former values.

Updating FPGA Frameworks to the Current Version

Introduction

If you have implemented your FPGA application using an older version of the RTI FPGA Programming Blockset, and you want to use it with a current version, the FPGA framework must be updated. This involves only a few internal modifications that do not affect the blocks' inputs and outputs or their parameters.

Updating the RTI FPGA Programming Blockset

If you have implemented your FPGA application using RTI FPGA Programming Blockset Version 1.1 and higher, and want to use it with RTI FPGA Programming

Blockset 3.11, the framework will automatically update itself to the current framework version.

The update handles all the subsystems in the model/subsystem. The parameters of the blocks remain unchanged after updating to the current framework version.

Troubleshooting

Introduction

When working with the RTI FPGA Programming Blockset there are some problems which can arise.

Problems and Their Solutions

Overview

Known problems and their solution are grouped by their occurrence:

- General modeling issues on page 201
- Implementing the FPGA model on page 201
- Building an FPGA application on page 204
- Building a processor application on page 207
- Running the FPGA application on a dSPACE system on page 207

General modeling issues

Copying of model parts takes a long time Copying the RTI FPGA Programming Blockset's function blocks might take too long. As of RTI FPGA Programming Blockset 3.3, the blockset automatically analyzes the FPGA model and reassigns new hardware resources. The analysis might take too long.

You can speed up the copy & paste process for the current session by deactivating the automatic reassignment of new hardware resources:

- In the MATLAB Command Window, type the following two lines:

```
global G_RTIFPGA_COPYBLOCK_CALLBACK_ENABLE
G_RTIFPGA_COPYBLOCK_CALLBACK_ENABLE = false
```

Implementing the FPGA model

The following errors might occur in implementing the FPGA model via the Xilinx block library and trying to perform actions like offline simulation or FPGA build. The messages appear in the MATLAB command prompt and/or in a MATLAB or Simulink error dialog box.

Incorrect display of sample time information When you use Simulink's Sample Time Display, the displayed sample time colors might be wrong when the model is in the compiled state, for example, during the offline simulation. This is caused by a display problem when using the Xilinx block library.

- Always update your model (**`Ctrl+D`**) before you interpret the colors. After the model update, the colors are correct.

Error message (Solver types)

```
The periodic sample time ... is not allowed because the ratio
of this sample time over base rate (...) is greater than the
maximum value of uint32.
```

or

```
Invalid setting for fixed-step size (...) in model '...'.
All sample times in your model must be an integer multiple of
the fixed-step size.
```

This message might appear in updating the Simulink model including an FPGA subsystem (Update Diagram, Ctrl-D) if the solver options are configured as a Fixed-step type. Xilinx supports solver options configured as a Variable-step type only. The diagram update can be launched explicitly or triggered implicitly when you run an offline simulation, an FPGA build process, etc.

- If FPGA subsystems are part of the model, always choose a Variable step solver to perform an offline simulation, an FPGA build process, etc. (model mode: FPGA-Build/Offline Simulation).
- Choose a Fixed-step solver only if there are no more FPGA subsystems in the model (model mode: Processor-Build).

Error message (Signal periods)

```
The period of the signal driving the enable of this block is
inappropriate. The period must be an integer multiple of the
gcd of signals driving the following input ports:
d : ...
enable period: ...
```

This message might appear in updating the Simulink model including an FPGA subsystem (Update Diagram, Ctrl-D). The diagram update can be launched explicitly or triggered implicitly when you run an offline simulation, an FPGA build process, etc. The error message is reported by an RTI FPGA Interface block of the RTI FPGA Programming Blockset.

- Control the data path driving the reporting RTI FPGA Interface block.
- If a Down Sample block from the Xilinx blockset was used in this path, make sure that there is always a corresponding Up Sample block before driving the RTI FPGA Interface block.
- Make sure that the downsampling factor of the whole path exactly matches the upsampling factor. For example, there could be two cascaded Down Sample blocks each with factor 2 and one Up Sample block with factor 4.
- If a Xilinx block defining its own sample period, for example, a Counter block, drives the RTI FPGA Interface block, make sure that the defined sample period matches the offline simulation period specified in the FPGA_SETUP_BL block.

- If the sample period must be higher than the system period, for example, if the counter has to run at a lower rate to count slower, use an Up Sample block before driving the RTI FPGA Interface block. The factor by which the driving block runs slower must be upsampled precisely.

Error message (FPGA_SETUP_BL block)

```
All Xilinx Blocks must be contained in a level of hierarchy
with a System Generator Token.
```

This error might appear if there is no block on the same hierarchy level as the FPGA_SETUP_BL block within the FPGA subsystem. This happens, for example, if all the other blocks in the FPGA model are encapsulated by subsystems, i.e., only subsystems are on the same level as the FPGA_SETUP_BL block.

1. Place at least one block on the same hierarchy level as the FPGA_SETUP_BL block. This might be a dummy block, for example, a Constant block from the Xilinx library or an FPGA_IO_READ_BL block from the RTI FPGA Programming Blockset.
2. Restart the MATLAB session.
3. Restart the offline simulation.

Error message (Resource Estimator block)

```
Error using ==>resourceestimatoraction>checkForSysgenToken at 51
The Resource Estimator requires instancing of a Sysgen Token
in the same Subsystem
```

This error might appear if you try to use a Xilinx Resource Estimator block in an FPGA subsystem.

- Use the Update Diagram command (Ctrl-D) in the model to instantiate a System Generator Setup block (Sysgen Token).
- Restart the resource estimation.

Error message (Offline simulation)

```
No data is transmitted to and from an FPGA subsystem during
an offline simulation although the corresponding interface
blocks are modelled and parameterized correctly.
```

This error appears if the FPGA subsystem has been replaced by an FPGA model INI file in the Processor Setup block. FPGA Model INI files cannot be simulated.

- Check the settings in the Processor Setup block and make sure that the affected FPGA subsystem is assigned to the FPGA board you want to simulate. It is not enough to include the FPGA subsystem in the model.

Warning message (Sample time inheritance)

```
Warning: Source '.../FPGA_XDATA_WRITE_BL1/phs_write_register/
phs//rtlib_sim_model/Data Store Read' specifies that its sample
time (-1) is back-inherited. You should explicitly specify the
sample time of sources.
```

This warning appears if the offline simulation model of an RTI FPGA block tries to inherit the sample time of its corresponding block in the processor model.

- This is the intended behavior of the RTI FPGA blocks, specified by the default settings. The message can be ignored.

- You can suppress the message:
  - You can disable this diagnostic by setting the **Source block specifies -1 sample time** diagnostic to `none` in the Sample Time group on the Diagnostics pane of the Configuration Parameters dialog box.
  - You can explicitly set the **Simulation ports sample time** to a value other than -1 for each affected RTI FPGA block on its Parameters page.

Error message (Invalid Simulink handle)

```
Trouble running xlUpdateIcon(3617,0003662109375,-1,-1,-1,
{'\fontsize{11pt}\bf In ',},{' ',},{'',},'','bc55d28f');;
Error was: Error using ==> get_param
```

This error might appear if the Xilinx System Generator uses an invalid Simulink handle format due to specific regional settings.

- Change your Windows regional settings to `English`

  or
- Change your Windows settings for the decimal point to `point` instead of `comma`.

Warning message (Broken connections)

```
Warning: Matching "Goto" for "From" '...' not found
```

This warning appears if the offline simulation connections of corresponding blocks in the FPGA subsystem and the processor model are broken. This might happen if you simulate an FPGA subsystem in a processor model and then change the assignment in the Processor Setup block from the FPGA subsystem to the FPGA model INI file with the FPGA subsystem remaining in the model.

- This warning can be ignored for RTI FPGA blocks because the broken connections are restored whenever they are required for offline simulation. When the warning appears no offline simulation is performed with the affected blocks.
- You can eliminate this warning by regenerating the RTI FPGA block structures.
  - For blocks of the Processor Interface, perform an Update Diagram command (Ctrl-D)
  - For blocks of the FPGA Interface, open the FPGA Setup block, reselect the current FPGA framework and close the dialog by clicking OK.

Building an FPGA application

The following errors might occur in building an FPGA application. They appear in the MATLAB Command Window and/or in a MATLAB or Simulink error dialog.

Path length exceeds maximum length During the build process the path length can get unpredictably long, as shown by the following example:

```
<model_folder>\<modelname>_rtiFPGA\
  <modelname>_<ApplicationID\sysgen\hdl_netlist\cm.runs\
  cm_c_counter_binary_v12_0_0_synth_1\
  cm_c_counter_binary_v12_0_0*.*
```

It is not possible to specify a maximum build path length that is guaranteed to work.

The windows operating system limits the maximum path length to 260 characters. This limitation might lead to miscellaneous error messages during file system operations.

1. Minimize the total length of the working folder, the FPGA build folder, the model name, and the name of the FPGA subsystem to be built. For details on specifying the FPGA build folder, refer to Parameters Page (FPGA_SETUP_BL) (RTI FPGA Programming Blockset - FPGA Interface Reference 🕮). For details on minimizing the total length of the working folder, refer to Xilinx Answer Record 52787 (http://www.xilinx.com/support/answers.html).
2. Restart the build process.

Error message (Build path)

```
RTI Build Error
An error occurred in the build of the Xilinx System Generator
Error in compiling/generating the netlist.
```

… and …

```
Standard exception: XNetlistEngine:
An exception was raised:
com.xilinx.sysgen.netlist.NetlistInternal: couldn't run
/coregen.exe:
```

There are several reasons for this error:

- One typical reason for this error is a too-long build path, which leads to problems when the Xilinx tools perform file system operations. For details, refer to Path length exceeds maximum length on page 204.
- The Xilinx CORE Generator might run into memory problems when using the Java virtual machine. The reason might be either not enough or too much reserved memory.
  1. Check the CORE Generator log file to determine the exact error reason.
     - Use a file search to find the log file `second_pass_coregen.txt` in the subfolders of the RTI FPGA build folder `<WorkingDir>/<ModelName>_rtiFPGA/...`
     - Open the log file and find the following error: `ERROR:coreutil:195 - Could not create Java virtual machine – JVM`
  2. Increase the memory size available for Java objects in MATLAB.
     - Set the **Java Heap Size** value to at least 768 MB in the **General - Java Heap Memory** page of MATLAB's **Preferences** dialog.
  3. See Xilinx Answer Records 20708 and 20780 for further details (http://www.xilinx.com/support/answers.html).
- The Xilinx CORE Generator fails to generate a core for a Constant Multiplier (CMult) configured with a factor of 1 and a bit width of 1.
  1. Check the CORE Generator log file to determine the exact error reason.
     - Use a file search to find the log file `second_pass_coregen.txt` in the subfolders of the RTI FPGA build folder `<WorkingDir>/<ModelName>_rtiFPGA/...`

  - Open the log file and find the following error: `ERROR:sim - PortBWidth: Value '1' is out of range '[2..64]'`
2. Replace all CMult blocks configured as described above by Delay blocks. This results in the same behavior, but a Delay block only needs one register entry, whereas a CMult block allocates a DSP block.

Error message (Lack of resources)

```
Starting mapping...
------------------------------------------------------------
RTI Build Error
Mapping failed because of overmapping. Your design is probably
too large for the FPGA you use. For further details, refer to
the Xilinx log file, which is located in:
...
```

This error appears if the FPGA application does not fit on the FPGA due to a lack of resources. The application is too large for the FPGA device.

- Reduce the resource requirements by removing parts of the FPGA model if applicable.
- Check the bit widths of the signals used in the design. The higher the bit widths, the higher the resource requirements.
- Change the implementation-specific settings for each Xilinx block. Several Xilinx blocks, for example, the Mult, Counter, CMult or AddSub blocks, provide an Implementation page in their block dialogs:
  - Try using embedded multipliers/DSP48 blocks or not.
  - Try using behavioral HDL or Cores.
  - Try using distributed RAM or Block RAM.
  - Try using optimization for area instead of speed.
- Reduce the resource-dependent settings in the RTI FPGA Interface blocks, for example, the buffer depth.
- Reduce the overall number of RTI FPGA Interface blocks used.
- Use the Xilinx Resource Estimator block to quickly check which decisions save resources. For Resource Estimator usage, see Error message (Resource Estimator block) on page 203.

Error message (Timing constraints)

```
RTI Build Error
At least one timing constraint of your system was not met.
You can analyze your design with the Timing Analyzer.
For further details, refer to the log file located in:
...
```

or

```
RTI Build Error
Mapping was not successful. For further details, refer to
the Xilinx log file, which is located in:
...
```

This error appears if the FPGA application cannot run on the FPGA due to timing issues. The design cannot run at the FPGA's clock frequency.

- Run a Timing Analysis to determine exactly which data path of your model does not meet timing constraints.

- Make the path a multicycle path by downsampling. Use the Xilinx blocks Down Sample and Up Sample in the data path.
- Add additional latency to the data path by:
  - Inserting dedicated Xilinx Register or Delay blocks in the data path.
  - Increasing the Latency parameter of Xilinx blocks within the data path.
- Change the implementation-specific settings for each Xilinx block. Several Xilinx blocks, for example, the Mult, Counter, CMult or AddSub blocks, provide an Implementation page in their block dialogs:
  - Use Cores instead of behavioral HDL.
  - Use optimization for speed instead of area.
  - Enable pipelining.

Building a processor application

The following errors might occur when performing a Simulink Coder build to generate a processor application. They appear in the MATLAB Command Window and/or in a MATLAB or Simulink error dialog.

Error message (Wrong model mode)

```
Model contains dSPACE FPGA blocks.
```

This error appears when you try to start a Simulink Coder build process for a model that still contains FPGA subsystems.

- Switch the model mode from FPGA-Build / Offline Simulation to Processor-Build by using the Processor Setup block (PROC_SETUP_BLx). This step temporarily removes all FPGA blocks from the model.

Running the FPGA application on a dSPACE system

The following errors might occur in running the built FPGA application on a dSPACE system. You can find the source of the errors by experimenting with dSPACE's ControlDesk.

Error message (No FPGA application loaded)

```
ds5203_init(0x00): No FPGA application loaded. (302)
```

This error appears when you download a processor application that tries to access an FPGA application but no FPGA application is running on the FPGA board/module.

- Choose the programming options into ram or into flash for the FPGA application in the Processor Setup block (PROC_SETUP_BLx) before performing a Simulink Coder build process to download the processor application.
- If you have chosen the into flash programming option, the autoboot option from the FPGA board's/module's flash memory has to be enabled and the dSPACE hardware has to be switched off and on once to start programming the FPGA from the flash.

Error message (Incompatible FPGA model INI file)

```
ds5203_program(0x00): The model ini-File you use is
incompatible to your Hardware! (332)
```

This error appears when you try to use an FPGA model INI file to program an FPGA board or FPGA module the INI file is not intended for.

- Choose the correct FPGA framework for your hardware configuration. There are FPGA frameworks available for each possible combination of FPGA boards or FPGA modules and I/O modules (piggyback boards).
- Adapt your hardware system, by mounting the correct I/O module on the FPGA board or FPGA module. The FPGA model INI file might be correct but not suitable to the hardware configuration.
- Check your dSPACE installation. If you try to use a new FPGA board or module with an old dSPACE installation, the new board or module might not be recognized as compatible by the software. Refer to the user documentation of the RTI FPGA Programming Blockset to get information on the supported hardware.

Error (Simulation results)

```
The online simulation results do not match the offline
simulation results from the Simulink model.
```

This error might appear for several reasons:

- Synchronous vs. asynchronous FPGA and processor applications:
  - In a Simulink simulation, the communication between the FPGA model and the processor model is deterministic. Both applications run synchronously on a common time base.
  - On the dSPACE hardware, the FPGA application runs independently of the processor model. Each application runs asynchronously with its own time base.
  - The values of the Simulink offline simulation and the online simulation on dSPACE hardware should match, but not the timing of the communication between FPGA and processor application.
- Offline simulation period vs. FPGA period
  - To guarantee precise results, the offline simulation period must match the FPGA's period. (This is in contrast to online simulation, which can run at a higher sample period because simulating the FPGA's period takes a lot of computing power).
  - If the FPGA application contains timing-dependent parts such as discrete integrators that accumulate by a time factor, you must be sure to adapt the time factor to the FPGA’s period before building the FPGA application if the offline simulation was performed at a different sample period and therefore with a different time factor.

General note Even if no errors occurred during modeling, simulating, and building, your FPGA application is not guaranteed to work correctly. Some combination of factors might prohibit the execution of your FPGA application, e.g., combinations of the operating temperature, the power of the entire hardware system, technological limitations of the FPGA, the sample rate, etc. Some FPGA applications do not start at all, others start correctly but fail during operation.

Due to the complexity of FPGA applications and their dependency on the used FPGA hardware, the Xilinx design tools cannot ensure that all possible design problems are detected during the analysis and simulation of the FPGA model.

To solve the problem, you have to modify the FPGA model and make sure that you consider generally accepted FPGA design rules to ensure a stable and reliable

FPGA application (refer to Audience profile on page 9). For more information, contact dSPACE Support.