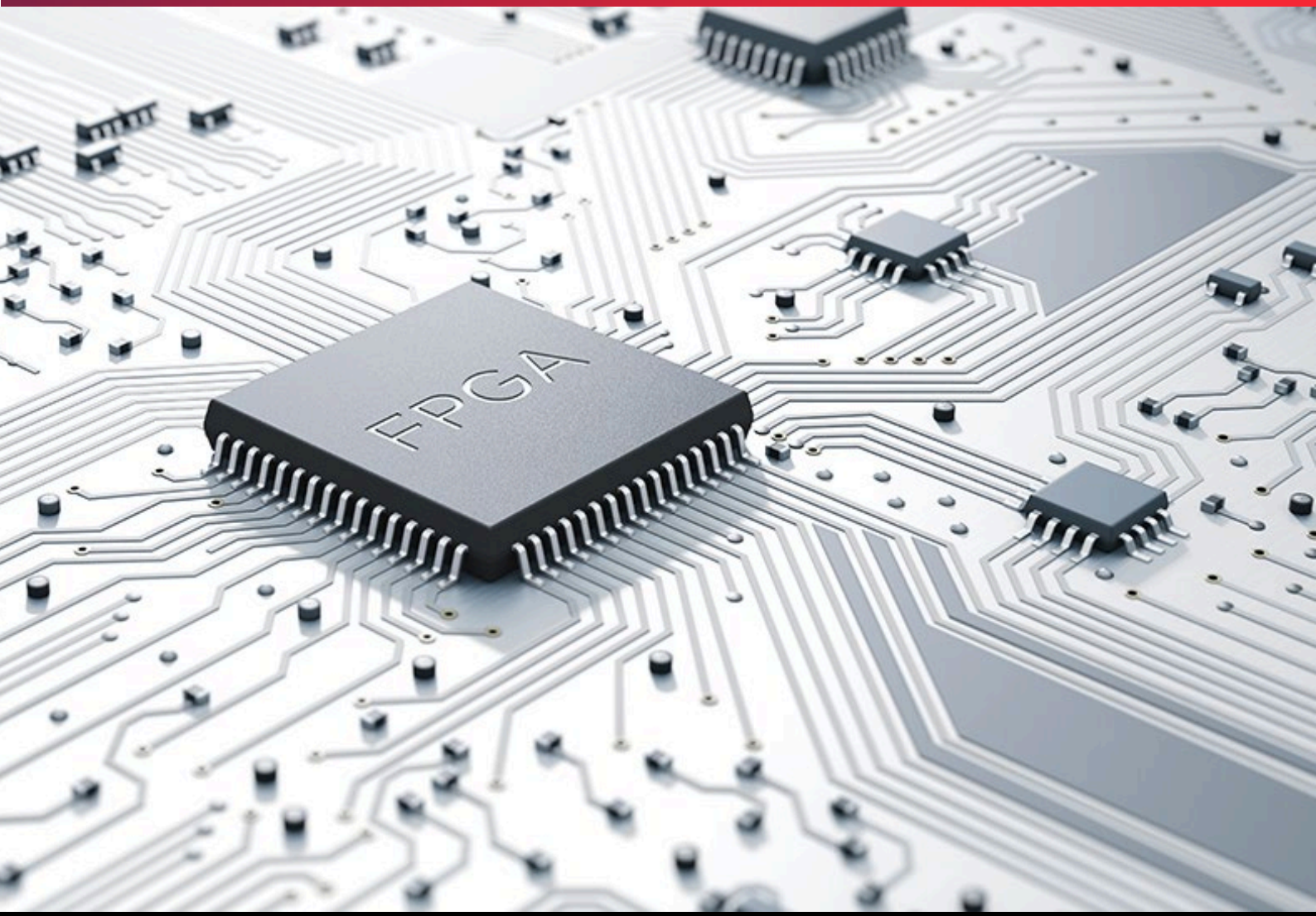
VECTOR

VT System FPGA Manager

User Programmable FPGA

User Manual

Version 3.5.2



English

vector.com

Imprint

Vector Informatik GmbH
Ingersheimer Straße 24
D-70499 Stuttgart

Table of Contents

1 Introduction 6

1.1 VT System FPGA Manager at a Glance 7

1.2 About this User Manual 8
- 1.2.1 Navigational Aids and Conventions 8
- 1.2.2 Latest Information 9
- 1.2.3 Certification 9
- 1.2.4 Warranty 9
- 1.2.5 Support 9
- 1.2.6 Registered Trademarks 9

2 Basic Concepts 11

2.1 Hardware Concept 12

2.2 Tool Chain 13

3 The VT System FPGA Manager 14

3.1 Prerequisites 15

3.2 Configuring the VT System FPGA Manager 15
- 3.2.1 Setting the Workbench Directory 16
- 3.2.2 Setting up the Simulink® Model Editor and DSP Builder® Libraries 16
- 3.2.3 Setting up the Quartus® Compiler 16
- 3.2.4 Example Projects 16

3.3 Managing Projects 16
- 3.3.1 Creating new Projects 16
- 3.3.2 Creating a New Multi Module Project 17
- 3.3.3 Creating a New Project For The VT7900A 19
- 3.3.4 Opening Projects 19
- 3.3.5 Renaming Projects 20
- 3.3.6 Deleting Projects 20
- 3.3.7 Editing Projects 20
- 3.3.8 Compiling Projects 24
- 3.3.9 Testing and Persisting Projects 26
- 3.3.10 Working with Project References 27
- 3.3.11 Clearing the User FPGA 27
- 3.3.12 Updating a VT7900A Configuration 27

3.4 Module Specific Settings 28
- 3.4.1 VT5838 Specific Settings 28

3.5 Command Line Interface (CLI) 28
- 3.5.1 Getting Help 29
- 3.5.2 Importing a Project Into a Workbench 29
- 3.5.3 Exporting a Project From The Workbench 30
- 3.5.4 Deleting a Project From The Workbench 31
- 3.5.5 Compiling a Project 31
- 3.5.6 Downloading a Project In Test Mode 31
- 3.5.7 Downloading a Project In Persisting Mode 32
- 3.5.8 Clearing a User FPGA 33

4 Writing HDL Code for VT System FPGA Modules 35
4.1 Working with the FPGA Manager 36
4.2 Coding with VHDL 36
4.3 Folder Structure 36
4.4 Template File 36
4.5 Adding/Removing/Changing System Variables 39
4.6 Using Quartus® 39

5 Modeling the Behavior of VT System FPGA Modules (DSP Builder) 40
5.1 Using the Model File Template 41
5.2 Simulation 44
5.3 Compiling and Download to FPGA 44
5.4 Adding/Removing/Changing System Variables 45
5.5 Integrating HDL Generating Simulink models 45

6 Modeling the Behavior of VT System FPGA Modules (HDL Coder) 48
6.1 Integrating The Generated Model Code 49
6.2 Mapping The Model Signals 49
6.3 Compiling and Download to FPGA 49
6.4 Adding/Removing/Changing System Variables 49

7 Inter Board Communication (IBC) 50
7.1 General Concept 51
7.2 Configuration in the VT System FPGA Manager 51
7.3 Hardware Connections 52
7.4 Performance Considerations 54
7.5 Inter Board Communication Plus (IBC+) 55
7.5.1 Configuration in the FPGA Manager 55
7.5.2 Performance Considerations 56

8 Signal Reference 57
8.1 Clock and Reset Signal 58
8.2 Additional Signals 58
8.3 Debugging 58
8.4 Time Stamps 62
8.5 VT1004A 62
8.6 VT1104 64
8.7 VT2004A 64
8.8 VT2516A 65
8.9 VT2710 66

8.10 VT2808 67
8.11 VT2816 68
8.12 VT2848 68
8.13 VT5838 69
8.13.1 Hardware Interface 75
8.14 VT7900A 79

9 Troubleshooting 81
9.1 Problems and Solutions 82

1 Introduction

This chapter contains the following information:

1.1	VT System FPGA Manager at a Glance	page 7
1.2	About this User Manual	page 8
	Navigational Aids and Conventions	
	Latest Information	
	Certification	
	Warranty	
	Support	
	Registered Trademarks	

1.1 VT System FPGA Manager at a Glance

Overview

For some VT System modules Vector offers a special version equipped with a user programmable FPGA. This allows for implementing custom functionality with direct access to the VT System module's hardware and very high performance. This makes the VT System modules even more flexible and applicable for almost any kind of application.

To aid you in the process of developing custom FPGA functionality for your FPGA enabled VT System modules Vector provides the VT System FPGA Manager. This utility helps to manage your projects and simplifies the process of compiling and downloading them to VT System modules. However the VT System FPGA Manager is no HDL code editor or modeling tool. Instead you can use your favorite IDE or alternatively Simulink® to actually write or model the HDL code implementing your desired functionality.

1.2 About this User Manual

1.2.1 Navigational Aids and Conventions

To find information quickly

This user manual provides you with the following navigational aids:

> At the beginning of each chapter you will find a summary of the contents
> The header shows which chapter and paragraph you are located in
> The footer shows which version the user manual refers to

Conventions

The following two charts show the spelling and symbol conventions used in this manual.

Style	Utilization
bold	Fields, interface elements, window and dialog names in the software. Accentuation of warnings and notes. **[OK]** Buttons are denoted by square brackets **File | Save** Notation for menus and menu commands
CANoe	Legally protected proper names and side notes.
`Source code`	File name and source code.
Hyperlink	Hyperlinks and references.
<Ctrl>+<S>	Notation for shortcuts.

Symbol	Utilization
	You can obtain supplemental information here.
	This symbol calls your attention to warnings.
	You can find additional information here.
	Here is an example that has been prepared for you.
	Step-by-step instructions provide assistance at these points.
	Instructions on editing files are found at these points.
	This symbol warns you not to edit the specified file.

1.2.2 Latest Information

Additional technical information

You may find additional technical information about your VT System:

> in the CANoe online help,
> on the Vector website www.vector.com (e.g. application notes), and
> in your CANoe installation.

Reference: You may find the **latest version of this manual** in your CANoe installation (start menu ⇨ CANoe ⇨ Help).

1.2.3 Certification

Certified Quality Management System

Vector Informatik GmbH has ISO 9001:2008 certification.
The ISO standard is a globally recognized quality standard.

CE Compliance

All VT System products comply with CE regulations.

1.2.4 Warranty

Limitation of warranty

We reserve the right to change the contents of the documentation and the software as well as the hardware design without notice. Vector Informatik GmbH assumes no liability for correct contents or damages which are resulted from the usage of the VT System FPGA Manager user manual. We are always grateful for references to mistakes or for suggestions for improvement, so as to be able to offer you even better-performing products in the future.

1.2.5 Support

Need support?

You can get through to our hotline by calling

+49 (711) 80670-200

or you can send a problem report to CANoe Support.

1.2.6 Registered Trademarks

Registered trademarks

All trademarks mentioned in this user manual, including those registered to third parties, are governed by the respective trademark laws and are the property of their respective owners. All trademarks, trade names or company names are or can be trademarks or registered trademarks of their particular owners. All rights which are not expressly allowed are reserved. Failure to explicitly note any given trademark within this user manual does not imply that a third party does not have rights to it.

> Windows is a trademark of the Microsoft Corporation.
> EtherCAT® is registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany.

- MATLAB® and Simulink® are registered trademarks of The MathWorks, Inc.
- Intel® and Quartus® are registered trademarks of Intel Corporation.

2 Basic Concepts

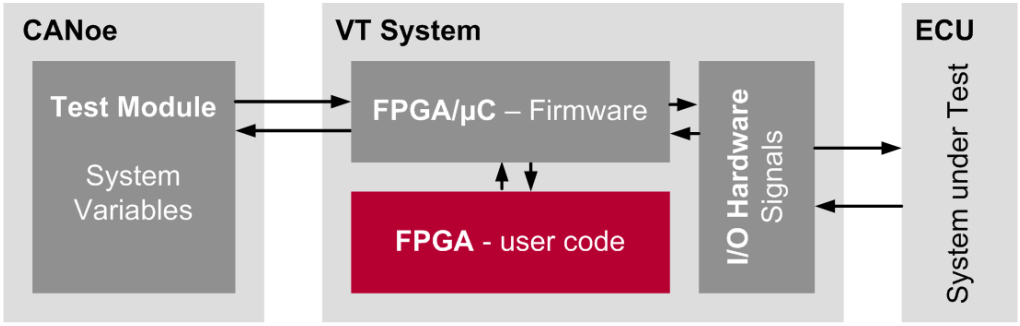
This chapter contains the following information:

2.1	Hardware Concept	page 12
2.2	Tool Chain	page 13

2.1 Hardware Concept

User FPGA

On the VT System modules basically FPGAs from Intel® are used. Usually on the processor board there is one main FPGA which is responsible for the data and signal processing and accessing the I/O hardware. On the special FPGA processor board there is a second User FPGA which is user programmable as the following figure shows:



The User FPGA has no direct connection to CANoe or the I/O Hardware on the VT System module and therefore to the ECU. So the focus during creating your own functionality has to be set only to the data processing but not how to access the I/O hardware or CANoe. The controlling of the I/O hardware and the preparing of the raw data will be done by the main FPGA. The User FPGA will get the prepared raw measurement data cyclic from the main FPGA and the main FPGA will also output the data from the User FPGA via the I/O hardware when the signal changes. The communication with CANoe will be handled by the main FPGA in the same way. As the User FPGA is only signal based and has no direct hardware access for example switching relays or setting measurement or output ranges have to done as usual in your test sequence within CANoe.

Programming the User FPGA will be done very convenient by the VT System FPGA Manager via the firmware without the need of additional programming hardware or tools. The User FPGA communicates with the following signals:

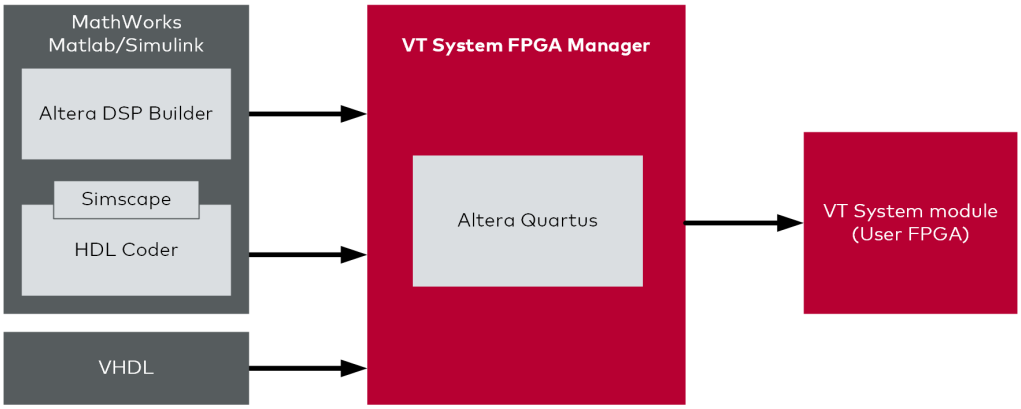
- Communication with CANoe will be done via system variables. They can be defined with the VT System FPGA Manager. Signals to CANoe will be updated cyclic.
- Input signals from the I/O hardware will be available cyclic at the User FPGA.
- Output signals to the I/O hardware will be updated on signal change.

2.2 Tool Chain

Supported design entries

As shown in the following figure the VT System FPGA Manager supports two design entries for the User FPGA:

- Graphical modeling with Simulink® with a special block set provided from Intel®
- Modeling with Simscape®, code generation with HDL Coder®
- Description with VHDL



The compilation of the project will be done with Intel® Quartus®, the design environment for Intel® FPGAs. This will be done automatically in the background because Quartus® is completely controlled by the VT System FPGA Manager. Downloading the compiled code will be done by the firmware also controlled by the VT System FPGA Manager.

Main tasks

So the main tasks of the VT System FPGA Manager are:

- **Project Management**:
  Add, change, and delete projects
- **Configuration**:
  Add, change, and delete System Variables
- **Compilation**:
  Translating the user code to gate level
- **Programming**:
  Download the compiled user code to the User FPGA temporarily or permanent

3 The VT System FPGA Manager

This chapter contains the following information:

3.1	Prerequisites	page 15
3.2	Configuring the VT System FPGA Manager Setting the Workbench Directory Setting up the Simulink® Model Editor and DSP Builder® Libraries Setting up the Quartus® Compiler Example Projects	page 15
3.3	Managing Projects Creating new Projects Creating a New Multi Module Project Creating a New Project For The VT7900A Opening Projects Renaming Projects Deleting Projects Editing Projects	page 16

3.1 Prerequisites

Overview

Before using the VT System FPGA Manager please make sure the following applications are installed correctly:

- Quartus® by Intel® version 19.1 or newer is mandatory for all source files to work properly (in most cases the free Quartus® Prime Lite suffices)
- Device files for Cyclone IV FPGAs by Intel® (necessary for most modules)
- Devices files for Cyclone V FPGAs by Intel® (necessary for VT5838)

If you plan to create your FPGA code by modeling it in Simulink® you also need the following software products:

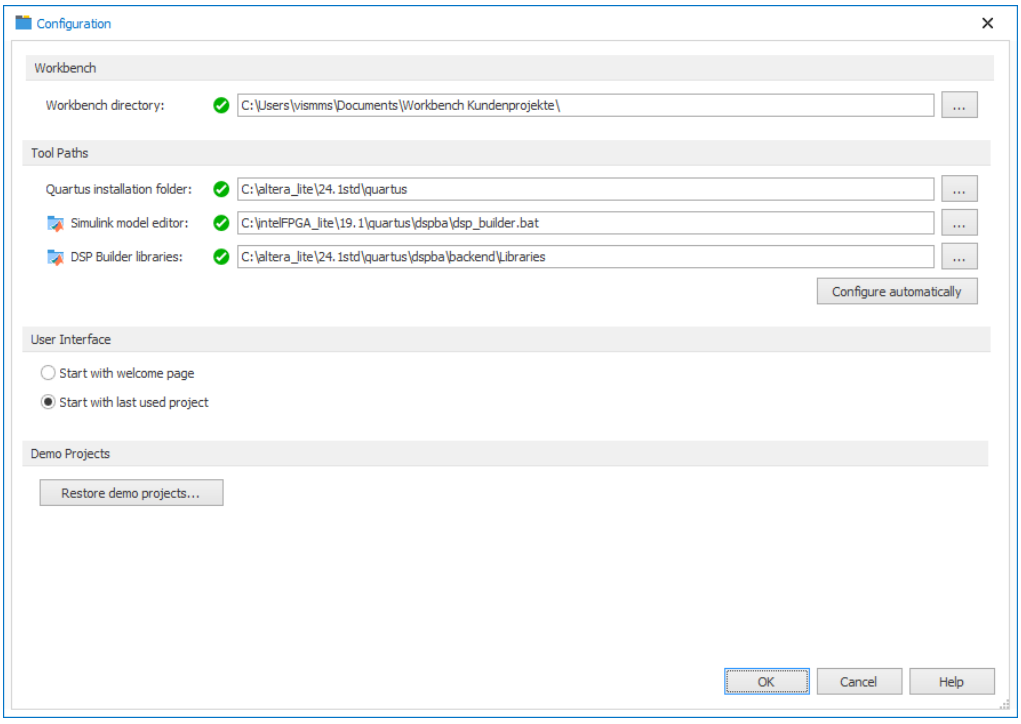
- MATLAB® version 10.3 (r2021a) or older with Simulink® by Mathworks
- DSP Builder Standard by Intel® (19.1 recommended)

Note: Vector recommends to use MATLAB® version 2011b or later. Although earlier versions work in most cases, too, there might be stability problems when using the DSP Builder Simulink® block set.

3.2 Configuring the VT System FPGA Manager

First configuration

Before using the VT System FPGA Manager a quick configuration is necessary. When the VT System FPGA Manager is started for the first time the configuration dialog is automatically shown. If you wish to change the settings later on you can always open the configuration dialog manually via **Tools|Options...**.



3.2.1 Setting the Workbench Directory

Workbench directory

The workbench directory is the folder in which all newly created projects will be stored. It is recommended to create a new, empty folder for this purpose. Please also make sure that you have write access to the folder and that enough hard disk space is available to store your projects.

3.2.2 Setting up the Simulink® Model Editor and DSP Builder® Libraries

Modeling in Simulink

If you wish to create your FPGA code by modeling it in Simulink® this setting is important. It specifies the path to the executable that will be used to open your .mdl files. Usually this is not MATLAB® but rather a batch file called **dsp_builder.bat**. This file is supplied by Intel® to open Simulink® with correctly set up block sets.

The DSP Builder Libraries path points to the location where the DSP Builder libraries are stored on your hard drive. Usually this is a subfolder in the Quartus® installation directory as shown in the image above.

3.2.3 Setting up the Quartus® Compiler

Compiling FPGA projects

The VT System FPGA Manager uses the Quartus® compiler to compile your FPGA projects. There is no extra setting in the options dialog for specifying a path to this compiler. Instead the path is read from the system's environment variable **PATH**. This variable should be modified automatically when installing Quartus®.

3.2.4 Example Projects

Examples in workbench directory

When setting up the VT System FPGA Manager for the first time two example projects will be copied to your workbench directory. The first project is created with MATLAB Simulink while the second project consists of manually written HDL code. The purpose of these projects is to demonstrate the basic functionalities of VT System FPGA modules and to serve as a starting point for your own projects. A detailed description of every example project can be found in the project's files, either the Simulink model or the main HDL code file.

3.3 Managing Projects

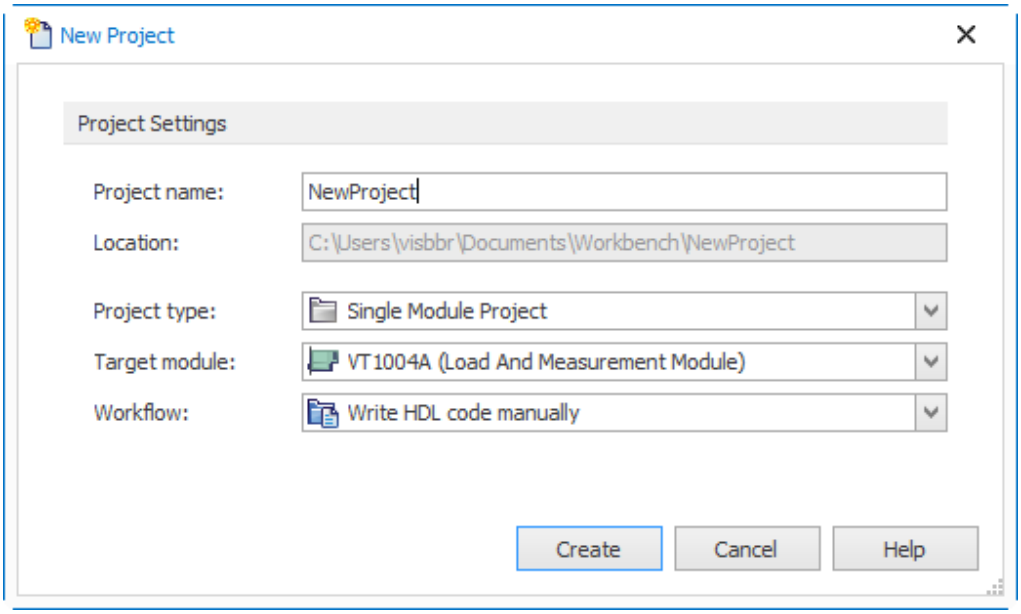
Overview

The VT System FPGA Manager allows you to create new projects, open projects for editing and delete existing projects.

3.3.1 Creating new Projects

Project creation dialog

To create a new project choose **File|New project...** or click the corresponding button in the toolbar. Alternatively perform a right click into the Project Explorer and click on **[New project...]**. This opens the project creation dialog.



In this dialog the following settings have to be made:

- **Project name**:
  Specifies the name of the new project. Please note that the name has to meet the naming conventions and must be unique.
- **Location**:
  This shows where the new project will be located on your hard drive. This setting can only be changed by changing the project's name or your workbench directory.
- **Project type:**
  You can choose between a single module project (for one VT module) and a multi module project (for multiple VT modules).
- **Target module**:
  Here the VT System module you are developing for has to be selected.
- **Workflow**:
  This setting lets you choose between three workflows. On the one hand you can write the HDL code for your project manually. On the other you can choose to create a Simulink® model and generate the HDL code from that model automatically. The third option is to model your project with Simscape and generate the code via HDL Coder. All workflows are explained in detail in the following chapters.

3.3.2 Creating a New Multi Module Project

Project creation

Creating a multi module project is a two step process. First of all you create an empty multi module project. Then you can add one or more sub-projects.

In order to create a multi module project, choose the project type **Multi Module Project** in the project creation dialog.

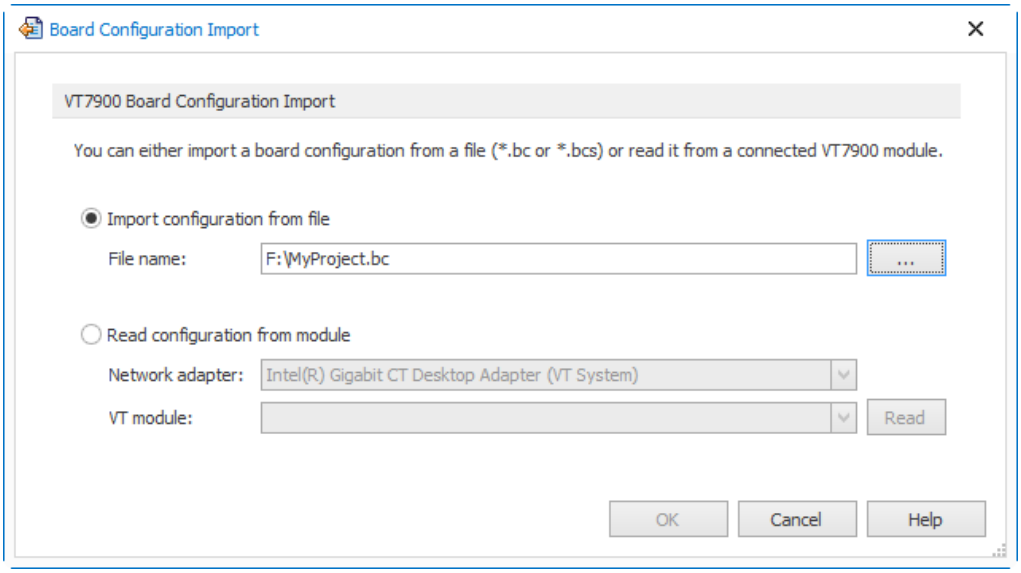
Adding sub-projects

To add a sub project, right-click the multi module project in the Project Explorer. Then choose **New sub-project...** from the context menu. Alternatively you can choose **File|New sub-project...** from the menu.

3.3.3 Creating a New Project For The VT7900A

Board configuration import

When creating a User FPGA project for the VT7900A it is required to import a board configuration file. This file contains information about the application board mounted to the VT7900A. In addition it defines which features of the VT7900A User FPGA will be available for programming. The board configuration files have the extension .bc or .bcs and can be created by using the VT System Application Board Designer. This tool is shipped with CANoe, just like the VT System FPGA Manager.



In this dialog you have two options:

> **Import configuration from file**:
This allows you to load a .bc or .bcs file previously saved with the VT System Application Board Designer.

> **Read configuration from module**:
This allows you to read the configuration from a connected VT7900A module. Please make sure that CANoe is not running in the background. Otherwise the connected module may not be detected.

3.3.4 Opening Projects

Opening projects

To open a project for editing, simply double click on the project in the Project Explorer. You may open multiple projects at once. A tab will be created for each project allowing you to quickly switch between them. On application start-up the last used project will be automatically re-opened. This behavior can be changed via the options dialog.

3.3.5 Renaming Projects

Renaming projects

To rename a project perform a right click on the project in the Project Explorer and choose **[Rename...]**. This will change the name of the project's folder on the disk and will modify the contents of several project files.

Warning: please make sure that you close all applications that may have access to files in your project folder (e.g. Windows Explorer, Simulink®, code editors etc.). Otherwise the VT System FPGA Manger might not be able to rename the project folder.

3.3.6 Deleting Projects

Deleting projects

To delete a project click on **File| Delete project...**. Alternatively you may perform a right click on the project in the Project Explorer and choose **[Delete...]**. The project's folder will then be removed from your workbench directory.

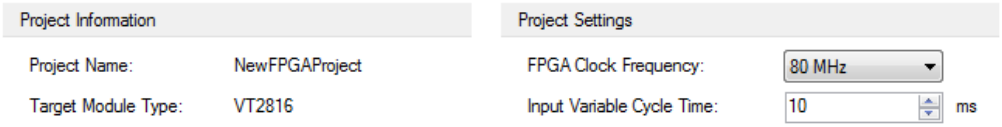
3.3.7 Editing Projects

Editing projects

After a project has been opened in the VT System FPGA Manger it can be edited.

Project settings

The top of the project editor shows some general project information, like the project's name and the targeted VT System module.

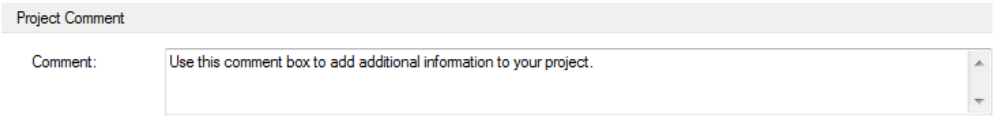


Here you can also make general settings that are available in every type of project, no matter which workflow you are using:

- **FPGA Clock Frequency**:
  This is the clock frequency to be used by the FPGA. The more complex your HDL code gets the lower the frequency should be. If the frequency is too high for your project's complexity timing errors may arise. In that case the Quartus® compiler will generate adequate warning messages.
- **Input Variable Cycle Time**:
  This setting specifies the cycle time in ms at which the signals from the User FPGA to CANoe will be updated.

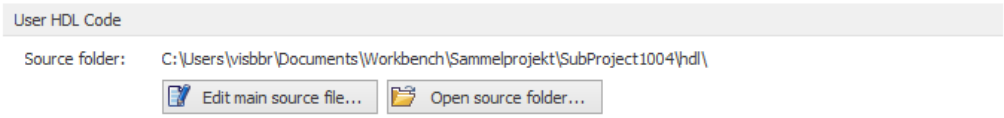
Project comment

The comment box allows you to add metadata to your project. For example it can be used to describe the functionality of your project or store information like the creation date or the author.



Manual workflow

This section only applies if you have created a project using the manual HDL coding workflow.

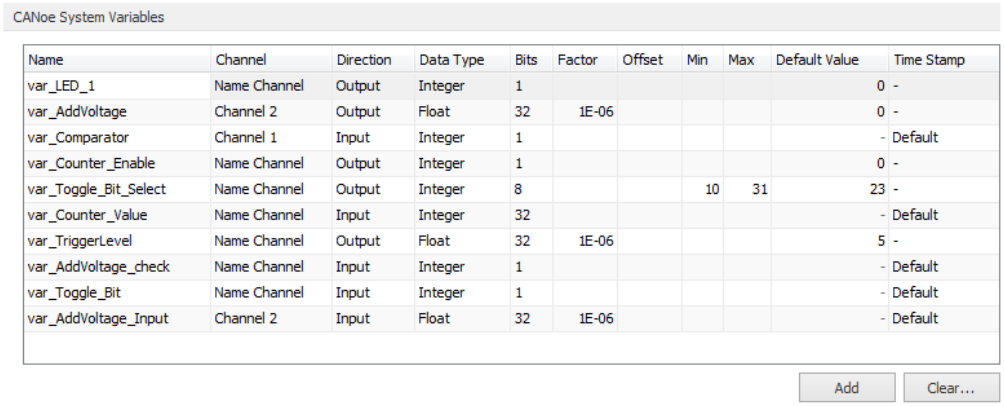


The second group in the project editor shows the path to your project's HDL source files. Via the button **[Open Source Folder...]** you can quickly open this folder from within the VT System FPGA Manager.

For detailed information on how to create a VT system FPGA project by manually writing the HDL code please refer to the corresponding chapter Writing HDL Code for VT System FPGA Modules.

System variables

The third group in the project editor lets you manage the CANoe system variables for your project.



Name	Channel	Direction	Data Type	Bits	Factor	Offset	Min	Max	Default Value	Time Stamp
var_LED_1	Name Channel	Output	Integer	1					0	-
var_AddVoltage	Channel 2	Output	Float	32	1E-06				0	-
var_Comparator	Channel 1	Input	Integer	1					-	Default
var_Counter_Enable	Name Channel	Output	Integer	1					0	-
var_Toggle_Bit_Select	Name Channel	Output	Integer	8			10	31	23	-
var_Counter_Value	Name Channel	Input	Integer	32					-	Default
var_TriggerLevel	Name Channel	Output	Float	32	1E-06				5	-
var_AddVoltage_check	Name Channel	Input	Integer	1					-	Default
var_Toggle_Bit	Name Channel	Input	Integer	1					-	Default
var_AddVoltage_Input	Channel 2	Input	Float	32	1E-06				-	Default

On the one hand these variables can be used to send measurement values from the User FPGA to CANoe. On the other hand stimulation values can be transferred from CANoe to the User FPGA.

Adding/removing system variables

To add and remove variables in your project you can use the corresponding buttons in the lower right corner of the project editor. Alternatively you can open a shortcut menu by right clicking into the System Variable table.

The number of available system variables is 128 for input and 128 for output. For the VT5838 it is 256 in both cases.

Naming system variables

All system variables' names must conform to the CANoe naming conventions. For example names must not start with a cypher or contain blanks. Please see the CANoe online help for a full overview of the naming rules. Additionally the system variables' names may not use any reserved keywords. These include the names of automatically generated inputs and outputs (e.g. "debug_LED_", "in_signal_" or "out_dac_").

Configuring system variables

Each of the system variables offers the following settings:

Name	This is the name that will be displayed in CANoe. The name is also used in the HDL code as well as the Simulink® model. On every channel each system variable name must be unique and meet certain naming constraints.
Channel	In CANoe each system variable is assigned to one specific channel of the VT System module. That channel is specified via this setting.
Direction	Here you can determine if the variable shall be an input (measurement) variable or an output (stimulation) variable.
Data Type	This setting specifies if the variable shall contain floating point or integer numbers.
Bits	Specifies the number of bits used to store this variable's values. The value range and the resolution of the system variable are depending on this setting.
Factor	Every value that is transmitted from the User FPGA to CANoe is multiplied by this factor. Values transmitted from CANoe to the User FPGA are divided by this factor. See the explanation of factor and offset below for more details. This is an optional setting. If no value is entered a default value of 1.0 is assumed.
Offset	Every value that is transmitted from the User FPGA to CANoe is increased by this offset. Values transmitted from CANoe to the User FPGA are decreased by the offset. See the explanation of factor and offset below for more details. This is an optional setting. If no value is entered a default value of 0.0 is assumed.
Min	This setting specifies the minimum value that is displayed in CANoe for this system variable. This is an optional setting. If no value is entered no minimum value will be displayed.
Max	This setting specifies the maximum value that is displayed in CANoe for this system variable. This is an optional setting. If no value is entered no maximum value will be displayed.
Default Value	For output (stimulation) variables a default value can be specified with this setting. Until the system variable has been set for the first time in CANoe, it has this default value.

Time Stamp	By default all input (measurement) variables will be updated in CANoe with the same time stamp. This time stamp is computed by the VT System module automatically every time the variable values are transmitted to CANoe. By using this setting you can assign your variables to time stamps other than the default one. All variables assigned to a custom time stamp will then use this time stamp's value when being updated in CANoe. The non-default time stamps will appear as signals in the VHDL code or as blocks in the Simulink model. You can then write your own values to these time stamp signals/blocks.

Note: The **factor** and **offset** settings are applied in two directions. Values that are transferred from the User FPGA to CANoe are multiplied by the factor and then increased by the offset. Values transferred from CANoe to the User FPGA are decreased by the offset and then divided by the factor.

Note: All **system variables** will be automatically made available in the Simulink® model (if used) or the HDL code every time the project is saved. Please do not attempt to add, remove or modify system variables directly in the model or HDL code files. Always use the VT System FPGA Manager to perform these operations.

Note: Changing project settings or system variables will result in a new module ID. CANoe recognizes a module by its ID. Therefore a settings or variables change will result in CANoe recognizing a new module which is not compatible with a different version of the same User FPGA project.

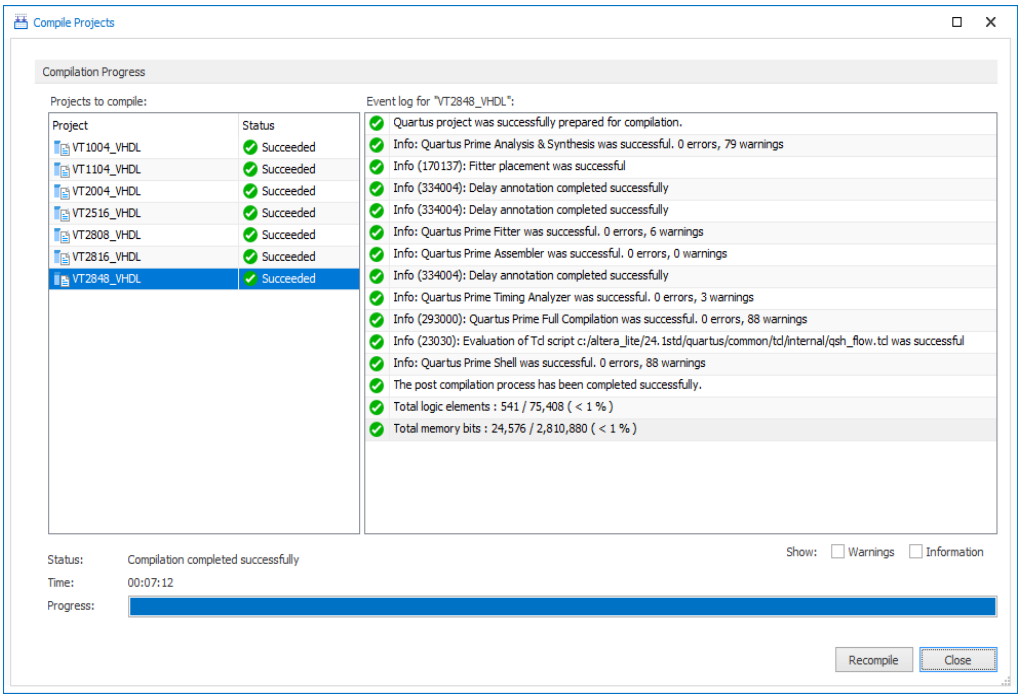
3.3.8 Compiling Projects

Compiling projects

To compile a project select **Project|Compile** from the menu bar or choose **Compile...** from the project's context menu in the project explorer. This will open up the compilation dialog.

Compilation dialog

The compilation dialog uses the Quartus® compiler to compile your FPGA project. All of the compiler's outputs will be displayed in the event log.



By default only success and error messages are displayed. To show warnings and additional information simply activate the corresponding checkboxes below the event log. Above the event log the current status of the compilation process and the time taken is displayed.

Compiling multi module projects

When dealing with multi module projects you can compile all contained sub-projects at once. To do so make sure issue the compile command on the multi module project, not one of the sub-projects.

Troubleshooting

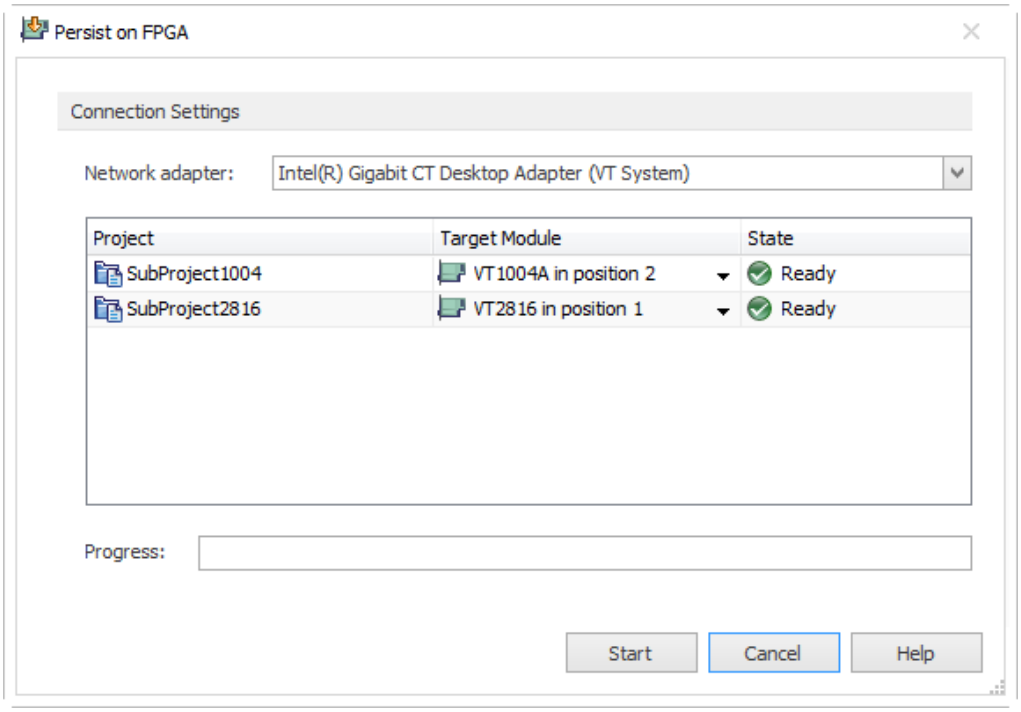
If the compilation of your project fails according error messages will be shown in the event log. These should help you fix any problems preventing a successful compilation.

In some cases it might also be helpful to open your project directly with Quartus® to gain additional information on the error. To do so, navigate to your project's folder in the workbench. Open the subfolder **quartus/SYN** where you can find the **.qpf** project file that can be directly opened with Quartus®.

3.3.9 Testing and Persisting Projects

Overview

After your project has been successfully compiled it can be downloaded to the VT System module. The VT System FPGA Manager offers two ways of downloading the project, one for testing and one for persisting the project.



Testing projects

When you choose to download a project for testing the project will only be stored temporarily on the VT System module. That means that it will be gone after the next power cycle. This function is intended to be used during active development of your project since it allows quickly performing development/testing iterations.

To download a project for testing purposes click on **Project|Test**. Alternatively you may perform a right click on the project in the Project Explorer and choose **[Test...]**.

Persisting projects

When you choose to persist a project the project will be stored permanently on the VT System module. This function is intended to be used when the project has been completed and is ready to be used.

To persist a project click on **Project|Persist**. Alternatively you may perform a right click on the project in the Project Explorer and choose **[Persist...]**.

Multi module projects	When dealing with multi module projects you can test/persist all contained sub-projects at once. To do so make sure issue the test/persist command on the multi module project, not one of the sub-projects.

3.3.10 Working with Project References

Overview	The VT System FPGA Manager allows you to work with projects that are not located in the workbench directory. This might be useful in cases where the project has to be stored externally, e.g. to be managed by a version control system like SVN.
Adding a reference	To add an externally stored project as a reference choose **File|Add project reference...** from the menu bar. Alternatively you can issue the same command from the Project Explorer's shortcut menu. Then navigate to the project folder and open the project.xml file located in that folder. The project is now added as a reference to your workbench. This is indicated by a slightly different icon in the Project Explorer.
Working with a reference	Referenced projects are treated exactly like regular projects. This means that all of the above information on project editing, compiling and persisting apply here as well.
Deleting a reference	When you delete a referenced project in the VT System FPGA Manager only the reference will be removed. The actual files of your project won't be deleted from your hard drive.

3.3.11 Clearing the User FPGA

Overview	With the VT System FPGA Manager you can easily clear the User FPGA. This will remove the project currently downloaded to the FPGA and thus disable any custom functionality. **Warning**: Clearing the User FPGA cannot be undone.
Clearing the FPGA	To clear the User FPGA click on **More| Clear User FPGAs**. Then select the network adapter your VT System is connected to. Now all available FPGA enabled VT System modules will be shown. Select the one you want to clear and press **Start**.
Restoring the User FPGA's functionality	As mentioned above the clearing of the User FPGA cannot be undone. If you want to restore its previous functionality you have to re-download the corresponding project.

3.3.12 Updating a VT7900A Configuration

Overview	It may occur that you have to make changes to your VT7900A board configuration after you have created an User FPGA project for your application board. In that case you have to update the board configuration in your User FPGA project.
Importing the new configuration	To update the board configuration oft he current project use the button **Update board configuration...** under **Project Information**. This will open up a dialog where you can either choose the updated .bc or .bcs file, or read the updated configuration from a connected VT7900A module.

Warning

Warning: Importing a new configuration will save all unsaved changes you have made to your project. In addition most oft he code files and/or your Simulink® model will be modified (e.g. some signals may be added or removed). Thus it is highly recommended to create a backup of your project before updating the board configuration.

3.4 Module Specific Settings

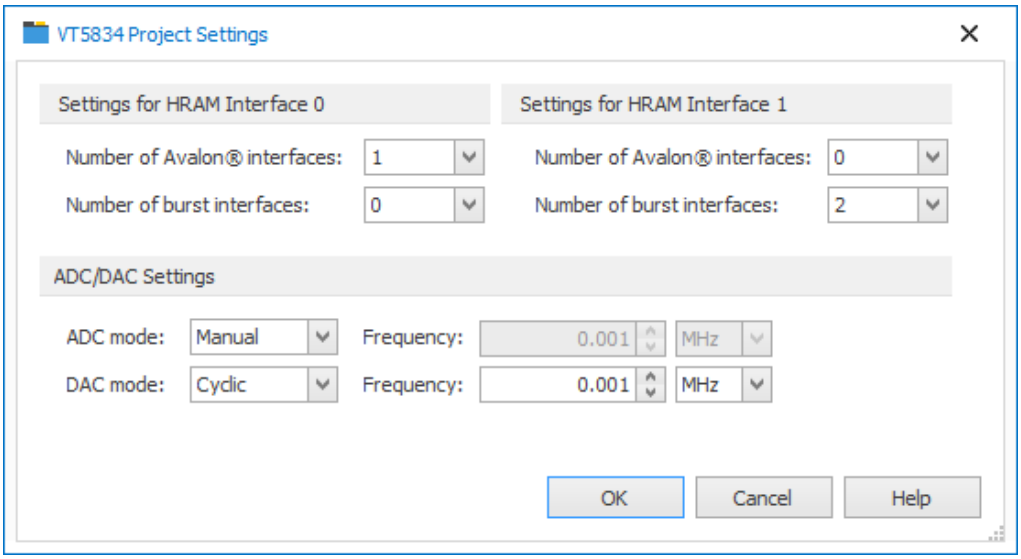
Overview

Some VT System Modules offer specific configuration options. These can be accessed by pressing **Module specific settings**... under **Project Settings**.

3.4.1 VT5838 Specific Settings

Overview

For the VT5838 several module specific settings are available:



HRAM Interfaces

In this section you can configure how many Avalon and burst interfaces you want to use on HRAM interface 0 resp. 1. The VHDL/MATLAB interface will be adjusted automatically depending on these settings.

ADC/DAC

Here you can configure the update behavior of the digital to analog and analog to digital converters. Two modes are available:

- **Manual**: when this mode is selected, the conversion has to be triggered manually. For details on the involved signals see the VT5838 signal description below.
- **Cyclic**: in this mode the conversion will be triggered automatically using the specified cycle time.

3.5 Command Line Interface (CLI)

Overview

The VT System FPGA Manager provides an extensive command line interface. Via this interface you can access all relevant functionality of this tool programmatically.

Getting started
Many of the settings you can make in the GUI of VT System FPGA Manager are also valid for operation in command line mode. Therefore it is recommended to start VT System FPGA Manager in GUI mode first. Make all settings that are important for you and make sure that your projects can be compiled correctly. After that you are good to go with the command line mode.

Parameters
Most parameters can be used in a long or a short form (e.g. “-action” or “-a”). Both will work equally. See the detailed command descriptions below to see which parameters are available.

The order of the parameters does not matter. You can specify them in any order.

In the following chapters all placeholders will be formatted like *this* (blue and italic) for better readability. When executing the commands will you will need to replace these placeholders with meaningful values.

Project names
Many commands require a project name to be specified. For basic projects this is simply the name as it appears in the GUI (which is the same as the name of the project’s folder on the file system). Multi module projects are identified in the same way.

If you want to specify a sub-project of a multi module project use the syntax *NameOfMultiModuleProject*.*NameOfSubProject* (both names are concatenated with a dot in between).

Exit codes
> **0:**
The command was executed successfully.
> **Values other than 0:**
An error occurred. See output for details.

3.5.1 Getting Help

Command
> **Long form:**
VTSFPGAManager.exe -help
> **Short form:**
VTSFPGAManager.exe -h

Parameters
None

Description
This command will print detailed instructions on how to use the command line interface. Also several usage examples will be shown.

Examples
Print the help to the command line:
VTSFPGAManager.exe -h

3.5.2 Importing a Project Into a Workbench

Command
> **Long form:**
VTSFPGAManager.exe -action import -file “*FILE*” [-workbench “*WB*”]
> **Short form:**
VTSFPGAManager.exe -a import -f “*FILE*” [-w “*WB*”]

Parameters
> **-action (-a)**
This parameter specifies the action to be performed. In order to import a project it must be set to “import”.

> **-file (-f)**
This parameter specifies the .zip file to import. It is recommended to wrap the parameter in quotes in order to avoid issues with blank characters.

> **-workbench (-w), optional**
This optional paremeter specifies the workbench directory into which the project shall be imported. If this parameter is not specified, the currently configured workbench of the VT System FPGA Manager instance is used. It is recommended to wrap the parameter in quotes in order to avoid issues with blank characters.

Description

This command is used to import a previously exported project (.zip file) into the VT System FPGA Manager workbench.

Examples

Import "MyProject.zip" into the default workbench:
VTSFPGAManager.exe -a import -f "C:\ProjectArchive\MyProject.zip"

Import "MyProject.zip" into a different workbench:
VTSFPGAManager.exe -a import -f "C:\MyProject.zip" -w "C:\AnotherWorkbench"

3.5.3 Exporting a Project From The Workbench

Command

> **Long form:**
VTSFPGAManager.exe -action export -project *PRJ* -file "*FILE*" [-workbench "*WB*"]

> **Short form:**
VTSFPGAManager.exe -a export -p *PRJ* -f "*FILE*" [-w "*WB*"]

Parameters

> **-action (-a)**
This parameter specifies the action to be performed. In order to export a project it must be set to "export".

> **-project (-p)**
This parameter specifies the name of the project as it appears in the GUI. Quotes surrounding the name are not required, since project names may not contain any problematic characters.

> **-file (-f)**
This parameter specifies the .zip file to export to. It is recommended to wrap the parameter in quotes in order to avoid issues with blank characters. If the file already exists it will be overwritten.

> **-workbench (-w), optional**
This optional paremeter specifies the workbench directory from which the project shall be exported. If this parameter is not specified, the currently configured workbench of the VT System FPGA Manager instance is used. It is recommended to wrap the parameter in quotes in order to avoid issues with blank characters.

Description

This command is used to export a project from the VT System FPGA Manager workbench to a .zip file.

Examples

Export "MyProject" to a .zip file:
VTSFPGAManager.exe -a export -p MyProject -f "C:\ProjectArchive\MyProject.zip"

3.5.4 Deleting a Project From The Workbench

Command
> **Long form:**
VTSFPGAManager.exe -action delete -project *PROJ* [-workbench "*WB*"]
> **Short form:**
VTSFPGAManager.exe -a delete -p *PROJ* [-w "*WB*"]

Parameters
> **-action (-a)**
This parameter specifies the action to be performed. In order to delete a project it must be set to "delete".
> **-project (-p)**
This parameter specifies the name of the project as it appears in the GUI. Quotes surrounding the name are not required, since project names may not contain any problematic characters.
> **-workbench (-w), optional**
This optional paremeter specifies the workbench directory from which the project shall be deleted. If this parameter is not specified, the currently configured workbench of the VT System FPGA Manager instance is used. It is recommended to wrap the parameter in quotes in order to avoid issues with blank characters.

Description
This command is used to delete a project from the VT System FPGA Manager workbench.

Examples
Delete the project "MyProject" from the default workbench:
VTSFPGAManager.exe -a delete -p MyProject

Delete the project "MyProject" from another workbench:
VTSFPGAManager.exe -a delete -p MyProject -w "C:\AnotherWorkbench"

3.5.5 Compiling a Project

Command
> **Long form:**
VTSFPGAManager.exe -action compile-project *PROJ*
> **Short form:**
VTSFPGAManager.exe -a compile -p *PROJ*

Parameters
> **-action (-a)**
This parameter specifies the action to be performed. In order to compile a project it must be set to "compile".
> **-project (-p)**
This parameter specifies the name of the project as it appears in the GUI. Quotes surrounding the name are not required, since project names may not contain any problematic characters.

Description
This command is used to compile a project. The settings (Quartus and Simulink paths) are used as configured in the GUI.

Examples
Compile the project "MyProject":
VTSFPGAManager.exe -a compile -p MyProject

3.5.6 Downloading a Project In Test Mode

Command
> **Long form:**
VTSFPGAManager.exe -action test -project *PROJ* -modules *MOD* [-network *X*]

> **Short form:**
VTSFPGAManager.exe -a test -p *PROJ* -m *MOD* [-n *X*]

>

Parameters

> **-action (-a)**
This parameter specifies the action to be performed. In order to download a project in test mode it must be set to “test”.

> **-project (-p)**
This parameter specifies the name of the project as it appears in the GUI. Quotes surrounding the name are not required, since project names may not contain any problematic characters.

> **-modules (-m)**
Via this parameter you can specify which of the connected modules shall be affected by the operation. You can choose from the following options:

  > **first:** The project is only downloaded to the first matching module.

  > **all:** The project is downloaded to all matching modules.

  > **x:** Zero based index of the module to be downloaded to. Only matching modules are considered.

  > **x,y,z,...:** Comma separated list of zero based module indices to be downloaded to. Only matching modules are considered.

> **-network (-n), optional**
This parameter specifies the index of the network adapter to be used for VT System access. If this parameter is not specified, the network adapter last used in the GUI will be used. To see which network adapters are available, please open the GUI.

Description

This command is used to download a project in test mode to one or more modules. Please note that the project must have been successfully compiled beforehand.

Examples

Download the project “MyProject” to all matching modules:
VTSFPGAManager.exe -a test -p MyProject -m all

Download the project to the first matching module on the first network adapter:
VTSFPGAManager.exe -a test -p MyProject -m first -n 0

Download the project only to the third matching module:
VTSFPGAManager.exe -a test -p MyProject -m 2

3.5.7 Downloading a Project In Persisting Mode

Command

> **Long form:**
VTSFPGAManager.exe -action persist -project *PROJ* -modules *MOD* [-network *X*]

> **Short form:**
VTSFPGAManager.exe -a persist -p *PROJ* -m *MOD* [-n *X*]

Parameters

> **-action (-a)**
This parameter specifies the action to be performed. In order to download a project in persist mode it must be set to “persist”.

> **-project (-p)**
This parameter specifies the name of the project as it appears in the GUI. Quotes surrounding the name are not required, since project names may not contain any problematic characters.

- **-modules (-m)**
  Via this parameter you can specify which of the connected modules shall be affected by the operation. You can choose from the following options:
  - **first:** The project is only downloaded to the first matching module.
  - **all:** The project is downloaded to all matching modules.
  - **x:** Zero based index of the module to be downloaded to. Only matching modules are considered.
  - **x,y,z,...:** Comma separated list of zero based module indices to be downloaded to. Only matching modules are considered.
- **-network (-n), optional**
  This parameter specifies the index of the network adapter to be used for VT System access. If this parameter is not specified, the network adapter last used in the GUI will be used. To see which network adapters are available, please open the GUI.

Description

This command is used to download a project in persisting mode to one or more modules. Please note that the project must have been successfully compiled beforehand.

Examples

Download the project to the first matching module:
VTSFPGAManager.exe -a persist -p MyProject -m first

Download the project to all matching modules on the first network adapter:
VTSFPGAManager.exe -a persist -p MyProject -m all -n 0

Download the project to the first three matching modules:
VTSFPGAManager.exe -a persist -p MyProject -m 0,1,2

3.5.8 Clearing a User FPGA

Command

- **Long form:**
  VTSFPGAManager.exe -action clear -modules *MODS* [-network *X*]
- **Short form:**
  VTSFPGAManager.exe -a clear -m *MODS* [-n *X*]

Parameters

- **-action (-a)**
  This parameter specifies the action to be performed. In order to clear the User FPGA of a connected VT System Module this must be set to “clear”.
- **-modules (-m)**
  Via this parameter you can specify which of the connected modules shall be affected by the operation. You can choose from the following options:
  - **first:** Only the first module is cleared.
  - **all:** All modules are cleared.
  - **x:** Zero based index of the module to be cleared.
  - **x,y,z,...:** Comma separated list of zero based module indices to be cleared.
- **-network (-n), optional**
  This parameter specifies the index of the network adapter to be used for VT System access. If this parameter is not specified, the network adapter last used in the GUI will be used. To see which network adapters are available, please open the GUI.

Description

This command is used to clear the User FPGA of one or more connected VT System modules.

Examples

Clear the User FPGAs of all connected modules:
VTSFPGAManager.exe -a clear -m all

Clear only the second connected module on a specific network adapter:
VTSFPGAManager.exe -a clear -m 1 -n 3

4 Writing HDL Code for VT System FPGA Modules

This chapter contains the following information:

4.1	Working with the FPGA Manager	page 36
4.2	Coding with VHDL	page 36
4.3	Folder Structure	page 36
4.4	Template File	page 36
4.5	Adding/Removing/Changing System Variables	page 39
4.6	Using Quartus®	page 39

4.1 Working with the FPGA Manager

FPGA Manager

If you want to write your VHDL code manually you must chose this workflow in the VT System FPGA Manager when you create a new project. In the source folder which is directly accessible from the VT System FPGA Manager a VHDL file with the top level entity will be created. Starting from this top level entity you can write your VHDL code with the editor of your choice. After VHDL coding is finished the project can be compiled and downloaded onto the User FPGA with the VT System FPGA Manager.

4.2 Coding with VHDL

Coding with VHDL

If you want to write your own VHDL files please remember to follow proper coding guidelines to assure that your code is working correctly with the used hardware. We recommend reading the Quartus® **handbook** by Intel® before writing your own code. This concerns especially the following chapters: **Recommended Design Practices** and **Recommended HDL Coding Styles**.

4.3 Folder Structure

Working folders

The work folder of your project consists of different subfolders.
These are:

- **bin**
  This folder contains the file for programming the FPGA and does therefore not need to be changed.
- **hdl**
  This folder contains the VHDL file created by the VT System FPGA Manager. If you create additional VHDL files for your project they should also be stored in this folder to guarantee that the VT System FPGA Manager can find them.
- **quartus**
  This folder folder contains the Quartus® II files which are created for your project. These files are usually used in the background by the VT System FPGA Manager without opening Quartus®. This folder contains the subfolders **hdl** and **syn**.
  - **hdl**
    This folder contains basic hdl files which should not be changed in any way.
  - **syn**
    This folder contains the Quartus® files of your project.

The **.qpf** file is the corresponding Quartus® project file.

4.4 Template File

VHDL template

By creating a new project with the VHDL workflow the VT System FPGA Manager creates a VHDL template file. This will be stored in the subfolder:

...\hdl and must remain in this folder for the VT System FPGA Manager to use it properly. The template file contains the basic structure needed for a correct usage of the User FPGA.

Example file

```
-- ExampleProject_GN VHDL code file

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_STD.all;
use IEEE.MATH_REAL.all;


ENTITY ExampleProject_GN IS
  PORT (

    -- @CMD=SYSVARSTART
    Sysvar1_ch0 : out std_logic_vector(31 downto 0) := (others => '0');
    Sysvar2_ch0 : in std_logic_vector(31 downto 0);
    -- @CMD=SYSVAREND

    -- @CMD=TIMESTAMPSTART
    -- @CMD=TIMESTAMPEND

    clk         : IN  std_logic;                                    -- Clock.clk
    areset      : IN  std_logic;                                    -- .reset
    h_areset    : IN  std_logic;

    in_adc_1    : IN  std_logic_vector(31 DOWNTO 0);                -- in_adc_1.wire
    in_adc_2    : IN  std_logic_vector(31 DOWNTO 0);                -- in_adc_2.wire
    in_adc_3    : IN  std_logic_vector(31 DOWNTO 0);                -- in_adc_3.wire
    in_adc_4    : IN  std_logic_vector(31 DOWNTO 0);                -- in_adc_4.wire
    in_adc_5    : IN  std_logic_vector(31 DOWNTO 0);                -- in_adc_5.wire
    in_adc_6    : IN  std_logic_vector(31 DOWNTO 0);                -- in_adc_6.wire
    in_adc_7    : IN  std_logic_vector(31 DOWNTO 0);                -- in_adc_7.wire
    in_adc_8    : IN  std_logic_vector(31 DOWNTO 0);                -- in_adc_8.wire
    in_adc_9    : IN  std_logic_vector(31 DOWNTO 0);                -- in_adc_9.wire
    in_adc_10   : IN  std_logic_vector(31 DOWNTO 0);                -- in_adc_10.wire
    in_adc_11   : IN  std_logic_vector(31 DOWNTO 0);                -- in_adc_11.wire
    in_adc_12   : IN  std_logic_vector(31 DOWNTO 0);                -- in_adc_12.wire

    out_dac_1   : OUT std_logic_vector(31 DOWNTO 0) := (OTHERS => '0');  -- out_dac_1.wire
    out_dac_2   : OUT std_logic_vector(31 DOWNTO 0) := (OTHERS => '0');  -- out_dac_2.wire
    out_dac_4   : OUT std_logic_vector(31 DOWNTO 0) := (OTHERS => '0');  -- out_dac_4.wire
    out_dac_3   : OUT std_logic_vector(31 DOWNTO 0) := (OTHERS => '0');  -- out_dac_3.wire

    in_adc_update   : IN  std_logic_vector(12 DOWNTO 1);
    in_invar_update : IN  std_logic;
    in_timestamp    : IN  std_logic_vector(63 DOWNTO 0);

    in_ibc_1   : IN std_logic_vector(0 DOWNTO 0) := (OTHERS => '0');  -- in_ibc_1.wire
    in_ibc_2   : IN std_logic_vector(0 DOWNTO 0) := (OTHERS => '0');  -- in_ibc_2.wire
    in_ibc_3   : IN std_logic_vector(0 DOWNTO 0) := (OTHERS => '0');  -- in_ibc_3.wire
    in_ibc_4   : IN std_logic_vector(0 DOWNTO 0) := (OTHERS => '0');  -- in_ibc_4.wire

    out_ibc_1   : OUT std_logic_vector(0 DOWNTO 0) := (OTHERS => '0');  -- out_ibc_1.wire
    out_ibc_2   : OUT std_logic_vector(0 DOWNTO 0) := (OTHERS => '0');  -- out_ibc_2.wire
    out_ibc_3   : OUT std_logic_vector(0 DOWNTO 0) := (OTHERS => '0');  -- out_ibc_3.wire
    out_ibc_4   : OUT std_logic_vector(0 DOWNTO 0) := (OTHERS => '0');  -- out_ibc_4.wire

    debug_LED_1 : OUT std_logic_vector(0 DOWNTO 0)  := (OTHERS => '0');  -- debug_LED_1.wire
    debug_LED_2 : OUT std_logic_vector(0 DOWNTO 0)  := (OTHERS => '0');  -- debug_LED_2.wire
    debug_LED_3 : OUT std_logic_vector(0 DOWNTO 0)  := (OTHERS => '0');  -- debug_LED_3.wire
    debug_LED_4 : OUT std_logic_vector(0 DOWNTO 0)  := (OTHERS => '0');  -- debug_LED_4.wire
    debug_LED_5 : OUT std_logic_vector(0 DOWNTO 0)  := (OTHERS => '0')   -- debug_LED_5.wire
    );
END;

ARCHITECTURE rtl OF ExampleProject_GN IS


BEGIN


END ARCHITECTURE rtl;  -- of  ExampleProject_GN
```

The picture above shows the VHDL template file for the VT2816 with two system variables (**Sysvar1_ch0** & **Sysvar2_ch0**) created according to the settings in the VT System FPGA Manager. The other input and output signals will be created automatically by the VT System FPGA Manager and depend on the used VT System module. Manually created system variables will always be restored to the settings in the VT System FPGA Manager when the project will be saved the next time. So it is not recommended to change them directly in the

VHDL file. This should also be considered for comments behind the signals. So additional comments should be placed outside the entity declaration. All output signals will be initialized with zeros so that they have a predetermined output value in case they aren't used in the project.

The VT System FPGA Manager adds suffixes to the name of the project and also to the names of the created system variables. The name of the entity always consists of your chosen name plus the suffix **_GN**. System variables always get a suffix based on the channel they are declared for. This means if you create a system variable and chose channel 1 for it the suffix will be **_ch1**. The suffix **_ch0** is used if the signal is mapped to the module instead of a single channel. The direction and the number of bits for the system variables are adjusted at the FPGA Manager. Please remember that variables with only one bit are not declared as **std_logic** but as **std_logic_vector(0 downto 0)**.

The other signals (including clock and reset signal) are a connection to the I/O hardware and will always be created automatically by the VT System FPGA Manager. These signals are dependent from the used VT System module and are described in the chapter Signal Reference.

The architecture of the top level entity is empty. Here is the entry to start writing VHDL code for the User FPGA.

4.5 Adding/Removing/Changing System Variables

Working with system variables

You can add, remove or change the settings of the system variables from your VHDL file by using the VT System FPGA Manager. In all cases you have to save your project afterwards in the VT System FPGA Manager to adapt these changes to the VHDL file. Manually editing the system variables in the VHDL file directly is not recommended as the complete top level entity will overwritten with the current settings in the VT System FPGA Manager by the next save of the project.

4.6 Using Quartus®

Quartus®

It is not intended and therefore not recommended to open Quartus®, because Quartus® is controlled completely by the VT system FPGA Manager. But if it is necessary to work with Quartus® directly you have to compile your project before downloading to the FPGA always with the VT system FPGA Manager, because the VT system FPGA Manager creates the final file which is needed to program the FPGA. Also downloading to the User FPGA is only possible with the VT system FPGA Manager. The basic VHDL code of the User FPGA is stored encrypted and not accessible. So using some functions like the RTL Viewer is not possible. In this case we recommend therefore creating a separate Quartus® project and including the code you want to analyze there.

5 Modeling the Behavior of VT System FPGA Modules (DSP Builder)

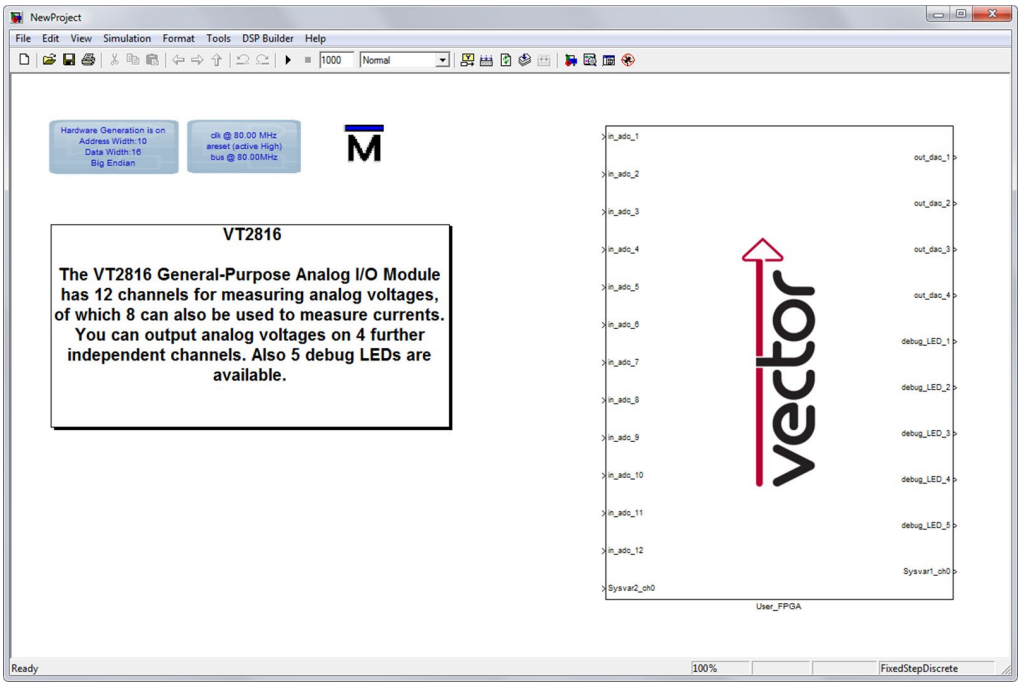
This chapter contains the following information:

5.1	Using the Model File Template	page 41
5.2	Simulation	page 44
5.3	Compiling and Download to FPGA	page 44
5.4	Adding/Removing/Changing System Variables	page 45

5.1 Using the Model File Template

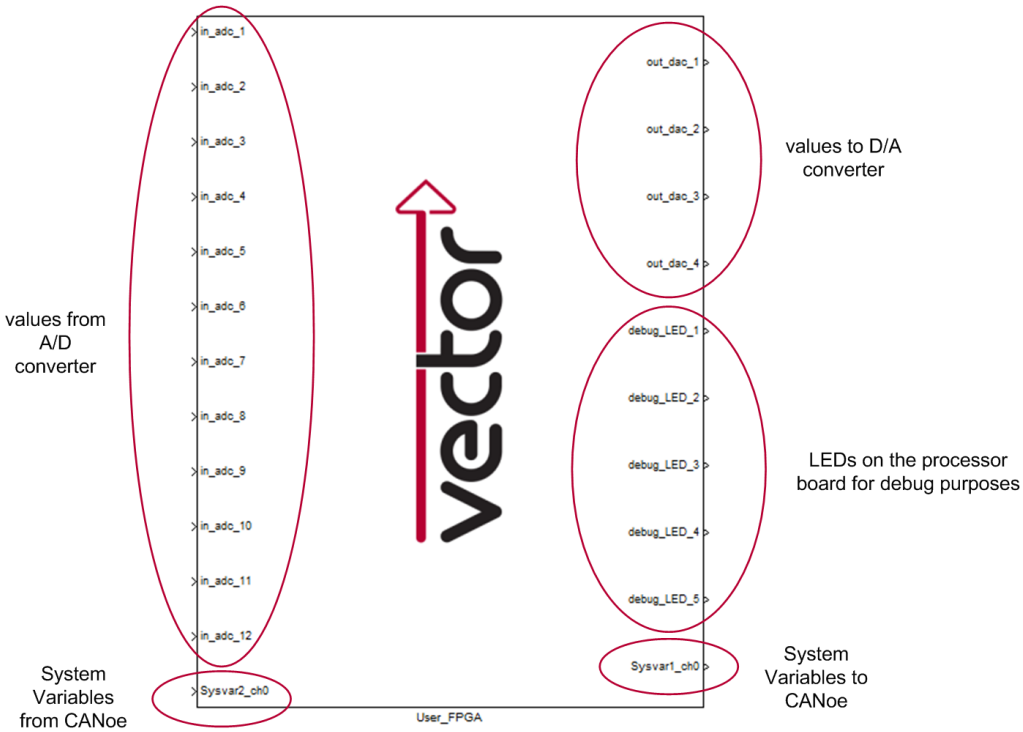
Simulink® model file

While creating a new FPGA project and choosing the Simulink® workflow, the VT System FPGA Manager will generate a Simulink® model file (.mdl). You have to open the Simulink® model file always with the VT system FPGA Manager and not directly with Simulink®. Otherwise the Intel® DSP Builder is not included correctly. After opening this file with the VT system FPGA Manager, dependent on the chosen VT System module the template appears:

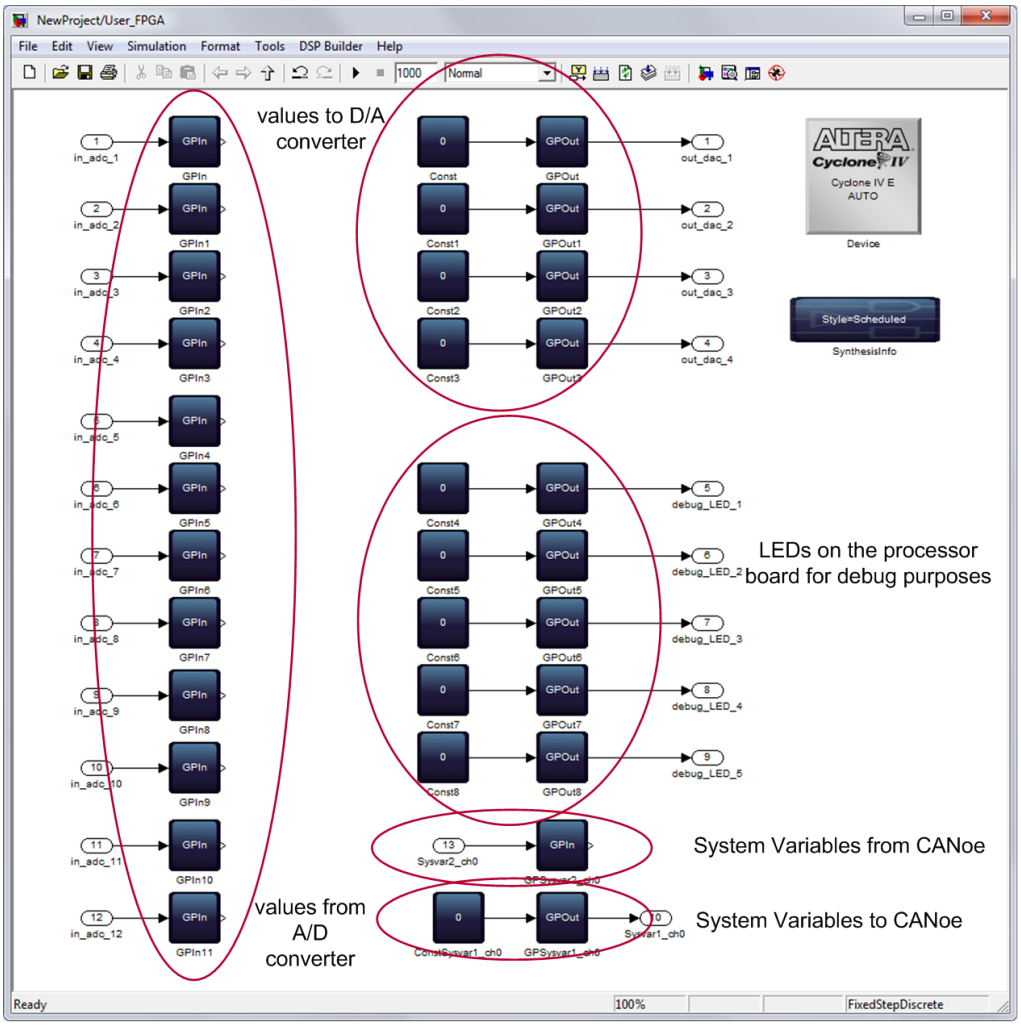


On the top left corner there are two blocks to adjust some basic settings like the used clock frequency or reset signal. These settings will be done by the VT System FPGA Manager and therefore it is not necessary to set these parameters manually.

The block on the right side includes the Simulink® model of the User FPGA. At this top level block all the available signals are listed. The following figure shows a detailed view of the top level block for the VT2816.



By double-clicking on this block the next hierarchy level appears.



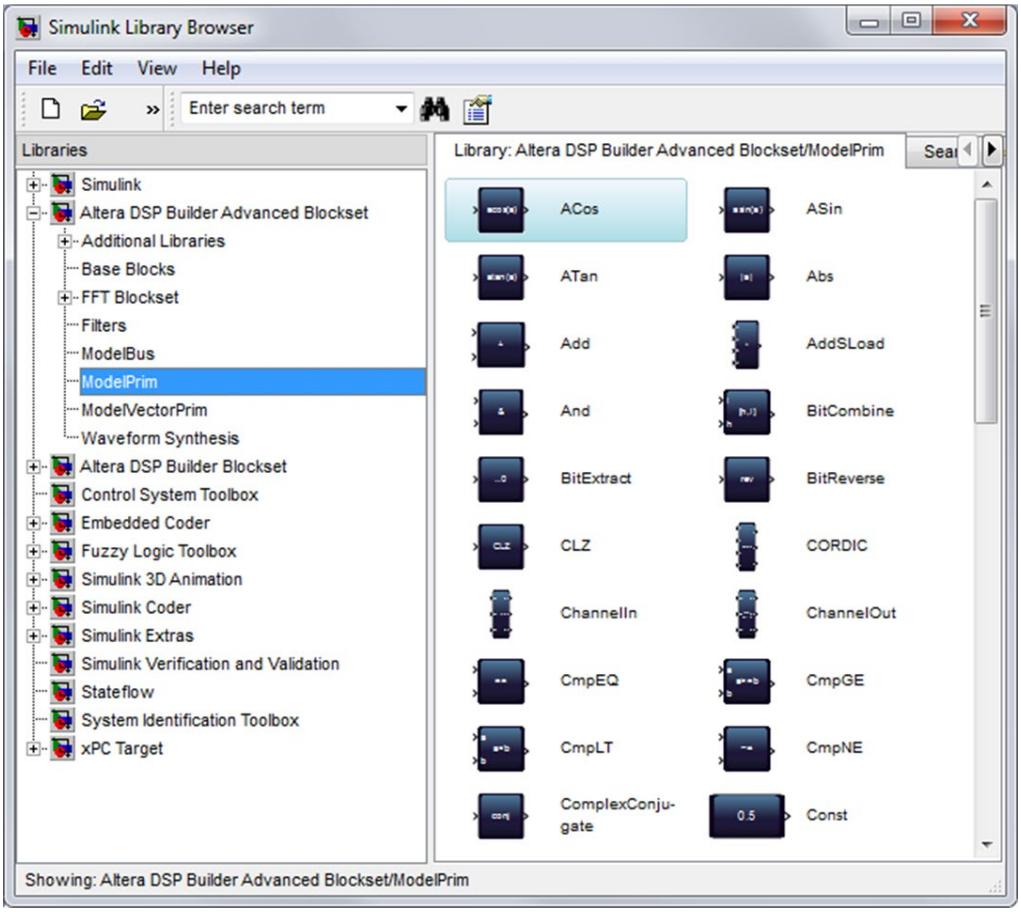
There are mainly two signals types available. System variables for communication with CANoe which are defined with the VT System FPGA Manager will automatically appear in the User FPGA model template. Signals for accessing the I/O hardware will appear also automatically dependent on which VT System module was chosen. In this figure the VT2816 is shown.

Basic signals like clock and reset are wired automatically and therefore have not to be considered in the User FPGA model. For debug purposes there are five LEDs on the processor board. All output signals are initial set to zero. You can find a detailed description of the signals in the chapter Signal Reference.

Intel® DSP Builder Advanced Blockset

At this level you can use blocks from the Intel® DSP Builder Advanced Blockset to create a model. The connection to the in- or output signals has to be done with the special in- and output blocks GPin and GPout which are also included in the Intel® DSP Builder Advanced Blockset and automatically placed by the template. If Intel® DSP Builder is installed correctly the Intel® DSP Builder Advanced Blockset is available in the Simulink® Library Browser which can be opened under **View|Library Browser**:

Simulink® Library Browser



There is also another block set available, the Intel® DSP Builder Blockset. As Intel® does not recommend using this block set for new developments, this block set is not supported by the VT System FPGA Manager.

For more detailed information about the Intel® DSP Builder and Simulink® it is recommended to read the documentation at www.intel.com and www.mathworks.com.

5.2 Simulation

After modeling the User FPGA with the Intel® DSP Builder Advanced Blockset the model can be simulated cycle accurate in Simulink® like normal Simulink® models by setting up the simulation and then pressing the start simulation button. Therefore also the normal Simulink® block sets can be used on the top level of the User FPGA model to create a test bench. Simulink® blocks can be used for simulation but they will not be translated to VHDL code.

5.3 Compiling and Download to FPGA

When pressing the start simulation button the corresponding VHDL Code for the User FPGA model is automatically generated and put in the folder where also the manually written VHDL files are stored. After code generation the compiling

of the project and downloading to the FPGA will be done with the VT System FPGA Manager again.

5.4 Adding/Removing/Changing System Variables

Working with system variables

When system variables have to be added, removed or changed this has always to be done with the VT System FPGA Manager and not directly in Simulink®. The modifications do not only affect the User FPGA model, but also some settings in the firmware and CANoe. So the model file has to be closed and the modifications have to be done with the VT System FPGA Manager. After saving the VT System FPGA Manager the changes will be adapted to the .mdl file and the User FPGA model can be opened again. If the system variables will be changed in the Simulink® model directly, the changes will be overwritten with the system variables settings in the VT System FPGA Manager when the model file is opened again after saving the project in the VT System FPGA Manager.

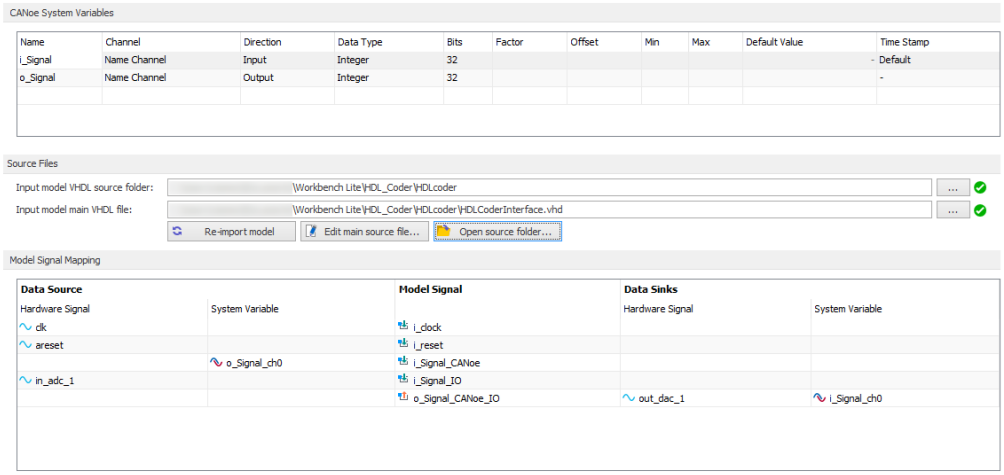
5.5 Integrating HDL Generating Simulink models

FPGA Manager

If you want to integrate existing HDL code generated using a Simulink® model the 'Model in Simscape, generate code with HDL Coder' option can be used. It is selected as a workflow when creating a new project. It allows the use of a GUI in the FPGA manager to connect signals in the VHDL code of the project without the need to connect them inside the code itself manually.

Source Files

For the FPGA manager to find and add the interface signals of the Simulink® model the top-level VHDL file of this model and its source folder must be pasted into the appropriate places of the FPGA manager (as seen in the picture below). In addition, the model folder must not be in the HDL folder of the user FPGA project. This is because the FPGA manager creates a copy of the model VHDL files and puts them into the HDL folder. If the interface of the Simulink® model changed the copies can be updated by pressing the 'Re-import model' button in the FPGA manager.



Model Signal Mapping

To connect the signals of the model using the FPGA manager GUI the 'Model Signal Mapping' table is used.

'Data Source' lists all source signals available in the project, divided into signals from the module itself (Hardware Signal) and CANoe signals created in the table above (System Variable).

'Data Sinks' lists all signals that can be driven by the model signals. They are also divided into signals from the module itself (Hardware Signal) and CANoe signals created in the table above (System Variable).

The 'Model Signal' column lists all interface signals from the main VHDL file of the Simulink® model. Only one source signal can be connected to a model signal. A model signal can be connected to multiple sink signals. Not all model signals need to be connected using the GUI. They can also be connected manually in the VHDL code. (See 'Hybrid user FPGA projects'.)

Hybrid User FPGA Projects

This option in the user FPGA manager can also be used to create hybrid projects that incorporate both, manual written VHDL code and a Simulink® model. This can be achieved by not connecting all ingoing model signals to a source signal in the GUI. Instead a signal can be manually connected in the VHDL code. For this reason, a record is created in the architecture. The signal allocation process and the port map must not be modified. Instead create another process (if necessary) and connect manually created signals to the appropriate signals of the record. Outgoing signals of the model can be connected using the same method, although they can be connected in both, the manually written code and the manager GUI.

The example below shows the Simulink® model interface and the architecture of the user FPGA project.

Model Entity

```
ENTITY HDLCoderInterface IS
  PORT (
    i_clock           : IN  std_logic;
    i_reset           : IN  std_logic;
    i_Signal_CANoe    : IN  std_logic_vector(31 DOWNTO 0);
    i_Signal_IO       : IN  std_logic_vector(31 DOWNTO 0);
    i_Signal_Manually : IN  std_logic_vector(31 DOWNTO 0);
    o_Signal_CANoe_IO : OUT std_logic_vector(31 DOWNTO 0) := (OTHERS => '0');
    o_Signal_Manually : OUT std_logic_vector(31 DOWNTO 0) := (OTHERS => '0')
    );
END HDLCoderInterface;
```

*_GN Architecture

```
ARCHITECTURE rtl OF User IS

  -- manually added signals
  SIGNAL s_Manually_1 : std_logic_vector(31 DOWNTO 0) := (OTHERS => '0');
  SIGNAL s_Manually_2 : std_logic_vector(31 DOWNTO 0) := (OTHERS => '0');

  --

  -- @CMD=MODELRECORDSTART
  TYPE t_HDLCoderInterfaceSignals IS RECORD
    i_clock           : std_logic;
    i_reset           : std_logic;
    i_Signal_CANoe    : std_logic_vector(31 DOWNTO 0);
    i_Signal_IO       : std_logic_vector(31 DOWNTO 0);
    o_Signal_CANoe_IO : std_logic_vector(31 DOWNTO 0);
    i_Signal_Manually : std_logic_vector(31 DOWNTO 0);
    o_Signal_Manually : std_logic_vector(31 DOWNTO 0);
  END RECORD;
--
  SIGNAL s_HDLCoderInterfaceSignals : t_HDLCoderInterfaceSignals := (
    i_clock           => '0',
    i_reset           => '0',
    i_Signal_CANoe    => (OTHERS => '0'),
    i_Signal_IO       => (OTHERS => '0'),
    o_Signal_CANoe_IO => (OTHERS => '0'),
    i_Signal_Manually => (OTHERS => '0'),
    o_Signal_Manually => (OTHERS => '0')
    );
  -- @CMD=MODELRECORDEND

BEGIN

  proc_ManuallyConnected : PROCESS (ALL)
  BEGIN  -- PROCESS proc_ManuallyConnected
    i_Signal_Manually <= s_Manually_1;
    s_Manually_2      <= o_Signal_Manually;
  END PROCESS proc_ManuallyConnected;
```

```
  --

  -- @CMD=MODELMAPSTART
  proc_SignalAllocation : PROCESS (ALL)
  BEGIN  -- PROCESS proc_SignalAllocation
    s_HDLCoderInterfaceSignals.i_clock        <= clk;
    s_HDLCoderInterfaceSignals.i_reset        <= areset;
    s_HDLCoderInterfaceSignals.i_Signal_CANoe <= o_Signal_ch0;
    s_HDLCoderInterfaceSignals.i_Signal_IO    <= in_adc_1;
    i_Signal_ch0                              <= s_HDLCoderInterfaceSignals.o_Signal_CANoe_IO;
    out_dac_1                                 <= s_HDLCoderInterfaceSignals.o_Signal_CANoe_IO;
  END PROCESS proc_SignalAllocation;
--
  inst_HDLCoderInterface_1 : ENTITY work.HDLCoderInterface
    PORT MAP (
      i_clock           => s_HDLCoderInterfaceSignals.i_clock,
      i_reset           => s_HDLCoderInterfaceSignals.i_reset,
      i_Signal_CANoe    => s_HDLCoderInterfaceSignals.i_Signal_CANoe,
      i_Signal_IO       => s_HDLCoderInterfaceSignals.i_Signal_IO,
      o_Signal_CANoe_IO => s_HDLCoderInterfaceSignals.o_Signal_CANoe_IO,
      i_Signal_Manually => s_HDLCoderInterfaceSignals.i_Signal_Manually,
      o_Signal_Manually => s_HDLCoderInterfaceSignals.o_Signal_Manually
      );
  -- @CMD=MODELMAPEND

END ARCHITECTURE rtl;  -- of  User
```

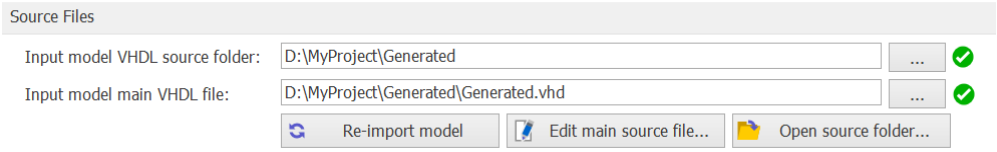
6 Modeling the Behavior of VT System FPGA Modules (HDL Coder)

This chapter contains the following information:

6.1	Integrating The Generated Model Code	page 49
6.2	Mapping The Model Signals	page 49
6.3	Compiling and Download to FPGA	page 49
6.4	Adding/Removing/Changing System Variables	page 49

6.1 Integrating The Generated Model Code

Once you have generated the HDL code from your model with HDL Coder® you can integrate it into your User FPGA project. Simply choose the main folder where the code is located as well as the main HDL code file:



In case you have made changes to your model, you can always re-import the generated code. To do so Click on 'Re-import model'. This will automatically update your User FPGA project where necessary.

6.2 Mapping The Model Signals

After importing the generated model code into your User FPGA project you can set up the model signal mapping.

Data Source

Each model signal can be mapped to one data source, which can either be a hardware signal or a user defined system variable.

Data Sinks

Additionally each model signal can be mapped to multiple data sinks, which can also be a hardware signal and/or a user defined system variable.

The required HDL code for these mappings will automatically be generated by the VT System FPGA Manager when saving or compiling your project.

6.3 Compiling and Download to FPGA

When pressing the start simulation button the corresponding VHDL Code for the User FPGA model is automatically generated and put in the folder where also the manually written VHDL files are stored. After code generation the compiling of the project and downloading to the FPGA will be done with the VT System FPGA Manager again.

6.4 Adding/Removing/Changing System Variables

Working with system variables

When system variables have to be added, removed or changed this has always to be done with the VT System FPGA Manager and not directly in Simulink®. The modifications do not only affect the User FPGA model, but also some settings in the firmware and CANoe. So the model file has to be closed and the modifications have to be done with the VT System FPGA Manager. After saving the VT System FPGA Manager the changes will be adapted to the .mdl file and the User FPGA model can be opened again. If the system variables will be changed in the Simulink® model directly, the changes will be overwritten with the system variables settings in the VT System FPGA Manager when the model file is opened again after saving the project in the VT System FPGA Manager.

7 Inter Board Communication (IBC)

This chapter contains the following information:

7.1	General Concept	page 51
7.2	Configuration in the VT System FPGA Manager	page 51
7.3	Hardware Connections	page 52
7.4	Performance Considerations	page 54
7.5	Inter Board Communication Plus (IBC+) Configuration in the FPGA Manager Performance Considerations	page 55

7.1 General Concept

General concept

More sophisticated User FPGA projects may involve multiple VT System modules. In these scenarios it might be required to synchronize these modules and exchange data between them in a timely manner. This is where the **inter board communication** feature (or short **IBC**) comes in. It allows you to establish a very fast communication bus between the VT modules in your project.

From a hardware based point of view the inter board communication is realized as a ring bus. Thus each participating VT module is connected via two cables (input and output) to its neighbours. One of the participating modules acts as a master and controls the timing of the data transmission on all modules.

The data exchanged between the modules is organized as 32bit signals. Each signal is written by exactly one of the participating modules. All other modules my read the signal, if required. You can use up to 256 signals in your User FPGA project.

7.2 Configuration in the VT System FPGA Manager

Prerequisites

In order to use the inter board communication feature, you need to create a multi module project with at least two sub-projects. See chapter 3.3.2 for details.

Enabling IBC

To enable the inter board communication for a multi module project, enable the check box **Enable inter board communication** on the project's configuration page.

Setting the IBC master

One of the sub-projects must act as the communication master to control the data flow between the modules. To select the master, use the drop down list **Communication master** on the project's configuration page.

Defining IBC signals

In the table **Signals** on the project's configuration page you can define the signals that are exchanged between all participating VT modules. For each signal you can change the following settings:

> **Name:**
The name of the signal.

> **Writer:**
Specifies which of the particpating VT modules will write this signal.

> **Readers:**
Here you can specify which of the particpating VT modules will have read access to this signal.

Using IBC signals in HDL code

Configured IBC signals will appear in the user HDL code file once the multi module project has been saved. The signal's name will be extended by the postfix "_in_ibc" resp. "_out_ibc" according to whether the signal is read or written by the current VT module's project.

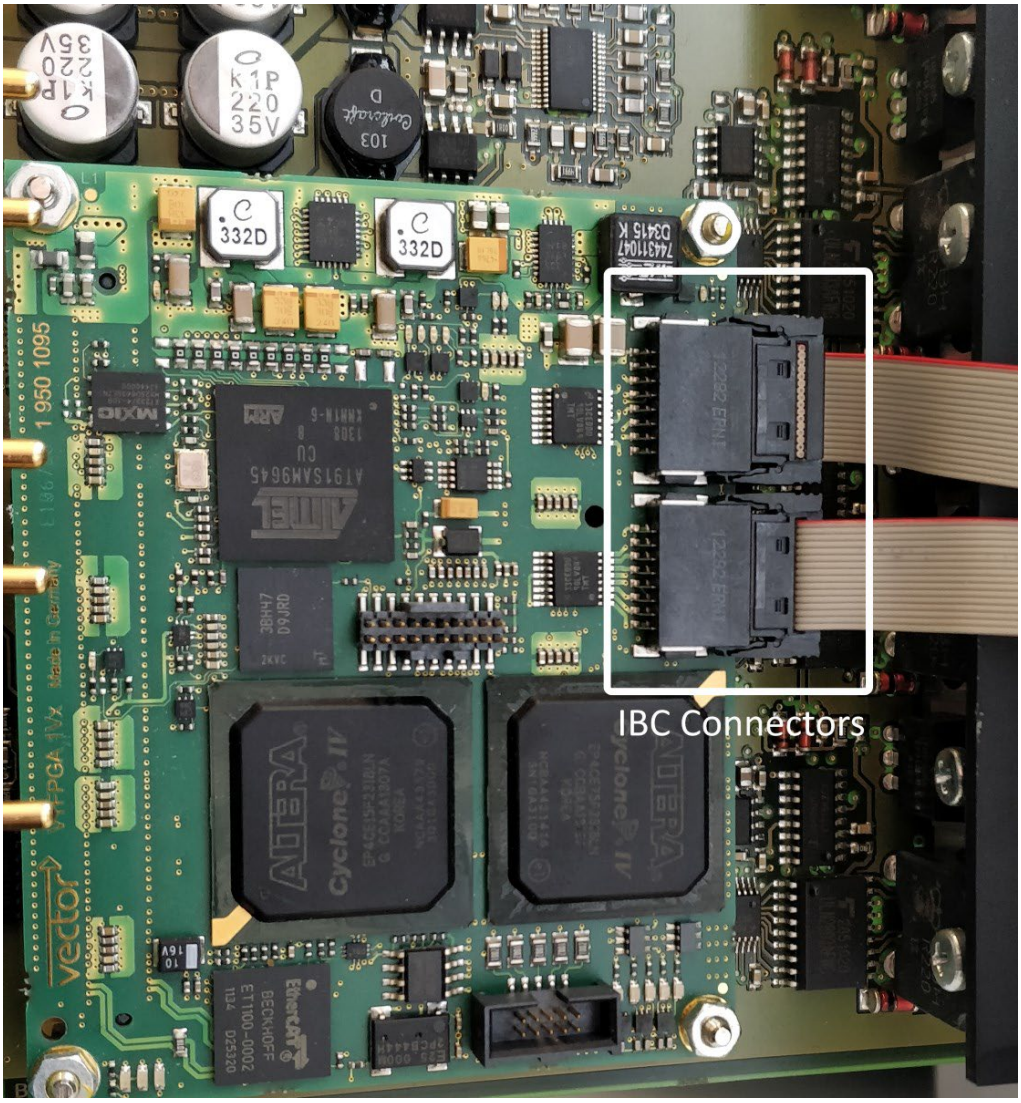
Using IBC signals in MATLAB Simulink®

Configured IBC signals will appear in the Simulink® model file once the multi module project has been saved. The signal's name will be extended by the postfix "_in_ibc" resp. "_out_ibc" according to whether the signal is read or written by the current VT module's project.

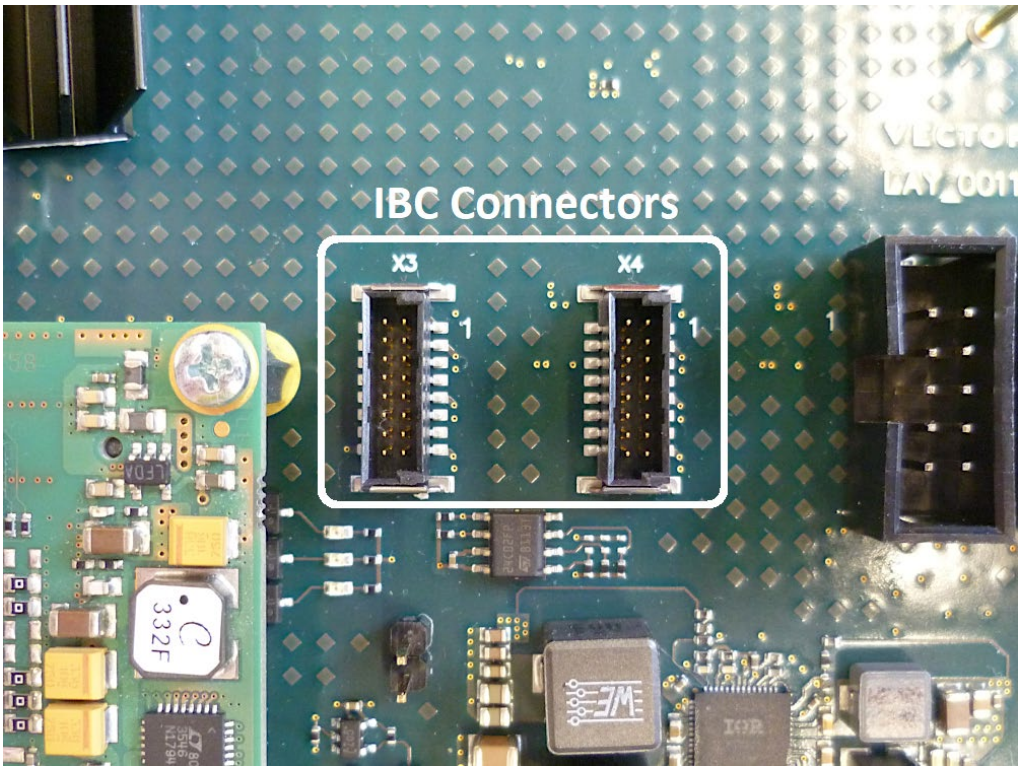
7.3 Hardware Connections

Hardware connections

The two IBC connectors are located on the right hand side of the VT logic board for most of the VT modules.

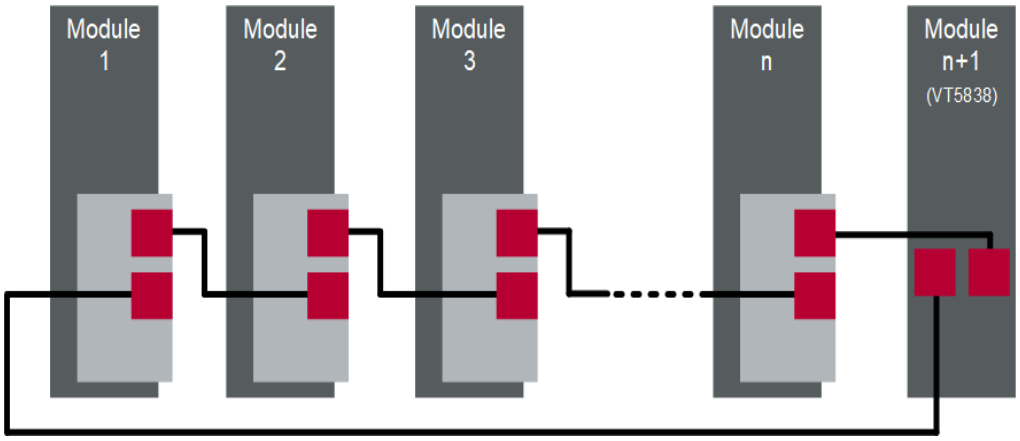


For the VT5838 they are located on the base board instead of the logic board.



The cables need to be installed crosswise. Thus you need to connect each module's top IBC connector to the next module's bottom connector.

From the last module you then go back to the first module in order to close the communication ring.



7.4 Performance Considerations

Single Writer

The most common scenario when using inter board communication is having exactly one module writing signals and n other modules reading these signals. In this scenario it takes up to 1.6µs to transmit a single signal from one module to the next.

In order to estimate the time it takes to transmit multiple signals from the writer to all readers on the bus the following formula can be used:

[transmission time] ≤ [number of signals] × [number of modules] × 1.6µs

Multiple Writers

When allowing multiple modules on the bus to write signals, a transmission time estimation is not possible anymore. This is due to the fact that any signal on the bus might get delayed when another writer starts to send a signal with a higher priority.

Thus in the best case scenario the transmission time for multiple writers is the same as for the single witer case described above. But the worst case transmission time may become much higher, if the different writers interfere with each other.

Recommendations

> The maximum frequency for writing output data should be 1 MHz (one update every microsecond). Faster update rates may result in an overflow of the output buffer.

> You can use the variable in_ibc_outvar_busy to check the status of the output buffer. If the variable is 1, the output buffer is full and no more data should be sent.

7.5 Inter Board Communication Plus (IBC+)

General Concept

This option is only supported by the VT5838.

More sophisticated VT5838 projects may involve 2 modules. In these scenarios it might be required to synchronize these modules and exchange data between them in a timely manner. This is where the **inter board communication plus** feature (or short **IBC+**) comes in. It allows you to establish a very fast communication bus between the VT modules in your project. Compared to the normal IBC connection it allows für higher speed and parallel data transfers.

From a hardware-based point of view the inter board communication plus is realized as a direct connection between 2 modules. This connection uses a different cable from the normal IBC.

The data exchanged between the modules is organized as 32-bit signals. Each signal is written by exactly one of the participating modules.

The IBC+ uses a connector on the front of the module. (see chapter 7.3, JTAG for a picture)

7.5.1 Configuration in the FPGA Manager

Prerequisites

To use the inter board communication feature, you need to create a multi module project with two VT5838 sub-projects. See chapter 3.3.2 for details.

Enabling IBC+

To enable the IBC+ for a multi module project, enable the check box **Enable inter board communication plus** on the project's configuration page.

System ID & Interface Number

They need to be the same for both modules. This can be used to differentiate between different VT5838 projects using the IBC+.

Defining IBC+ signals

In the table **Signals** on the project's configuration page, you can define the signals that are exchanged between all participating VT modules. For each signal you can change the following settings:

> **Name:**
> The name of the signal.

> **Direction:**
> Specifies which of the particpating VT modules will write this signal.

Up to four lanes can be used to transmit the data parallel.

Using IBC+ signals in HDL code

Configured IBC+ signals will appear in the user HDL code file once the multi module project has been saved. The signal's name will be extended by the postfix "in_ibc_plus" resp. "out_ibc_plus" according to whether the signal is read or written by the current VT module's project.

Using IBC+ signals in MATLAB Simulink®

Configured IBC+ signals will appear in the Simulink® model file once the multi module project has been saved. The signal's name will be extended by the postfix "in_ibc_plus" resp. "out_ibc_plus" according to whether the signal is read or written by the current VT module's project.

7.5.2 Performance Considerations

Transmission time per lane	[transmission time] ≥ 500 ns
Recommendations	> The maximum frequency for writing output data should be 2 MHz. Faster update rates may result in data loss. > Multiple lanes can be used in parallel for higher average transfer. > You can use the variable in_ibc_plus_lane_block_tx_ready to check the status of the output buffer. If the bit for the lane is 1, the output buffer is full and no more data should be sent.

8 Signal Reference

In this chapter you find the following information:

8.1	Clock and Reset Signal	page 58
8.2	Additional Signals	page 58
8.3	Debugging	page 58
8.4	Time Stamps	page 62
8.5	VT1004A	page 62
8.6	VT1104	page 64
8.7	VT2004A	page 64
8.8	VT2516A	page 65
8.9	VT2710	page 66
8.10	VT2808	page 67
8.11	VT2816	page 68
8.12	VT2848	page 68
8.13	VT5838 Hardware Interface	page 69
8.14	VT7900A	page 79

8.1 Clock and Reset Signal

Signals

The clock and the reset signal (**clk** and **areset**) are only available for VHDL coding. For the modeling with Simulink® the clock and reset signal are used automatically. The clock frequency can set to 10, 40 or 80 MHz in the VT System FPGA Manager; the reset is always high active and can't be changed.

8.2 Additional Signals

The VT2004A and VT2816 FPGA modules don't support the following signals.

ADC update signal

This signal exists only for modules that are able to measure analog signals. The ADC update signal (**in_adc_update; in_adc_data_valid in case of the VT5838**) is set to high for one clock cycle when new values from the ADC are received. This allows to recognize a new value even if it is identical to the previous value.

Variable update

The variable update signal (**in_invar_update**) is set to high for one clock cycle when new signals (excluding the ADC update signals) are received. This includes signals send from CANoe.

Timestamp

The timestamp signal (**in_timestamp**) is a 64bit value that shows the current time of the module. This value can be send as a timestamp to CANoe and is explained in the chapter „Time Stamps".

The following signals are only supported by the VT5838.

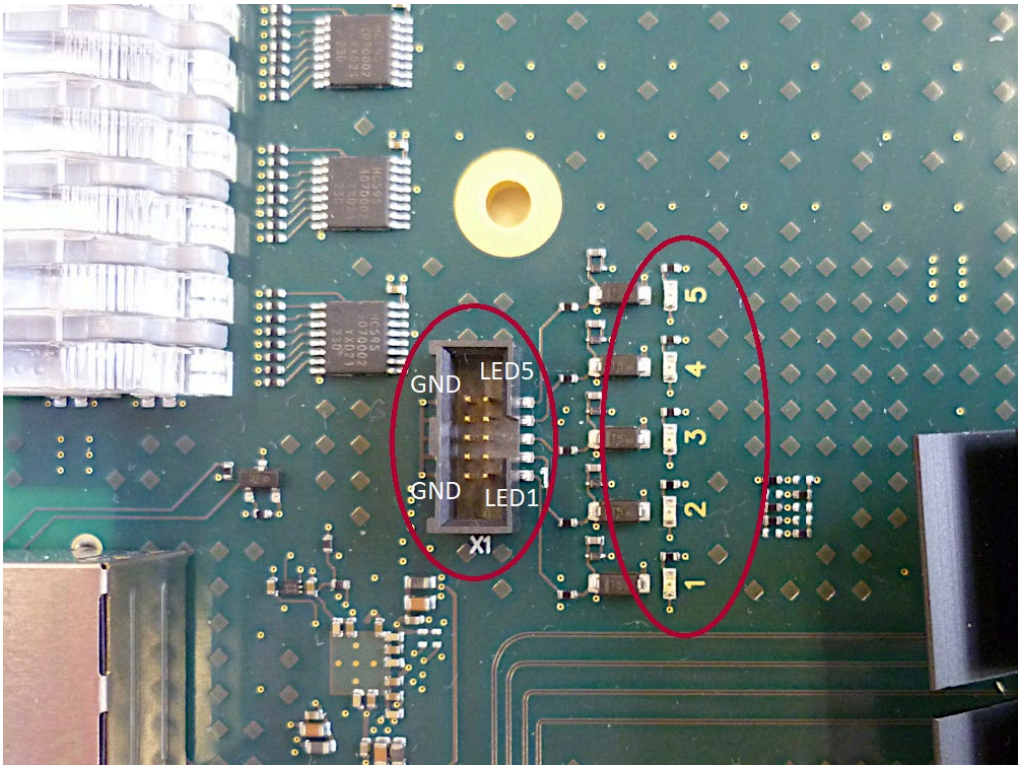
Frontpanel LEDs

The user FPGA can control the red (**frontpanel_LED_red**) and green (**frontpanel_LED_green**) frontpanel LEDs. This uses a register for the communication. A real-time output is therefore not possible.

8.3 Debugging

Debug LEDs

For debug purposes it is possible to use five LEDs located on the FPGA processor board. The FPGA processor board is a stacked PCB located near to the front panel on the VT System module. For the VT5838 the LEDs are located on the base board.



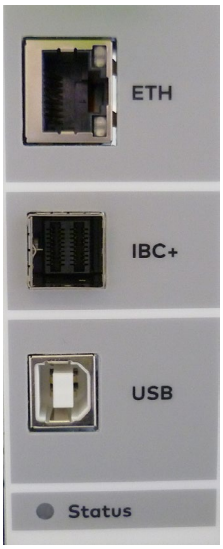
The LEDs are controlled with simple Boolean signals, named **debug_LED_1** to **debug_LED_5**. These signals can also be watched with an oscilloscope by connecting a cable at the provided connector on the processor board. The configuration of this male connector is:

PIN	Configuration
1	LED1
2	GND
3	LED2
4	GND
5	LED3
6	GND
7	LED4
8	GND
9	LED5
10	GND

This option only exists for the VT5838.

JTAG

The front USB can be used for a JTAG connection. The necessary hardware is already included on the module so no USB-Blaster™ is needed. The Quartus® USB-Blaster™ drivers are still necessary.

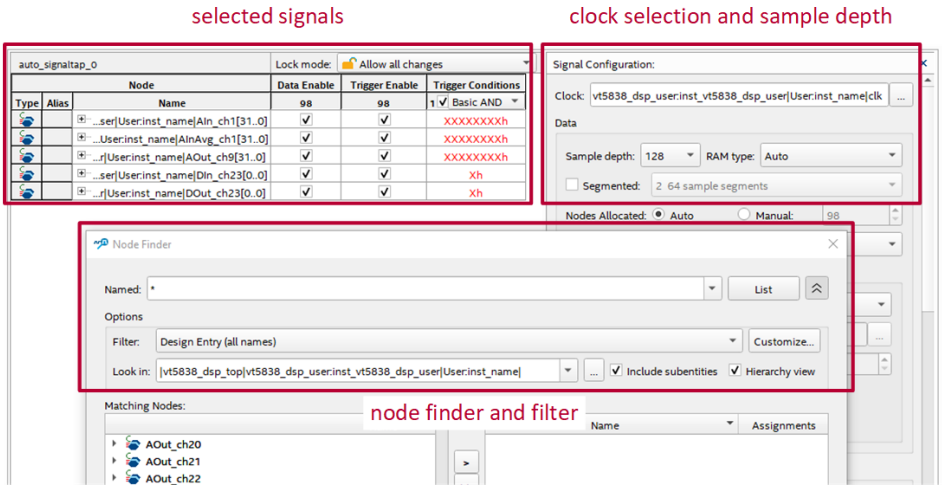


Do the following steps to use a signal tap with the VT5838.

1. Install the drivers for the USB-Blaster™ which are part of Quartus®.
2. Connect the front USB port of the VT5838 to the PC.
3. Open the .qpf file in Quartus®. This file is in the following folder of the project: `quartus\SYN`.
4. Oben the **Signal Tap Logic Analyzer** window via **Tools** in the menubar.
5. Save the signal tap (preferrably in the **SYN** folder) via the toolbar and enable the signal tap for the project in the opening panel.
6. Select in the **Signal Configuration** on the righthand side a clock signal to use the signal tap. It is recommended to select the clock signal used for the User FPGA project.

Note: It is possible that no signals are displayed. Then you have to compile the FPGA project via the VT System FPGA Manager (see chapter 3.3.8).

7. By double clicking in the main part of the **Signal Tap Logic Analyzer** window, you open the **Node Finder** window to insert signals. It is recommended to set the filter to **Design Entity (all names)** for all signals to show up.
8. Click on the **[List]** button. Signals can be inserted in their entirety or only specific bits. For the measurement of non-continuous signals it is recommended to use a trigger.
9. Close the **Node Finder** window.
10. Set the **Sample depth** in the **Signal Confiuration** so that the signal tap measures enough values for an evaluation.
11. Save the signal tap via the toolbar after entering all settings.
12. Start the compilation via the ribbon of the VT System FPGA Manager. It is possible that the VT System FPGA Manager does not recognize a change in the signal tap file. In this case start via the toolbar in the **Signal Tap Logic Analyzer** window the compilation and stop it immediately. Afterwards the VT System FPGA Manager should recognize the change.
13. Download the project via the ribbon of the VT System FPGA Manager.
14. Select the JTAG hardware in the **Signal Tap Logic Analyzer** window. If the hardware was recognized and selected correctly the window should show **Ready to acquire**. If it shows **Invalid JTAG configuration** then the signal tap must be saved and the project compiled again.
15. In the **Signal Tap Logic Analyzer** window, you can start the measurement once or continuously. If a trigger is set, the measurement waits for the next trigger event.
16. If you save the signal tap after the measurement, then the measurement data will also be saved.



8.4 Time Stamps

Default time stamps

By default all input (measurement) variables will be updated in CANoe with the same time stamp. This time stamp is computed by the VT System module automatically every time the variable values are transmitted to CANoe.

Custom time stamps

In the System Variables table in the VT System FPGA Manager you can assign your variables to time stamps other than the default one. All variables assigned to a custom time stamp will then use this time stamp's value when being updated in CANoe.

The non-default time stamps will appear as signals in the VHDL code or as blocks in the Simulink model. You can then write your own values to these time stamp signals/blocks.

Name	Direction	Data Type	Description
out_time_stamp_group1 ... out_time_stamp_group32	OUT	uint64	Holds the current time stamp for all measurement System Variables associated with this time stamp group.

8.5 VT1004A

VT1004A channels

The VT1004A has 4 analog/digital input channels.

The input data of the ADC is forwarded to the User FPGA. The ADC has a sample rate of 250 kHz and a resolution of 16 Bit. This resolution will be converted for the User FPGA, so that a signed int32 value is available cyclically every 4 µs in µV. Negative values are represented by a two's complement.

The digital input state of a channel is also available at the User FPGA with some delay.

Name	Direction	Data Type	Unit	Cycle Time	Description
in_adc_1 ... in_adc_4	IN	int32	µV	typ. 4 µs (delay typ. 20 µs)	data from ADC input channel 1 to 4

Name	Direction	Data Type	Delay Time	Description
In_signal_1 ... In_signal_4	IN	boolean	typ. 1 µs	data from digital input channel 1 to 4

8.6 VT1104

VT1104 channels

The VT1104 has 4 analog/digital input channels.

The input data of the ADC is forwarded to the User FPGA. The ADC has a sample rate of 250 kHz and a resolution of 16 Bit. This resolution will be converted for the User FPGA, so that a signed int32 value is available cyclically every 4 µs in µV. Negative values are represented by a two's complement.

The digital input state of a channel is also available at the User FPGA with some delay.

Name	Direction	Data Type	Unit	Cycle Time	Description
in_adc_1 ... in_adc_4	IN	int32	µV	typ. 4 µs (delay typ. 20 µs)	data from ADC input channel 1 to 4

Name	Direction	Data Type	Delay Time	Description
In_signal_1 ... In_signal_4	IN	boolean	typ. 1 µs	data from digital input channel 1 to 4

8.7 VT2004A

VT2004A channels

The VT2004A has 4 analog output channels.

The data to the 14 Bit DAC is only forwarded when the output signal changes. For outputting a signal, the curve type has to be set to User FPGA in CANoe. More information can be found in the CANoe help.

Name	Direction	Data Type	Unit	Cycle Time	Description
out_dac_1 ... out_dac_4	OUT	int32	µV	on change (delay typ. 1 µs)	data to DAC output channel 1 to 4

8.8 VT2516A

VT2516A channels

The VT2516A has 16 analog/digital input channels and 16 analog output channels.

The input data of the ADC is forwarded to the User FPGA. The ADC has a resolution of 12 Bit and works in multiplexed mode. The resolution will be converted for the User FPGA, so that a signed int32 value is available cyclically in µV. Negative values are represented by a two's complement. Because of the multiplexed sampling of the channels, the cycle time of the input signals varies.

The data to the 8 Bit DAC is only forwarded when the output signal changes. For outputting a signal, the curve type has to be set to User FPGA in CANoe. More information can be found in the CANoe help.

The digital input state of a channel is also available at the User FPGA with some delay.

Name	Direction	Data Type	Unit	Cycle Time	Description
in_adc_1 ... in_adc_16	IN	int32	µV	min.20 µs max.40 µs	data from ADC input channel 1 to 16
out_dac_1 ... out_dac_16	OUT	int32	µV	on change (delay typ. 1 µs)	data to DAC output channel 1 to 16

Name	Direction	Data Type	Delay Time	Description
In_signal_1 ... In_signal_16	IN	boolean	typ. 1 µs	data from digital input channel 1 to 16

8.9 VT2710

VT2710 channels

The VT2710 has 17 channels that can be used to map user defined system variables. These channels include one module channel, four piggy channels, two SPI channels, two UART channel, two RS-485 channels, two I2C channels, two LVDS channels and two DIO channels.

For each of the two topmost connectors of the VT2710 the same set of variables exists. Their functionality depends on the User FPGA configuration in CANoe. The following table shows the variables for connector 1 (left). The variables for connector 2 look identical, but use “con2” instead of “con1” in their names. In addition there are four variables to control the two LVDS channels of the VT2710.

Name	Direction	Description
io_in_con1_dio1	IN	Digital input 1 from connector 1.
io_in_con1_dio2_rs232_rx	IN	Digital input 2 from connector 1. Can also be used as RS-232 RX.
io_in_con1_dio3	IN	Digital input 3 from connector 1.
io_in_con1_dio4_rs485_rx	IN	Digital input 4 from connector 1. Can also be used as RS-485 RX.
io_in_con1_dio5	IN	Digital input 5 from connector 1.
io_in_con1_dio6	IN	Digital input 6 from connector 1.
io_in_con1_dio7_i2c_sda	IN	Digital input 7 from connector 1. Can also be used as I2C SDA.
io_in_con1_dio8_i2c_scl	IN	Digital input 8 from connector 1. Can also be used as I2C SCL.

Name	Direction	Description
io_out_con1_dio1_rs232_tx	OUT	Digital output 1 from connector 1. Can also be used as RS-232 TX.
io_out_con1_dio2	OUT	Digital output 2 from connector 1.
io_out_con1_dio3_rs485_tx	OUT	Digital output 3 from connector 1. Can also be used as RS-485 TX.
io_out_con1_dio4	OUT	Digital output 4 from connector 1.
io_out_con1_dio5_rs485_oe	OUT	Digital output 5 from connector 1. Can also be used as RS-485 output enable.
io_out_con1_dio6	OUT	Digital output 6 from connector 1.
io_out_con1_dio7_i2c_sda	OUT	Digital output 7 from connector 1. Can also be used as I2C SDA.
io_out_con1_dio8_i2c_scl	OUT	Digital output 8 from connector 1. Can also be used as I2C SCL.

Name	Direction	Description
lvds_in_1 and lvds_in_2	IN	LVDS input from LVDS connector 1/2.
lvds_out_1 and lvds_out_2	OUT	LVDS output to LVDS connector 1/2.

I/O Delay

The following table shows the delays for data input and output. These delays were measured between the physical connector and the User FPGA.

Channel	Input Delay (Connector -> User FPGA)	Output Delay (User FPGA -> Connector)
DIO 1-2	80ns	20ns
DIO 3-6	80ns	20ns
DIO 7-8	80ns	400ns
RS-232	200ns	130ns
RS-485	50ns	50ns
I2C	100ns	100ns

8.10 VT2808

VT208 channels

The VT2808 has 8 analog input channels, each providing a voltage and a current value.

The input data of the ADC is forwarded to the User FPGA. The ADC has a sample rate of 250 kHz and a resolution of 16 Bit. This resolution will be converted for the User FPGA, so that a signed int32 value is available cyclically every 4 µs inf 1 µV/1 µA. Negative values are represented by a two's complement.

Name	Direction	Data Type	Unit	Cycle Time	Description
in_adc_voltage_1... in_adc_voltage_8	IN	int32	µV	typ. 4 µs (delay typ. 20 µs)	voltage data from ADC input channel 1 to 8
in_adc_current_1... in_adc_current_8	IN	int32	µA	typ. 4 µs (delay typ. 20 µs)	current data from ADC input channel 1 to 8

8.11 VT2816

VT2816 channels

The VT2816 has 12 analog input channels and 4 analog output channels.

The input data of the ADC is forwarded to the User FPGA. The ADC has a sample rate of 250 kHz and a resolution of 16 Bit. This resolution will be converted for the User FPGA, so that a signed int32 value is available cyclically every 4 µs in µV. Negative values are represented by a two's complement.

The data to the 14 Bit DAC is only forwarded when the output signal changes. For outputting a signal, the curve type has to be set to User FPGA in CANoe. More information can be found in the CANoe help.

Name	Direction	Data Type	Unit	Cycle Time	Description
in_adc_1 ... in_adc_12	IN	int32	µV	typ. 4 µs (delay typ. 20 µs)	data from ADC input channel 1 to 12
out_dac_1 ... out_dac_4	OUT	int32	µV	on change (delay typ. 1 µs)	data to DAC output channel 1 to 4

8.12 VT2848

VT2848 channels

The VT2848 has 48 digital input channels and 48 digital output channels.

It is recommended to use channels 1 to 16 for input purposes and channels 33 to 48 for output purposes.

The input state of a channel is available at the User FPGA with some delay.

A change of the output state is forwarded also with some delay. For outputting a signal, the curve type has to be set to User FPGA in CANoe. More information can be found in the CANoe help.

Name	Direction	Data Type	Delay Time	Description
in_signal_1 ... in_signal_48	IN	boolean	typ. 1 µs	data from digital input channel 1 to 48
out_signal_1 ... out_signal_48	OUT	boolean	typ. 1 µs	data to digital output channel 1 to 48

8.13 VT5838

VT5838 channels

The VT5838 has 16 digital IO channels. In addition, it has 8 analog input channels and 10 output channels.

The VT5838 also supports 2 RS422 channels, 2 LVDS channels and 2 HyperRAM ICs with 128 Mb each.

For custom purposes a piggie board can be designed. To communicate with this board 58 digital IO signals are available.

For custom signaling 16 red LEDs and 16 green LEDs on the frontpanel can be set.

The input data of the ADC is forwarded to the User FPGA. The ADC has a frequency of up to 3.2 MHz and a resolution of 16 Bit. The update frequency depends on the module specific settings in the FPGA manager. This resolution will be converted for the User FPGA, so that a signed int32 value is available. Negative values are represented by a two's complement.

The data width of the DAC is 16 bit. It has a frequency of up to 2 MHz. For outputting a signal, the curve type has to be set to User FPGA in CANoe. More information can be found in the CANoe help.

The phy used by the VT5838 is an ADIN1300.
The user FPGA used by the VT5838 is an Altera Cyclone V 5CGXFC9E6F31I7N.

Name	Direction	Data Type	Unit	Cycle Time	Description
in_adc_1 ... in_adc_8	IN	int32	µV	between 200 ns and 1 ms	data from ADC input channel 1 to 8
out_dac_1 ... out_dac_14	OUT	int32	µV	between 500 ns and 1 ms	data to DAC output channel 1 to 14
in_adc_data_valid	IN	boolean			marks the validity of incoming ADC data
out_adc_sample	OUT	boolean			triggers the ADC
in_dac_ready	IN	boolean			indicates that the DAC unit is ready
in_dac_load	IN	boolean			
out_dac_conv	OUT	boolean			triggers the DAC

Name	Direction	Data Type	Delay Time	Description
in_digital_1 ... in_digital_16	IN	boolean	typ. 100 ns	data from digital input channel 1 to 16
out_digital_1 ... out_digital_16	OUT	boolean	typ. 100 ns	data to digital output channel 1 to 16
out_digital_tristate_1 ... out_digital_tristate_16	OUT	boolean		set output channel to active output ('0') or high impedance ('1')

Name	Direction	Description
in_serial_rs422_1 ... in_serial_rs422_2	IN	RS422 serial data in
out_serial_rs422_1 ... out_serial_rs422_2	OUT	RS422 serial data out
out_serial_rs422_enable_1 ... out_serial_rs422_enable_2	OUT	RS422 enable signal
in_serial_lvds_1 ... in_serial_lvds_2	IN	LVDS data serial in
out_serial_lvds_1 ... out_serial_lvds_2	OUT	LVDS data serial out

Name	Direction	Description
out_hram0_avalon_select	OUT	vector for the avalon select signal
in_hram0_avalon_waitrequest	IN	vector for the avalon wait request signal
out_hram0_avalon_write	OUT	vector for the avalon write signal
out_hram0_avalon_read	OUT	vector for the avalon read signal
out_hram0_avalon_address_1 ...	OUT	24 bit avalon address for writing and reading

Name	Direction	Description
out_hram0_avalon_address_2		
out_hram0_avalon_writedata_1 ... out_hram0_avalon_writedata_2	OUT	32 bit avalon output data
in_hram0_avalon_readdata_1 ... in_hram0_avalon_readdata_2	IN	32 bit avalon input data
in_hram0_burst_busy	IN	vector for burst busy signal
in_hram0_burst_grant	IN	vector for burst grant signal
out_hram0_burst_write	OUT	vector for burst write signal
out_hram0_burst_read	OUT	vector for burst read signal
in_hram0_burst_readdata_valid	IN	signals when the readdata is valid
out_hram0_burst_be_1 ... out_hram0_burst_be_2	OUT	burst enable signal
out_hram0_burst_count_1 ... out_hram0_burst_count_2	OUT	selection which data bytes should be written or read
out_hram0_burst_address_1 ... out_hram0_burst_address_2	OUT	24 bit burst address for writing and reading
out_hram0_burst_writedata_1 ... out_hram0_burst_writedata_2	OUT	32 bit burst output data
in_hram0_burst_readdata_1 ... in_hram0_burst_readdata_2	OUT	32 bit burst input data

Name	Direction	Description
out_hram1_avalon_select	OUT	vector for the avalon select signal

Name	Direction	Description
in_hram1_avalon_waitrequest	IN	vector for the avalon wait request signal
out_hram1_avalon_write	OUT	vector for the avalon write signal
out_hram1_avalon_read	OUT	vector for the avalon read signal
out_hram1_avalon_address_1 ... out_hram1_avalon_address_2	OUT	24 bit avalon address for writing and reading
out_hram1_avalon_writedata_1 ... out_hram1_avalon_writedata_2	OUT	32 bit avalon output data
in_hram1_avalon_readdata_1 ... in_hram1_avalon_readdata_2	IN	32 bit avalon input data
in_hram1_burst_busy	IN	vector for burst busy signal
in_hram1_burst_grant	IN	vector for burst grant signal
out_hram1_burst_write	OUT	vector for burst write signal
out_hram1_burst_read	OUT	vector for burst read signal
in_hram1_burst_readdata_valid	IN	signals when the readdata is valid
out_hram1_burst_be_1 ... out_hram1_burst_be_2	OUT	burst enable signal
out_hram1_burst_count_1 ... out_hram1_burst_count_2	OUT	selection which data bytes should be written or read
out_hram1_burst_address_1 ... out_hram1_burst_address_2	OUT	24 bit burst address for writing and reading
out_hram1_burst_writedata_1 ... out_hram1_burst_writedata_2	OUT	32 bit burst output data
in_hram1_burst_readdata_1 ...	OUT	32 bit burst input data

Name	Direction	Description
in_hram1_burst_readdata_2		

Name	Direction	Description
in_application_board_clk_1 ... in_application_board_clk_2	IN	clock signal from an application board
in_application_board_io_input_1 ... in_application_board_io_input_44	IN	data from digital input channel 1 to 44 of the application board
out_application_board_io_output_1 ... out_application_board_io_output_44	OUT	data to digital output channel 1 to 44 of the application board
out_application_board_io_control_1 ... out_application_board_io_control_44	OUT	select if a pin is used as an input or output pin

Name	Direction	Description
frontpanel_LED_green_1 ... frontpanel_LED_green_16	OUT	control of the green LEDs on the frontpanel
frontpanel_LED_red_1 ... frontpanel_LED_red_16	OUT	control of the red LEDs on the frontpanel

Name	Direction	Description
in_ibc_plus_lane_error_1...4	IN	Signalizes an error on this specific lane. - Bit 0: link error - Bit 1: interface version mismatch - Bit 3: system id mismatch
in_ibc_plus_lane_block_tx_ready	IN	Signalizes that the output FIFO isn't full and data can be sent. (Each bit represents another lane.)

Name	Direction	Description
in_ibc_plus_lane_block_tx_ack	IN	Signalizes that the output was sent. (Each bit represents another lane.)
in_ibc_plus_lane_block_tx_valid	OUT	Signals that new data should be sent. (Each bit represents another lane.)
in_ibc_plus_lane_block_rx_valid	IN	Signalizes a new input value. (Each bit represents another lane.)

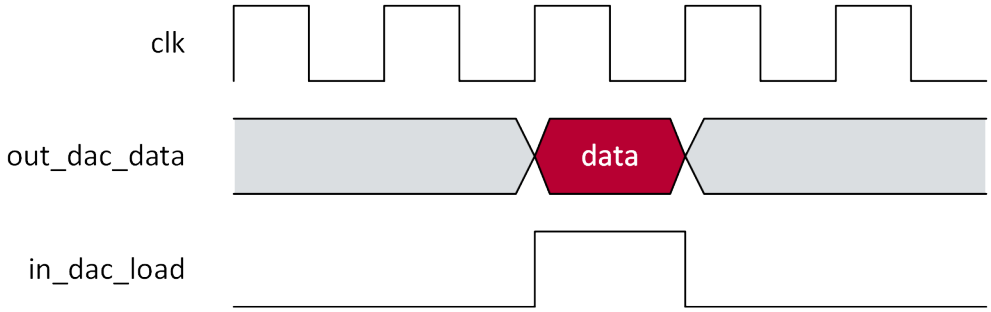
Name	Direction	Description
out_phy_reset_n	OUT	Low active reset of the phy.
in_phy_int_n	IN	Low active phy interrupt.
out_phy_led	OUT	RJ45 LED control. - Bit 0: - Bit 1: - Bit 2:
out_phy_mdc	OUT	Phy MDC signal.
inout_phy_mdio	INOUT	Phy MDIO signal.
in_mac_address_valid	IN	Signalizes if the module has a valid MAC address.
in_mac_address_value	IN	Shows the MAC address if it's valid.
in_eth_reset	IN	Ethernet reset.
in_phy_led_0	IN	Phy LED_0 signal. (Needs hardware revision 0x10.)
in_phy_link_status	IN	Phy LINK_ST signal. (Needs hardware revision 0x10.)
in_eth_clock	IN	Ethernet clock.
in_eth_gmii_rx_clock_en	IN	GMII rx clock enable.
in_eth_gmii_rx_en	IN	GMII rx enable.
in_eth_gmii_rx_err	IN	GMII rx error.
in_eth_gmii_rx_data	IN	GMII rx data.
in_eth_gmii_tx_clock_en	IN	GMII tx clock enable.
in_eth_gmii_tx_en	OUT	GMII tx enable.
in_eth_gmii_tx_err	OUT	GMII tx error.
in_eth_gmii_tx_data	OUT	GMII tx data.

Name	Direction	Description
in_hw_revision	IN	Hardware revision number of the baseboard.

8.13.1 Hardware Interface

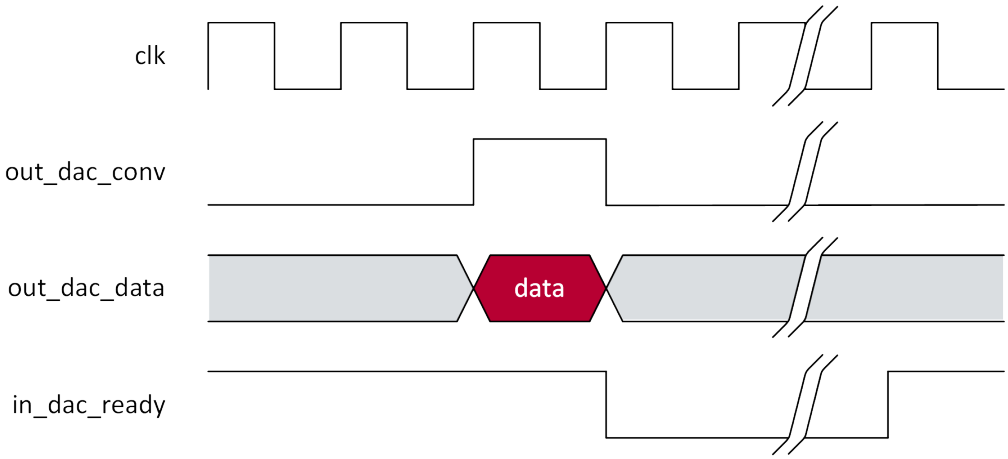
Analog output cyclic

The data will be output with a fixed frequency that can be set in the manager. The acceptance of thedata is signaled with load.



Analog output manual

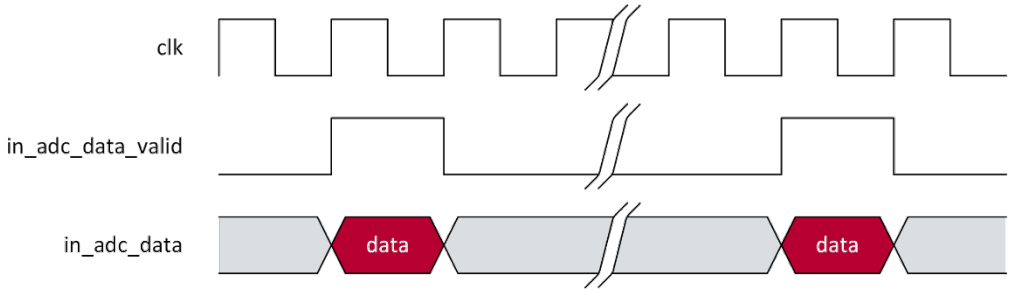
The data will be send when the conv signal is set. Ready being low signals that the data is send to the DAC. It is only allowed to output data when the ready signal is high.



Analog input cyclic

The measurement data is provided with a cyclic frequency that can be set in the manager. Valid signals when new data is received.

The ADC uses a buffer to store a single measurement data before it is transferred to the FPGA. This results in a delay for getting the data. When the valid signal ('in_adc_data_valid') is set the data from the buffer is transferred to the FPGA while the measurement data is written into the buffer. This results in a delay for the measurement data of one read cycle.

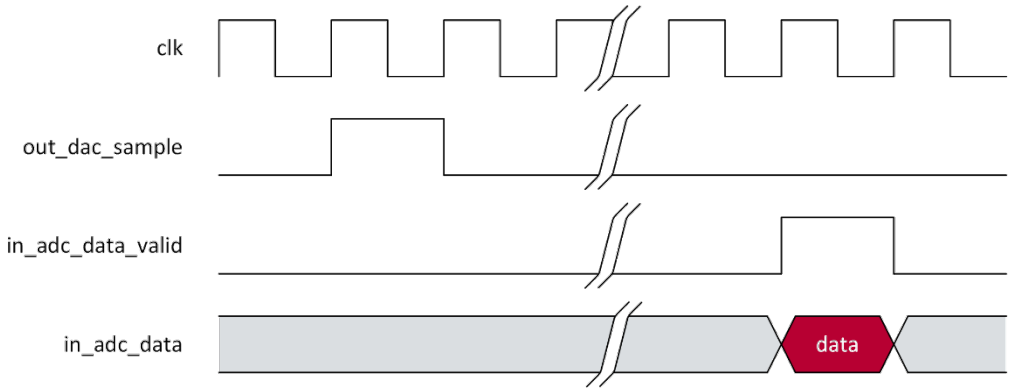


Analog input manual

The sample signal triggers all ADCs of the activated channels. Valid signals when new data is received.

The ADC uses a buffer to store a single measurement data before it is transferred to the FPGA. This results in a delay for getting the data. When the trigger signal (‘out_dac_sample’) is set the data from the buffer is transferred to the FPGA and received when the valid signal ('in_adc_data_valid’) is set. The measurement data is written to the buffer at the same time.

This means that you have to set the trigger signal a second time to receive the current measurement data.

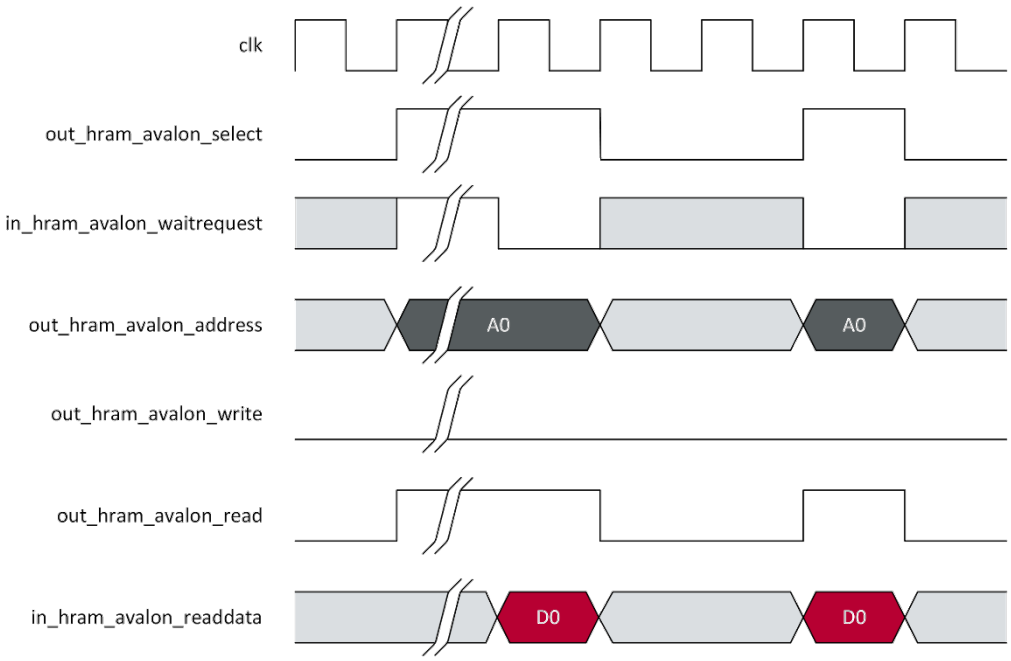


Avalon/Burst addresses

The data in the RAM is saved as blocks of 4 bytes with 1 byte per address. The addresses are also structured as blocks of 4. This means that 4 consecutive addresses link to the same 4 bytes of data in the RAM and are therefore identical (e.g. address 0 = 1 = 2 = 3, 4 = 5 = 6 = 7, ...). This means that the least 2 bits of the address are ignored.

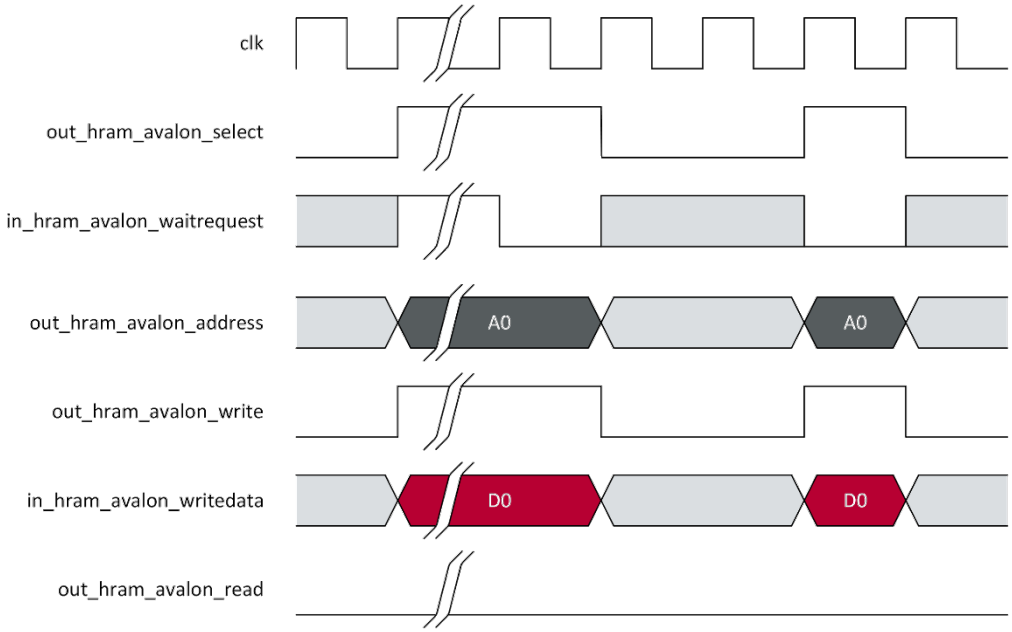
Avalon read

The read access is started by selecting the avalon interface and setting the read signal and the address. The data is valid when the waitrequest signal changes to low.



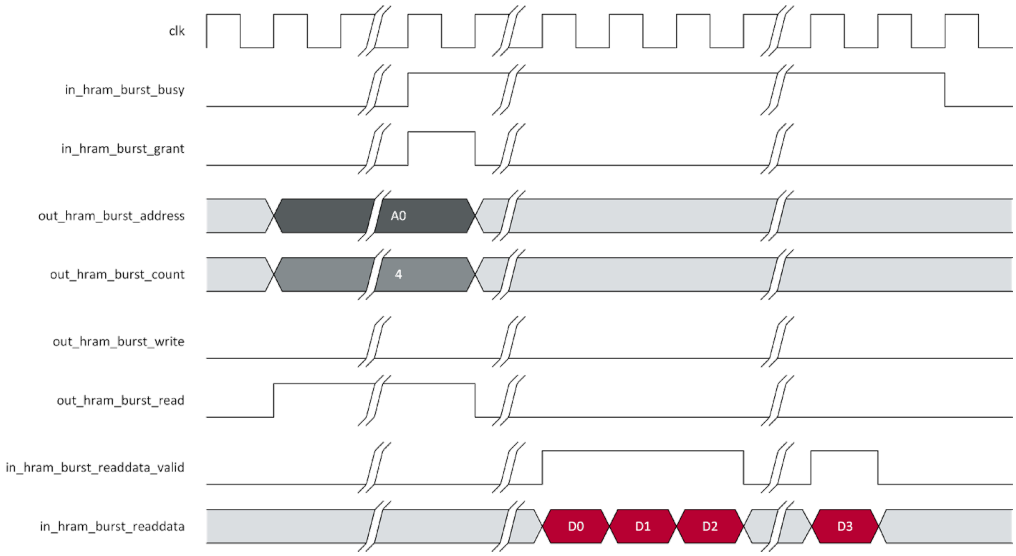
Avalon write

The write access is started by selecting the Avalon interface and setting the write signal, write data and address. The access is finished when the waitrequest signal changes to low.



Burst read

The read access is started by setting the read signal and the first address. The busy and grant signals need to be low. Count signals the number of data words used. The read access is acknowleged by setting the grant signal. Busy shows that the access is active. Valid signals when new data is received.

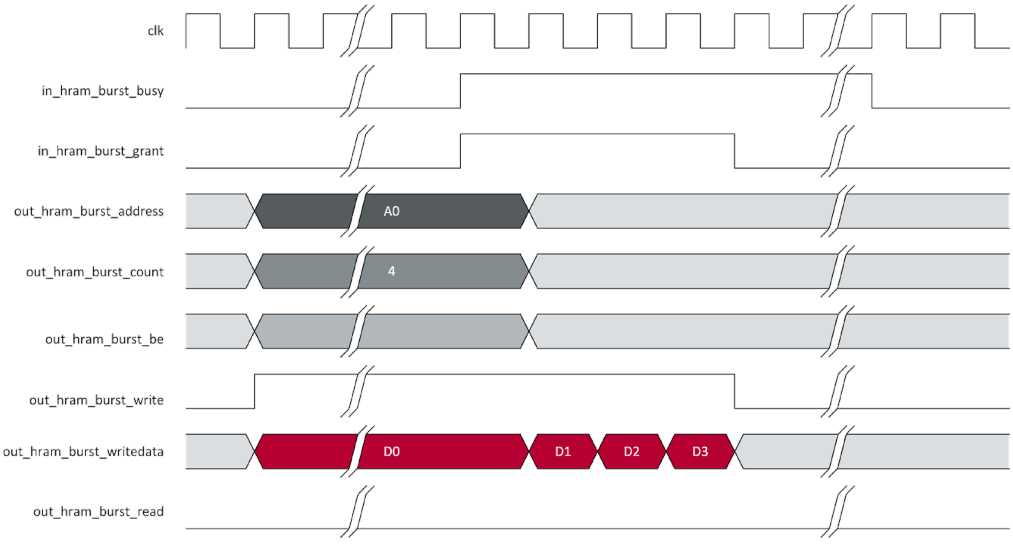


Burst write

The write access is started by setting the write signal and the first address. The busy and grant signals need to be low. Count signals the number of data words used. The words are transmitted in every cycle if the grant signals is set.

out_hram_burst_be:

- 1111: writes full 4 bytes
- 0011: writes lower 2 bytes
- 1100: writes upper 2 bytes
- 0001: writes byte 0 only
- 0010: writes byte 1 only
- 0100: writes byte 2 only
- 1000: writes byte 3 only



8.14 VT7900A

VT7900A channels

The number of channels as well as their names are not fixed for the VT7900A. Instead they can be freely configured by using the VT System Application Board Designer.

The available signals also depend on the board configuration used for this project.

Name	Direction	Data Type	Description
In_i2c_scl1	IN	boolean	I²C 1 Clock (Pin State)
out_i2c_scl1	OUT	boolean	I²C 1 Clock (Output)
in_i2c_sda1	IN	boolean	I²C 1 Data (Pin State)
out_i2c_sda1	OUT	boolean	I²C 1 Data (Output)

Name	Direction	Data Type	Description
inout_databus_d0 ... inout_databus_d15	IN/OUT	boolean	Databus Data
out_databus_d0_7_out_active	OUT	boolean	Databus Output Active Data 0-7
out_databus_d8_15_out_active	OUT	boolean	Databus Output Active Datat 8-15

Name	Direction	Data Type	Description
in_spi_miso	IN	boolean	SPI MISO
in_spi_n_mirq	IN	boolean	SPI Interrupt Input (Low Active)
out_spi_spck	OUT	boolean	SPI Clock
out_spi_mosi	OUT	boolean	SPI MOSI
out_spi_n_mcs0 ... out_spi_n_mcs3	OUT	boolean	SPI Chip Select (Low Active)

Name	Direction	Data Type	Description
out_dout_0 ... out_dout_3	OUT	boolean	Digital Output

Name	Direction	Data Type	Description
in_din_0 ... in_din_3	IN	boolean	Digital Input

Name	Direction	Data Type	Description
out_databus_a1 ... out_databus_a5	OUT	boolean	Databus Address
out_databus_n_wr	OUT	boolean	Databus Write Enable (Low Active)
out_databus_n_rd	OUT	boolean	Databus Read Enable (Low Active)
out_databus_n_bhe	OUT	boolean	Databus Reserved
out_databus_n_pclk	OUT	boolean	Databus Clock
out_databus_n_conv	OUT	boolean	Databus Conversion Start (Low Active)
out_databus_n_reset	OUT	boolean	Databus Reset (Low Active)
in_databus_n_sirq	IN	boolean	Databus Additional Interrupt Input (Low Active)
in_databus_n_wait	IN	boolean	Databus Wait (Low Active)
in_databus_n_busy	IN	boolean	Databus Busy (Low Active)
in_databus_n_irq0	IN	boolean	Databus Interrupt Input 0 (Low Active)
in_databus_n_irq1	IN	boolean	Databus Interrupt Input 1 (Low Active)

9 Troubleshooting

In this chapter you find the following information:

9.1 Problems and Solutions page 82

9.1 Problems and Solutions

FAQ: The VT System FPGA Module is not Recognized

Problem

When trying to test or persist a project in the VT System FPGA Manager your VT System module does not appear in the target module selection box.

Solution

> Make sure the module is correctly inserted into the VT System rack
> Check that the VT System is turned on and properly connected to the computer
> Verify that no other applications that may access the VT System (e.g. CANoe) are running
> Check your network adapter's settings in the Windows control panel (see CANoe online help for details)

FAQ: MATLAB® Crashes When Using the DSP Builder Block Set

Problem

When opening the Intel® DSP Builder standard block set in the Simulink® library browser MATLAB® crashes.

Solution

> The crashes should only occur when opening the DSP Builder standard block set. For programming the User FPGA this block set is not needed. Instead use the advanced block set as described in the chapters above.
> This problem seems to occur only in DSP Builder 12.0. So if possible try to install more up to date DSP Builder versions.

FAQ: DSP Builder does not Create HDL Code due to a Missing License

Problem

When trying to create HDL code from a Simulink® model the DSP Builder shows an error message regarding a missing license.

Solution

Please make sure you have a valid DSP Builder license file. Usually this is a **.dat** file. Store this file in some safe place on your hard drive. Then make sure the environment variable **LM_LICENSE_FILE** exists and is pointing to your license file. After that reboot your computer. On the next start-up DSP Builder should find the license file and generate the HDL code properly.

FAQ: Demo projects are broken or missing

Problem

The VT System FPGA Manager is shipped with several sample projects for all supported VT System modules. But how can you restore such a sample to its original state after you have modified or deleted it?

Solution

Go to the options dialog and look for a section called “Demo Projects”. Here you can restore each of the sample projects. Please be careful if you have made any modifications that you would like to keep. These changes will get lost when you restore the corresponding project.

FAQ: Quartus® does not Compile HDL Code due to a Missing License

Problem

When trying to compile your project from the VT System FPGA Manager you get an error about a missing license.

Solution

Quartus® needs a license file to compile your projects. This license file is part of your CANoe installation and can be found in **\Exec32\VTSFPGAAssets\lic_VT_User_FPGA.dat**.

To allow Quartus® finding this license file the user environment variable **LM_LICENSE_FILE** has to be set accordingly. Usually this is done automatically by the VT System FPGA Manager. In some cases a reboot of the computer may be required to make the changes visible to Quartus®.

If this variable is not created and set automatically, you may do so manually. To do so, make sure the user environment variable **LM_LICENSE_FILE** exists (if not, create it). Then set its value to the absolute path of the file **lic_VT_User_FPGA.dat** in your CANoe installation. If the variable already exists, append a semicolon and the file path to the variable’s value.

In most cases the should solve the problem. If the error still occurs, start the Quartus application. Go to **Tools | License Setup**. Insert the contents of the computer’s environment variable **LM_LICENSE_FILE** into the field **License file**. Now you should see a new entry called **Vector Informatik** in the list **Licensed AMPP/MegaCore functions**.

FAQ: User FPGA project is not recognized by CANoe although it was downloaded onto the User FPGA

Problem

A project was downloaded onto the User FPGA but is not recognized by CANoe. The module appears as normal a VT System module in the VT System Configuration dialog.

Solution

The CANoe configuration may have been created before you downloaded the User FPGA project onto the VT System module. Thus the CANoe configuration contains only the base version of the VT System module.

When you download your project onto the User FPGA later on, CANoe will keep treating the configured module as a base version. Your User FPGA project will be ignored by CANoe.

To resolve this issue you need to delete the module in the VT System configuration dialog in CANoe. Then you need add your User FPGA enabled module again, e.g. by using the **Adapt to connected hardware** function.

FAQ: The timing requirements for a project are not met

Problem

Not meeting the timing requirements can lead to unpredictable behavior, e.g. wrong calculation results.

Solution

Try reducing the User FPGA frequency. If this is not possible, e.g. a specific frequency is needed, try using a different seed for the compile process.

You can find the option for a different seed in Quartus® Prime:

Assignments | **Settings** | **Compiler Settings** | **Advanced Settings (Fitter).** Change **Fitter Initial Placement Seed** value.

In addition you can use the **Design Space Explorer II** (**Tools** | **Launch Design Space Explorer II**) to make a seed sweep. You can find this option under **Exploration** | **Exploration Points**.

The use of a different seed is only recommended for experienced users (see Intel: SEED).

VECTOR >

More Information

- > News
- > Products
- > Demo Software
- > Support
- > Training Classes
- > Addresses

www.vector.com