RTI FPGA Programming Blockset

Handcode Interface 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

Contents

About This Guide

Introduction

This guide provides information about the RTI FPGA Programming Blockset handcode implementation flow. You will learn how to implement an FPGA application for dSPACE hardware, for example, the DS5203 FPGA Board. You will also learn how to access the FPGA application from a processor application running on a processor board.

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).
- Verifiable experience on implementing HDL code (VHDL or Verilog).
- Implementing FPGA designs with Xilinx® design tools.

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.

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

Basic feature of the blockset

The handcode interface of the RTI FPGA Programming Blockset allows you to integrate a custom, code-based FPGA application in a dSPACE hardware and software environment.

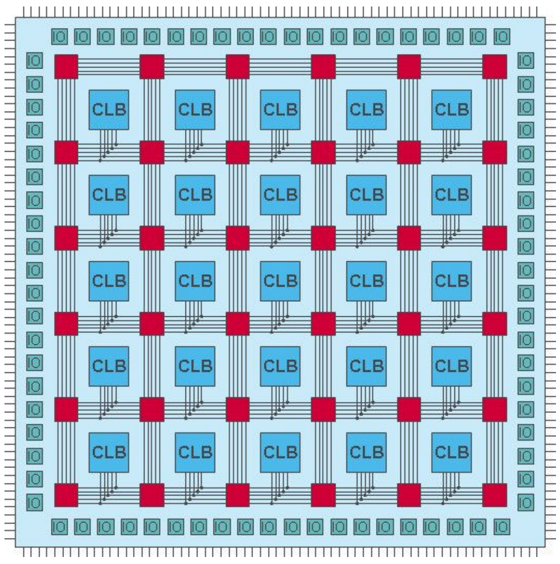
Where to go from here

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.

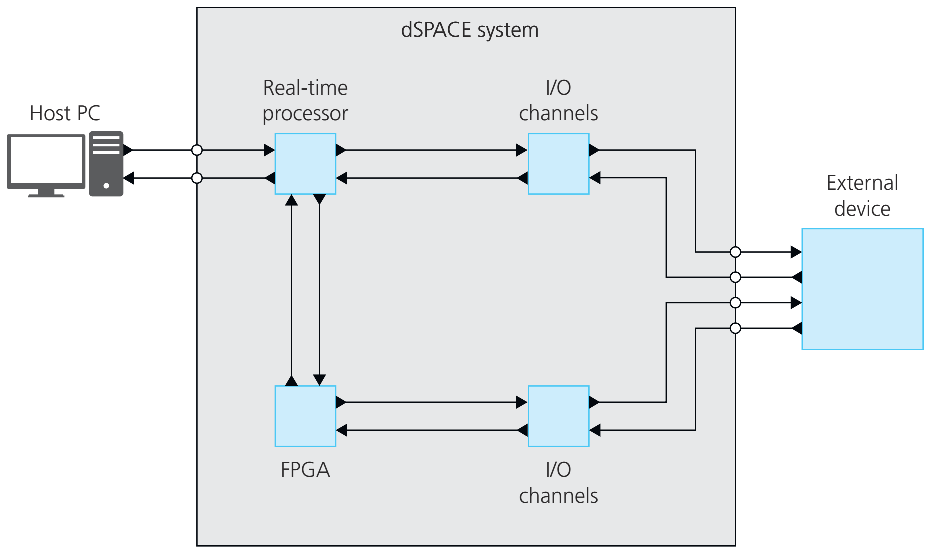
Basics on the System Architecture

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

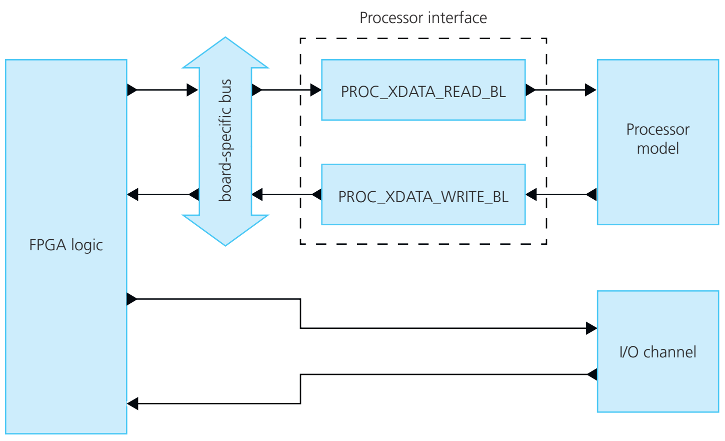
  Device that can be connected to the dSPACE system, such as an ECU.

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 real-time processor. Furthermore, script functions let you execute functions via command line or M file.

Overview of the blockset



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. 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 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 ConfigurationDesk. This represents a connection to the real-time processor in your FPGA model by a function port that you can connect to a related model port block.

For details on adding FPGA applications to the signal chain in ConfigurationDesk, refer to Adding FPGA Applications to the Signal Chain (ConfigurationDesk Real-Time Implementation Guide 🕮).

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 initialize the framework. For more information on the script functions, refer to Script Functions Supporting the Handcode Interface (RTI FPGA Programming Blockset Script Interface Reference 🕮).

Related topics

Basics

Basics on Exchanging Data Between Processor and FPGA.. 13
Exchanging Data With the Processor Model.. 15

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 processor interface provides 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 15.

Details on the access types

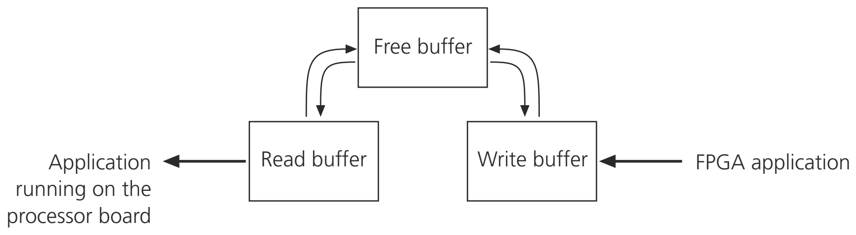
Register access Register access lets you access a scalar value in the register. The data is identified by the specified channel number. The values are transmitted element by element.

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.

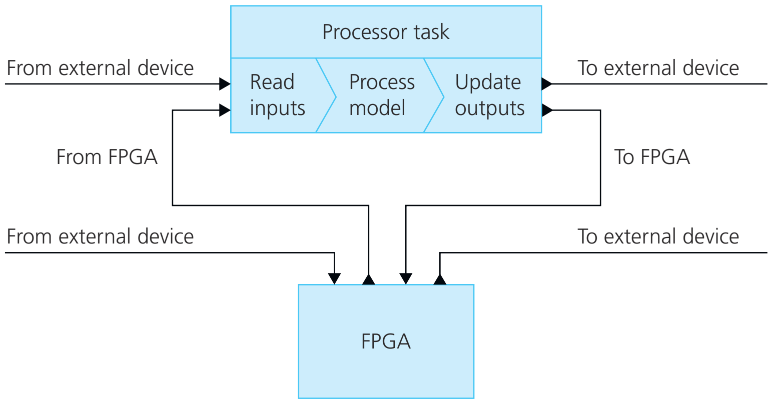
Buffer access Buffer access lets you access a vector value in the data buffer. One specific value of the data is identified by the specified channel number and the position within the buffer.

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.



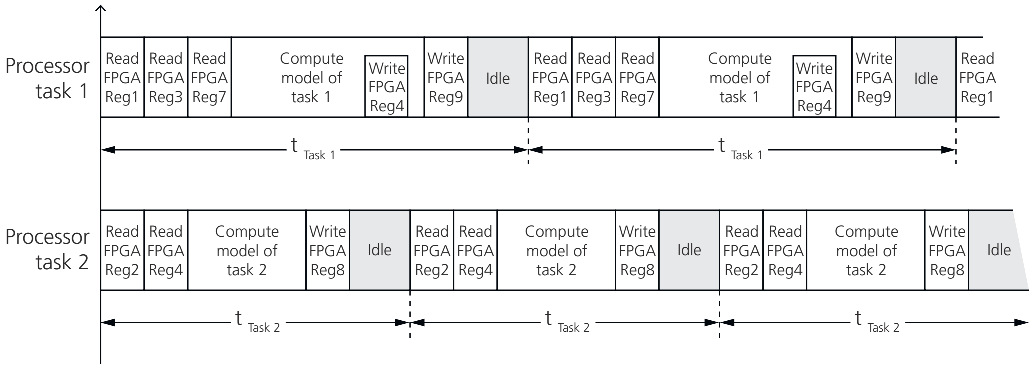
Processing the data exchange

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

Exchanging Data With the Processor Model

Introduction

Modeling the data exchange between the FPGA application and the processor application depends on the hardware used.

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:
	64 bit		

Data Storage Area	Data Width	Data Storage Size	Data Format
Buffer	32 bit 64 bit	32 32-bit buffers and 32 64-bit buffers Each buffer with up to 32,768 elements	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

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 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. 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.
	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.

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.

Read/write access in the FPGA application

The selected FPGA framework provides specific read and write functions that exchange data via the board-specific bus to the processor model. For instructions on implementing these functions, refer to Specifying the FPGA I/O Interface on page 35. For information on the available read and write functions, refer to RTI FPGA Programming Blockset - FPGA Handcode Interface Reference 📖.

The values in the FPGA application are processed as fixed-point values. The data exchange with the processor model requires data type conversion between fixed-point and floating-point values. Data conversion is automatic in these I/O functions. The fixed-point format to be used can be set as an I/O function parameter.

Read/write access in the processor model

If you use a MicroAutoBox II, a MicroLabBox, or a PHS-bus-based system you can implement the processor interface with interface blocks of the RTI FPGA

Programming Blockset or with data exchange functions of board-specific Real-Time Libraries (RTLib). If you use a MicroAutoBox III or SCALEXIO system, you implement the processor interface in ConfigurationDesk. Refer to Adding FPGA Applications to the Signal Chain (ConfigurationDesk Real-Time Implementation Guide 📖).

Using the RTI FPGA Programming Blockset 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 I/O functions in the FPGA application. The simplest way to create and configure these blocks in the processor model is to use the `Generate` command from the Interface page in the PROC_SETUP_BL block dialog.

For detailed instructions, refer to How to Generate a Processor Model Interface on page 44.

Using the Real-Time Library The board-specific RTLib provides C functions for data exchange to handcode the processor interface in the processor application.

Refer to the following function references:

- PHS-bus-based system: Data Exchange Functions (DS5203 RTLib Reference 📖)
- MicroAutoBox II: Data Exchange Functions (MicroAutoBox II RTLib Reference 📖)
- MicroLabBox: Data Exchange Functions (MicroLabBox RTLib Reference 📖)

Related topics

Basics

Software Tools for Working With the Handcode Interface of the RTI FPGA Programming Blockset

Introduction

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

Tools from MathWorks®

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

For compatibility information, refer to Required MATLAB Releases (Installing dSPACE Software 📖).

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 Handcode Interface of the RTI FPGA Programming Blockset requires the following products of the Vivado Design Suite:

Xilinx Vivado Xilinx Vivado 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 Vivado compiler, 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 FPGA application is handcoded, 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

Basics

Typical Workflow... 28

Supported Hardware

Introduction

FPGA frameworks are available for the following dSPACE boards.

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 and real-time libraries (RTLib) for MicroLabBox. To use I/O functions from the RTI FPGA Programming Blockset and RTI blocksets/RtLib for MicroLabBox in the same application, you have to use the flexible I/O framework for MicroLabBox. Refer to Features of the MicroLabBox Flexible I/O Framework on page 88.

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

Feature	Description
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 📖).

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.
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 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.	▪ ADC (Type A), Analog In 11[1)]: 8 parallel A/D converters with 16-bit resolution and a conversion 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.	ADC (Type A), Analog In 14[1)]: 14 parallel A/D converters with 16-bit 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.

Feature	DS1552 Multi-I/O Module	DS1552B1 Multi-I/O Module	DS1554 Engine Control I/O Module
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.

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)

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.

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 ▪ 11,700 kbit total block RAM ▪ 125 MHz hardware clock frequency	XILINX® KINTEX®-7-410T ▪ 406,720 logic cells ▪ 1,540 DSP slices ▪ 5,663 kbit maximum distributed RAM ▪ 28,620 kbit total block RAM ▪ 125 MHz hardware clock frequency	XILINX® KINTEX® UltraScale™ KU035 ▪ 444,343 system logic cells ▪ 1,700 DSP slices ▪ 5,908 kbit maximum distributed RAM	XILINX® KINTEX® UltraScale+™ KU15P ▪ 1,143,450 system logic cells ▪ 1,968 DSP slices ▪ 9,800 kbit maximum distributed RAM ▪ 34,600 kbit total block RAM

Feature	DS2655 (7K160)	DS2655 (7K410)	DS6601	DS6602
			▪ 19,000 kbit total block RAM ▪ 125 MHz hardware clock frequency	▪ 36,000 kbit UltraRAM ▪ 125 MHz hardware clock frequency
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 25.			
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 📖).

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.

Feature	DS2655M1	DS2655M2	DS6651
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 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.	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.

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 📖)

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.

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 ▪ 840 DSP slices ▪ 4000 kbit distributed RAM ▪ 16020 kbit block RAM ▪ 100 MHz hardware clock frequency	Xilinx Kintex® XC7K410T ▪ approx. 407 k logic cells: ▪ 63550 Kintex-7 slices ▪ 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 28.	
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.	

Feature	DS5203 (7K325)	DS5203 (7K410)
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.
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 🕮).

Typical Workflow

Introduction

There is a typical workflow you must follow, if you want to integrate a code-based FPGA application in a dSPACE hardware and software environment.

Workflow steps

The RTI FPGA Programming Blockset handcode workflow consists of the following steps, which are explained in detail later on.

1. Copy a dSPACE FPGA framework from the dSPACE installation folder to a custom folder.

   For further information, refer to Preparing Your Environment on page 32.

2. Handcode the functionality of the FPGA application by changing the contents of a customizable VHDL or Verilog file included in the dSPACE FPGA framework.

   Note

   For MicroAutoBox III/SCALEXIO systems, only VHDL files are included.

   For further information, refer to Specifying the FPGA Functionality on page 32.
3. Specify the I/O interface of the FPGA application in a MATLAB M file included in the dSPACE FPGA framework (Handcode FPGA framework INI file).

   For further information, refer to Specifying the FPGA I/O Interface on page 35.
4. Write the I/O interface configuration to the VHDL or Verilog code automatically via a MATLAB function provided by the RTI FPGA Programming blockset.

   For further information, refer to Configuring the FPGA Code With the Specified I/O Interface on page 38.
5. Use Xilinx Vivado to build the FPGA application. A preconfigured Xilinx Vivado project is part of the dSPACE FPGA framework.

   For further information, refer to How to Build the FPGA Application on page 39.
6. Build a dSPACE FPGA model INI file from the Xilinx Vivado results via a MATLAB function provided by the RTI FPGA Programming blockset.

   For further information, refer to Generating an FPGA Model INI File on page 41.
7. The following steps depend on the hardware used.

   For a MicroAutoBox II, a MicroLabBox, or a PHS-bus-based system:

   - Add the processor interface to your processor model. Perform one of the following methods:
     - Use the Processor Interface blocks from the RTI FPGA Programming Blockset combined with the FPGA model INI file to program and access the FPGA board.

       For further information, refer to How to Build a Processor Application on page 49.
     - Add data exchange functions of the Real-Time Library to your processor application. Refer to Handcoding the Processor Interface on page 54.

   For a MicroAutoBox III or SCALEXIO system:

   - Add the FPGA model INI file as an FPGA custom function to ConfigurationDesk. Refer to Adding FPGA Applications to the Signal Chain (ConfigurationDesk Real-Time Implementation Guide 🕮).

     You implement the communication between the FPGA application and the processor application via model ports in ConfigurationDesk.

Related topics

Basics

Modeling Aspects

Introduction

Describes general modeling aspects that have no particular context.

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.

- If you use a MicroAutoBox II, a MicroLabBox, or a PHS-bus-based system, you implement the data exchange between the processor application and the FPGA application by generating the processor interface which uses the blocks from the RTI FPGA Programming Blockset's Processor Interface library.
- If you use a MicroAutoBox III or a SCALEXIO system, you implement the data exchange between the processor application and the FPGA application by using related model port blocks in the behavior model.

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. Make sure that the data paths are synchronized.

Tracing FPGA signals

The FPGA signals can be accessed only if you have connected them to the processor application.

Related topics

Basics

Detailed Instructions on the Handcode Workflow

Introduction

Provides detailed descriptions of each of the workflow steps.

Where to go from here

Information in this section

Preparing Your Environment

Introduction

The FPGA framework defines the available I/O functions and all their parameters required to configure and use a specific FPGA board.

Specifying the FPGA framework

Before you start integrating your FPGA application into the dSPACE hardware, you must specify the FPGA framework that corresponds to your FPGA board. This provides information on all the available I/O functions and their parameters, and further data required to use a specific FPGA board.

You will find the available FPGA frameworks in `<RCP_HIL_InstallationPath>\MATLAB\RTIFPGA\Frameworks`. You must choose one of the subfolders and copy it to a user-defined location where you have modify rights. The path to the destination folder will now be referenced as `<FPGA_Framework>`.

Next step

Now, you must specify the FPGA functionality, refer to Specifying the FPGA Functionality on page 32.

Related topics

Basics

Specifying the FPGA Functionality

Introduction

To implement FPGA functionality, you must customize a specific file from the Xilinx Vivado project.

Customizing the architecture section

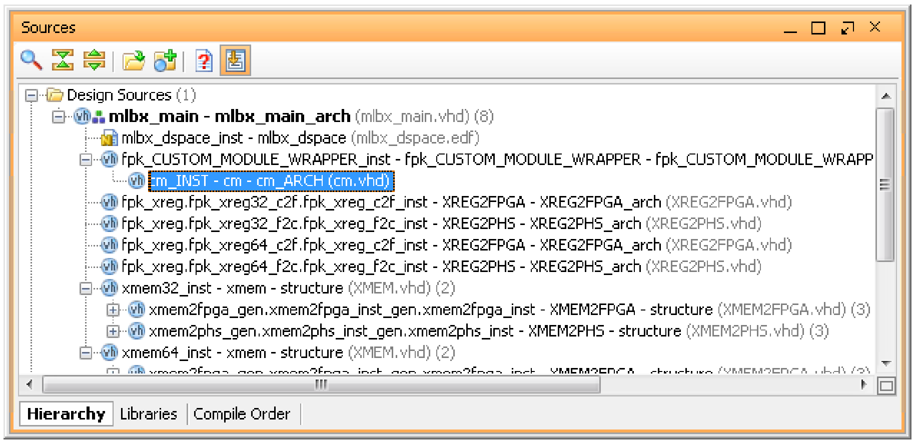
There are different Xilinx Vivado projects available for coding with VHDL and Verilog.

Note

For a SCALEXIO or MicroAutoBox III system, only VHDL-coded projects are available.

Using VHDL code Open the following Xilinx Vivado project with the Xilinx Project Manager: `<FPGA_Framework>\HC_Vivado\HC_Vivado2015.xpr`

Open the `cm.vhd` file from the project's source files.



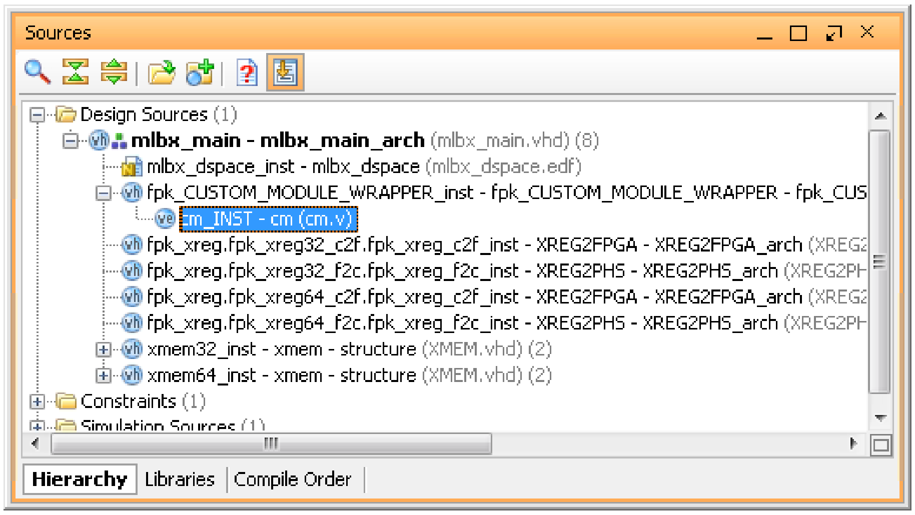
```
Design Sources (1)
  mlbx_main - mlbx_main_arch (mlbx_main.vhd) (8)
    mlbx_dspace_inst - mlbx_dspace (mlbx_dspace.edf)
    fpk_CUSTOM_MODULE_WRAPPER_inst - fpk_CUSTOM_MODULE_WRAPPER - fpk_CUSTOM_MODULE_WRAPP
      cm_INST - cm - cm_ARCH (cm.vhd)
    fpk_xreg.fpk_xreg32_c2f.fpk_xreg_c2f_inst - XREG2FPGA - XREG2FPGA_arch (XREG2FPGA.vhd)
    fpk_xreg.fpk_xreg32_f2c.fpk_xreg_f2c_inst - XREG2PHS - XREG2PHS_arch (XREG2PHS.vhd)
    fpk_xreg.fpk_xreg64_c2f.fpk_xreg_c2f_inst - XREG2FPGA - XREG2FPGA_arch (XREG2FPGA.vhd)
    fpk_xreg.fpk_xreg64_f2c.fpk_xreg_f2c_inst - XREG2PHS - XREG2PHS_arch (XREG2PHS.vhd)
    xmem32_inst - xmem - structure (XMEM.vhd) (2)
      xmem2fpga_gen.xmem2fpga_inst_gen.xmem2fpga_inst - XMEM2FPGA - structure (XMEM2FPGA.vhd) (3)
      xmem2phs_gen.xmem2phs_inst_gen.xmem2phs_inst - XMEM2PHS - structure (XMEM2PHS.vhd) (3)
    xmem64_inst - xmem - structure (XMEM.vhd) (2)
```

At the end of the file you will find the architecture section that has to be customized to implement the FPGA functionality.

```
ARCHITECTURE beh OF CUSTOM_MODULE_CW IS

-----------------------------------------------------------------
-- Specify custom code from here on...                       ---
-----------------------------------------------------------------

  -------------------
  -- FPGA Inverter --
  -------------------

SIGNAL data_invert : STD_LOGIC_VECTOR(31 DOWNTO 0);

BEGIN

  -- Input data
  data_invert <= (not xreg_000_dout) + "01";

  -- Inverted output data
  xreg_000_din <= data_invert;

  -- Inverted output data analogIO loop
  dac0_data <= data_invert(31) & data_invert(13 downto 0);
  xreg_001_din(15 downto 0) <= adc0_data(15 downto 0);
  xreg_001_din(31 downto 16) <= X"0000" when adc0_data(15) = '0' else
                                X"FFFF";

  -- LED state
  led_out <= data_invert(31);
  usr_0_interrupt <= data_invert(31);

  -- LED state DigIO loop
  digio_00_out <= data_invert(31);
  digio_00_oe <= '1';
  usr_1_interrupt <= digio_02_in;

END beh;
```

Using Verilog code Open the following Xilinx Vivado project with the Xilinx Project Manager:
`<FPGA_Framework>\HC_Vivado\HC_Vivado2015_verilog.xpr`

Open the `cm.v` file from the project's source files.



```
Design Sources (1)
  mlbx_main - mlbx_main_arch (mlbx_main.vhd) (8)
    mlbx_dspace_inst - mlbx_dspace (mlbx_dspace.edf)
    fpk_CUSTOM_MODULE_WRAPPER_inst - fpk_CUSTOM_MODULE_WRAPPER - fpk_CUS
      cm_INST - cm (cm.v)
    fpk_xreg.fpk_xreg32_c2f.fpk_xreg_c2f_inst - XREG2FPGA - XREG2FPGA_arch (XREG2
    fpk_xreg.fpk_xreg32_f2c.fpk_xreg_f2c_inst - XREG2PHS - XREG2PHS_arch (XREG2PH
    fpk_xreg.fpk_xreg64_c2f.fpk_xreg_c2f_inst - XREG2FPGA - XREG2FPGA_arch (XREG2
    fpk_xreg.fpk_xreg64_f2c.fpk_xreg_f2c_inst - XREG2PHS - XREG2PHS_arch (XREG2PH
    xmem32_inst - xmem - structure (XMEM.vhd) (2)
    xmem64_inst - xmem - structure (XMEM.vhd) (2)
Constraints (1)
Simulation Sources (1)
```

At the end of the file you will find the architecture section that has to be customized to implement the FPGA functionality.

```
  //----------------------------------------------------------------
  //  Specify custom code from here on...                        ---
  //----------------------------------------------------------------


  //------------------
  //-- FPGA Inverter --
  //------------------
  wire [31:0] data_invert;

  // Input data
  assign data_invert = (~xreg_000_dout) + 32'h0001;

  // Inverted output data
  assign xreg_000_din = data_invert;

  // Inverted output data analogIO loop
  assign dac0_data = {data_invert[31], data_invert[13:0]};
  assign xreg_001_din[15:0] = adc0_data[15:0];
  assign xreg_001_din[31:16] = (adc0_data[15] == 1'b0) ? 16'h0 : 16'hFFFF;

  // LED state
  assign led_out = data_invert[31];
  assign usr_0_interrupt = data_invert[31];

  // LED state DigIO loop
  assign digio_00_out = data_invert[31];
  assign digio_00_oe = 1'b1;
  assign usr_1_interrupt = digio_02_in;

endmodule // End of CUSTOM_MODULE_CW
```

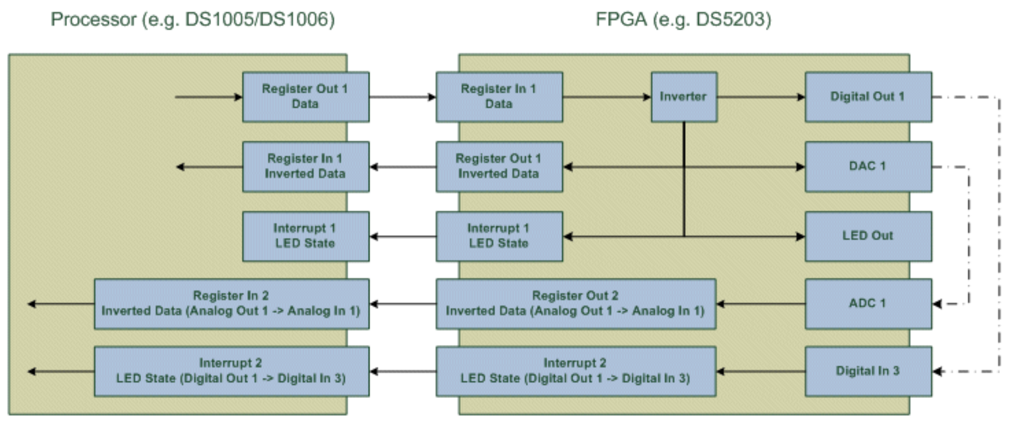
To connect your FPGA application to the I/O of the FPGA board and to the processor application, you have to use the I/O functions defined in the specified FPGA framework. These I/O functions provide several channels for **Register In**, **Buffer Out**, **Digital Out**, **Interrupt**, etc. The I/O functions and their parameters, and the number of channels, depend on the specified FPGA framework.

The signals of the `cm` entity form the interface to all the available I/O functions. A detailed description of all the available I/O functions including their parameters and the interface can be found in RTI FPGA Programming Blockset - FPGA Handcode Interface Reference 🕮.

You can use this information to customize the architecture section of the `cm` file to implement your FPGA functionality.

Example

The initial architecture section is predefined to implement an FPGA application. Below is the example of the onboard I/O FPGA framework of the DS5203.


Processor (e.g. DS1005/DS1006)
FPGA (e.g. DS5203)
Register Out 1 Data
Register In 1 Inverted Data
Interrupt 1 LED State
Register In 2 Inverted Data (Analog Out 1 -> Analog In 1)
Interrupt 2 LED State (Digital Out 1 -> Digital In 3)
Register In 1 Data
Register Out 1 Inverted Data
Interrupt 1 LED State
Register Out 2 Inverted Data (Analog Out 1 -> Analog In 1)
Interrupt 2 LED State (Digital Out 1 -> Digital In 3)
Inverter
Digital Out 1
DAC 1
LED Out
ADC 1
Digital In 3

Data can be written from the processor to the FPGA via **Register** channel 1. The data is interpreted as mV signals and inverted on the FPGA. The inverted data is directly written back to the processor via **Register** channel 1. It is also written to the **DAC** channel 1. **ADC** channel 1 reads the output of **DAC** channel 1 if the external connection is closed as illustrated. **Register** channel 2 writes the **ADC** channel 1 values to the processor. The sign bit of the inverted signal directly toggles the **LED** output of the DS5203 and requests an interrupt on line 1. The sign bit is also written to **Digital Out** channel 1. **Digital In** channel 3 reads the **Digital Out** channel 1 data and requests an interrupt on line 2 if the external connection is closed as illustrated.

Next step

Now, you must specify the FPGA I/O interface, refer to Specifying the FPGA I/O Interface on page 35.

Related topics

Basics

Specifying the FPGA I/O Interface

Introduction

The specified FPGA framework contains a handcode FPGA framework INI file that you must customize according to the I/O functions used.

Handcode FPGA framework INI file

The handcode FPGA framework INI file is a MATLAB M file. It is included in the selected FPGA framework folder and is called `hc_fpga_framework_ini_<Framework_Name>.m`, for example, `hc_fpga_framework_ini_DS5203_XC7K325T.m`. The file contains configuration parameters for the first and the last channel of all the available I/O functions of the selected FPGA framework. Each I/O function used in your FPGA application must be specified according to the example code in the file. I/O functions that are not used should be commented out to minimize the required FPGA resources, by entering `%`. The file is pre-initialized for the handcoding example application described in Specifying the FPGA Functionality on page 32. You have to configure the file within the section enclosed by `BEGIN USER CODE` and `END USER CODE`.

A detailed description of all the available I/O functions including their parameters and the interface can be found in RTI FPGA Programming Blockset - FPGA Handcode Interface Reference 🕮.

Additional instructions when using a SCALEXIO system

When you use a SCALEXIO system, you have to specify the slot positions of the mounted I/O modules (up to five modules). If you use inter-FPGA communication, you must also specify the I/O module slots that provide the inter-FPGA communication interface.

```
% ======       <FPGA Base Board> IO Modul Selection      =======
...
% IO Frameworks to choose from:
% - IO_fpga_framework_ini_DS2655M1
% - IO_fpga_framework_ini_DS2655M2
% - IO_fpga_framework_ini_DS2655_InterFPGA
% - IO_fpga_framework_ini_DS6651
  ioFrmIni = { ...
     'IO_fpga_framework_ini_DS2655M1', ...    % IO Module in slot 1
     '', ...                                  % IO Module in slot 2
     '', ...                                  % IO Module in slot 3
     '', ...                                  % IO Module in slot 4
     '', ...                                  % IO Module in slot 5
 };
...
```

For each I/O module and inter-FPGA interface used, you have to copy the entire I/O function section and modify it, for example, you have to adapt the I/O module number `ioModuleNr`.

```
%>>>>>>>> DS2655M1 IO Module >>>>>>>>
 % ====== IO-Module Number: =======
 ioModuleNr = 1;    % IO-Module Number (used Slot of IO Module)
%--------------------------------
% ------ Digital Out -------
%--------------------------------
 hcfw.IOProperties.Out.Fct(1 + ioOutOffset(ioModuleNr)).Parameter(7).Init
        = {'1'}; % High supply (0 for 5V, 1 for 3.3V)
...
```

Example

If you use the DS5203 (7K325) with onboard I/O framework, the corresponding handcode FPGA framework INI file is called `hc_fpga_framework_ini_DS5203_XC7K325T.m`.

The following example shows that 128 channels of the I/O function Register Out are available, but only the first and second channels are used. The entries for the second channel have been added by copying one of the existing function configurations and adapting them. The channel number has been changed and the parameter values have been adapted. The entries for the last channel have been commented out. For an I/O function that you want to use in your custom code, you must set its `HcUsed` parameter to '1'.

Note

If you only specify the `HcUsed` property of an I/O function in the board-specific `hc_fpga_framework_ini` file, all the parameters of this function are set to their initial values. To overwrite a value, you have to explicitly specify the related parameter.

If you set the `HcUsed` property to '0', you have to comment out the specified parameters.

```
%---------------------------------
% ------ Register Out 1 -------
%---------------------------------
hcfw.PHSProperties.Out.Fct(1).HcUsed = {'1'};                  % IO function is used in custom code
hcfw.PHSProperties.Out.Fct(1).HcCustomName = {'Inverted Data'}; % Custom name of the IO function
hcfw.PHSProperties.Out.Fct(1).Parameter(1).Init = {'0'};       % Binary Point position (0..32)
hcfw.PHSProperties.Out.Fct(1).Parameter(2).Init = {'signed'};  % Data Format (signed/unsigned/floating-point)
hcfw.PHSProperties.Out.Fct(1).Parameter(3).Init = {'0'};       % Register Group ID (0..63), 0 for ungrouped
%---------------------------------
% ------ Register Out 2 -------
%---------------------------------
hcfw.PHSProperties.Out.Fct(2).HcUsed = {'1'};                  % IO function is used in custom code
hcfw.PHSProperties.Out.Fct(2).HcCustomName =
{'Inverted Data (Analog Out 1 -> Analog In 1)'}; % Custom name of the IO function
hcfw.PHSProperties.Out.Fct(2).Parameter(1).Init = {'0'};       % Binary Point position (0..32)
hcfw.PHSProperties.Out.Fct(2).Parameter(2).Init = {'signed'};  % Data Format (signed/unsigned/floating-point)
hcfw.PHSProperties.Out.Fct(2).Parameter(3).Init = {'0'};       % Register Group ID (0..63), 0 for ungrouped
% ...
```

Next step

Now you must configure the FPGA code with the specified I/O interface, refer to Configuring the FPGA Code With the Specified I/O Interface on page 38.

Related topics

Basics

Configuring the FPGA Code With the Specified I/O Interface

Introduction

Before you can build the FPGA application, you must configure the FPGA code with the I/O interface.

Configuring the FPGA code using the script interface

After you have specified the I/O interface in the handcode FPGA framework INI file (refer to Specifying the FPGA I/O Interface on page 35) and saved it, you can use the script interface to initialize the framework. This means, that the FPGA code is automatically configured with the specified I/O interface.

The script interface can be used in the MATLAB Command Window or in an M file.

The syntax is:

```
rtifpga_scriptinterface('HCFPGAFrameworkInit',
‘<FPGA_Framework>’,
‘<HC_FPGA_Framework_INI>’)
```

- <FPGA_Framework>: Path to the specified FPGA framework
- <HC_FPGA_Framework_INI>: Name of the handcode FPGA framework INI file without file name extension

The script interface generates a unique ID for this application and modifies the `cm.v` file accordingly.

Example

If you have specified the DS5203 (7K325) with onboard I/O framework that you have copied to `D:\Work\FPGAApplications`, the command looks like this:

```
rtifpga_scriptinterface('HCFPGAFrameworkInit',
'D:\Work\FPGAApplications\DS5203',
'hc_fpga_framework_ini_DS5203_XC7K325T')
```

Additional instructions when using DS2655M2 or DS6651 I/O modules

If you use DS2655M2 Digital I/O Modules and/or DS6651 Multi-I/O Modules, you have to copy files from the RTL folders of the I/O module frameworks to your working folder once and customize them.

Perform the following steps:

1. Copy all files from one of the following RTL folders to the RTL folder of your project.

I/O Module Type	Folder Path
DS2655M2:	`<RCP_HIL_InstallationPath>\MATLAB\RTIFPGA\Frameworks\DS2655M2\RTL\`
DS6651:	`<RCP_HIL_InstallationPath>\MATLAB\RTIFPGA\Frameworks\DS6651\RTL\`

   If you use both I/O module types, perform the following steps before you copy the files for the other I/O module type.

2. Replace the part `XXXX_MODULENO_XXXX` in the names of the copied files with the module number.

For example, rename `IO_custom_drv_ctrl_XXXX_MODULENO_XXXX.vhd` to `IO_custom_drv_ctrl_1.vhd` if the I/O module is installed in the I/O module slot 1.

3. If you use several I/O modules of the same type, copy the renamed files and replace the module number part of the file name with the other module numbers.

   For example, copy `IO_custom_drv_ctrl_1.vhd` and rename the copied file to `IO_custom_drv_ctrl_2.vhd` if the other I/O module is installed in the I/O module slot 2.
4. Open the copied files and replace the string `XXXX_MODULENO_XXXX` with the relevant module number.
5. If you use both I/O module types, repeat steps 1 … 4 with the other I/O module type.
6. Open Xilinx Vivado and add all copied files to your handcode project via File - Add Sources.

Next step

Now, you can build the FPGA application, refer to How to Build the FPGA Application on page 39.

Related topics

Basics

Typical Workflow.. 28

How to Build the FPGA Application

Objective

The Xilinx Vivado project is used to start the build process of the FPGA application.

Precondition

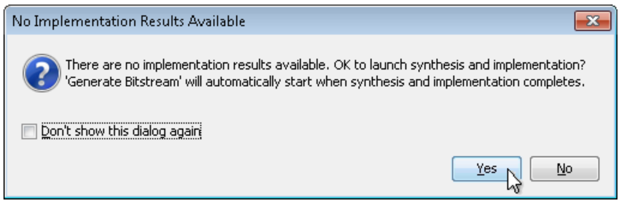
The built FPGA application can be integrated into the dSPACE environment only, if you have configured the I/O interface correctly. The required steps are described in Detailed Instructions on the Handcode Workflow on page 31.

Method

How to build the FPGA application

1 In the Flow Navigator, click Generate Bitstream.

2 If the No Implementation Results Available dialog appears, click Yes.



No Implementation Results Available

There are no implementation results available. OK to launch synthesis and implementation?
'Generate Bitstream' will automatically start when synthesis and implementation completes.

Result

If no error messages are displayed, the build process has been successfully finished and the FPGA bitstream has been created.

Note

If you use a DS6602 FPGA Base Board, the build process issues a critical warning about a specific timing requirement.
For more information, refer to Problems and Their Solutions on page 97.

Even if no errors occurred during handcoding, 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 code.

To solve the problem, you have to modify the FPGA code 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 7). For more information, contact dSPACE Support.

Next step

Now you must generate a corresponding FPGA model INI file, refer to Generating an FPGA Model INI File on page 41. For SCALEXIO systems, this includes generating the files required for an FPGA custom function to be loaded in ConfigurationDesk.

Related topics

Basics

Typical Workflow..28

Generating an FPGA Model INI File

Introduction

In this step, you will generate an FPGA model INI file that will contain the generated bitstream and relevant information on implementing the processor communication.

Generating the FPGA model INI file using the script interface

After you have built the bitstream, you can use the script interface to create a new FPGA model INI file that is application-specific because it includes a bitstream.

The script interface can be used in the MATLAB Command Window or in an M file.

The syntax is:

```
rtifpga_scriptinterface('HCFPGAModelIniGenerate',
‘<FPGA_Framework>’,
‘<HC_FPGA_Framework_INI>’)
```

- <FPGA_Framework>: Path to the specified FPGA framework
- <HC_FPGA_Framework_INI>: Name of the handcode FPGA framework INI file without file extension

Result

The generated FPGA model INI file can be found in the current MATLAB working directory. It is called `<HcApplIDString>_<ApplicationID>.ini` where `<HcApplIDString>` is the application name specified in the handcode FPGA framework INI file and `<ApplicationID>` is the unique ID generated by the `HCFPGAFrameworkInit` function of the script interface, refer to Configuring the FPGA Code With the Specified I/O Interface on page 38.

Example

If you have specified the DS5203 (7K325) with onboard I/O framework that you have copied to `D:\Work\FPGAApplications`, the command looks like this:

```
rtifpga_scriptinterface('HCFPGAModelIniGenerate',
'D:\Work\FPGAApplications\DS5203',
'hc_fpga_framework_ini_DS5203_XC7K325T')
```

Next step

If you use a MicroAutoBox II, a MicroLabBox, or a PHS-bus-based system, you must implement the processor interface based on the generated FPGA model INI file, refer to Using the FPGA Model INI File for the Processor Interface on page 42.

If you use a MicroAutoBox III or a SCALEXIO system, you can add the FPGA application to ConfigurationDesk as an FPGA custom function. Refer to How to Add FPGA Applications to ConfigurationDesk (ConfigurationDesk I/O Function Implementation Guide 🕮).

Related topics

Basics

Using the FPGA Model INI File for the Processor Interface

Introduction

Using the FPGA Model INI file to implement the processor interface of the processor application for the MicroAutoBox II, the MicroLabBox, or the PHS-bus-based system.

Generating RTI interface blocks

To implement the communication between the FPGA board and the processor board, you can implement a processor model containing RTI blocks corresponding to the implemented FPGA I/O interface. These blocks are provided by the Processor Interface of the RTI FPGA Programming Blockset.

The processor board also controls the download of the bitstream.

All this information is contained in the generated FPGA model INI file, which has to be imported to the processor model. For further information, refer to Modeling the Processor Communication on page 44.

Including the bitstream to the real-time application

The generated FPGA model INI file includes the bitstream to implement the functionality of the FPGA application. If you use the Processor Interface of the RTI FPGA Programming Blockset to build the processor application, the bitstream is automatically included to the real-time application. Refer to How to Build a Processor Application on page 49.

If you use RTLib to model the processor application, you must copy the bitstream out of the FPGA model INI file and include it to the real-time application. Refer to How to Extract Bitstream Files from FPGA Model INI Files on page 58.

Related topics

Basics

Using RTI to Implement the Processor Interface

Introduction

You can use the Processor Interface sublibrary of the RTI FPGA Programming Blockset to implement the processor model and the handcoded FPGA application to the MicroAutoBox II, the MicroLabBox, or the PHS-bus-based system.

Where to go from here

Information in this section

Modeling the Processor Communication

Introduction

The communication between the FPGA application and the processor application must be implemented in a corresponding processor model.

Where to go from here

Information in this section

How to Generate a Processor Model Interface

Objective

The RTI interface 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 application 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 model INI file. It therefore makes sense to generate the corresponding preconfigured interface blocks automatically instead of adding them to the processor model and configuring them manually.

Note

This step is totally different when you use a MicroAutoBox III/SCALEXIO system. Refer to the ConfigurationDesk user documentation to inform about generating the model interface.

Preconditions

- Your platform is a MicroAutoBox II, a MicroLabBox, or a PHS-bus-based system.

- The I/O functions for reading data, writing data, and interrupt handling in the FPGA application should be completely configured before the corresponding blocks are generated for the processor model.

Method

To generate a processor model interface

1. Add a PROC_SETUP_BL block to the processor model.
2. Double-click the block to open its dialog.
3. Switch to the Advanced page and add the generated FPGA model INI file to the file list.
4. Switch to the Unit page and 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.
5. Select the FPGA model INI file whose model interface you want to generate.
6. Switch to the Interface page and click `Generate` to generate the RTI blocks required for the processor model interface.

Result

You have created a new model containing RTI blocks from the Processor Interface library which are required to exchange data with the FPGA application. The blocks are configured with all the relevant settings of their counterparts in the FPGA application. Your FPGA application will contain PROC_XDATA_WRITE_BL blocks for input I/O functions, such as `Register In` or `Buffer In`, and PROC_XDATA_READ_BL blocks for output I/O functions, such as `Register Out` or `Buffer Out`.

I/O functions that are only relevant on the FPGA board, for example, `ADC` or `DAC` functions, are not given counterparts in the processor model interface.

You can now 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 I/O functions.

If you already implemented the processor model 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 processor interface blocks. You can also check compatibility with the current FPGA application 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 block for one FPGA application. Otherwise the FPGA board number will be replaced in any processor interface block used anywhere in the model.

Related topics

Basics

How to Trigger Interrupt-Driven Tasks

Objective

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

Basics

You need an interrupt I/O function in the FPGA application and its counterpart in the processor model to trigger a function-call subsystem by a value coming from the FPGA.

Preconditions

- Your platform is a MicroAutoBox II, a MicroLabBox, or a PHS-bus-based system.
- You have implemented an `Interrupt` I/O function from the handcode FPGA framework in your FPGA application. You must specify a channel and a descriptive name for the interrupt channel used as parameters.

Method

To trigger an interrupt-driven task

1. Add a PROC_INT_BL block to the processor model. You can do this by instantiating this block from the Processor Interface library in the processor model or by generating the processor model interface via the PROC_SETUP_BL block and moving it from the temporary interface model to the processor model. For detailed instructions, refer to How to Generate a Processor Model Interface on page 44.

   The PROC_INT_BL block must have the same channel number as the related `Interrupt` I/O function in the FPGA application.

Result

When the `Interrupt` I/O function in the FPGA application requests an interrupt, the function-call subsystem is triggered that is connected to the PROC_INT_BL block in the processor model.

Related topics

HowTos

Modeling Aspects for RTI-MP Systems

Introduction

There are some points to note, when using the Processor Interface library for an MP 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 divided 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 🕮.

Implementing a master MP model

The master MP model is implemented as described above. It contains the blocks from the Processor Interface library that refers to the specified FPGA application.

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. The slave MP model must contain the blocks for the processor model interface including a PROC_SETUP_BL block on its top level. The Processor Interface blocks in the slave MP model must correspond to its included FPGA application.

Related topics

Basics

Details on MP Systems (RTI and RTI-MP Implementation Guide 🕮)

Working with the Processor Model

Introduction

When you have implemented the processor model, you can simulate its behavior and build the processor application.

Where to go from here

Information in this section

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 in different model modes

You can simulate the processor model in these model modes:

- FPGA-Build / Offline simulation
- Processor-Build

In both model modes, the processor model only contains the blocks from the processor 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 applications.

Note

When you use a MicroAutoBox III or SCALEXIO system, the simulation behavior of the processor model is different to processor models for a MicroAutoBox II, a MicroLabBox, or a PHS-bus-based system. For further information, refer to Model Port Block Behavior During Simulink Simulation and Real-Time Execution (Model Interface Package for Simulink - Modeling Guide 📖).

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 - Start from the menu bar or using the `Ctrl+T` shortcut.

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

How to Build a Processor Application

Objective

Before you can execute 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 the Simulink Coder. When you specify the system target file that corresponds to your hardware, for example, `rti1007.tlc` for a DS1007 modular system, most of the relevant configuration parameters are set automatically, and many others can be customized. For general information, refer to Building and Downloading the Model (RTI and RTI-MP Implementation Guide 📖).

When you use a SCALEXIO/MicroAutoBox III system, the build process of the configured processor application is managed by ConfigurationDesk. Model separation is not relevant.

Preconditions

- 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 after the build process, the real-time hardware must be registered, for example via the Platform Manager in ControlDesk.

Method

To build a processor application

1. Switch the model mode to Processor-Build in the PROC_SETUP_BL block.
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.
3. Choose Tools - Code Generation - Options from the menu bar of the model to open the Code Generation dialog.
4. Specify the required build options in the Configuration Parameters dialog. Check the configuration, for example, the selected system target file and the specified fixed step size.

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

Tip

If the build options were already specified, you can also start the build process directly via the `Ctrl+B` shortcut or via the **Tools - Code Generation - Build Model** command from the menu bar of the model.

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 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 further information on programming the FPGA, refer to How to Create Burn Applications on page 50.

For example, if you built the *MyDemo* application for a DS1007 modular system, 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 Create Burn Applications..50

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 63.

Related topics

Basics

Experimenting with an FPGA Application.. 63

References

CreateBurnApplication (RTI FPGA Programming Blockset Script Interface Reference 🕮)

Using RTLib to Implement the Processor Interface

Introduction

You can use board-specific C functions to implement the processor application and the handcoded FPGA application on the MicroAutoBox II, the MicroLabBox, or the PHS-bus-based system.

Where to go from here

Information in this section

Information in other sections

MicroAutoBox II:

Compiling, Linking and Downloading an Application (MicroAutoBox II RTLib Reference 🕮)
Information about the batch files, makefiles, and linker command files that support your program development.

Board Initialization (MicroAutoBox II RTLib Reference 📖)
Before you can use the FPGA Type 1 module, you have to perform the initialization process.

MicroLabBox:

Compiling, Linking and Downloading an Application (MicroLabBox RTLib Reference 📖)
Information about the batch files, makefiles, and linker command files that support your program development.

FPGA Initialization (MicroLabBox RTLib Reference 📖)
Before you can use the I/O FPGA application, you have to perform the initialization process.

DS1006 modular system:

Compiling, Linking and Downloading an Application (DS1006 RTLib Reference 📖)
Information about the batch files, makefiles, and linker command files that support your program development.

Board Initialization (DS5203 RTLib Reference 📖)
Before you can use the DS5203 FPGA Board, you have to perform the initialization process.

DS1007 modular system:

Compiling, Linking and Downloading an Application (DS1007 RTLib Reference 📖)
Information about the batch files, makefiles, and linker command files that support your program development.

Board Initialization (DS5203 RTLib Reference 📖)
Before you can use the DS5203 FPGA Board, you have to perform the initialization process.

Handcoding the Processor Interface

Objective

RTLib functions let you implement the processor interface and handle the FPGA interrupts.

Data exchange

With RTLib, you can implement the processor's read and write access to the FPGA data storage via data exchange functions.

Accessing FPGA data storage Depending on the platform, FPGA data storage, data direction, and bit width, you have to use different data exchange functions to exchange data. The data exchange functions have the following naming convention for the different use cases:

```
<FPGA_board>_<data_direction>_<data_storage>
```

Placeholder	Possible Entry	Meaning
FPGA_board	DS5203	DS5203 FPGA Board
	fpga_tp1	MicroAutoBox II
	IoFpga	MicroLabBox
data_direction	read	Reading data from the FPGA application
	write	Writing data to the FPGA application
data_storage	reg	Accessing a 32-bit register
	reg_grp	Accessing a 32-bit register group
	reg_grp_mixed	Accessing one of the following register groups: ▪ 64-bit register group ▪ Register group with 32-bit and 64-bit data values
	buf	Accessing a 32-bit buffer
	buf64	Accessing a 64-bit buffer

Data type conversion Data exchange with the FPGA application requires data type conversion, which is configured by the `scaling` and `mode` parameters.

The `scaling` parameter correlates with the binary point position parameter of the FPGA application for signed and unsigned data types. The `scaling` parameter is ignored if the FPGA application uses floating-point data. Predefined symbols let you configure the `scaling` parameter. The following table shows examples of predefined symbols for reading a register of an FPGA application implemented on the DS5203 FPGA Board.

Processor Application		FPGA Application
Symbol of the Scaling Parameter	**Meaning**	**Binary point position**
`DS5203_READ_SCALE_BIN_PT_0`	Scaling factor: $1/2^0$=1.00	0
`DS5203_READ_SCALE_BIN_PT_3`	Scaling factor: $1/2^3$=0.125	3

The mode parameter lets you configure the data type used in the FPGA application with predefined symbols.

Data exchange example This example shows how to write data in signed mode with a scaling factor of 2^{16} (Binary point position = 16) to register 1 of a DS5203 FPGA Board:

```
dsfloat scaling_factor = DS5203_WRITE_SCALE_BIN_PT_16;
ds5203_write_reg(
  DS5203_1_BASE,   //PHS-bus base address
  1,               //number of the register
  &data,           //pointer to data variable
  &scaling_factor,
  DS5203_DATA_MODE_SIGNED); // data processing mode (data type)
```

Interrupt handling

The FPGA boards provide interrupt channels that you can handle by using the RTLib interrupt functions. The handling of the interrupts depends on the platform you use.

DS5203 FPGA Board Before interrupts can be read from the FPGA application, you must enable the PHS-bus interrupt handler and the DS5203 interrupt channel. To do this, call `install_phs_int_vector` and `DS5203_enable_int`. You can query for pending interrupts with `ds5203_pending_int`. An interrupt is pending if it was requested but not yet acknowledged by the program.

You can read pending interrupts with `ds5203_read_int`. After you evaluate the read interrupt, you must acknowledge the interrupt. To do this, call `ds5203_ack_int`.

The following example shows you how to use interrupts with the DS5203 FPGA Board:

```
/*
  Compile and download with down1006/1007:
  down1007 ds5203_test.c ds5203_123456789A0002.c
*/
#include <ds5203.h>
#include <phsint.h>
extern void *fpga_prgrm_data_ptr_123456789A0002;
static UInt32 isr_counter;
void isr_0(void)
{
  if (ds5203_pending_int(DS5203_1_BASE, DS5203_INT_SRC_1) > 0)
  {
    isr_counter++;
    ds5203_ack_int(DS5203_1_BASE, DS5203_INT_SRC_1);
  }
}
int main(void)
{
  ULong64 applId = 0x123456789A0002LL;
  /* program and init FPGA with bitstream which generates interrupts */
  ds5203_prgr_dat *prgr_dat_123456789A0002 = (ds5203_prgr_dat*)
fpga_prgrm_data_ptr_123456789A0002;
  ds5203_program(DS5203_1_BASE, prgr_dat_123456789A0002);
  ds5203_init(DS5203_1_BASE, &applId);
  /* initialize and enable RTLib and DS5203 PHS-bus interrupts */
  install_phs_int_vector(DS5203_1_BASE, 0, isr_0);
  RTLIB_INT_ENABLE();
  ds5203_enable_int(DS5203_1_BASE, DS5203_INT_SRC_1);
  while(1)
  {
    RTLIB_BACKGROUND_SERVICE();
  }
  return 0;
}
```

MicroAutoBox II You must enable the interrupt channel for the FPGA. To do this, call `fpga_tp1_enable_int`. The interrupt handling is performed by the interrupt controller on the MicroAutoBox II base board.

MicroLabBox Before interrupts can be read from the FPGA application, you must enable the interrupt channel for the FPGA. To do this, call `IoFpga_enable_int`. You must register the interrupt channels of the FPGA application to connect them to the interrupt service routine (ISR) of MicroLabBox. To do this, call `IoFpga_register_isr`. Pending interrupts are automatically acknowledged to reset the associated flag in the interrupt flag register.

The following example shows you how to use interrupts with MicroLabBox:

```
/*
  Compile and download with down1202:
  down1202 ds1202_test.c ds1202_123456789A0002.c
*/
#include <brtenv.h>
static UInt32 isr_counter;
void isr_0(void)
{
  isr_counter++;
}

int main(void)
{
  ULong64 applId = 0x123456789A0002ULL;
  /* init RTLib and IoFpga  */
  RTLIB_INIT();
  IoFpga_init(&applId);
  /* register interrupt handler */
  IoFpga_register_isr(isr_0, IOFPGA_INT_SRC_1);
  /* RTLib global interrupt enable */
  RTLIB_INT_ENABLE();
  IoFpga_enable_int(IOFPGA_INT_SRC_1);
  while(1)
  {
    RTLIB_BACKGROUND_SERVICE();
  }
  return 0;
}
```

Reference information

MicroAutoBox II:
- Data Exchange Functions (MicroAutoBox II RTLib Reference 🕮)
- Interrupt Functions (MicroAutoBox II RTLib Reference 🕮)

MicroLabBox:
- Data Exchange Functions (MicroLabBox RTLib Reference 🕮)
- Interrupt Functions (MicroLabBox RTLib Reference 🕮)

PHS-bus-based system:
- Data Exchange Functions (DS5203 RTLib Reference 🕮)
- Interrupt Functions (DS5203 RTLib Reference 🕮)

How to Extract Bitstream Files from FPGA Model INI Files

Objective

Extracting bitstream file from the FPGA model INI file to include the bitstream file to the processor application.

Preconditions

- The FPGA model INI file is generated.
- A tool to extract ZIP files is installed.

Method

To extract a bitstream file from an FPGA model INI file

1. Change the file-type extension of the FPGA model INI file from `.ini` to `.zip`.
   For example: `FPGA_73536B10F7204F.ini` to `FPGA_73536B10F7204F.zip`.
2. Extract the ZIP file. The C file with the name of the FPGA model INI file is the bitstream file of your FPGA application.
3. Copy the bitstream file to the project folder of your processor application.
   For example: Copy `FPGA_73536B10F7204F.c` to a local folder.

Result

You extracted the bitstream file.

Next step

You can now include the bitstream file to the processor application. Refer to How to Include Bitstream Files to Handcoded Processor Applications on page 59.

Related topics

Basics

How to Include Bitstream Files to Handcoded Processor Applications

Objective

You must include the bitstream file of the FPGA model INI file in the processor application so the processor application is able to program the FPGA.

Precondition

The bitstream file is extracted from the FPGA model INI file.

Possible methods

- You can edit a custom makefile to include the bitstream file. Refer to Method 1.
- You can copy the C code of the bitstream file to the processor application code. Refer to Method 2.

Method 1

To include a bitstream file by editing a custom makefile

1 Copy the `DsBuildTemplate.mk` makefile of your processor board's RTLib to your application folder. For example, copy `<RCP_HIL_InstallationPath>\DS1007\RTLib\DsBuildTemplate.mk` of the DS1007 PPC Processor Board.

2 Rename the makefile <application_name>.mk.

3 Edit the `CUSTOM_SRC_FILES` section of the makefile by adding the name of the bitstream file. For example:

```
CUSTOM_SRC_FILES = FPGA_73536B10F7204F.c
```

Method 2

To include a bitstream file by copying C code

1 Copy the C code of the bitstream file to the handcoded processor application.

Result

You included the bitstream file to the handcoded processor application.

Related topics

Basics

Typical Workflow.. 28

How to Download FPGA Applications and Program the FPGA

Objective

The processor application handles the download of the FPGA application and the programming of the FPGA.

Basics

If you handcode the processor application for a MicroAutoBox II or a PHS-bus-based system, you have to call an RTLib function to download the bitstream and program the FPGA. You must call the FPGA programming function after the processor board was initialized and before the FPGA is initialized.

Note

Due to MicroLabBox's loading mechanism, no function is required for downloading and programming the FPGA module. If the bitstream file is included, the FPGA is programmed during the initialization of MicroLabBox. To initialize MicroLabBox, call the functions `RTLIB_INIT()` and `IoFpga_init(&appID)`.

Possible methods

- If you use MicroAutoBox II, refer to Method 1.
- If you use a PHS-bus-based system, refer to Method 2.

Method 1

To download FPGA application and program the FPGA of the MicroAutoBox II

1. Initialize the processor board by calling `init()`.
2. Download the FPGA application and program the FPGA by calling `fpga_tp1_program`.
3. Initialize the FPGA board by calling `fpga_tp1_init`.
4. If you want to enable or disable that the FPGA application stored in the flash memory starts automatically after power-up, call `fpga_tp1_enable_autoboot` or `fpga_tp1_disable_autoboot`, respectively. The functions overwrite the settings of the MicroAutoBox II Configuration Tool.

   FPGA applications stored in the RAM/SRAM cannot start automatically after power-up.

Method 2

To download FPGA application and program the DS5203 FPGA Board

1. Initialize the processor board by calling `init()`.
2. Download the FPGA application and program the FPGA by calling `DS5203_program`.
3. Initialize the FPGA board by calling `ds5203_init`.

Result

After you build the real-time application, the real-time application can download the FPGA application and program the FPGA.

Example

This example shows you how to use the FPGA programming function for a MicroAutoBox II in general:

```
extern void *fpga_prgrm_data_ptr_<AppID>;
fpga_tp1_prgr_dat *datastruct = (fpga_tp1_prgr_dat*)fpga_prgrm_data_ptr_<AppID>;
fpga_tp1_program(base, datastruct);
```

Assuming that the created model INI file contains `FPGA_TP1_MAIN_84303E38761313.c`, the program function and the data structure look like the following example.

Extract from the above-mentioned C file:

```
#include <dstypes.h>
UInt8 bitstream_84303E38761313[];
UInt32 compat_boards_maj_84303E38761313[];
UInt32 compat_boards_min_84303E38761313[];
struct
{
   UInt32 model_blkst_major;
   UInt32 model_blkst_minor;
   UInt32 model_blkst_maintenence;
   UInt8 *bitstream_pt;
   UInt32 length;
   UInt32 length_unpckd;
   UInt32 *compatible_boards_list_maj_pt;
   UInt32 *compatible_boards_list_min_pt;
   UInt32 compatible_boards_list_length;
   UInt32 fw_id;
   UInt32 mode;
} prgr_dat_84303E38761313 =
{
   1,  /* blockset major version */
   1,  /* blockset minor version */
   0,  /* blockset maintenance version */
   (UInt8*) &bitstream_84303E38761313, /* pointer to bitstream */
   0x3199e,   /* length compressed (= 203,166 bytes) */
   0x441f80,  /* length decompressed (= 4,464,512 bytes) */
   (UInt32*) &compat_boards_maj_84303E38761313, /* pointer to comp boards major */
   (UInt32*) &compat_boards_min_84303E38761313, /* pointer to comp boards minor */
   1,  /* number of compatible boards */
   1,  /* framework identifier */
   (0x02 | 0x04 | 0x08)
   /* programming mode: FPGA_TP1_PRGRM_SRM | FPGA_TP1_PRGRM_COM |
FPGA_TP1_PRGRM_DBGOUT */
};
void *fpga_prgrm_data_ptr_84303E38761313 = (void*) &prgr_dat_84303E38761313;
UInt32 compat_boards_maj_84303E38761313[] =
{
   2
};
UInt32 compat_boards_min_84303E38761313[] =
{
   0
};
```

```
UInt8 bitstream_84303E38761313[] =
{
   249,255,255,255,255,249,4,4,
   249,8,8,249,16,16,0,0,
   0,187,17,34,0,68,249,8,
   ...
}
...
```

Extract from the main program:

```
extern void *fpga_prgrm_data_ptr_84303E38761313;
fpga_tp1_prgr_dat *datastruct = (fpga_tp1_prgr_dat*)
fpga_prgrm_data_ptr_84303E38761313;
fpga_tp1_program(base, datastruct);
```

Related topics

Basics

Typical Workflow.. 28

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 50.

Note

Even if no errors occurred during handcoding, 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 code.

To solve the problem, you have to modify the FPGA code 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 7). 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

If you want to access an FPGA signal, you must provide it to the processor model.

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.

Features Provided by Specific FPGA Boards

Introduction	The following features are provided only by specific FPGA boards.

Where to go from here

Information in this section

Handcoding Inter-FPGA Communication

Introduction

The DS5203 FPGA Board and the SCALEXIO FPGA base boards support inter-FPGA communication.

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 67.

- 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 I/O function	▪ 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 68.
3) Theoretical maximum

Implementing the inter-FPGA interface

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

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

- Implementing Inter-FPGA Communication via IOCNET on page 68
- Implementing Inter-FPGA Communication via MGT Modules on page 71
- Implementing Inter-FPGA Communication via I/O Module Slots on page 72
- Implementing Inter-FPGA Communication Between DS5203 FPGA Boards on page 83

Handcoding 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:

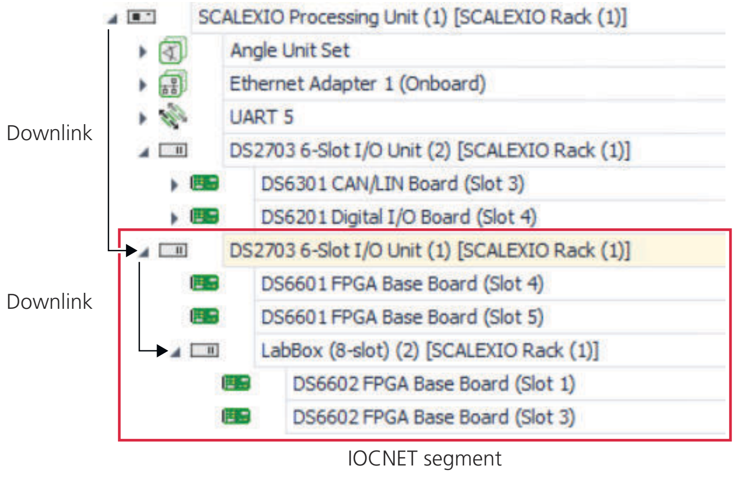
    Typ. 1 Gbit/s, max. 1.6 Gbit/s theoretical
- The latency is typ. 1 µs per network hop. Refer to Calculating the Latency (IOCNET) on page 70.
- Each 32-bit channel can address up to 1024 data values.
- Each 64-bit channel can address up to 512 data values.
- The data type is a raw data type: UFix_32_0 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.

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.

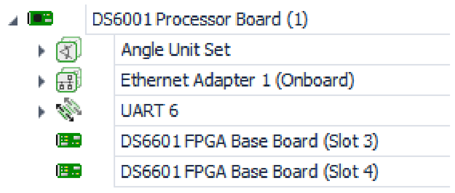
To calculate the latency, you have to count the IOCNET routers between the FPGA base boards, including the onboard routers.

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)

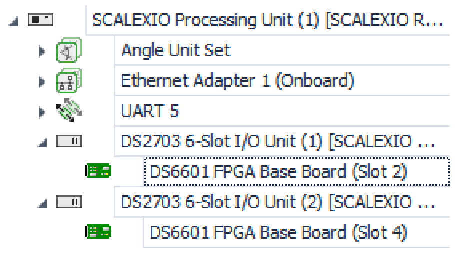
These are the network hops:

- 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
```

Handcoding 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** I/O function.
- 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 I/O Functions of the DS660X_MGT Framework (RTI FPGA Programming Blockset - FPGA Handcode Interface Reference 🕮).

Related topics

Basics

Handcoding 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 76.
- The maximum data rate is 582.3 Mbit/s with default values. Refer to Calculating the Data Rate and Latency (SCALEXIO) on page 80.
- The latency is 72 ns … 96 ns with default values. Refer to Calculating the Data Rate and Latency (SCALEXIO) on page 80.
- 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

You add the inter-FPGA functionality to the FPGA application by configuring the `hc_fpga_framework_ini_<SCALEXIO FPGA base board>.m` file of the SCALEXIO FPGA base board framework.

Refer to Specifying the FPGA I/O Interface on page 35.

Location of SCALEXIO FPGA base board frameworks with inter-FPGA communication:

- DS2655 (7K160) FPGA Base Board:

  `<RCP_HIL_InstallationPath>\MATLAB\RTIFPGA\Frameworks\DS2655_XC7K160T`
- DS2655 (7K410) FPGA Base Board:

  `<RCP_HIL_InstallationPath>\MATLAB\RTIFPGA\Frameworks\DS2655_XC7K410T`
- DS6601 FPGA Base Board:

  `<RCP_HIL_InstallationPath>\MATLAB\RTIFPGA\Frameworks\DS6601_XCKU035`
- DS6602 FPGA Base Board:

  `<RCP_HIL_InstallationPath>\MATLAB\RTIFPGA\Frameworks\DS6602_XCKU15P`

Specifying the inter-FPGA communication

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

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. It is not recommended to implement a bus without bus synchronization or to change the default bus parameter values 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 I/O Functions of the Inter-FPGA Interface Framework (RTI FPGA Programming Blockset - FPGA Handcode 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 inter-FPGA I/O functions. 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 functions that use the same communication channel within the same FPGA application. For example, you cannot use an **I-FPGA In** function on channel number 1 if there is already an **I-FPGA Out** function on this channel, but you can use an **I-FPGA In** function on channel number 2.

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 77.

General configuration aspects

NOTICE

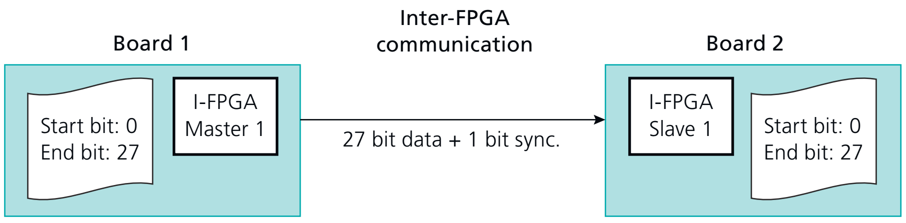
An incorrect configuration might damage the electrical interface.
If you configure both ends of an inter-FPGA connection bus to write on the bus, the connection results in a short circuit. This short circuit might damage the electrical interface of the used I/O module slots. In multiprocessor applications,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

- Make sure that you implement for each I-FPGA Out function an I-FPGA In function as counterpart on the other FPGA board.
- Make sure that the inter-FPGA I/O functions implemented on the other FPGA board uses the same Startbit and Endbit to send or receive the 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

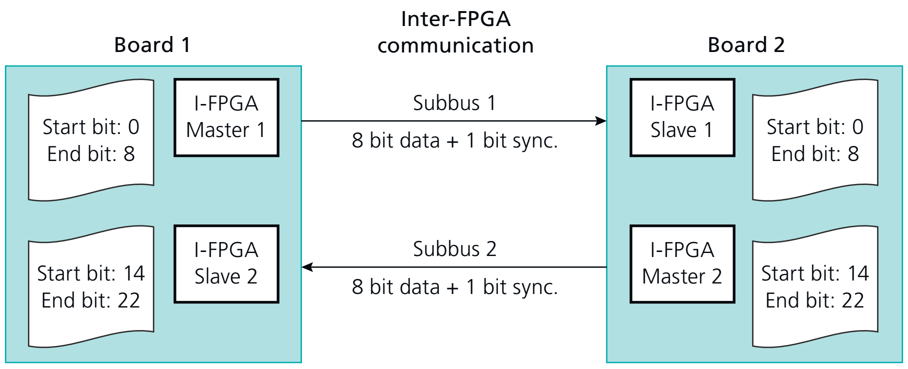
The hardware connections, the implemented FPGA code, and the function settings for the bus configuration have to match to get the required data transfer.

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.

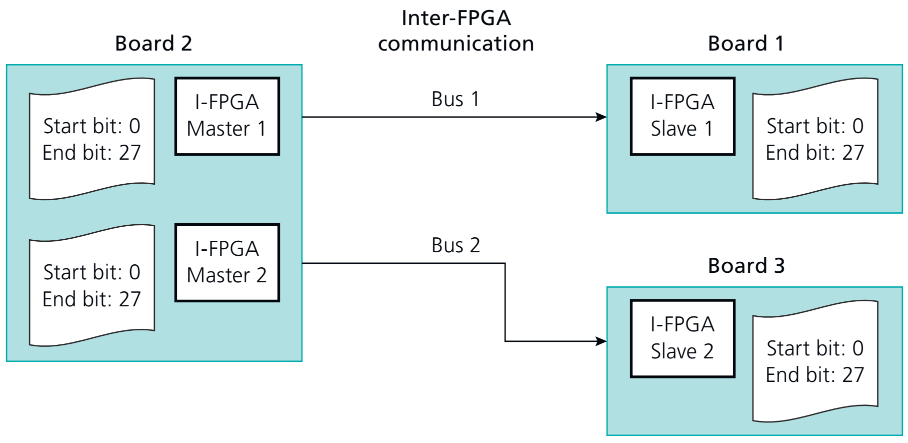


Inter-FPGA communication in both directions with two subbuses In the following example, the FPGA applications can exchange data with a data width of 8 bits in both directions. At the inports and outports of an I-FPGA Block, the position of the LSB is always 0. The offset configured by the start bit for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

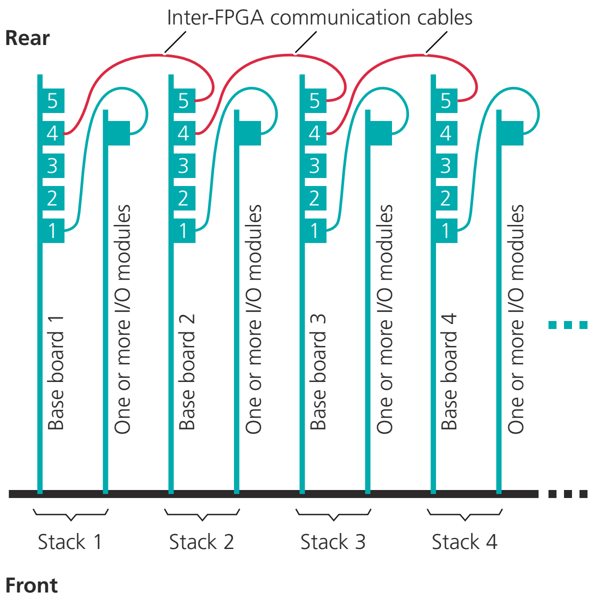


Inter-FPGA communication between multiple SCALEXIO FPGA base boards In the following example with three SCALEXIO FPGA base boards, the FPGA application of board 2 can directly send data to board 1 and board 3. The FPGA application that runs on board 2 uses two inter-FPGA interfaces, one interface for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

References

I/O Functions of the Inter-FPGA Interface Framework (RTI FPGA Programming Blockset - FPGA Handcode Interface Reference 📖)

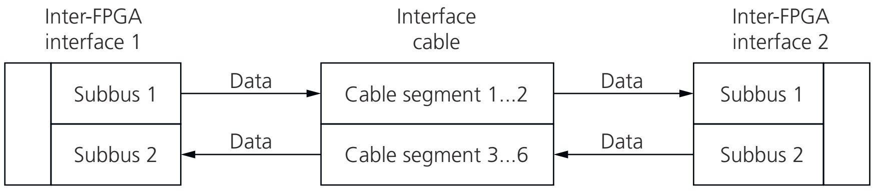
How to Determine the Bit Ranges for Inter-FPGA Subbuses Between SCALEXIO FPGA Base Boards

Objective

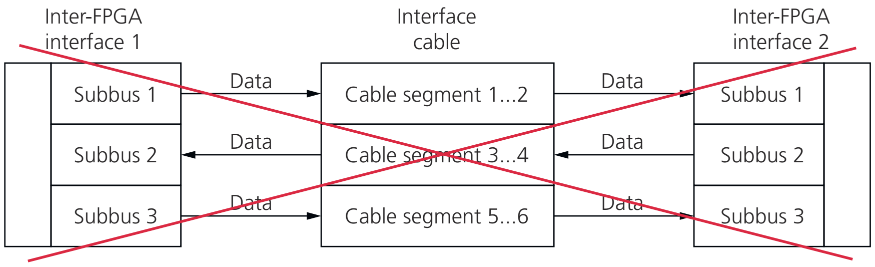
When you implement subbuses to send and receive data via an inter-FPGA communication bus between SCALEXIO FPGA base boards, you must observe limitations for the bit ranges of subbuses.

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
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 79.
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 79.
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 functions to the FPGA code 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**	**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**	**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	**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.
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	**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 and cable segment 3 for the I-FPGA Out function. 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

I/O Functions of the Inter-FPGA Interface Framework (RTI FPGA Programming Blockset - FPGA Handcode Interface Reference 📖)

Calculating the Data Rate and Latency (SCALEXIO)

Calculating data rate

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

$DataRate_{Max}$ = I-FPGA_Clock · (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.

$\mathtt{DataRate}_{\mathtt{Max,Bus}}$ = 125 MHz · (28 - 1) Bit / 6

$\mathtt{DataRate}_{\mathtt{Max,Bit}}$ = 125 MHz · 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.

$\mathtt{DataRate}_{\mathtt{Max,Bus}}$ = 125 MHz · 28 Bit / 6

$\mathtt{DataRate}_{\mathtt{Max,Bit}}$ = 125 MHz · 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:

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

The formula use the following parameters:

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

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

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

$T_{\text{inter-FPGA Master}}$ = $\mathtt{ClockPeriod}_{\text{inter-FPGA}}$ · 2 ClockCycles

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

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

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

- `Latency = 56 ns + FilterDepth · 8 ns`
  - If jitter occurs: `Latency = 56 ns + (FilterDepth + 1) · 8 ns`
  - If spikes occur: `Latency = 56 ns + (FilterDepth + [0 … FilterDepth]) · 8 ns`
  - Maximal: `Latency = 64 ns + (2 · FilterDepth) · 8 ns`

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

$\mathtt{Latency}_{\mathtt{Min}}$ `= 56 ns + 2 · 8 ns = 72 ns`

$\mathtt{Latency}_{\mathtt{Max}}$ `= 64 ns + (2 · 2)· 8 ns = 96 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:

  `Latency = 40 ns`
  - If jitter occurs: `Latency = 44 ns`
- If the filter depth is set to >0:

  `Latency = 36 ns + FilterDepth · 4 ns`
  - If jitter occurs: `Latency = 36 ns + (FilterDepth + 1) · 4 ns`
  - If spikes occur: `Latency = 36 ns + (FilterDepth + [0 … FilterDepth]) · 4 ns`
  - Maximal: `Latency = 40 ns + (2 · FilterDepth) · 4 ns`

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

$\mathtt{Latency}_{\mathtt{Min}}$ `= 36 ns + 2 · 4 ns = 44 ns`

$\mathtt{Latency}_{\mathtt{Max}}$ `= 40 ns + (2 · 2)· 4 ns = 56 ns`

Handcoding 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

Handcoding 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 functions 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 85.
- The latency is 90 ns … 120 ns with default values. Refer to Calculating the Data rate and Latency (PHS-Bus-Based System) on page 85.
- 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 of the inter-FPGA I/O function. The maximum data width of a subbus is `Endbit - Startbit`. You have to make sure that the sending I/O function provides the same bit range as the receiving I/O function.

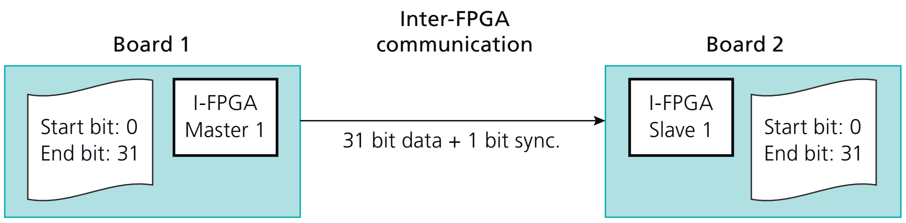
You cannot use an **I-FPGA Master** function and an **I-FPGA Slave** function with the same channel number in the same FPGA application.

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

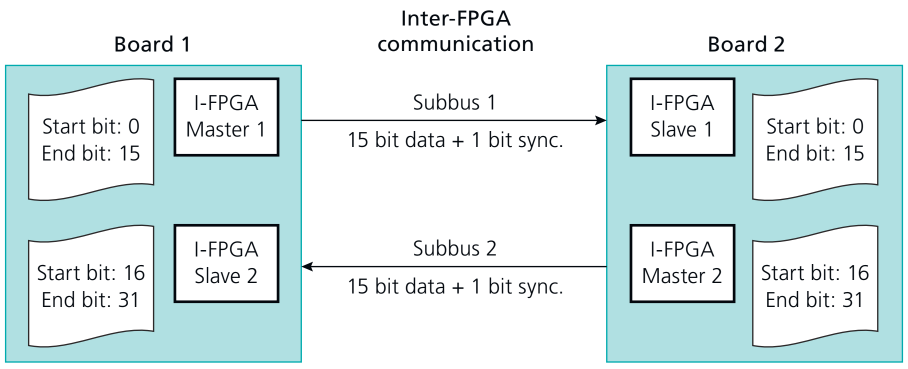
The hardware connections, the implemented FPGA code, and the function settings for the bus configuration have to match to get the required data transfer.

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.. 66

References

I/O Functions of the DS5203 with Onboard I/O Frameworks (RTI FPGA Programming Blockset - FPGA Handcode 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 function settings for the clock, the bit length and the data width.

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

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

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

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

The formula use the following parameters:

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

$T_{\texttt{I-FPGA In}}$:	Latency of the I-FPGA Slave FPGA function. $T_{\texttt{I-FPGA In}} = 10\ \texttt{ns}$

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:

- $\texttt{Latency} = 70\ \texttt{ns} + \texttt{FilterDepth} \cdot 10\ \texttt{ns}$
  - If jitter occurs: $\texttt{Latency} = 70\ \texttt{ns} + (\texttt{FilterDepth} + 1) \cdot 10\ \texttt{ns}$
  - If spikes occur: $\texttt{Latency} = 70\ \texttt{ns} + (\texttt{FilterDepth} + [0 \ldots \texttt{FilterDepth}]) \cdot 10\ \texttt{ns}$
  - Maximal: $\texttt{Latency} = 80\ \texttt{ns} + (2 \cdot \texttt{FilterDepth}) \cdot 10\ \texttt{ns}$

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

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

$\texttt{Latency}_{\texttt{Max}}$ `= 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.

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

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

MicroLabBox: Using Remaining I/O Channels with RTI/RTLib

Introduction

Only the **DS1202 FPGA I/O Type 1 (Flexible I/O)** framework lets you use remaining I/O channels with the RTI blocksets/Real-Time Libraries (RTLib) for MicroLabBox.

Where to go from here

Information in this section

Features of the MicroLabBox Flexible I/O Framework

Introduction

The **DS1202 FPGA IO Type 1 (Flexible I/O)** framework lets you handcode real-time applications for MicroLabBox together with the RTI1202 blockset, the RTI Electric Motor (EMC) blockset, and the Real-Time Library (RTLib) for MicroLabBox.

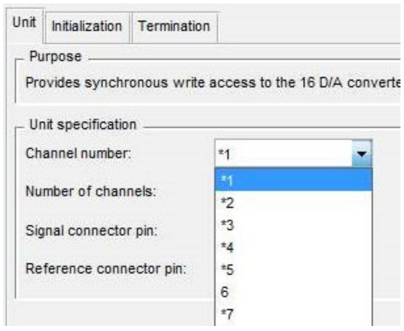
An I/O channel of MicroLabBox is either used by the standard I/O features of MicroLabBox or by a custom FPGA application. The standard I/O features of MicroLabBox let you access the I/O interface with RTI blocksets/RTLib for MicroLabBox. These features are automatically implemented into the FPGA if the 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 RTI blocksets/RTLib for MicroLabBox.

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 RTI blocksets/RTLib for MicroLabBox.
- The framework provides the same I/O functions 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 89.
- 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.



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 RTI blocksets/RTLib for MicroLabBox, 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 handcoding 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 (`DS1302_XC7K325T`) until you have the final version.
2. Copy the `cm` file, the `hc_fpga_framework_ini_DS1302.m` file, and custom code to the DS1202 FPGA I/O Type 1 (Flexible I/O) framework (`DS1302_XC7K325T_FLEXIBLEIO`).
3. Build the final FPGA application.
4. Implement the processor interface with RTI blocksets/RTLib for MicroLabBox to access the remaining I/O channels.
5. Build the processor application.

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

Related topics

Basics

Detailed Instructions on the Handcode Workflow....................31
Features of the MicroLabBox Flexible I/O Framework....................88

DS6602: Accessing the DDR4 RAM

Introduction

The *DS6602 (KU15P) FPGA Base Board* framework lets you access the DDR4 RAM of the FPGA base board.

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 functions 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.

Memory access modes

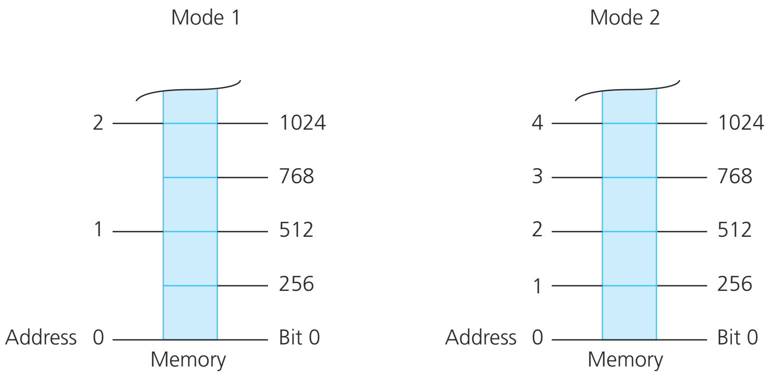
The RTI FPGA Programming Blockset provides two memory access modes:

- 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

I/O Functions of the DS6602 FPGA Base Board Framework (RTI FPGA Programming Blockset - FPGA Handcode Interface Reference 📖)

Initializing the DDR4 RAM of the DS6602

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 📖)

DS6601, DS6602: Coding a Customized MGT Protocol

Introduction

The *DS6601 (KU035) FPGA Base Board* and the *DS6602 (KU15P) FPGA Base Board* frameworks let you customize MGT protocols.

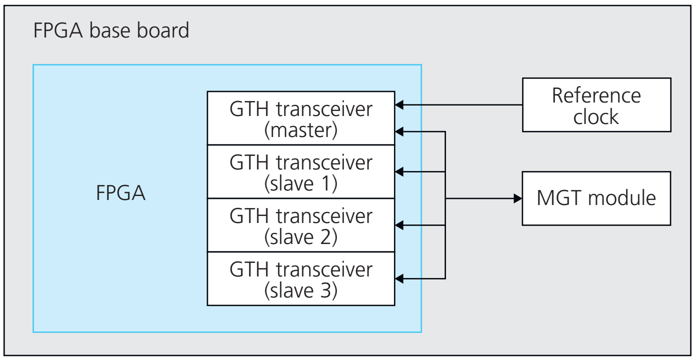
Customizing MGT Protocols

Supported platforms

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

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 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.

Component	Description
	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.

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 the handcode. 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"
    ```

Related topics

Basics

MGT Out (RTI FPGA Programming Blockset - FPGA Handcode Interface Reference 📖)

References

MGT In (RTI FPGA Programming Blockset - FPGA Handcode Interface Reference 📖)

Troubleshooting

Introduction

When working with the RTI FPGA Programming Blockset there are some problems which can arise.

Problems and Their Solutions

Building an FPGA application

If you use a DS6602 FPGA Base Board, the build process issues a critical warning that the design failed to meet the timing requirements:

Implementation (1 critical warning)

- Route Design (1 critical warning)
  - [Timing 38-282] The design failed to meet the timing requirements. Please see the timing summary report for details on the timing violations.
    - Intra-Clock Paths

```
Clock:    clk_out4_clk_ctrl_kup
Type:     Pulse Width Violation
Required:3.333
Actual:   4.000
Slack:    -0.667
At Pin:   FeedbackBaseBoardTemplate_inst/…/REFCLK
          …/io_bus_master_dlyctrl_gen.delayctrl_REPLICATED_0/REFCLK
          …/io_bus_master_dlyctrl_gen.delayctrl_REPLICATED_0_1/REFCLK
          …/io_bus_master_dlyctrl_gen.delayctrl_REPLICATED_0_2/REFCLK
          …/io_bus_master_dlyctrl_gen.delayctrl_REPLICATED_0_3/REFCLK
          …/io_bus_master_dlyctrl_gen.delayctrl_REPLICATED_0_4/REFCLK
          …/io_bus_master_dlyctrl_gen.delayctrl_REPLICATED_0_5/REFCLK
```

Solution Ignore this warning because the warning is issued in error.