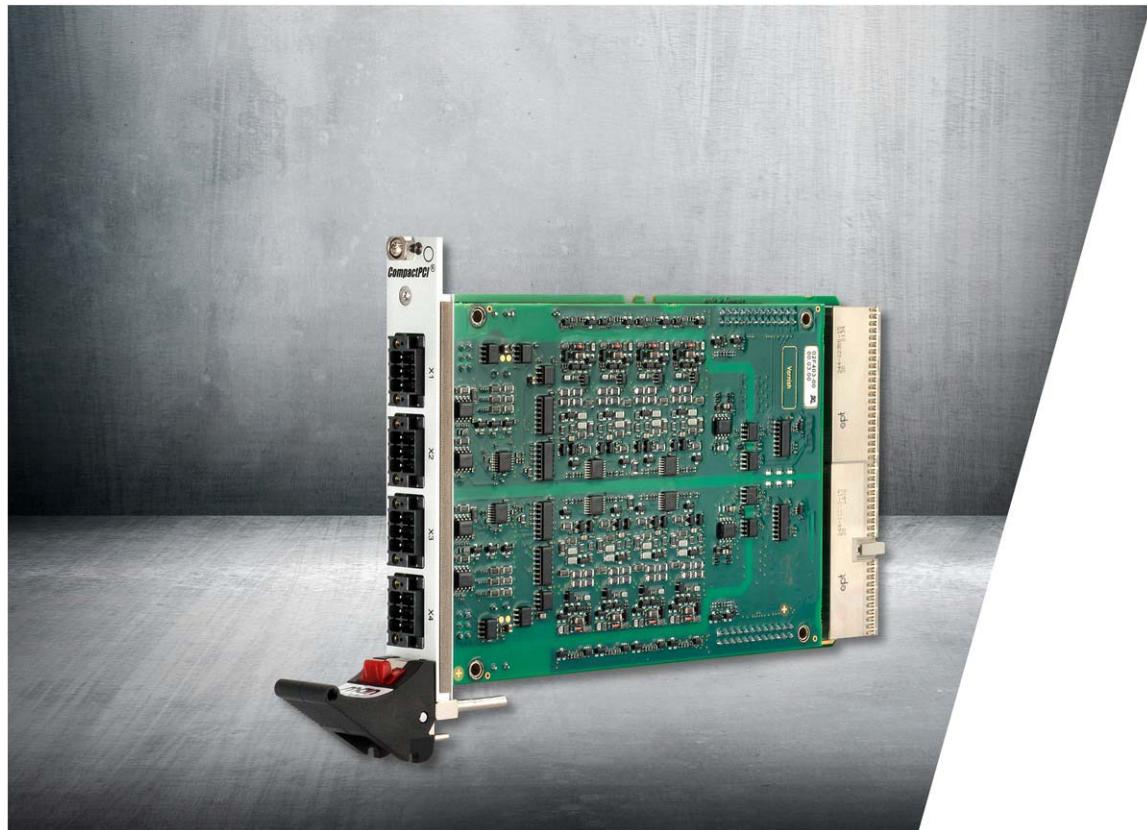


F403

Binary I/O Card for Railways
3U CompactPCI



User Manual

mcn

Contents

1 Product Overview	12
1.1 Product Description	12
1.2 External Interfaces	13
1.3 Board Layout	14
1.4 Block Diagram	15
1.5 Technical Data	16
1.6 Product Identification.....	18
2 Getting Started	19
2.1 Connecting and Starting	19
2.2 Installing Driver Software	19
2.3 Demo Application	20
2.3.1 Configuring the Geographical Address.....	20
2.3.2 Configuring the UART Interface.....	21
2.3.2.1 Linux	21
2.3.2.2 Windows.....	21
2.3.2.3 VxWorks	22
2.3.3 Compiling and Running the Demo Application	22
3 Functional Description.....	23
3.1 Power Supply.....	23
3.2 Binary I/O.....	23
3.2.1 Principle of Operation	23
3.2.2 Pulsed Input Current.....	24
3.2.3 Front Connectors	25
3.3 Debug LED	26
3.4 CompactPCI	26
3.4.1 Geographical Addressing	26
4 Hardware/Software Interface	27
4.1 PCI Configuration	27
4.2 Protocol Layer Overview	28
4.2.1 DSFI2 PCI Device.....	29
4.3 I/O Controller Functions.....	30
4.3.1 Basic I/O Functions	30
4.3.1.1 Inputs	30
4.3.1.2 Outputs.....	30
4.3.2 Output Watchdog.....	31
4.3.3 Output Comparison.....	32
4.3.4 Output Overload	32
4.3.5 Voltage Supervision	33
4.3.6 Current Drain Trigger.....	34
4.4 F403 Application Programming Interface (API).....	35
4.4.1 F403_Config()	36
4.4.2 F403_SetOutput()	39
4.4.3 F403_SetGroupOutputs()	40
4.4.4 F403_GetGroupInputs().....	41
4.4.5 F403_GetGroupStatus().....	42
4.4.6 F403_ClearGroupError()	44
4.4.7 F403_InterruptStatus().....	46

4.5	DSFI2 Layer	47
4.5.1	Functions.....	47
4.5.2	Configuration	48
4.5.3	DSFI2 Roles	49
4.5.3.1	DSFI2 Master	49
4.5.3.2	DSFI2 Slave	49
4.5.4	I/O Controller Identification	50
4.5.5	DSFI2 Integration on F403.....	50
4.5.6	DSFI2 API	51
4.5.6.1	DSFI2_init()	51
4.5.6.2	DSFI2_Config()	51
4.5.6.3	DSFI2_Function()	52
4.5.6.4	DSFI2_send().....	53
4.5.6.5	DSFI2_receive().....	54
4.5.7	DSFI2 Function Profile	55
4.5.7.1	DSFI2 Common Function Profile.....	55
4.5.7.2	I/O-Specific Function Profile.....	56
4.5.7.3	DSFI2 Handler Function Profile	57
4.5.8	Configuration	58
4.5.8.1	Link	58
4.5.8.2	Parameter Configuration	59
4.5.8.3	Operational State	59
4.5.9	I/O Controller Functions	60
4.5.9.1	Basic I/O Functions	60
4.5.9.2	Output Watchdog	60
4.5.9.3	Output Comparison	60
4.5.9.4	Output Overload	61
4.5.9.5	Voltage Supervision.....	61
4.5.9.6	Current Drain Trigger	62
4.5.10	Interrupt Handling	63
4.5.11	DSFI2 Message Layout.....	63
4.5.12	DSFI2 Message Fields	64
4.5.13	DSFI2 Message Field Coding.....	64
4.5.13.1	INSTANCE	64
4.5.13.2	MASTERID.....	65
4.5.13.3	FUNCTION	65
4.5.13.4	ERR.....	66
4.5.14	DSFI2 Master (Application).....	67
4.5.14.1	DSFI2 Master Services.....	67
4.5.14.2	DSFI2 Master States	67
4.5.15	DSFI2 Slave (I/O Controller).....	68
4.5.15.1	DSFI2 Slave Services	68
4.5.15.2	DSFI2 Slave Base Class	68
4.5.15.3	DSFI2 I/O Controller	78
4.5.16	DSFI2 Handler Unit	81
4.5.16.1	SetOutputValue	85
4.5.16.2	GetOutputValue.....	85
4.5.16.3	RegisterTest	85
4.5.16.4	GetInputValue	85
4.5.16.5	GetDSFI2Header	86
4.5.16.6	GetInstanceInterrupt	86

4.6	STP Layer	87
4.6.1	Framing and Bit Stuffing	88
4.6.1.1	Reset	88
4.6.1.2	CRC	88
4.6.1.3	STP Telegram Layout.....	89
4.6.1.4	STP Special Characters	89
4.7	Telegram Layout	90
5	Maintenance	91
5.1	Updating Firmware in the FPGA	91
5.1.1	Installing the Update Tool.....	91
5.1.1.1	Windows.....	91
5.1.1.2	Linux	91
5.1.1.3	VxWorks	91
5.1.2	Using the Update Tool.....	92
5.1.3	Using the Tool with the F403	92

Figures

Figure 1.	Front interfaces	13
Figure 2.	Board layout – top view.....	14
Figure 3.	Block diagram.....	15
Figure 4.	Product labels.....	18
Figure 5.	I/O stage	24
Figure 6.	Pulsed input current	24
Figure 7.	Protocol layer communication.....	28
Figure 8.	Hardware-software interface to F403	28
Figure 9.	Current Drain Trigger.....	34
Figure 10.	F403 API.....	35
Figure 11.	DSFI2 instances	48
Figure 12.	DSFI2 subsystem	49
Figure 13.	DSFI2 common function profile.....	55
Figure 14.	I/O-specific function profile	56
Figure 15.	DSFI2 handler function profile.....	57
Figure 16.	DSFI2 configuration	58
Figure 17.	Current Drain Trigger.....	62
Figure 18.	DSFI2 message overview.....	63
Figure 19.	DSFI2 Master states	67
Figure 20.	DSFI2 Slave I/O function profile	68
Figure 21.	DSFI2 common slave function profile	69
Figure 22.	DSFI2 Slave Configuration of State Machine	70
Figure 23.	Interrupt information of handler unit.....	81
Figure 24.	DSFI2 handler function profile	82
Figure 25.	STP layer communication	87
Figure 26.	STP overview.....	88
Figure 27.	STP CRC	88
Figure 28.	STP telegram.....	89
Figure 29.	Request layout	90

Tables

Table 1.	Pin assignment of front connector X1.....	25
Table 2.	Pin assignment of front connector X2.....	25
Table 3.	Pin assignment of front connector X3.....	25
Table 4.	Pin assignment of front connector X4.....	25
Table 5.	Signal mnemonics of front connectors X1 to X4	26
Table 6.	Debug LED.....	26
Table 7.	PCI address map	27
Table 8.	DSFI2 PCI device.....	29
Table 9.	PCI configuration header	29
Table 10.	Voltage supervision	33
Table 11.	Error status for API functions.....	35
Table 12.	F403_Config() parameters per I/O channel	36
Table 13.	F403_Config() common parameter	37
Table 14.	F403_Config() function arguments	37
Table 15.	struct F403_IoGrpConfig	38
Table 16.	struct F403_BoardConfig_t	38
Table 17.	F403_SetOutput() function arguments	39
Table 18.	F403_SetGroupOutputs() function arguments.....	40
Table 19.	F403_GetGroupInputs() function arguments.....	41
Table 20.	F403_GetGroupStatus() function arguments.....	42
Table 21.	struct F403_ErrStatus_t for F403_GetGroupStatus()	43
Table 22.	F403_ClearGroupError() function arguments	44
Table 23.	struct F403_ErrStatus_t for F403_ClearGroupError().....	45
Table 24.	F403 InterruptStatus() function arguments.....	46
Table 25.	struct F403 InterruptStatus_t for F403 InterruptStatus()	46
Table 26.	DSFI2 instances	50
Table 27.	MISC gpin connections	50
Table 28.	MISC interrupt_status connections	50
Table 29.	DSFI2_init() return codes.....	51
Table 30.	DSFI2_Config() function arguments.....	52
Table 31.	DSFI2_Config() return code.....	52
Table 32.	struct DSFI_Function	52
Table 33.	DSFI2_Function() function arguments.....	53
Table 34.	DSFI2_Function() return code.....	53
Table 35.	DSFI2_send() function arguments	53
Table 36.	DSFI2_send() return code	53
Table 37.	DSFI2_receive() function arguments	54
Table 38.	DSFI2_receive() return parameters	54
Table 39.	DSFI2 common function overview.....	55
Table 40.	I/O-specific function overview	56
Table 41.	DSFI2 handler function overview.....	57
Table 42.	I/O controller parameters.....	59
Table 43.	DSFI2 handler parameters	59
Table 44.	Voltage Supervision	61
Table 45.	DSFI2 data objects	64
Table 46.	REQ message header field INSTANCE	64
Table 47.	DSFI2 instances	64
Table 48.	Message header field MASTERID.....	65
Table 49.	REQ message header field FUNCTION	65
Table 50.	ACK message header field ERR	66
Table 51.	Master configuration states	67
Table 52.	Slave configuration states.....	70
Table 53.	DSFI2 Slave Variables	71
Table 54.	Configuration states for DSFI2 slave variables	71

Table 55.	Mapping of DSFI2 slave variables	72
Table 56.	Mapping of DSFI2_DeviceId	72
Table 57.	DSFI2_ConfigState	73
Table 58.	DSFI2_InterruptStatus	73
Table 59.	DSFI2_InterruptMask	73
Table 60.	SetMasterId function	74
Table 61.	SetOperational function	74
Table 62.	SetConfig function	74
Table 63.	GetConfig function	75
Table 64.	SetDefaults function	75
Table 65.	GetDefaults function	75
Table 66.	GetInterrupt function	76
Table 67.	GetStatusVariables function	76
Table 68.	GetDeviceId function	76
Table 69.	DSFI2 error cases	77
Table 70.	Address space of DSFI2 I/O controller	78
Table 71.	INTERRUPT_STATUS states	79
Table 72.	INTERRUPT_MASK states	79
Table 73.	parameter_valid register	80
Table 74.	DSFI2 handler unit address space	83
Table 75.	INSTANCE_INTERRUPT mapping	84
Table 76.	INSTANCE_INTERRUPT states	84
Table 77.	DSFI2Header	84
Table 78.	SetOutputValue function	85
Table 79.	GetOutputValue function	85
Table 80.	RegisterTest function	85
Table 81.	GetInputValue function	85
Table 82.	GetDSFI2Header function	86
Table 83.	GetInstanceInterrupt function	86
Table 84.	STP special characters	89

About this Document

This user manual is intended only for system developers and integrators, it is not intended for end users.

It describes the design, functions and connection of the product. The manual does not include detailed information on individual components (data sheets etc.).



F403 product page with up-to-date information and downloads:
www.men.de/products/f403/

History

Issue	Comments	Date
E1	First issue	2013-11-28
E2	Added Chapter 2.3 Demo Application on page 20, Updated Chapter 2.2 Installing Driver Software on page 19 and Chapter 4.4 F403 Application Programming Interface (API) on page 35.	2013-12-17
E3	Updated format. Minor editorial changes and improved structure. Reworked Chapter 3.5 demo application.	2014-11-05
E4	General update, new structure	2016-07-19

Conventions



Indicates important information or warnings concerning situations which could result in personal injury, or damage or destruction of the component.



Indicates important information concerning electrostatic discharge which could result in damage or destruction of the component.



Indicates important information or warnings concerning proper functionality of the product described in this document.



The globe icon indicates a [hyperlink](#) that links directly to the Internet. When no globe icon is present, the hyperlink links to specific information within this document.

<i>Italics</i>	Folder, file and function names are printed in <i>italics</i> .
Mono	A monospaced font type is used for hexadecimal numbers, listings, C function descriptions or wherever appropriate. Hexadecimal numbers are preceded by "0x".
<i>Comment</i>	Comments embedded into coding examples are shown in green text.
IRQ# /IRQ	Signal names followed by a hashtag "#" or preceded by a forward slash "/" indicate that this signal is either active low or that it becomes active at a falling edge.
In/Out	Signal directions in signal mnemonics tables generally refer to the corresponding board or component, "in" meaning "to the board or component", "out" meaning "from the board or component".

Product Safety

Electrostatic Discharge (ESD)



Computer boards and components contain electrostatic sensitive devices. Electrostatic discharge (ESD) can damage components. To protect the PCB and other components against damage from static electricity, you should follow some precautions whenever you work on your computer.

- Power down and unplug your computer system when working on the inside.
- Hold components by the edges and try not to touch the IC chips, leads, or circuitry.
- Use a grounded wrist strap before handling computer components.
- Place components on a grounded antistatic pad or on the bag that came with the component whenever the components are separated from the system.
- Only store the product in its original ESD-protected packaging. Retain the original packaging in case you need to return the product to MEN for repair.

Legal Information

Changes

MEN Mikro Elektronik GmbH ("MEN") reserves the right to make changes without further notice to any products herein.

Warranty, Guarantee, Liability

MEN makes no warranty, representation or guarantee of any kind regarding the suitability of its products for any particular purpose, nor does MEN assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including, without limitation, consequential or incidental damages. TO THE EXTENT APPLICABLE, SPECIFICALLY EXCLUDED ARE ANY IMPLIED WARRANTIES ARISING BY OPERATION OF LAW, CUSTOM OR USAGE, INCLUDING WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE OR USE. In no event shall MEN be liable for more than the contract price for the products in question. If buyer does not notify MEN in writing within the foregoing warranty period, MEN shall have no liability or obligation to buyer hereunder.

The publication is provided on the terms and understanding that:

1. MEN is not responsible for the results of any actions taken on the basis of information in the publication, nor for any error in or omission from the publication; and
2. MEN is not engaged in rendering technical or other advice or services.

MEN expressly disclaims all and any liability and responsibility to any person, whether a reader of the publication or not, in respect of anything, and of the consequences of anything, done or omitted to be done by any such person in reliance, whether wholly or partially, on the whole or any part of the contents of the publication.

Conditions for Use, Field of Application

The correct function of MEN products in mission-critical and life-critical applications is limited to the environmental specification given for each product in the technical user manual. The correct function of MEN products under extended environmental conditions is limited to the individual requirement specification and subsequent validation documents for each product for the applicable use case and has to be agreed upon in writing by MEN and the customer. Should the customer purchase or use MEN products for any unintended or unauthorized application, the customer shall indemnify and hold MEN and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim or personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that MEN was negligent regarding the design or manufacture of the part. In no case is MEN liable for the correct function of the technical installation where MEN products are a part of.

Qualified Personnel

The product/system described in this documentation may be operated only by personnel qualified for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Conformity

MEN products are no ready-made products for end users. They are tested according to the standards given in the Technical Data and thus enable you to achieve certification of the product according to the standards applicable in your field of application.

RoHS

Since July 1, 2006 all MEN standard products comply with RoHS legislation.

Since January 2005 the SMD and manual soldering processes at MEN have already been completely lead-free. Between June 2004 and June 30, 2006 MEN's selected component suppliers have changed delivery to RoHS-compliant parts. During this period any change and status was traceable through the MEN ERP system and the boards gradually became RoHS-compliant.

WEEE Application



The WEEE directive does not apply to fixed industrial plants and tools. The compliance is the responsibility of the company which puts the product on the market, as defined in the directive; components and sub-assemblies are not subject to product compliance.
In other words: Since MEN does not deliver ready-made products to end users, the WEEE directive is not applicable for MEN. Users are nevertheless recommended to properly recycle all electronic boards which have passed their life cycle.

Nevertheless, MEN is registered as a manufacturer in Germany. The registration number can be provided on request.

Copyright © 2016 MEN Mikro Elektronik GmbH. All rights reserved.

Germany

MEN Mikro Elektronik GmbH
Neuwieder Straße 3-7
90411 Nuremberg
Phone +49-911-99 33 5-0

info@men.de
www.men.de

France

MEN Mikro Elektronik SAS
18, rue René Cassin
ZA de la Châtelaine
74240 Gaillard
Phone +33-450-955-312

info@men-france.fr
www.men-france.fr

USA

MEN Micro Inc.
860 Penllyn Blue Bell Pike
Blue Bell, PA 19422
Phone 215-542-9575

sales@menmicro.com
www.menmicro.com

China

MEN Mikro Elektronik (Shanghai) Co., Ltd.
Room 808-809, Jiaxing Mansion, No. 877 Dongfang Road
200122 Shanghai
Phone +86-21-5058-0961

sales@men-china.cn
www.men-china.cn

1 Product Overview

1.1 Product Description

Binary I/O for Railway

The F403 is a binary I/O CompactPCI board especially designed for railway applications. The card is used for input/output of digital signals with different voltage levels and ground references. It supports 16 bidirectional digital input/output channels, which are separated into four optically isolated groups.

EN 50155 Compliance

Its voltage range from 14.4 VDC to 154 VDC and its current output of 1 A at 24 V complies with EN 50155, which makes the board ready for immediate use in train applications.

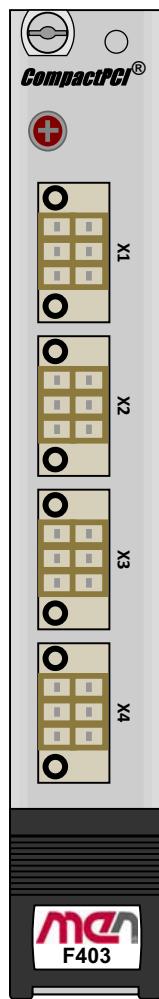
Safety Measures for Mobile Environments

The four front connectors are implemented by using spring cage terminal blocks causing only low wiring outlay and supporting fast installation.

The binary railway I/O supports all safety measures necessary for mobile environments like trains including voltage and temperature supervision and readback of outputs.

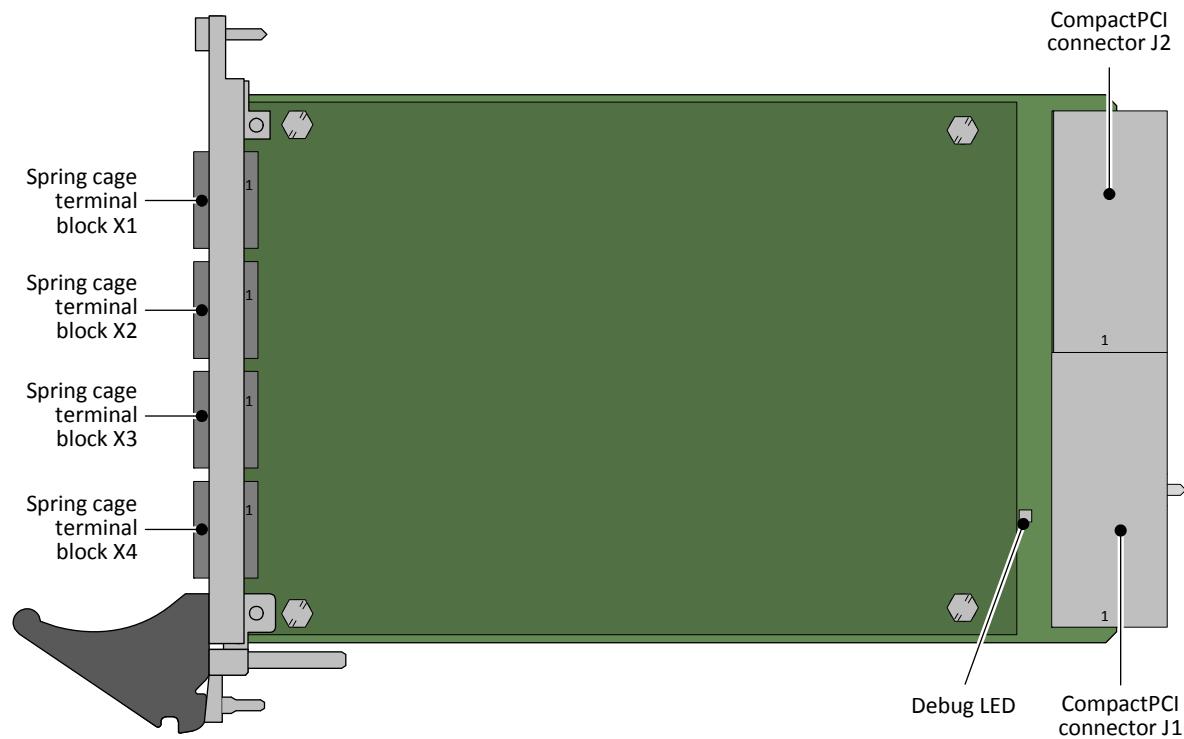
1.2 External Interfaces

Figure 1. Front interfaces



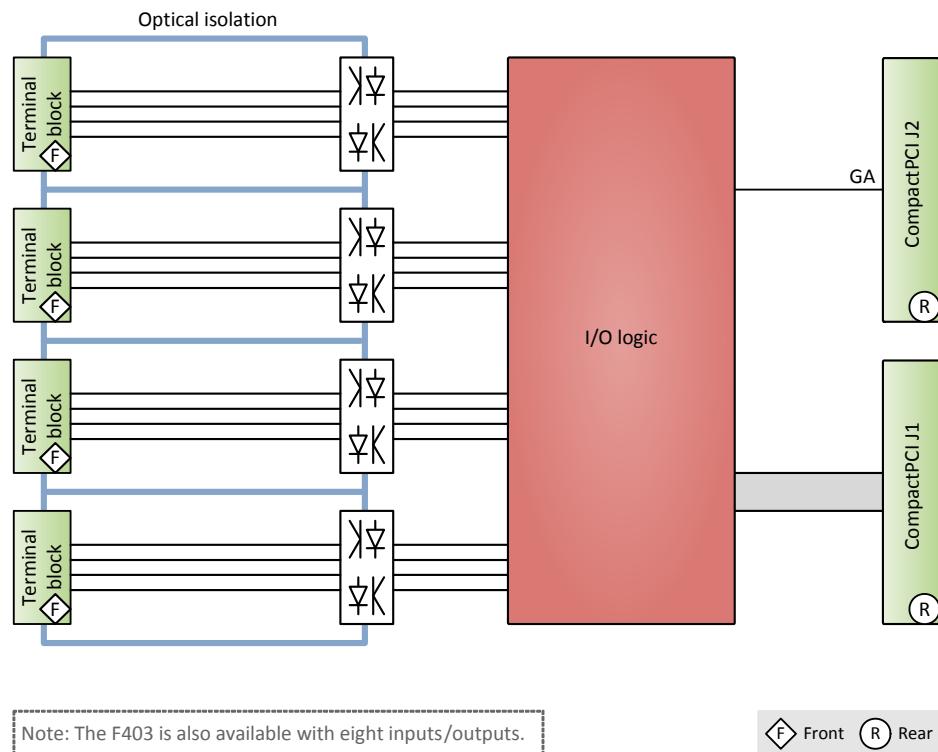
1.3 Board Layout

Figure 2. Board layout - top view



1.4 Block Diagram

Figure 3. Block diagram



1.5 Technical Data

Binary I/Os

- 16 binary signals
 - 4 optically isolated groups
 - 4 channels for each group
- The following I/O configurations are possible:
 - 16 inputs/outputs, or
 - 16 inputs (no outputs), or
 - 8 inputs/outputs
- Individual edge-triggered interrupts
- High-side output switches
- High output current: max. 1 A per channel at 24 V
- Temperature and voltage supervision

Output Characteristics

- Output voltage range
 - Limits continuous: 0 VDC to +138 VDC
 - Limits (duration <1s): 0 VDC to +154 VDC
- Switching time for output change: min. 400 µs (rise time) / min. 600 µs (fall time)

Input Characteristics

- Input voltage range
 - Limits continuous: -0.7 VDC to +138 VDC
 - Limits (duration <1s): -0.7 VDC to +154 VDC
- Input voltage of external supply voltage
 - Can be configured individually for each group
 - Nominal: +24 VDC to +110 VDC
 - Limits continuous: +16.8 VDC to +138 VDC
 - Limits (duration <1s): +14.4 VDC to +154 VDC
- Switching threshold: 40% (+15%/-15%) of external supply voltage

Front Interfaces

- 4 spring cage terminal blocks

CompactPCI Bus

- Compliance with CompactPCI Core Specification PICMG 2.0 R3.0
- Peripheral slot
- 32-bit/33-MHz PCI Bus, or
- 32-bit/66-MHz PCI Bus
- V(I/O): +3.3 V
- J2 connector with geographical addressing for distinguishing boards in a system with several boards

Electrical Specifications

- Isolation voltage:
 - 1500 VAC between isolated side and digital side
 - 1500 VAC between the channels
- Supply voltage/power consumption from CompactPCI:
 - +5V (+5%/-5%), 130 mA typ.

Mechanical Specifications

- Dimensions: conforming to CompactPCI specification for 3U boards
- Front panel: 4 HP with ejector
- Weight: 292 g

Environmental Specifications

- Temperature range (operation):
 - -40..+85°C (qualified components)
 - Airflow: 1.0 m/s
- Temperature range (storage): -40..+85°C
- Relative humidity (operation): max. 95% non-condensing
- Relative humidity (storage): max. 95% non-condensing
- Altitude: -300 m to +3000 m
- Shock: 50 m/s², 30 ms (EN 61373)
- Vibration (function): 1 m/s², 5 Hz to 150 Hz (EN 61373)
- Vibration (lifetime): 7.9 m/s², 5 Hz to 150 Hz (EN 61373)
- Conformal coating (standard)

MTBF

- 418 612 h @ 40°C according to IEC/TR 62380 (RDF 2000)

Safety

- Flammability
 - UL 94V-0
- Electrical Safety
 - Insulation measurement test according to EN 50155 (10.2.9.1)
 - Voltage withstand test according to EN 50155 (10.2.9.2)
 - Information technology equipment test according to EN 60950

EMC Conformity

- EN 55011 (radio disturbance)
- IEC 61000-4-2 (ESD)
- IEC 61000-4-3 (electromagnetic field immunity)
- IEC 61000-4-4 (burst)
- IEC 61000-4-5 (surge)
- IEC 61000-4-6 (conducted disturbances)

Software Support

- The F403 is supported by standard OS UART drivers



See the MEN website for a demo application for Linux, Windows and VxWorks:
www.men.de/products/f403/#downl

1.6 Product Identification

MEN user documentation may describe several different models and/or design revisions of the F403. You can find information on the article number, the design revision and the serial number on a label affixed to the board.

- Article number:** Indicates the product family and model. This is also MEN's ordering number. To be complete it must have 9 characters.
- Revision number:** Indicates the design revision of the product.
- Serial number:** Unique identification assigned during production.

If you need support, you should communicate these numbers to MEN.

Figure 4. Product labels



2 Getting Started

2.1 Connecting and Starting

You can use the following check list when installing the board in a system for the first time and with minimum configuration.

- » Power down the system.
- » Insert the F403 into a peripheral slot of your CompactPCI system, making sure that the backplane connectors are properly aligned.

Note: The peripheral slots of every CompactPCI system are marked by a circle on the backplane and/or at the front panel.

- » Power up the system.

2.2 Installing Driver Software

The F403 does not need any specific driver, since the 16550-compatible UART driver of the operating system can be used.

See [Chapter 4 Hardware/Software Interface on page 27](#) for more information.

The UART driver of the operating system has to be configured specifically for the F403. Port address, port name, IRQ and UART type have to be set.

A demo application for Linux, Windows and VxWorks is available.



See the MEN website for the demo application:
www.men.de/products/f403/#downl

See [Chapter 2.3 Demo Application on page 20](#) for configuration instructions.

The following UART types are supported:

- 16450: no FIFO
- 16550/16550A: 16-byte FIFO
- 16650: 32-byte FIFO
- 16750: 64-byte FIFO
- 16850: 128-byte FIFO

As the UART is only a virtual UART, physical parameters like baudrate, base clock, etc. are not relevant.

The UART interface can either be operated on memory mapped BAR or on I/O mapped BAR (with the default driver of operating system).

See [Chapter 4.1 PCI Configuration on page 27](#) for the PCI address map.

2.3 Demo Application

The demo application 13Y020-06 shows how to use the F403 with the DSFI2 API. It is a sample application for the input and output functions of the BINIO.



See the MEN website for the demo application:
www.men.de/products/f403/#downl

The demo application supports Linux, Windows and VxWorks.

Before using the demo application you have to configure the geographical address and UART connection.

2.3.1 Configuring the Geographical Address

The geographical address of the PCI slot[0..31] must be given in file *main.c*. In *main.c*, the address variable is called *G_GeoAddress* (line number 78). Change the value for the selected PCI slot.

The protocol reads the geographical address from the hardware and verifies it by comparing it with the address provided by the user. The F403 APIs will be available to communicate only if the geographical address matches the address provided by the user. Otherwise an error message will be displayed.

2.3.2 Configuring the UART Interface

The UART configuration of a communication interface is operating system specific.

2.3.2.1 Linux

For Linux the *setserial* command line tool is used to configure the UART interface. The sample UART configuration for Linux is given in the file *f403_init.sh*.

- » Required packages: *setserial*, *stty* (e.g., run *apt-get install setserial* from terminal to install *setserial* package).
- » Open terminal as root.
- » Adapt *setserial* and *stty* parameters in *f403_init.sh*.
- » Update the interface name */dev/ttysX* in the protocol configuration file (*protocol_config.h*, line number 74).

```
#else
static interface_t ifaceArray[] = {
    {"/dev/ttys0", IF_1},
    {"/dev/ttys1", IF_2},
};
#endif
```

The protocol stack needs to know which interface is configured. It gets the interface configuration from file *protocol_config.h*. In this file, more than one interface can be configured, along with a reference (e.g. *IF_1*). The configured interface reference has to be used in the API to access the interface over the F403 API.

2.3.2.2 Windows

The application uses the *MODE* command-line tool to configure the UART interface.

- » Configure the UART in file *main.c* (line number 117ff.):

```
#if defined(WIN32) || defined(_WIN32) || defined(__WIN32)
    HANDLE consInput;
    consInput = GetStdHandle(STD_INPUT_HANDLE);
    system("MODE COM3 BAUD=115200 PARITY=N RETRY=N DATA=8 STOP=1
to=on xon=off odsr=off octs=off dtr=off rts=off idsr=off");
    system("cls");
```

- » Update the interface name *COMx* in the protocol configuration file *protocol_config.h*, so the protocol stack knows which interface is configured (line number 66ff.):

```
#if defined(WIN32) || defined(_WIN32) || defined(__WIN32)
static interface_t ifaceArray[] = {
    {"COM3", IF_1},
};
```

The protocol stack needs to know which interface is configured. It gets the interface configuration from file *protocol_config.h*. In this file, more than one interface can be configured, along with a reference (e.g. *IF_1*). The configured interface reference has to be used in the API to access the interface over the F403 API.

2.3.2.3 VxWorks

For VxWorks there is no need to configure a UART interface. The default settings can be used.

- » Update the interface name `/tyCo/x` in the protocol configuration file (*protocol_config.h*, line number 70), so the protocol stack knows which interface is configured.

```
#elif defined(__VxWorks)
static interface_t ifaceArray[] = {
    {"/tyCo/1", IF_1},
};
```

The protocol stack needs to know which interface is configured. It gets the interface configuration from file *protocol_config.h*. In this file, more than one interface can be configured, along with a reference (e.g. `IF_1`). The configured interface reference has to be used in the API to access the interface over the F403 API.

2.3.3 Compiling and Running the Demo Application

Once you have made your configurations, you have to compile the source code:

- » Call *make/compile.bat* as per the operating system. This generates binary file *f403_example*.
- » Execute the binary to run the demo application.

3 Functional Description

The following describes the individual functions of the F403. Detailed information regarding individual components can be obtained from the data sheets or data books of the semiconductor manufacturer concerned.

3.1 Power Supply

The F403 is supplied with +5V via CompactPCI connector J1.

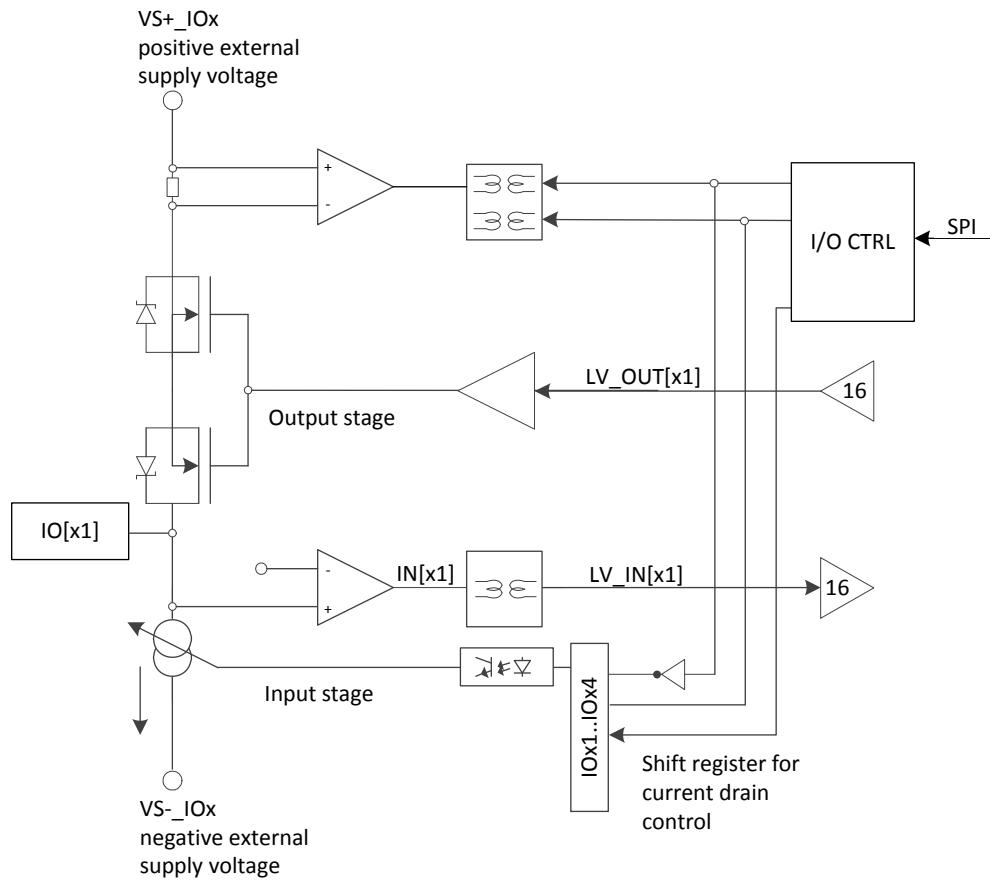
3.2 Binary I/O

The F403 supports 16 bidirectional digital input/output channels, which are separated into four optically isolated groups of four I/O channels each.

3.2.1 Principle of Operation

- See [Figure 3, Block diagram on page 15](#) for a general block diagram including isolation.
- See [Chapter 4.5.15.2.11 SetConfig on page 74](#) on how to configure the 16 binary inputs/outputs individually.

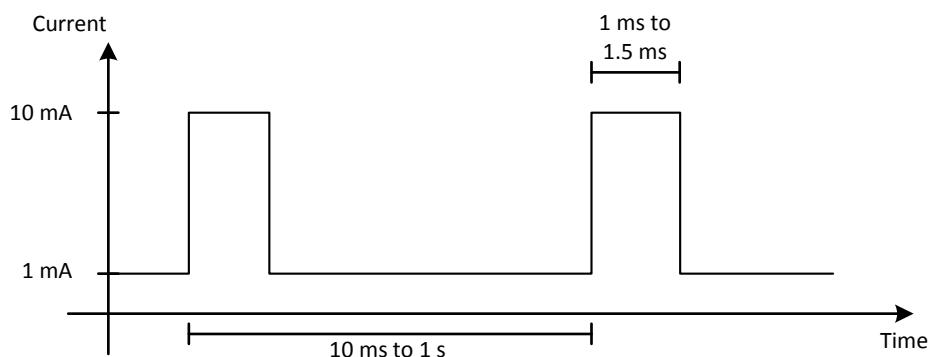
The isolated logic supply is regulated in a way that the positive rail is always VS+_IOx and the logic ground is always 5V +/-10% below VS+_IOx.

Figure 5. I/O stage

3.2.2 Pulsed Input Current

A pulsed current of typically 10 mA flows for approximately 1 ms, with a continuous current of 1.2 mA minimum in between the pulses. The pulse frequency can be set in steps of 10 Hz within the range of 1 Hz to 100 Hz (valid settings: 1, 10, 20..100 Hz).

See also [Chapter 4.5.15.2.11 SetConfig on page 74](#).

Figure 6. Pulsed input current

3.2.3 Front Connectors

The F403 provides four spring cage terminal blocks at the front panel.

Connector type:

- 6-pin spring cage terminal block, 3.5 mm

Mating connector:

- 6-pin printed-circuit board connector, 3.5 mm
e.g. Phoenix Contact DFMC 1.5/ 3-ST-3.5-LR (article number 1790496) or DFMC 1.5/ 3-STF-3.5 (article number 1790302)

Table 1. Pin assignment of front connector X1

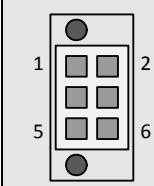
	1 VS+_IO1 3 IO[11] 5 IO[13]	2 VS-_IO1 4 IO[12] 6 IO[14]
---	-----------------------------------	-----------------------------------

Table 2. Pin assignment of front connector X2

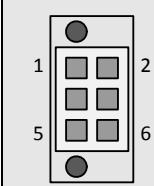
	1 VS+_IO2 3 IO[21] 5 IO[23]	2 VS-_IO2 4 IO[22] 6 IO[24]
--	-----------------------------------	-----------------------------------

Table 3. Pin assignment of front connector X3

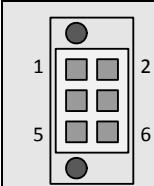
	1 VS+_IO3 3 IO[31] 5 IO[33]	2 VS-_IO3 4 IO[32] 6 IO[34]
---	-----------------------------------	-----------------------------------

Table 4. Pin assignment of front connector X4

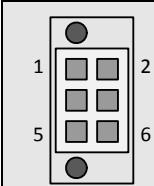
	1 VS+_IO4 3 IO[41] 5 IO[43]	2 VS-_IO4 4 IO[42] 6 IO[44]
---	-----------------------------------	-----------------------------------

Table 5. Signal mnemonics of front connectors X1 to X4

Signal	Direction	Function
VS+_IO[1:4]	in	Power input of I/O groups 1 to 4 +24 VDC to +110 VDC nominal
VS-_IO[1:4]	in	Power ground of I/O groups 1 to 4
IO[11:14]	in/out	Binary I/O of I/O group 1
IO[21:24]	in/out	Binary I/O of I/O group 2
IO[31:34]	in/out	Binary I/O of I/O group 3
IO[41:44]	in/out	Binary I/O of I/O group 4

3.3 Debug LED

See [Figure 2, Board layout – top view on page 14](#) for the position of the debug LED.

The debug LED indicates the firmware load status (factory or user image is loaded).

Table 6. Debug LED

Debug LED		Description
green	red	
on	on	The firmware is programmed with the factory image and no configuration error has occurred yet.
on	off	The firmware is programmed with the application image.
off	on	The firmware has returned to factory image after a configuration error.
off	off	The firmware is not loaded.

3.4 CompactPCI

The F403 supports a 32-bit 33-MHz CompactPCI interface fully compatible with CompactPCI specification PICMG 2.0 Rev. 3.0. The board works with 3.3V and tolerates 5V I/O.

Connector type of J1:

- 110-pin shielded, 2mm-pitch, 5-row receptacle according to IEC 917 and IEC 1076-4-101

The pin assignment of connector J1 as defined in the CompactPCI specification will not be repeated here.

3.4.1 Geographical Addressing

The F403 also provides CompactPCI J2 connector for geographical addressing. It provides the GA signals GA0 to GA4.

4 Hardware/Software Interface

This chapter is intended for software developers or board integrators who need deeper knowledge of the implementation details of the F403 interfaces and its internal connections.

4.1 PCI Configuration

The F403 has the following IDs on the PCI bus:

- PCI Device ID: 0x4D45
- PCI Vendor ID: 0x1A88
- Subsystem Device ID: 0x5A14
- Subsystem Vendor ID: 0xbc

Table 7. PCI address map

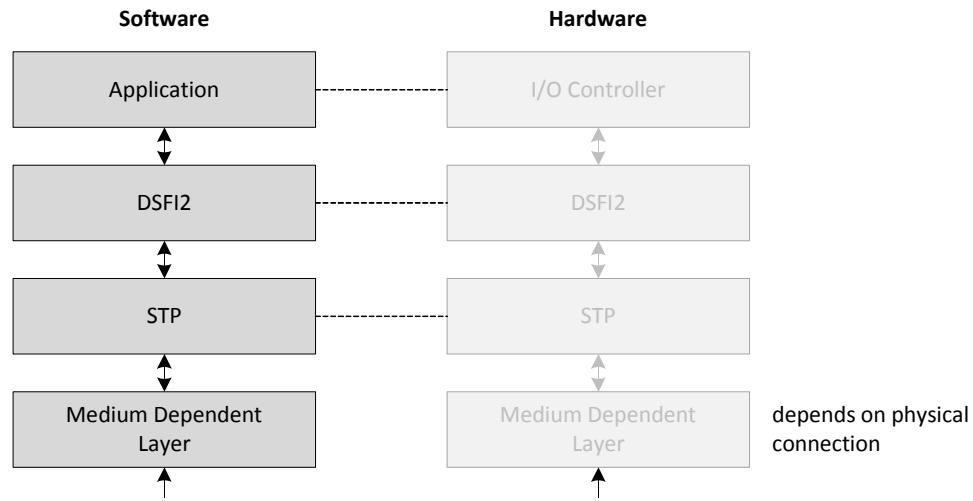
Offset	Size	BAR	Mapped	Description
0x0000	0x200	0	Memory (non-prefetched)	Chameleon table (FPGA)
0x0200	0x8	0	Memory (non-prefetched)	Serial Flash Controller
0x0280	0x80	0	Memory (non-prefetched)	Virtual UART interface of <i>DSFI2_HANDLER</i>
0x0000	0x8	1	I/O	Virtual UART interface of <i>DSFI2_HANDLER</i>

4.2 Protocol Layer Overview

The F403 appears on the PCI bus as a standard UART interface providing a 16550-compatible register interface. This ensures easy access to the board using all common operating systems.

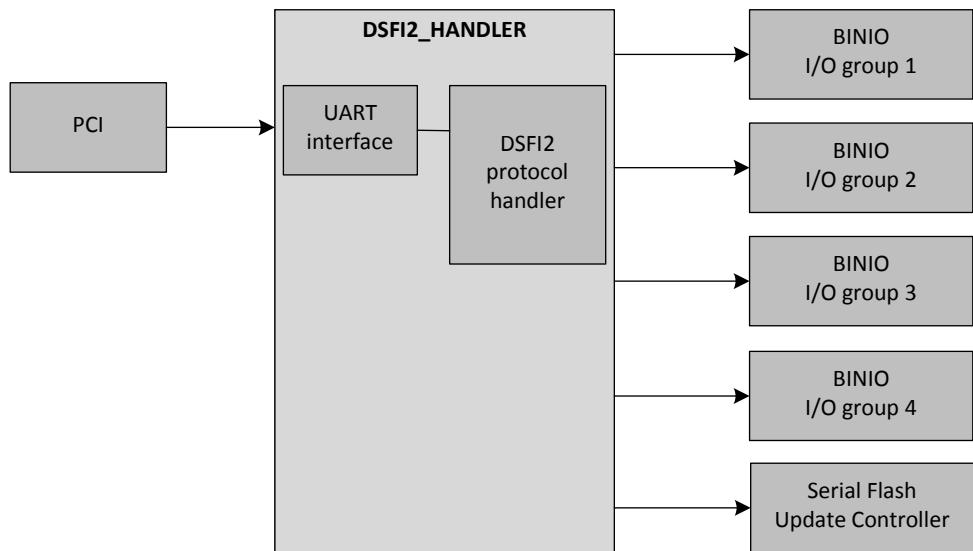
The communication between the software and the F403 is handled by the DSFI2 protocol stack, which consists of the DSFI2 protocol layer and an underlying protocol which is optimized for byte-based communication (STP protocol layer).

Figure 7. Protocol layer communication



The F403 is a DSFI slave. The F403 implements up to four I/O controllers for binary I/O.

Figure 8. Hardware-software interface to F403



4.2.1 DSFI2 PCI Device

The F403 PCI device uses one of the following methods for identification on the PCI bus:

Table 8. DSFI2 PCI device

Vendor ID	Device ID	Description
0x1A88	0x4D45	Chameleon table at BAR 0 (memory mapped) offset 0x0 Chameleon table includes a 16Z099-00 instance which is used as DSFI2 handler.
0x1A88	0x4432	Virtual UART with 16Z099-00 DSFI2 handler at BAR 0 (IO mapped) offset 0x0

The F403 contains a 32-bit PCI to Wishbone bridge. The PCI configuration header corresponds to the following table.

Table 9. PCI configuration header

Address	Byte					
	3	2	1	0		
00	Device ID (0x4D45)		Vendor ID (0x1A88)			
04	Status Register		Command Register			
08	Class Code (0x068000)			Revision ID 0xXX		
0C	BIST	Header Type	Latency Timer	Cache Line Size		
10	Base Address Register 0					
14	Base Address Register 1					
18	Base Address Register 2					
1C	Base Address Register 3					
20	Base Address Register 4					
24	Base Address Register 5					
28	Card Bus CIS Pointer					
2C	Subsystem ID (0x5A14)		Subsystem Vendor ID (0x00BC)			
30	Expansion ROM Base Address Register					
34	Reserved					
38	Reserved					
3C	Maximum Latency	Minimum Grant	Interrupt Pin 0x01	Interrupt Line		

4.3 I/O Controller Functions

The F403 binary I/O board can be operated by F403 API functions.

See [Chapter 4.4 F403 Application Programming Interface \(API\) on page 35](#) for the F403 features and corresponding F403 API functions.

4.3.1 Basic I/O Functions

Each I/O channel of an I/O controller instance can be configured as output or input only. The direction of I/O channels can be set by configuring the I/O controller instances with *F403_Config()*.

See [Chapter 4.4.1 F403_Config\(\) on page 36](#) for more information on the *F403_Config()* function.

4.3.1.1 Inputs

The input values of all channels of an instance are read by *F403_GetGroupInputs()*. The input value of an I/O channel is read even if a channel is configured as output.

See [Chapter 4.4.4 F403_GetGroupInputs\(\) on page 41](#) for more information on the *F403_GetGroupInputs()* function.

4.3.1.2 Outputs

The output values of all channels of an I/O group are set by *F403_SetGroupOutputs()*. A single I/O channel configured as output can be set by *F403_SetOutput()*.

The output value of a channel can be set regardless of its output mode (input only/output).

- See [Chapter 4.4.3 F403_SetGroupOutputs\(\) on page 40](#) for more information on the *F403_SetGroupOutputs()* function.
- See [Chapter 4.4.2 F403_SetOutput\(\) on page 39](#) for more information on the *F403_SetOutput()* function.

4.3.2 Output Watchdog

The output watchdog of I/O channels configured as output detects failures in communication with the I/O instance. When the output watchdog of an output channel is enabled, the output channel must be set by *F403_SetGroupOutputs()* or *F403_SetOutput()* periodically within the watchdog timeout period. The output watchdog can be enabled and configured by setting the corresponding parameter in the configuration of F403 with *F403_Config()*.

See [Chapter 4.4.3 F403_SetGroupOutputs\(\) on page 40](#) for more information on the *F403_SetGroupOutputs()* function.

See [Chapter 4.4.2 F403_SetOutput\(\) on page 39](#) for more information on the *F403_SetOutput()* function.

See [Chapter 4.4.1 F403_Config\(\) on page 36](#) for more information on the *F403_Config()* function.

If the output channel is not triggered within the watchdog timeout, a watchdog failure is detected and

- the output channel is disabled
- the failure is indicated as DSFI2 interrupt if the corresponding interrupt is enabled in configuration
- the failure can be read as output watchdog failure channel error by *F403_GetGroupStatus()*
- the failure can be cleared by *F403_ClearGroupError()*

▪ See [Chapter 4.4.5 F403_GetGroupStatus\(\) on page 42](#) for more information on the *F403_GetGroupStatus()* function.

▪ See [Chapter 4.4.6 F403_ClearGroupError\(\) on page 44](#) for more information on the *F403_ClearGroupError()* function.

4.3.3 Output Comparison

The comparison of I/O channels configured as output with the corresponding input channel detects failures of I/O channels. The comparison can be enabled by setting the corresponding parameter in the configuration of I/O controller instances with *F403_Config()*.

See [Chapter 4.4.1 F403_Config\(\) on page 36](#) for more information on the *F403_Config()* function.

If the input value of an output channel does not match the output value, a failure is detected and

- the output channel is disabled
 - the failure is indicated as DSFI2 interrupt if the corresponding interrupt is enabled in configuration
 - the failure can be read as FET failure by
 - the failure can be cleared by *F403_ClearGroupError()*
-
- See [Chapter 4.4.5 F403_GetGroupStatus\(\) on page 42](#) for more information on the *F403_GetGroupStatus()* function.
 - See [Chapter 4.4.6 F403_ClearGroupError\(\) on page 44](#) for more information on the *F403_ClearGroupError()* function.

4.3.4 Output Overload

I/O channels are monitored for channel overload. This feature is always enabled.

If an overload error of an output channel is detected,

- the output channel is disabled
 - the failure is indicated as DSFI2 interrupt if the corresponding interrupt is enabled in configuration
 - the failure can be read as overload failure by *F403_GetGroupStatus()*
 - the failure can be cleared by *F403_ClearGroupError()*
-
- See [Chapter 4.4.5 F403_GetGroupStatus\(\) on page 42](#) for more information on the *F403_GetGroupStatus()* function.
 - See [Chapter 4.4.6 F403_ClearGroupError\(\) on page 44](#) for more information on the *F403_ClearGroupError()* function.

4.3.5 Voltage Supervision

The following failures on the F403 are monitored and the status can be read by *F403_GetGroupStatus()*.

See [Chapter 4.4.5 F403_GetGroupStatus\(\) on page 42](#) for more information on the *F403_GetGroupStatus()* function.

Table 10. Voltage supervision

Failure	Description	Outputs
Overvoltage	Overvoltage on I/O channel supply voltage, I/O channel supply voltage over 154 V	No effect
Power not good	Power fail I/O channel supply voltage under 15.8 V	All outputs of I/O controller instance are disabled
Overvoltage on 5 V	Overvoltage on digital supply voltage (isolated 5 V) used for status monitoring	No effect
Shift error	Status monitoring failure	All outputs of I/O controller instance are disabled

If a voltage supervision error is detected,

- the failure is indicated as DSFI2 interrupt if the corresponding interrupt is enabled in configuration
- the failure can be read by *F403_GetGroupStatus()*
- the failure can be cleared by *F403_ClearGroupError()*

- See [Chapter 4.4.5 F403_GetGroupStatus\(\) on page 42](#) for more information on the *F403_GetGroupStatus()* function.
- See [Chapter 4.4.6 F403_ClearGroupError\(\) on page 44](#) for more information on the *F403_ClearGroupError()* function.

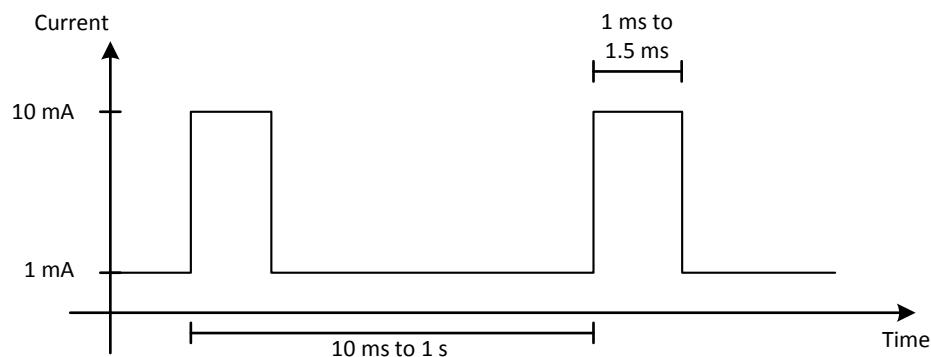
4.3.6 Current Drain Trigger

The input current of I/O channels configured as input (Current Drain Trigger, CDT) can be configured with the corresponding parameters in the configuration of I/O controller instances with *F403_Config()*.

See [Chapter 4.4.1 F403_Config\(\) on page 36](#) for more information on the *F403_Config()* function.

- CDT mode: continuously triggered, constant 1 mA, constant 10 mA
- CDT period: 1 Hz..100 Hz

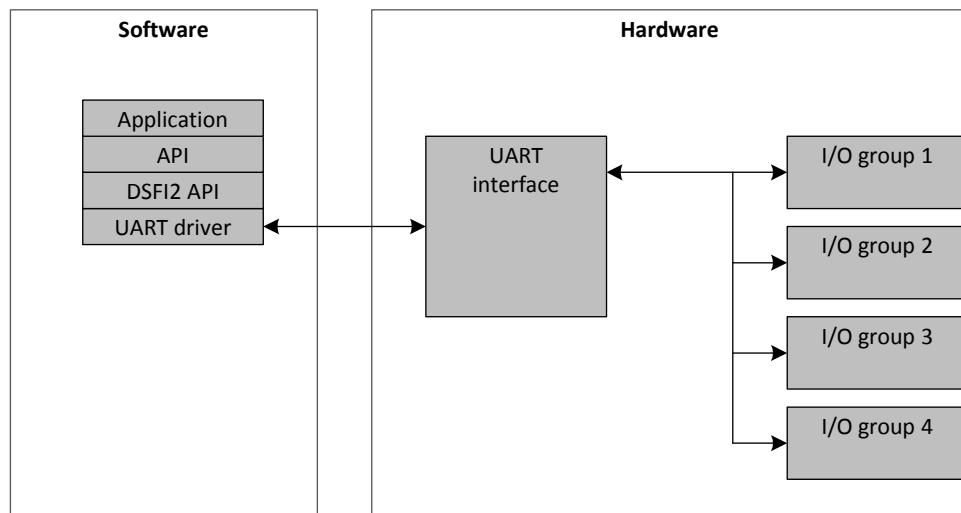
Figure 9. Current Drain Trigger



4.4 F403 Application Programming Interface (API)

The F403 API provides a set of functions to configure and operate the F403 binary I/O board. The API uses the UART interface of one F403 board and handles the DSFI2 protocol stack to operate the F403.

Figure 10. F403 API



All F403 API functions return the following error status.

Table 11. Error status for API functions

Configuration	Value	Description
<i>int textStatus</i>	<i>F403_SUCCESS</i>	Action was performed successfully. No communication error or slave status error occurred.
	<i>F403_ERROR</i>	An error occurred during the requested action.
	<i>F403_INVALID_DATA</i>	Invalid input parameter. Example: - a pointer value is NULL - the parameter array does not contain all the configurations of the I/O groups
	<i>F403_GEOADDR_ERROR</i>	The geographical address provided by the user does not match the F403 encoded address.
	Communication errors	An error code is generated in the communication.
<i>uint8_t intPending</i>	0 = no interrupt 1 = interrupt pending	Interrupt pending flag status is received in the acknowledgment.

4.4.1 F403_Config()

You can use this function to establish a connection to the F403 over DSFI2 protocol stack and configure all I/O controller instances and DSFI2 instance 0 (DSFI2 handler).

Perform the following steps for all I/O controller instances of F403 and DSFI2 instance 0 (DSFI2 handler):

I/O controller instances of the F403:

- » Configure instance by calling *DSFI2_Config()*

DSFI2 instance 0 (DSFI2 handler):

- » Call FN36 *Get DSFI2Header*
- » Check if the F403 is accessed
- » Check geographical address

Table 12. F403_Config() parameters per I/O channel

Parameter	Values	Description
Output mode	0 = input only, 1 = output	Sets the mode of the I/O channel
Output value	0 = open, 1 = closed	Sets the output value
Current drain trigger	Mode: 0 = constant 2 mA 1 = 10 mA 2 = triggered continuously 3 = triggered once Period: 0 = 100 Hz 1 = 90 Hz ... 9 = 10 Hz 10 = 1 Hz	Sets the current of the input channel
Output comparison	0 = disable, 1 = enable	Compares whether the output value is read back on the corresponding input channel. Disables output in case of error.
Output watchdog	0 = disable 1 = 8 ms 2 = 16 ms ... 15 = 120 ms	Disables the output channel if it is not set periodically. The timeout value can be configured.
Interrupt	0 = enable, 1= disable	Indication of asynchronous events: - Input value changed - I/O channel error - Common error

Table 13. F403_Config() common parameter

Parameter	Values	Description
Interrupt	0 = enable, 1= disable	Indication of asynchronous events: - Temperature failure - Voltage failure - Power-good failure

Parameters:

- Communication interface
- Geographical address
- Configuration parameter set of 4 I/O groups

Return values:

- Communication error code
- Interrupt pending flag

Function call:

```
struct F403_Status* F403_Config(uint8_t iFace, uint8_t geoAddr,
                                struct F403_BoardConfig_t * boardConfig);
```

Table 14. F403_Config() function arguments

Configuration Parameters	Description
<i>uint8_t iFace</i>	Communication interface
<i>uint8_t geoAddr</i>	Geographical address
<i>F403_BoardConfig_t * boardConfig</i>	Pointer to the configuration structure of all 4 I/O groups

Table 15. struct F403_IoGrpConfig

Configuration Parameters	Parameter Value (bitwise)	Description
<i>uint8_t ioGroup</i>	1 ... 4	I/O group number
<i>masterID</i>	-	Master ID
<i>uint8_t ioMode</i>	[3-0]: chann-4.. chann-1	Mode of the I/O channel in the form of channel mask
<i>uint8_t ioVal</i>	[3-0]: chann-4.. chann-1	Output value of the I/O channels in the form of channel mask
<i>uint8_t cdtMode</i>	[1-0]: channel-1 [3-2]: channel-2 [5-4]: channel-3 [7-6]: channel-4	Current drain trigger mode in the form of channel mask
<i>uint16_t cdtPeriod</i>	[3-0]: channel-1 [7-4]: channel-2 [11-8]: channel-3 [17-12]: channel-4	Current drain trigger period in the form of channel mask
<i>uint8_t outputCmp</i>	[3-0]: chann-4.. chann-1	Output comparison in the form of channel mask
<i>uint16_t outputWdt</i>	[3-0]: channel-1 [7-4]: channel-2 [11-8]: channel-3 [15-12]: channel-4	The timeout value of the watchdog timer in the form of channel mask
<i>uint8_t interrupt</i>	0 = enable, 1= disable	Interrupt enable/disable in the form of channel mask

Table 16. struct F403_BoardConfig_t

Configuration Parameters	Description
<i>F403_IoGrpConfig_t * ioGrp1Config</i>	Configuration parameters for I/O group 1
<i>F403_IoGrpConfig_t * ioGrp2Config</i>	Configuration parameters for I/O group 2
<i>F403_IoGrpConfig_t * ioGrp3Config</i>	Configuration parameters for I/O group 3
<i>F403_IoGrpConfig_t * ioGrp4Config</i>	Configuration parameters for I/O group 4
<i>uint8_t genlInterrupt</i>	General interrupt enable/disable

4.4.2 F403_SetOutput()

You can set a single output of one I/O controller instance by calling FN37 *SetOutputValue*.

Parameters:

- Communication interface
- I/O group number (1..4)
- Channel number (1..4)
- Output value (0 = open, 1 = closed)

Return values:

- Communication error code
- Interrupt pending flag

Function call:

```
struct F403_Status* F403_SetOutput(uint8_t iFace, uint8_t ioGroup, uint8
channel, uint8_t status);
```

Table 17. F403_SetOutput() function arguments

Configuration Parameters	Description
<i>uint8_t iFace</i>	Communication interface
<i>uint8_t ioGroup</i>	I/O group number
<i>uint8 channel</i>	Channel number of the I/O group
<i>uint8_t status</i>	status[bit 0] = 0: output switch open status[bit 0] = 1: output switch closed

4.4.3 F403_SetGroupOutputs()

You can set all outputs of one I/O controller instance at once by calling FN32 *SetOutputValues*.

Parameters:

- Communication interface
- I/O group number (1..4)
- Output value: 4 bit (0 = open, 1 = closed)

Return values:

- Communication error code
- Interrupt pending flag

Function call:

```
struct F403_Status* F403_SetGroupOutputs(uint8_t iFace, uint8_t ioGroup,
uint8_t status);
```

Table 18. F403_SetGroupOutputs() function arguments

Configuration Parameters	Description
<i>uint8_t iFace</i>	Communication interface
<i>uint8_t ioGroup</i>	I/O group number
<i>uint8_t status</i>	Value (bitwise): status[3..0] = chann-4..chann-1 status[x] = 0: output switch open status[x] = 1: output switch closed

4.4.4 F403_GetGroupInputs()

You can read all inputs of one I/O controller instance by calling FN34 *GetInputValues*.

Parameters:

- Communication interface
- I/O group number (1..4)

Return values:

- Input value: 4 bit (0 = low, 1 = high)
- Communication error code
- Interrupt pending flag

Function call:

```
struct F403_Status* F403_GetGroupInputs(uint8_t iFace, uint8_t ioGroup,
uint8_t * status);
```

Table 19. F403_GetGroupInputs() function arguments

Configuration Parameters	Description
<i>uint8_t iFace</i>	Communication interface
<i>uint8_t ioGroup</i>	I/O group number
<i>uint8_t * status</i>	Pointer to data buffer. The received input status will be saved here. Value (bitwise): status[3..0] = chann-4..chann-1 status[x] = 0: input low status[x] = 1: input high

4.4.5 F403_GetGroupStatus()

You can read the status and errors of one I/O controller instance and instance 0 (DSFI2 handler) by calling this function.

Perform the following steps for one I/O controller instance of the F403:

- » Read pending errors of I/O channels by calling FN128 *GetChannelError*
- » Read pending common errors by calling FN35 *GetError*
- » Clear pending errors by calling FN36 *ClearError*
- » Read voltage supervision status by calling FN130 *GetVsvStatus*

Perform the following step for DSFI2 instance 0 (DSFI2 handler):

- » Read internal status by calling FN35 *GetInputValue*

Parameters:

- Communication interface
- I/O group number (1..4)

Return values:

- Overload failure: 4 bit (0 = no error, 1 = error)
- FET failure: 4 bit (0 = no error, 1 = error)
- Output watchdog failure: 4 bit (0 = no error, 1 = error)
- I/O voltage supervision error: 4 bit (0 = no error, 1 = error)
- I/O voltage > 35 V (0 = < 35 V, 1 = > 35 V)
- Temperature failure (0 = no error, 1 = error)
- Supply voltage failure (0 = no error, 1 = error)
- FPGA status (0 = factory image, 1 = user image)
- Internal watchdog error (0 = no error, 1 = error)
- Communication error code
- Interrupt pending flag

Function call:

```
struct F403_Status* F403_GetGroupStatus(uint8_t iFace, uint8_t ioGroup,
                                         struct F403_ErrStatus_t * errStatus);
```

Table 20. F403_GetGroupStatus() function arguments

Configuration Parameters	Description
<i>uint8_t iFace</i>	Communication interface
<i>uint8_t ioGroup</i>	I/O group number
<i>F403_ErrStatus_t * errStatus</i>	Pointer to the <i>F403_ErrStatus</i> structure. Upon successful execution the structure will be updated.

Table 21. struct F403_ErrStatus_t for F403_GetGroupStatus()

Configuration Parameters	Description
<i>uint8_t overloadFail</i>	Overload failure Value (bitwise): overloadFail[3..0] = chann-4..chann-1 overloadFail[x] = 0: no error overloadFail[x] = 1: output overload error
<i>uint8_t fetFail</i>	FET failure Value (bitwise): fetFail[3..0] = chann-4..chann-1 fetFail[x] = 0: no error fetFail[x] = 1: output FET failure
<i>uint8_t wdtFail</i>	Output watchdog failure Value (bitwise): wdtFail[3..0] = chann-4..chann-1 wdtFail[x] = 0: no error wdtFail[x] = 1: output watchdog error
<i>uint8_t vsvError</i>	I/O voltage supervision error Value (bitwise): vsvError[0]: overvoltage error on I/O channel supply voltage vsvError[1]: reserved vsvError[2]: power fail of I/O channel supply voltage vsvError[3]: overvoltage on digital supply voltage (isolated 5 V) vsvError[x] = 0: no error vsvError[x] = 1: error pending
<i>uint8_t ioVoltage</i>	I/O voltage > 35 V Value: ioVoltage = 0: I/O channel supply voltage under 35 V ioVoltage = 1: I/O channel supply voltage over 35 V
<i>uint8_t temperatureFail</i>	Temperature failure Value: temperatureFail = 0: no error temperatureFail = 1: temperature failure
<i>uint8_t voltageFail</i>	Voltage failure Value: voltageFail = 0: no error voltageFail = 1: supply voltage failure
<i>uint8_t fpgaStatus</i>	FPGA status Value: fpgaStatus = 0: factory image loaded fpgaStatus = 1: user image loaded
<i>uint8_t intWdgFail</i>	Internal watchdog error Value: intWdgFail = 0: no error intWdgFail = 1: internal watchdog failure

4.4.6 F403_ClearGroupError()

You can clear pending errors of one I/O controller instance by calling this function.

Perform the following step for one I/O controller instance of the F403:

- » Clear pending errors by calling FN36 *ClearError*

Parameters:

- Communication interface
- I/O group number (1..4)
- Overload failure: 4 bit (0 = no effect, 1 = clear error)
- FET failure: 4 bit (0 = no effect, 1 = clear error)
- Output watchdog failure: 4 bit (0 = no effect, 1 = clear error)
- I/O voltage supervision error: 4 bit (0 = no effect, 1 = clear error)

Return values:

- Communication error code
- Interrupt pending flag

Function call:

```
struct F403_Status* F403_ClearGroupError(uint8_t iFace,
uint8_t ioGroup, struct F403_ClearError * clearError);
```

Table 22. F403_ClearGroupError() function arguments

Configuration Parameters	Description
<i>uint8_t iFace</i>	Communication interface
<i>uint8_t ioGroup</i>	I/O group number
<i>F403_ClearError * clearError</i>	Pointer to the <i>F403_ClearError</i> structure. Upon successful execution the structure will be updated.

Table 23. struct F403_ErrStatus_t for F403_ClearGroupError()

Configuration Parameters	Description
<i>uint8_t overloadFail</i>	Overload failure Value (bitwise): overloadFail[3..0] = chann-4..chann-1 overloadFail[x] = 0: no effect overloadFail[x] = 1: clear overload error
<i>uint8_t fetFail</i>	FET failure Value (bitwise): fetFail[3..0] = chann-4..chann-1 fetFail[x] = 0: no effect fetFail[x] = 1: clear output FET failure
<i>uint8_t wdtFail</i>	Output watchdog failure Value (bitwise): wdtFail[3..0] = chann-4..chann-1 wdtFail[x] = 0: no effect wdtFail[x] = 1: clear output watchdog error
<i>uint8_t vsvError</i>	I/O voltage supervision error Value (bitwise): vsvError[0]: Overvoltage error on IO channel supply voltage vsvError[1]: reserved vsvError[2]: Power fail of IO channel supply voltage vsvError[3]: Overvoltage on digital supply voltage (isolated 5V) vsvError[x] = 0: no effect vsvError[x] = 1: clear error

4.4.7 F403_InterruptStatus()

You can read and clear pending interrupts of the F403 by calling this function.

Perform the following steps:

- » Determine DSFI2 instances with pending interrupts by calling FN37 *GetInstanceInterrupt* of DSFI2 instance 0 (DSFI2 handler)
- » Read pending interrupts of all I/O controller instances by calling FN6 *GetInterrupt*
- » Clear pending interrupts of all I/O controller instances by calling FN9 *ClearInterrupt*
- » Read pending interrupts of DSFI2 instance 0 (DSFI2 handler) by calling FN6 *GetInterrupt*
- » Clear pending interrupts of DSFI2 instance 0 (DSFI2 handler) by calling FN9 *ClearInterrupt*

Parameter:

- Communication interface

Return values:

- Input value changed: 16 bit (0 = no interrupt, 1 = interrupt)
- I/O channel error: 16 bit (0 = no interrupt, 1 = interrupt)
- I/O common error: 16 bit (0 = no interrupt, 1 = interrupt)
- Temperature failure (0 = no interrupt, 1 = interrupt)
- Supply voltage failure (0 = no interrupt, 1 = interrupt)
- Communication error code
- Interrupt pending flag

Function call:

```
struct F403_Status* F403_InterruptStatus(uint8_t iFace,
    struct F403_InterruptStatus_t * intStatus);
```

Table 24. F403_InterruptStatus() function arguments

Configuration Parameters	Description
<i>uint8_t iFace</i>	Communication interface
<i>F403_InterruptStatus_t * intStatus</i>	Pointer to the <i>F403_InterruptStatus_t</i> structure. Upon successful execution the structure will be updated.

Table 25. struct F403_InterruptStatus_t for F403_InterruptStatus()

Configuration Parameters	Description
<i>uint8_t inputChange</i>	Input value changed Value (bitwise): wdtFail[3..0] = chann-4..chann-1 wdtFail[x] = 0: no effect wdtFail[x] = 1: clear output watchdog error
<i>uint8_t channelError</i>	I/O channel error
<i>uint8_t commonError</i>	I/O common error
<i>uint8_t temperatureFail</i>	Temperature failure
<i>uint8_t voltageFail</i>	Supply voltage failure

4.5 DSFI2 Layer

DSFI2 is a message-based communication protocol layer used to access the I/O controller on the F403.

Communication with the I/O controller is done via a virtual UART interface. For the host software it appears as a standard 16550 UART interface. Only the physics is omitted and the I/O controller is connected directly in order to avoid the high restrictions in performance when using a real UART interface, depending on the Baud rate setting.

Communication to the upper software layer is limited to a single path due to the UART interface. If more than one path is required to the upper layer interface of the DSFI2 layer, the upper layer is responsible for arbitration of the paths, e.g. the prioritization of requests from upper layers or the arbitration of multiple requests from upper layers.

The DSFI2 does not support buffering or retry of requests. This feature has to be implemented by upper level software if required.

The DSFI2 provides a unique software interface for different physical interfaces. A message-based communication protocol layer is used to access I/O controllers in an FPGA. An access is a transmission between a master and a slave. Each access consists of a request message (REQ) and a corresponding acknowledge message (ACK). The order of requests/acknowledges must not be changed.

Asynchronous events can be indicated by the slave by setting an interrupt flag in acknowledge messages (ACK). The DSFI2 layer has the following main tasks:

- Operation of the I/O controller with functions on a high abstraction level
- Addressing the I/O controller
- Identification of the I/O controller
- Assigning requests to the addressed I/O controller instance in a DSFI2 subsystem
- Reassigning acknowledges to the application which initiated the corresponding requests in a master implementation

4.5.1 Functions

Requests contain a function number that indicates which operation shall be performed. There are various functions that are divided as follows:

- Functions which have to be supported by every DSFI2 slave
- Functions which are specific to a single DSFI2 slave

A set of general functions is available in all I/O controllers. Additionally there are I/O controller specific functions.

4.5.2 Configuration

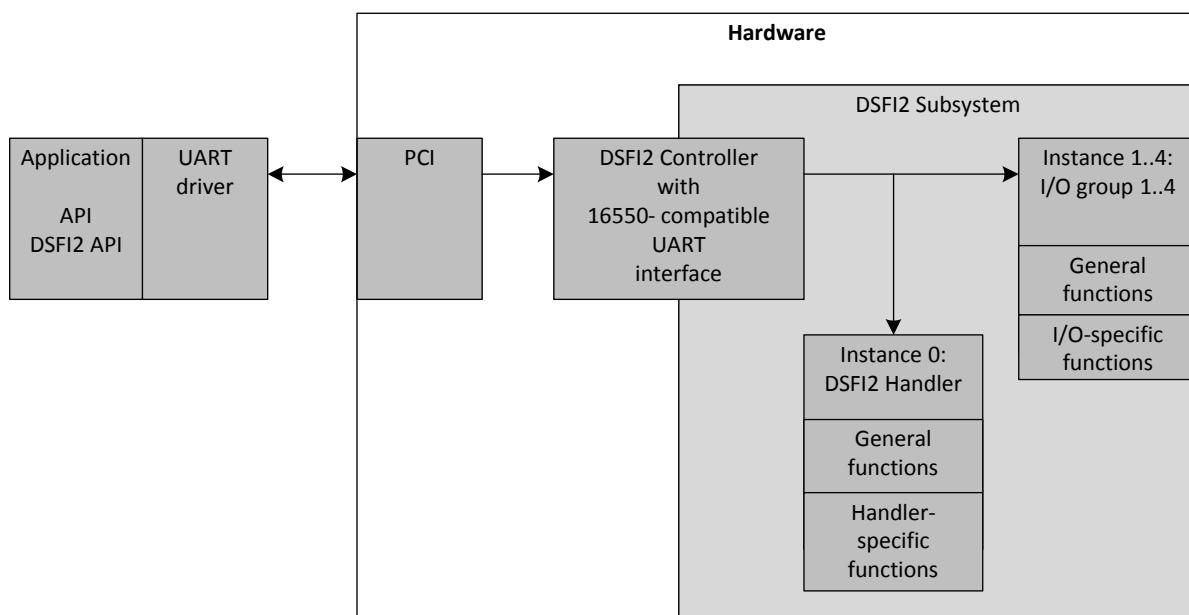
I/O controller operation is divided into a configuration phase and an operational phase. Configuration is performed by general functions that are equal for all DSFI2 I/O controllers.

An I/O controller is not operational until it has been configured by a DSFI2 master.

The DSFI2 layer is initialized by the *DSFI2.UL_initialize* function and configured by the *DSFI2.UL_parameter* function.

The DSFI2 protocol accesses the IP cores (DSFI2 instances) in the FPGA. Each IP core needs to be configured by the application before it can be used. For configuration and normal operation the application triggers predefined functions of the IP cores (e.g. set binary outputs). The F403 provides the following DSFI2 instances:

Figure 11. DSFI2 instances

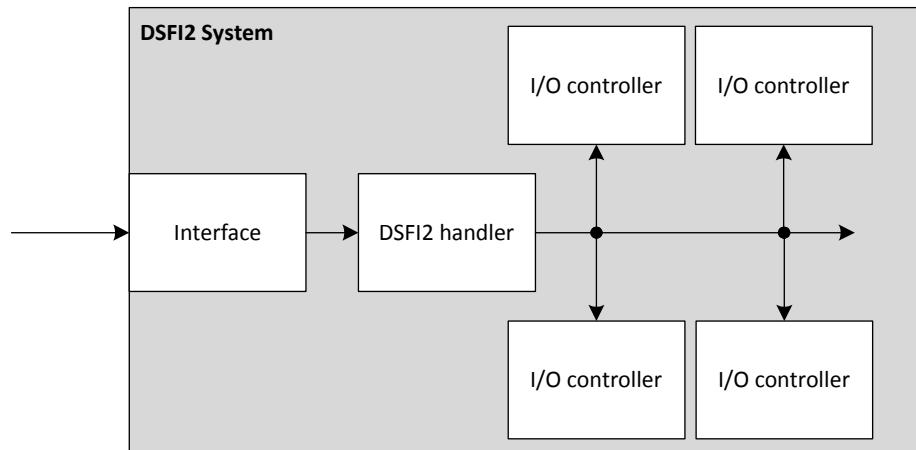


4.5.3 DSFI2 Roles

- Master: client sending request messages
- Slave: client sending acknowledge messages

A DSFI2 slave is an I/O controller behind the DSFI2 handler.

Figure 12. DSFI2 subsystem



4.5.3.1 DSFI2 Master

A DSFI2 master has to initialize a DSFI2 connection. The DSFI2 subsystem is scanned for available I/O controller instances, unless they are already known. The result is provided as a DSFI2 table.

A DSFI2 master gets one or more requests by the upper layer. The DSFI2 requests are assembled into DSFI2 request messages and handed to the lower layer as payload.

Received data from the lower layer is stored and disassembled into single acknowledges. Each acknowledge is evaluated for error or interrupt information and reassigned to the corresponding request. The acknowledge payload and status are sent back to the upper layer.

Interrupts that are transmitted in acknowledges are indicated to the upper layer.

4.5.3.2 DSFI2 Slave

A DSFI2 slave has access to the FPGA I/O controllers (upper layer).

Received data from the lower layer is disassembled into single requests. Each request is evaluated and the resulting read/write accesses to the addressed I/O controller are performed.

Read data and status are assembled to a DSFI2 acknowledge. If interrupts are pending, the header of acknowledge is extended by interrupt information and replied to the lower layer.

4.5.4 I/O Controller Identification

In the DSFI2 protocol each I/O controller provides identification information by itself. An instance-based addressing is used to identify all available I/O controllers without the need for the user to provide additional information.

The DSFI2 controller for the F403 is a virtual UART interface with DSFI2 subsystem. The DSFI2 subsystem is scanned for I/O controllers. Starting with DSFI2 instance 0, the device register of each instance is read until 0x3F is read (no device). The available I/O controller instances in the DSFI2 subsystem are provided as a DSFI2 table:

Table 26. DSFI2 instances

Instance	ID	Description	Detailed Specification
0	099	MISC	DSFI2_HANDLER including virtual UART interface
1	124	BINIO 1	Binary I/O controller for I/O group 1
2	124	BINIO 2	Binary I/O controller for I/O group 2
3	124	BINIO 3	Binary I/O controller for I/O group 3
4	124	BINIO 4	Binary I/O controller for I/O group 4
5	126	FLASH	Flash interface for update

I/O controller identification can be omitted if DSFI2 configuration is known.

4.5.5 DSFI2 Integration on F403

The gpin input of the MISC unit of the *DSFI2_HANDLER* is connected as follows.

Table 27. MISC gpin connections

gpin[x]	Connected signal
0	<i>t_fail_n</i>
1	<i>v_fail_n</i>
2	<i>pwgood</i>
3	<i>board_status(0)</i>
4	<i>board_status(1)</i>
else	0b0

The interrupt_status input of the MISC unit of the *DSFI2_HANDLER* is connected as follows.

Table 28. MISC interrupt_status connections

interrupt_status[x]	Connected signal
0	NOT <i>t_fail_n</i>
1	NOT <i>v_fail_n</i>
2	NOT <i>pwgood</i>
else	0b0

4.5.6 DSFI2 API

4.5.6.1 DSFI2_init()

This function initializes the DSFI layer and the layers below it.

- » Configure the lower protocol type
- » Initialize the lower layer

Return value

- Error code

Function call:

```
int DSFI_Init(void);
```

Table 29. DSFI2_init() return codes

Value	Description
DSFI_SUCCESS	The initialization is successful
DSFI_ERROR	An error occurred during the initialization process
DSFI_PROT_NOT_SET	The lower protocol is not configured correctly

4.5.6.2 DSFI2_Config()

Write a 256-Byte configuration block of one DSFI2 instance by calling the following functions of the DSF2 instance.

- » Call FN8 *GetDeviceId* and check IP core ID
- » Unlink instance by calling FNO *SetMasterId* (master ID = 0)
- » Link instance by calling FNO *SetMasterId* (ID of application)
- » Configure instance by calling FN2 *SetConfig* with the corresponding parameter setting
- » Set instance to state operational by calling FN1 *SetOperational*

Parameters

- DSFI2 master ID
- Instance number
- IP core ID
- Pointer to payload data

Return values

- Error code

Function call:

```
int DSFI2_Config(uint8_t instance, uint8_t masterId, uint8_t geoAddr,
uint8_t * config, uint16_t configSize);
```

Table 30. DSFI2_Config() function arguments

Configuration	Description
<i>uint8_t instance</i>	I/O group number
<i>uint8_t masterId</i>	Master ID
<i>uint8_t geoAddr</i>	Geographical address
<i>uint8_t * config</i>	Pointer to the configuration block
<i>uint16_t configSize</i>	Size of the configuration block

Table 31. DSFI2_Config() return code

Value	Description
<i>DSFI_SUCCESS</i>	The configuration is successful
<i>DSFI_ERROR</i>	An error occurred during the configuration process
<i>DSFI_PROT_NOT_SET</i>	The lower protocol is not configured correctly
<i>DSFI_INVALID_DATA</i>	The configuration data pointer value is NULL or the size is 0
<i>DSFI_GEOADDR_ERROR</i>	The provided geographical address does not match the card address

4.5.6.3 DSFI2_Function()

This function calls one or several DSFI2 functions and handles DSFI2 request/acknowledge (*DSFI2_send()*, *DSFI2_receive()*).

Parameters per DSFI2 function call

- Instance number
- Function number
- Pointer to send payload data
- Pointer to receive payload data

Return value per DSFI2 function call

- Error code
- Received payload
- Received payload length

Function call:

```
int DSFI2_Function(struct DSFI_Function_t * nFunction, uint8_t size);
```

Table 32. struct DSFI_Function

Data	Description
<i>uint8_t instance</i>	Instance number
<i>uint8_t function</i>	Function number
<i>uint8_t * outData</i>	Pointer to data for transmission
<i>uint16_t outDataSize</i>	Size of the transmit data buffer
<i>uint8_t *inData</i>	Pointer to the data buffer for the received data
<i>uint8_t * inDataSize</i>	Size of the receive data buffer
<i>int errorStatus</i>	Error status after performing the action. It comprises the communication error and the slave status error.

Table 33. DSFI2_Function() function arguments

Configuration	Description
<i>nFunction</i>	Pointer to an array of the function data structures
<i>size</i>	Array size

Table 34. DSFI2_Function() return code

Value	Description
<i>DSFI_SUCCESS</i>	The function call is successful
<i>DSFI_ERROR</i>	An error occurred during the function call
<i>DSFI_INVALID_DATA</i>	The pointer value is NULL or the size is 0

4.5.6.4 DSFI2_send()

This function sends DSFI2 requests.

Parameters

- Instance number
- Function number
- Pointer to payload data

Return value

- Error code

Function call:

```
int DSFI2_Send(uint8_t instance, uint8_t function, uint8_t * dataP,
uint16_t dataLen);
```

Table 35. DSFI2_send() function arguments

Configuration	Description
<i>uint8_t instance</i>	Instance number
<i>uint8_t function</i>	Function number
<i>uint8_t * dataP</i>	Pointer to the user payload
<i>uint16_t dataLen</i>	Length of the user payload

Table 36. DSFI2_send() return code

Value	Description
<i>F403_SUCCESS</i>	The transmission was successful
<i>F403_ERROR</i>	Error in transmission

4.5.6.5 DSFI2_receive()

This function receives DSFI2 acknowledge.

Parameters

- Pointer to the buffer for payload data
- Buffer size

Return values

- Error code
- Received payload

Function call:

```
int DSFI2_Receive(uint8_t * dataP, uint8_t *dataLen);
```

Table 37. DSFI2_receive() function arguments

Configuration	Description
<i>uint8_t * dataP</i>	Buffer pointer to the user payload
<i>uint8_t * dataLen</i>	Pointer to the length of the buffer for the payload. After reception the received length is saved here.

Table 38. DSFI2_receive() return parameters

Value	Description
<i>F403_SUCCESS</i>	The reception was successful
<i>F403_ERROR</i>	Error in reception
<i>F403_CRC_ERROR</i>	A parity bit error is detected during reception.
<i>Error status</i>	Upon successful reception, the error status from the DSFI frame header will be returned.
<i>dataP</i>	Upon successful reception the payload data will be written to the user data buffer.
<i>dataLenP</i>	The value of the data length will be updated with the received payload length, when the receive operation is successful.

4.5.7 DSFI2 Function Profile

Each DSFI2 IP core provides two sets of functions which can be used by the application: common functions which are identical for each IP core and IP core specific functions.

4.5.7.1 DSFI2 Common Function Profile

Figure 13. DSFI2 common function profile

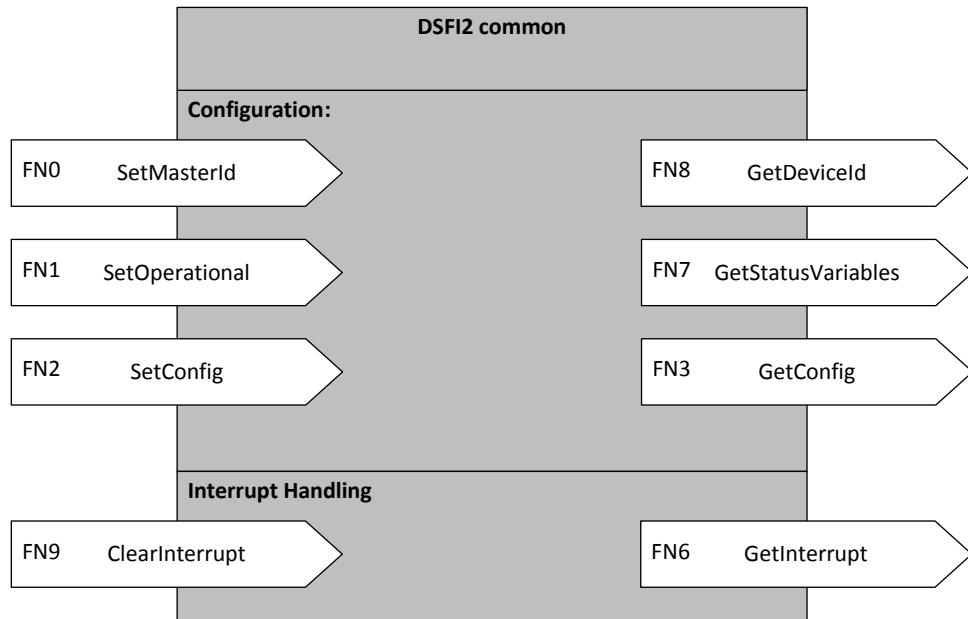


Table 39. DSFI2 common function overview

FN	Function	State	Description
0	<i>SetMasterId</i>	UNLINKED, any for master id = 0	Master ID = 0: unlink instance Master ID != 0: link instance to master
1	<i>SetOperational</i>	CONFIGURE	Set instance to operational mode
2	<i>SetConfig</i>	CONFIGURE	Write configuration block
3	<i>GetConfig</i>	CONFIGURE, OPERATIONAL	Read configuration block
6	<i>GetInterrupt</i>	CONFIGURE, OPERATIONAL	Read pending interrupts
7	<i>GetStatusVariables</i>	CONFIGURE, OPERATIONAL	Read status block
8	<i>GetDeviceId</i>	CONFIGURE, OPERATIONAL	Read ID of instance
9	<i>ClearInterrupt</i>	OPERATIONAL	Clear pending interrupts

4.5.7.2 I/O-Specific Function Profile

Figure 14. I/O-specific function profile

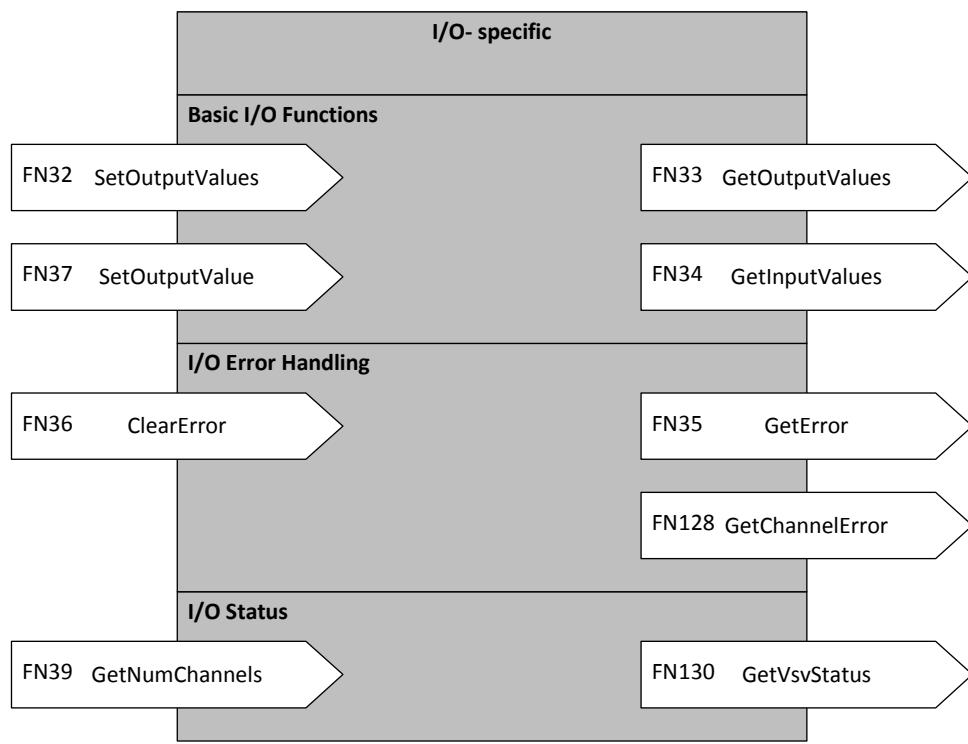


Table 40. I/O-specific function overview

FN	Function	State	Description
32	<i>SetOutputValues</i>	OPERATIONAL	Set all binary outputs of instance
33	<i>GetOutputValues</i>	CONFIGURE, OPERATIONAL	Read status of all binary outputs of instance
34	<i>GetInputValue</i>	OPERATIONAL	Read status of all binary inputs of instance
35	<i>GetError</i>	CONFIGURE, OPERATIONAL	Read pending errors
36	<i>ClearError</i>	OPERATIONAL	Clear pending errors
37	<i>SetOutputValue</i>	OPERATIONAL	Set a single output channel
39	<i>GetNumChannels</i>	CONFIGURE, OPERATIONAL	Read number of I/O channels
128	<i>GetChannelError</i>	CONFIGURE, OPERATIONAL	Read channel related errors
130	<i>GetVsvStatus</i>	CONFIGURE, OPERATIONAL	Read voltage supervision status

4.5.7.3 DSFI2 Handler Function Profile

Figure 15. DSFI2 handler function profile

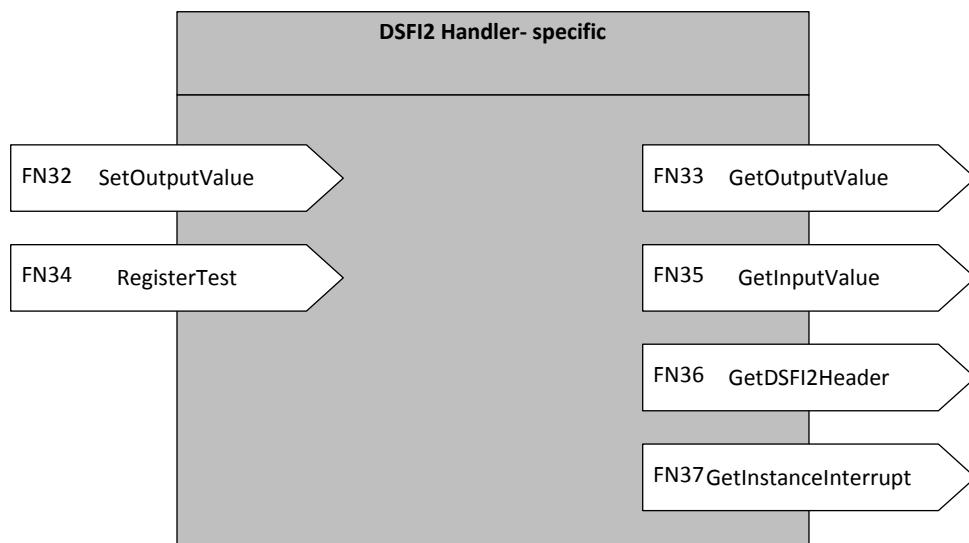


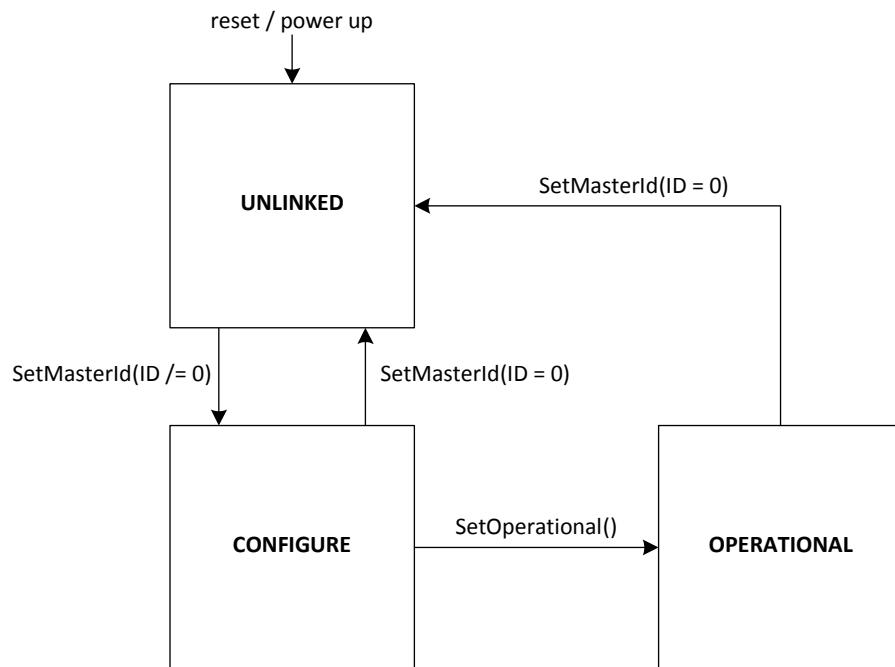
Table 41. DSFI2 handler function overview

FN	Function	State	Description
32	<i>SetOutputValue</i>	OPERATIONAL	Write internally used outputs - Clear internal watchdog failure
33	<i>GetOutputValue</i>	CONFIGURE, OPERATIONAL	Read internally used outputs
34	<i>RegisterTest</i>	any	Communication test
35	<i>GetInputValue</i>	CONFIGURE, OPERATIONAL	Read F403 status - Temperature failure - Voltage failure - Power good - Board status (FPGA factory/ user image loaded) - Internal watchdog failure
36	<i>GetDSFI2Header</i>	any	Read the board ID, revision
37	<i>GetInstanceInterrupt</i>	CONFIGURE, OPERATIONAL	Read interrupts of DSFI2 instances to determine which instance has pending interrupts

4.5.8 Configuration

Each of the DSFI2 instances must be configured by application before it can be used. Therefore, a logical connection to a single application part (master) must be established before the instance can be configured and operated.

Figure 16. DSFI2 configuration



4.5.8.1 Link

Initially each instance enters the UNLINKED state after reset/power up. That means that no master is able to control the DSFI2 instance in this state. The instance must be linked to an application part (master) before the instance can be configured.

The instance is linked to the master by the FNO *SetMasterId* function. The instance enters the CONFIGURE state from its initial state (UNLINKED).

4.5.8.2 Parameter Configuration

If an instance is in CONFIGURE state, all configuration parameters can be written by FN2 *SetConfig*. The configuration parameters can be read back by FN3 *GetConfig*.

The following parameters can be configured for each I/O controller instance of the F403:

Table 42. I/O controller parameters

Parameter	Values	Description
Output mode	Input only/ output	Set mode of I/O channel
Output value	Open/closed	Set output value
Current drain trigger	Continuous/ constant, period (1 Hz..100 Hz)	Set current of input channel
Output comparison	Enable/disable	Compare whether output value is read back on corresponding input channel. Disable output in case of error.
Output watchdog	Disable/ 8..120 ms	Disable output channel when not set periodically
Interrupt	Enable/disable	Indication of asynchronous events: - Input value changed - I/O channel error - Common error

The following parameters can be configured for the DSFI2 handler instance of the F403:

Table 43. DSFI2 handler parameters

Parameter	Values	Description
Watchdog clear (Output value bit 0)	0b0/0b1	Clear internal watchdog timeout indication
Interrupt	Enable/disable	Indication of asynchronous events: - Temperature failure - Voltage failure - Power-good failure

4.5.8.3 Operational State

When an instance is configured, it can be set to OPERATIONAL state by FN1 *SetOperational*. In this state the instances are fully operational. Configuration cannot be changed in OPERATIONAL state.

4.5.9 I/O Controller Functions

4.5.9.1 Basic I/O Functions

Each I/O channel of an I/O controller instance can be configured as output or input only. The direction of I/O channels can be set by configuring the I/O controller instances with FN2 *SetConfig*.

The number of I/O channels of an I/O controller instance can be read by FN39 *GetNumChannels*. For F403 all 4 I/O controller instances support 4 channels.

The input values of all channels of an instance are read by FN34 *GetInputValues*. The input value of an I/O channel is read even if a channel is configured as output.

The output values of all channels of an I/O controller instance are set by FN32 *SetOutputValues*. The output values of all channels of an I/O controller instance can be read by FN33 *GetOutputValues*. A single I/O configured as output can be set by FN37 *SetOutputValue*.

The output value of a channel can be set regardless of its output mode (input only/output).

4.5.9.2 Output Watchdog

The output watchdog feature of I/O channels configured as output detects failures in communication with the I/O instance. When the output watchdog of an output channel is enabled, the output channel must be set by FN32 *SetOutputValues* or FN37 *SetOutputValue* periodically within the watchdog timeout period. The output watchdog can be enabled and configured by setting the corresponding parameter in the configuration of I/O controller instances with FN2 *SetConfig*.

If the output channel is not triggered within the watchdog timeout, a watchdog failure is detected and

- the output channel is disabled
- the failure is indicated as DSFI2 interrupt when the corresponding interrupt is enabled in instance configuration
- the failure can be read as channel error by FN35 *GetError*
- the failure can be read as output watchdog failure by FN128 *GetChannelError*
- the failure can be cleared by FN36 *ClearError*

4.5.9.3 Output Comparison

The comparison of I/O channels configured as output with the corresponding input channel detects failures of I/O channels. The comparison can be enabled by setting the corresponding parameter in the configuration of I/O controller instances with FN2 *SetConfig*.

If the input value of an output channel does not match the output value, a failure is detected and

- the output channel is disabled
- the failure is indicated as DSFI2 interrupt when the corresponding interrupt is enabled in instance configuration
- the failure can be read as channel error by FN35 *GetError*
- the failure can be read as FET failure by FN128 *GetChannelError*
- the failure can be cleared by FN36 *ClearError*

4.5.9.4 Output Overload

I/O channels are monitored for channel overload. This feature is always enabled.

If an overload error of an output channel is detected,

- the output channel is disabled
- the failure is indicated as DSFI2 interrupt when the corresponding interrupt is enabled in instance configuration
- the failure can be read as channel error by FN35 *GetError*
- the failure can be read as overload failure by FN128 *GetChannelError*
- the failure can be cleared by FN36 *ClearError*

4.5.9.5 Voltage Supervision

The following failures on the F403 are monitored and the status can be read by FN130 *GetVsvStatus*.

Table 44. Voltage Supervision

Failure	Description	Outputs
Overvoltage	Overvoltage on IO channel supply voltage IO channel supply voltage over 154 V	No effect
Power not good	Power fail IO channel supply voltage under 15.8 V	Disable all outputs of I/O controller instance
Overvoltage on 5 V	Overvoltage on digital supply voltage (isolated 5 V) used for status monitoring	No effect
Shift error	Status monitoring failure	Disable all outputs of I/O controller instance

If a common error/voltage supervision error is detected,

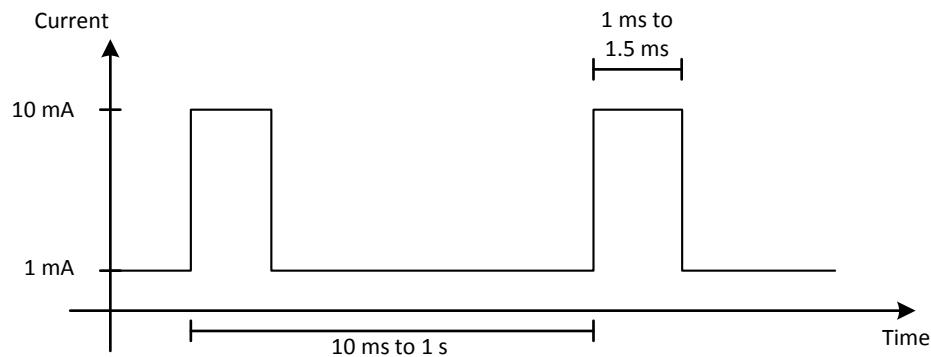
- the failure is indicated as DSFI2 interrupt when the corresponding interrupt is enabled in instance configuration
- the failure can be read as common error by FN35 *GetError*
- the failure can be cleared by FN36 *ClearError*

4.5.9.6 Current Drain Trigger

The input current of I/O channels configured as input (Current Drain Trigger, CDT) can be configured with the corresponding parameters in the configuration of I/O controller instances with FN2 SetConfig.

- CDT mode: continuously triggered, constant 1 mA, constant 10 mA
- CDT period: 1 Hz..100 Hz

Figure 17. Current Drain Trigger



4.5.10 Interrupt Handling

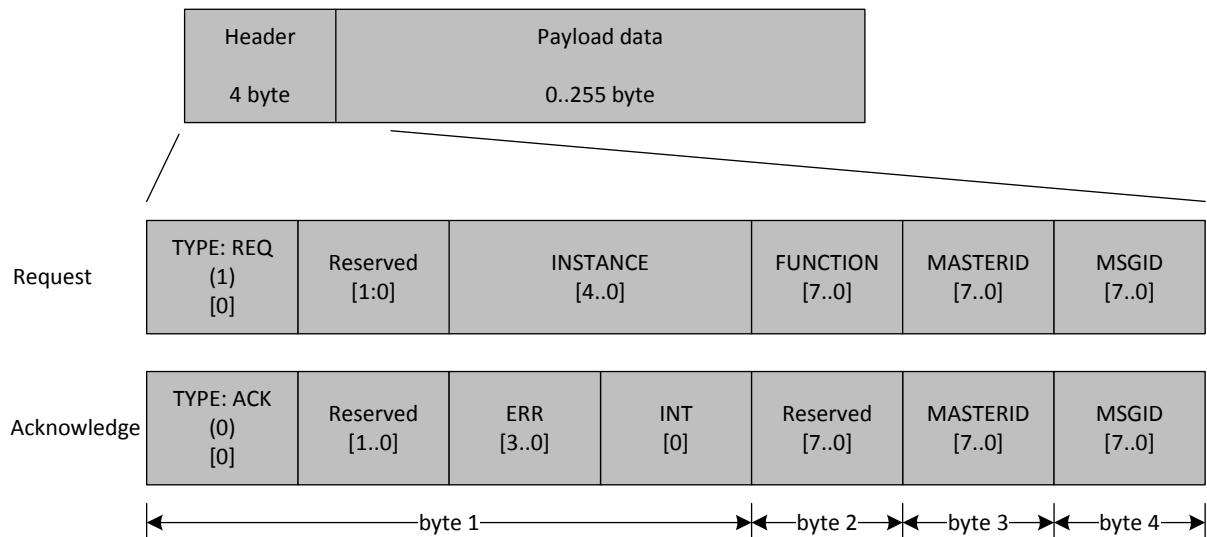
Asynchronous events of IP cores can be indicated as DSFI2 interrupts over the DSFI2 protocol. Pending interrupts are handled by application with DSFI2 functions of the corresponding instance.

DSFI2 interrupt generation of an instance is enabled/disabled in configuration by FN2 *SetConfig*. To determine which instance has pending interrupts, call FN37 *GetInstanceInterrupt* of the DSFI2 handler. To determine which interrupt of an instance is pending, call FN6 *GetInterrupt* of the corresponding instance. To clear pending interrupts in an instance call FN9 *ClearInterrupt* of the corresponding instance. To ensure that no interrupt is lost, the current state of instance interrupts is provided to the application.

4.5.11 DSFI2 Message Layout

The messages have the following layout:

Figure 18. DSFI2 message overview



4.5.12 DSFI2 Message Fields

The following data objects are used in DSFI2:

Table 45. DSFI2 data objects

Object	Element	Comment
DSFI2_REQ	TYPE	Binary 1
	INSTANCE	I/O controller instance
	FUNCTION	Function number
	MASTERID	Unique ID of requesting master
	MSGID	Unique sequence number
	Data	Payload data
DSFI2_ACK	TYPE	Binary 0
	ERR	Error status (see below)
	INT	Interrupt status (1 = interrupt pending)
	MASTERID	Unique ID of requesting master
	MSGID	Unique sequence number (matching REQ ID)
	Data	Payload data

4.5.13 DSFI2 Message Field Coding

4.5.13.1 INSTANCE

The REQ message header field INSTANCE is coded as follows:

Table 46. REQ message header field INSTANCE

Description	INSTANCE[4..0]
Instance number of protocol handler	0b000000
For requests	Instance number of the accessed DSFI2 slave

The following instances are available in the DSFI2 subsystem.

Table 47. DSFI2 instances

Instance	Description	Detailed Specification
0	MISC	DSFI2_HANDLER including MISC Unit
1	BINIO 1	Binary I/O controller for I/O group 1
2	BINIO 2	Binary I/O controller for I/O group 2
3	BINIO 3	Binary I/O controller for I/O group 3
4	BINIO 4	Binary I/O controller for I/O group 4
5	FLASH	Flash interface for update

4.5.13.2 MASTERID

The message field *MASTERID* is coded as follows:

Table 48. Message header field *MASTERID*

Description	MASTERID[7..0]
Invalid DSFI2 master ID (used to unlink a master from a slave)	0x00
Valid DSFI2 master ID	0x01..0xFF

4.5.13.3 FUNCTION

The REQ message field *FUNCTION* will identify the function that shall be performed on the DSFI2 slave. The functions are grouped as follows:

Table 49. REQ message header field *FUNCTION*

Function class	Function	FUNCTION[7..0]
Slave common	<i>FN0</i>	0x00
	<i>FN1</i>	0x01
	<i>FN2</i>	0x02

	<i>FN31</i>	0x1F
Slave group specific	<i>FN32</i>	0x20
	<i>FN33</i>	0x21

	<i>FN127</i>	0x7F
Slave specific	<i>FN128</i>	0x80
	<i>FN129</i>	0x81

	<i>FN255</i>	0xFF

4.5.13.4 ERR

The ACK message field ERR is coded as follows:

Table 50. ACK message header field ERR

Priority	Description	ERR[3..0]
1 (lowest)	No error	0x0
2	Wishbone timeout	0x1
3	Wrong MasterId	0x2
4	Invalid REQ	0x3
5	Slave unlinked	0x4
6	Reserved	0x5
7	Length error	0x6
8	Invalid instance	0x7
9	Invalid function	0x8
10	Slave unconfigured	0x9
11	Slave state error	0xA
12 (highest)	Internal error	Others

4.5.14 DSFI2 Master (Application)

4.5.14.1 DSFI2 Master Services

A DSFI2 master is used to operate DSFI2 I/O controllers. It sends DSFI2 request messages initiated by application software and receives the resulting DSFI2 acknowledges that were sent by the accessed DSFI2 slave (I/O controller).

4.5.14.2 DSFI2 Master States

A DSFI2 Master implements the following state flow.

Figure 19. DSFI2 Master states

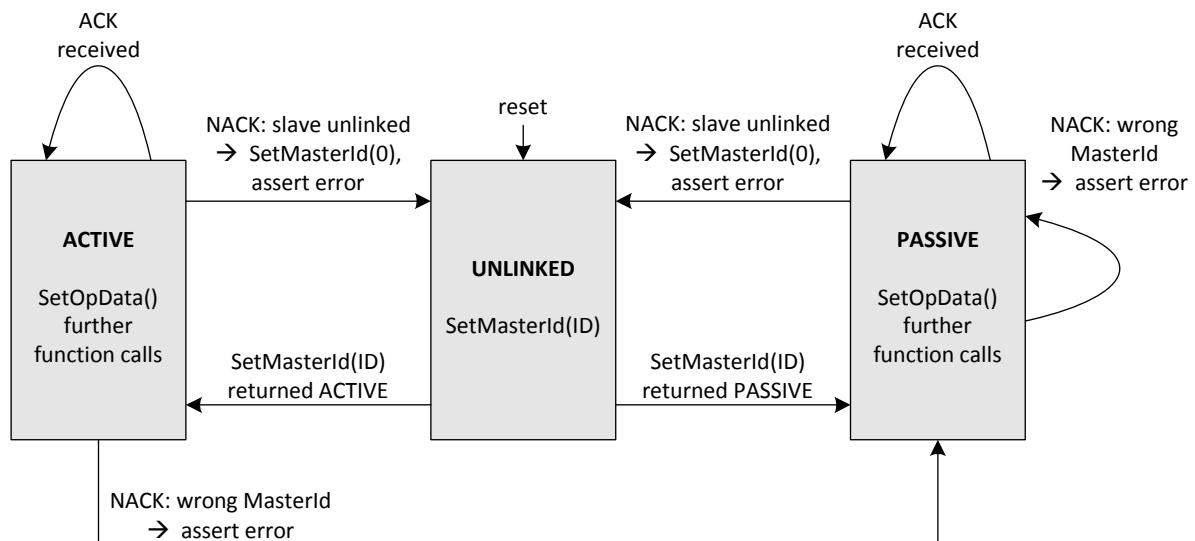


Table 51. Master configuration states

Configuration state	Description
UNLINKED	Link DSFI2 slave
ACTIVE	Control over slave gained. Configure slave and set slave to operational. Perform function calls for normal operation.
PASSIVE	Control over slave not gained. Check configuration of slave. Check variables during normal operation. Assert error, if slave does not behave as specified.

4.5.15 DSFI2 Slave (I/O Controller)

4.5.15.1 DSFI2 Slave Services

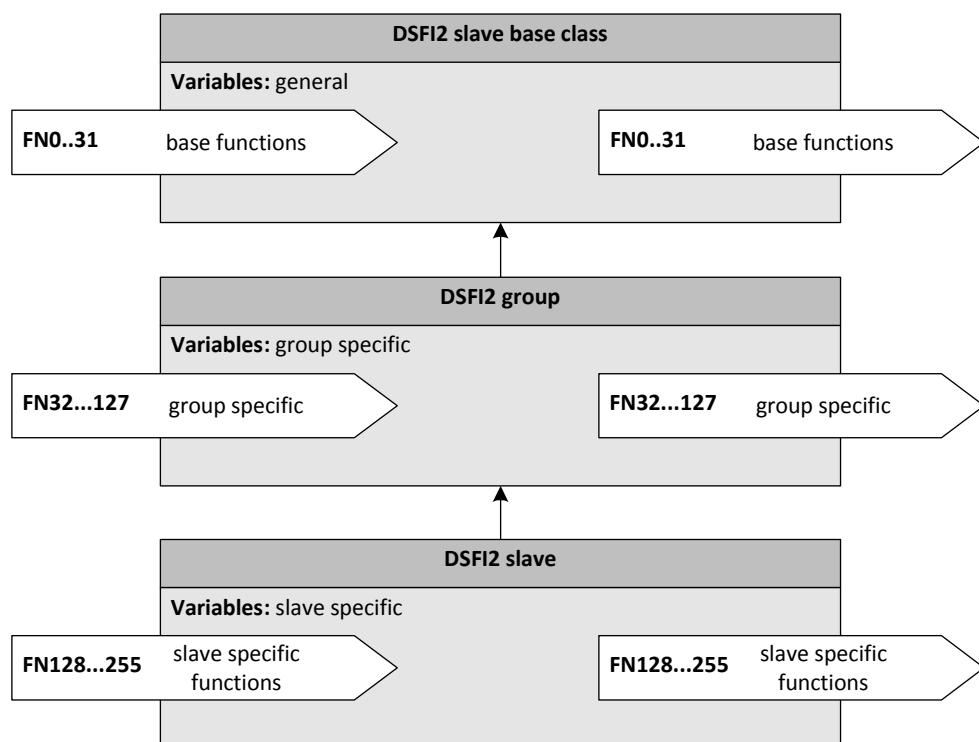
A DSFI2 slave is an IP core in an FPGA which is operated via the DSFI2 protocol. DSFI2 request messages which are initiated by application software are received by a DSFI2 handler. The DSFI2 handler performs the requested operations on the I/O controller and sends back a DSFI2 acknowledge message for each request.

4.5.15.2 DSFI2 Slave Base Class

4.5.15.2.1 DSFI2_Slave Inheritance

A DSFI2 slave is derived from the DSFI2 base class. All variables and functions of the parent class are available in the derived class.

Figure 20. DSFI2 Slave I/O function profile

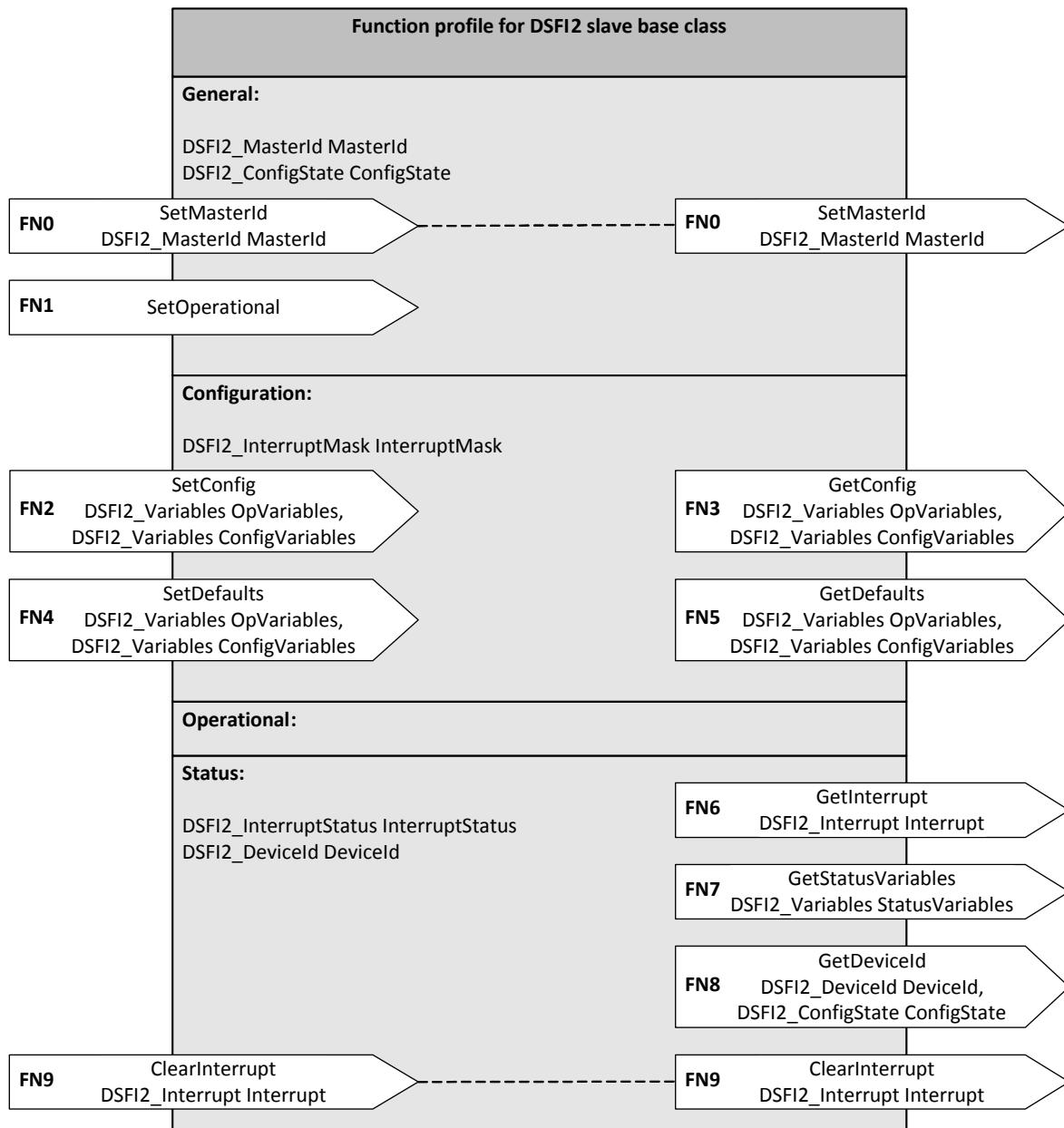


A common set of functions for all DSFI2 slaves is supported. A DSFI2 slave may implement additional DSFI2 slave specific functions.

4.5.15.2.2 DSFI2_Slave Function Profile

A DSFI2 slave supports the following functions:

Figure 21. DSFI2 common slave function profile



4.5.15.2.3 DSFI2 Slave Configuration of State Machine

Figure 22. DSFI2 Slave Configuration of State Machine

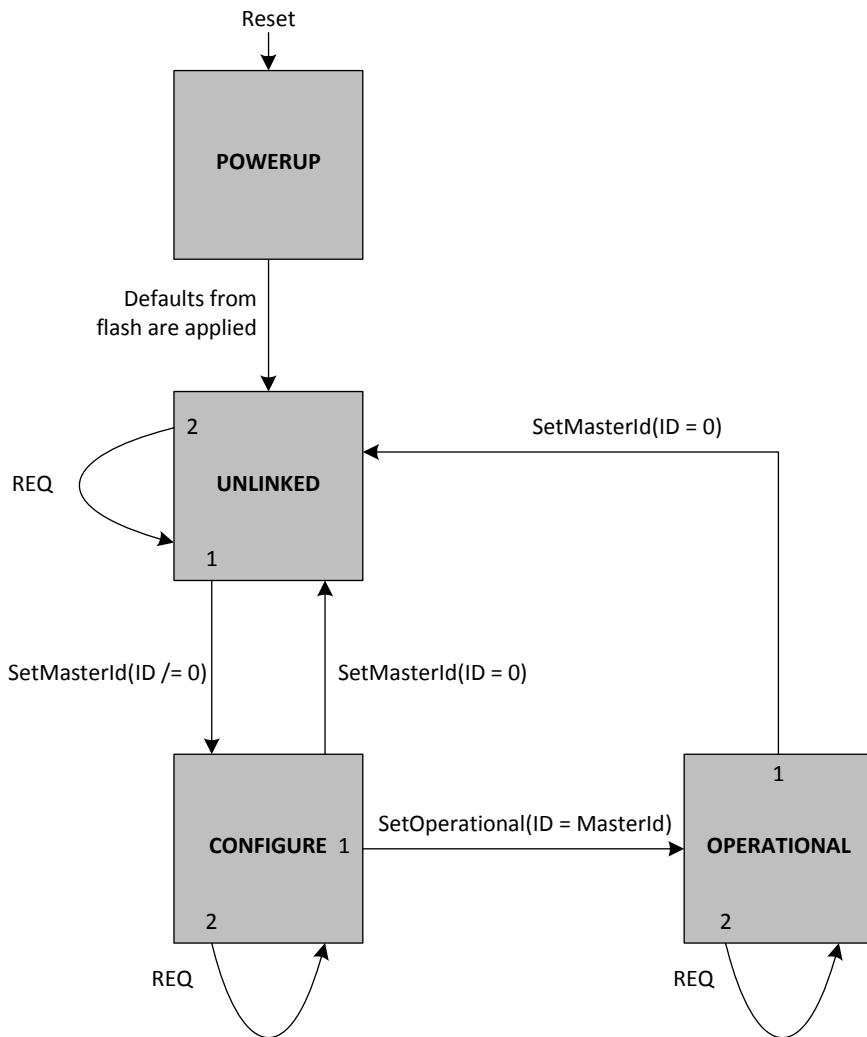


Table 52. Slave configuration states

Configuration State	Description
POWERUP	After power up the slave default values for variables are loaded from Flash and applied.
UNLINKED	Configuration by a DSFI2 master is required.
CONFIGURE 1	Slave is configured.
OPERATIONAL	Slave is fully configured and operational.

4.5.15.2.4 DSFI2 Slave Variables

The following permissions apply to variables of a DSFI2 slave, in the different configuration states:

Table 53. DSFI2 Slave Variables

Variable	Type	Description
<i>ConfigVariables</i>	<i>DSFI2_Variables</i>	Common and DSFI2 slave specific configuration variables
<i>DeviceId</i>	<i>DSFI2_DeviceId</i>	Included in ConfigVariables
<i>OpVariables</i>	<i>DSFI2_Variables</i>	Common and DSFI2 slave specific operational variables
<i>StatusVariables</i>	<i>DSFI2_Variables</i>	Common and DSFI2 slave specific status variables
<i>MasterId</i>	<i>DSFI2_MasterId</i>	ID of the configuring master is stored
<i>ConfigState</i>	<i>DSFI2_ConfigState</i>	Internal state of slave
<i>InterruptStatus</i>	<i>DSFI2_InterruptStatus</i>	Indicate interrupts
<i>InterruptMask</i>	<i>DSFI2_InterruptMask</i>	Enable/disable interrupts

Table 54. Configuration states for DSFI2 slave variables

Variable	Configuration State			
	POWERUP	UNLINKED	CONFIGURE	OPERATIONAL
<i>ConfigVariables</i>	-	-	R/W	R
<i>DeviceId</i>	-	-	R/W	R/W
<i>OpVariables</i>	-	-	R/W	R/W
<i>StatusVariables</i>	-	R/C	R/C	R/W/C
<i>MasterId</i>	-	R/W	R/W	R/W
<i>ConfigState</i>	-	R/W	R/W	R/W
<i>InterruptStatus</i>	-	-	R/C	R/W/C
<i>InterruptMask</i>	-	-	R/W	R

R: readable by function

W: writeable by function

C: changeable anytime by status

Configuration and operational variables are part of DSFI2 slave configuration and can be written and read back in CONFIGURE state. Status variables are not part of DSFI2 slave configuration and read values are unpredictable.

4.5.15.2.5 Mapping of DSFI2 Slave Variables

A DSFI2 slave has the following common mapping for internal variables.

Table 55. Mapping of DSFI2 slave variables

Variable	Offset	Description
<i>OpVariables</i>	0x00..0x7F	DSFI2 slave specific
<i>ConfigVariables</i>	0x00..0x7F	DSFI2 slave specific
	0x70..0x73	InterruptMask
	0x74..0x7F	Reserved
	0x00..0x7F	I/O controller specific
<i>StatusVariables</i>	0x70..0x73	InterruptStatus
	0x74..0x7B	Reserved
	0x78..0x79	DeviceId: device
	0x7A	DeviceId: variant
	0x7B	Reserved
	0x7C..0x7D	DeviceId: revision
	0x7E..0x7F	Reserved

The mapping of variables is equal for all DSFI2 slaves.

The *DSFI2_Variables* data type contains common and device specific variables and has a length of 0x80 bytes. DSFI2 slaves implement the *OpVariables* variable of the *DSFI2_Variables* data type. *ConfigVariables* are used for operational variables. DSFI2 slaves implement the *ConfigVariables* variable of the *DSFI2_Variables* data type. *ConfigVariables* are used for configuration variables. DSFI2 slaves implement the *StatusVariables* variable of the *DSFI2_Variables* data type. *StatusVariables* are used for variables which provide the status of the DSFI2 slave. *StatusVariables* have a length of 0x80 bytes. Status variables are not part of DSFI2 slave configuration and read values are unpredictable.

DSFI2 slaves implement the *MasterId* variable of the *DSFI2_MasterId* data type. *MasterId* is used to store the ID of the DSFI2 master which is linked to the slave. The data type *DSFI2_MasterId* is 8 bit and has the following range:

- Valid range: 0x00..0xFF
- Default value (invalid ID): 0x00

DSFI2 slaves implement the *DeviceId* variable of the *DSFI2_DeviceId* data type. The *DSFI2_DeviceId* data type is 64 bit and has the following mapping:

Table 56. Mapping of DSFI2_DeviceId

DSFI2_DeviceId [x]	Variable	Description
63..48	Reserved	Reserved
47..40	<i>REV_MAJ[7..0]</i>	Major revision
39..32	<i>REV_MIN[7..0]</i>	Minor revision
31..22	Reserved	Reserved
21..16	<i>VARIANT[5..0]</i>	Variant of device
15..10	Reserved	Reserved
9..0	<i>DEVICE[9..0]</i>	Device number of device

DSFI2 slaves implement the *ConfigState* variable of the *DSFI2_ConfigState* data type. *ConfigState* contains the configuration state of the DSFI2 slave. *ConfigState* is 1 byte and coded as follows:

Table 57. DSFI2_ConfigState

ConfigState[7..2]	ConfigState[1..0]	Configuration State
Reserved	0b00	POWERUP
Reserved	0b01	UNLINKED
Reserved	0b10	CONFIGURE
Reserved	0b11	OPERATIONAL

DSFI2 slaves implement the *InterruptStatus* variable of the *DSFI2_InterruptStatus* data type and are 32 bit. *InterruptStatus* contains pending interrupts.

Table 58. DSFI2_InterruptStatus

InterruptStatus [x]	Description
0	No interrupt pending
1	Interrupt pending

DSFI2 slaves implement the *InterruptMask* variable of the *DSFI2_InterruptMask* data type and is 32 bit. *InterruptMask* contains pending interrupts.

Table 59. DSFI2_InterruptMask

InterruptMask [x]	Description
0	Interrupt enabled
1	Interrupt disabled

4.5.15.2.6 Wrong MasterId

A DSFI2 slave checks the *DSFI2_MasterId*, which is transmitted with a REQ when the function of the REQ performs a write operation on a variable. If the transmitted *DSFI2_MasterId* does not match the *MasterId* of the master the slave is linked to, the ERR flag in the ACK is set to "Wrong MasterId" and internal variables are not changed.

4.5.15.2.7 Invalid Request

A DSFI2 slave checks whether a function of a REQ performs a valid operation on a variable. If an operation is not permitted, the ERR flag in the ACK shall be set to "Invalid REQ" and internal variables are not changed.

See [Table 53, DSFI2 Slave Variables on page 71](#) for variables permissions.

4.5.15.2.8 Slave UNLINKED

A DSFI2 slave evaluates the functions called in requests in the *UNLINKED* state. If a function other than *SetMasterId* is called when the DSFI2 slave is in the *UNLINKED* state, the ERR flag in the ACK is set to "Slave unlinked" and internal variables are not changed.

4.5.15.2.9 *SetMasterId*

The *SetMasterId* function sets the *MasterId* variable to the value of the *MASTERID* of the request message if the DSFI2 slave is in the *UNLINKED* state or if the *MasterIdIn* is *0x0*. The new value of the *MasterId* is returned in *MasterIdOut*.

See [Figure 22, DSFI2 Slave Configuration of State Machine on page 70](#) for the setting of the *ConfigState* variable.

If *MasterIdIn* is *0x0*, the *parameter_valid* register of the affected instance is set to *0x00*.

Table 60. SetMasterId function

Write variables	<i>MasterId, ConfigState</i>
Read variables	<i>MasterId</i>
Request payload	None
Acknowledge payload	<i>DSFI2_MasterId MasterIdOut</i>

4.5.15.2.10 *SetOperational*

The *SetOperational* function sets the *ConfigState* variable to *OPERATIONAL* if the DSFI2 slave is in the *CONFIGURE* state. Otherwise the *ERR* flag in the *ACK* is set to "Invalid REQ" and internal variables are not changed. The *parameter_valid* register of the affected instance is set to *0x01* to apply the configuration.

Table 61. SetOperational function

Write variables	<i>ConfigState</i>
Read variables	-
Request payload	None
Acknowledge payload	None

4.5.15.2.11 *SetConfig*

The *SetConfig* function configures the DSFI2 slave and its internal variables and

- updates *ConfigVariables* with the content of *ConfigVariablesIn*
- updates *OpVariables* with the content of *OpVariablesIn*

Table 62. SetConfig function

Write variables	<i>ConfigVariables, OpVariables</i>
Read variables	-
Request payload	<i>DSFI2_Variables OpVariablesIn, DSFI2_Variables ConfigVariablesIn</i>
Acknowledge payload	None

4.5.15.2.12 *GetConfig*

The *GetConfig* function provides the configuration of the internal variables of the DSFI2 slave and

- returns *ConfigVariables* in *ConfigVariablesOut*
- returns *OpVariables* in *OpVariablesOut*

Table 63. *GetConfig* function

Write variables	-
Read variables	<i>ConfigVariables, OpVariables</i>
Request payload	None
Acknowledge payload	<i>DSFI2_Variables OpVariablesOut,</i> <i>DSFI2_Variables ConfigVariablesOut</i>

4.5.15.2.13 *SetDefaults*

The *SetDefaults* function sets the default values of the slave. The default values are stored and read from slave in POWERUP state:

- store default of *ConfigVariables* with the content of *ConfigVariablesIn*
- store default of *OpVariables* with the content of *OpVariablesIn*

Table 64. *SetDefaults* function

Write variables	<i>ConfigVariables, OpVariables</i>
Read variables	-
Request payload	<i>DSFI2_Variables OpVariablesIn,</i> <i>DSFI2_Variables ConfigVariablesIn</i>
Acknowledge payload	None

4.5.15.2.14 *GetDefaults*

The *GetDefaults* function provides the configuration of the internal variables of the DSFI2 slave and

- returns the default of *ConfigVariables* in *ConfigVariablesOut*
- returns the default of *OpVariables* in *OpVariablesOut*

Table 65. *GetDefaults* function

Write variables	-
Read variables	<i>ConfigVariables, OpVariables</i>
Request payload	None
Acknowledge payload	<i>DSFI2_Variables OpVariablesOut,</i> <i>DSFI2_Variables ConfigVariablesOut</i>

4.5.15.2.15 *GetInterrupt*

The *GetInterrupt* function provides the current value of interrupts (variable *InterruptStatus*) in *InterruptOut* and clears all pending interrupts (interrupt variable: 0x0).

Table 66. GetInterrupt function

Write variables	<i>InterruptStatus</i>
Read variables	<i>InterruptStatus</i>
Request payload	None
Acknowledge payload	<i>DSFI2_InterruptStatus InterruptOut</i>

4.5.15.2.16 *GetStatusVariables*

The *GetStatusVariables* function provides the configuration of the internal variables of the DSFI2 slave and returns *StatusVariables* in *StatusVariablesOut*.

Table 67. GetStatusVariables function

Write variables	-
Read variables	<i>StatusVariables</i>
Request payload	None
Acknowledge payload	<i>DSFI2_Variables StatusVariablesOut</i>

4.5.15.2.17 *GetDeviceId*

The *GetDeviceId* function provides the device number, variant, revision and internal state of the DSFI2 slave and returns:

- *DeviceId* in *DeviceIdOut*
- *ConfigState* in *ConfigStateOut*

Table 68. GetDeviceId function

Write variables	-
Read variables	<i>DeviceId, ConfigState</i>
Request payload	None
Acknowledge payload	<i>DSFI2_DeviceId DeviceIdOut</i> <i>DSFI2_ConfigState ConfigStateOut</i>

4.5.15.2.18 Error Cases

In case of an error DSFI2 slaves behave as follows:

Table 69. DSFI2 error cases

Priority	Error	DSFI2 slave FSM state	Action
1 (highest)	Error in lower layers	Any	Ignore errors
2	Message received in POWERUP state	POWERUP	Ignore message
3	Message received with length < 3 byte (header)	Any	Ignore message
4	Message received with invalid length (not as expected by DSFI2 function)	Any	ACK(err = Length Error)
5	FNO <i>SetMasterId</i> with MasterId == MasterID of slave	CONFIGURE, OPERATIONAL	ACK(err = Slave State Error) Payload contains current MasterId of slave
6	FNO <i>SetMasterId</i> with MasterId /= 0	CONFIGURE, OPERATIONAL	ACK(err = No error) Payload contains current MasterId of slave
7	FN1..7, FN9..255 (any function except FNO <i>SetMasterId</i> , FN8 <i>GetDeviceId</i>)	UNLINKED	ACK(err = Slave unlinked)
8	FN1 <i>SetOperational</i> , FN2 <i>SetConfig</i> , FN4 <i>SetDefaults</i>	OPERATIONAL	ACK(err = Slave State Error)
9	MasterId of message /= MasterId of slave in functions: FN1 <i>SetOperational</i> , FN2 <i>SetConfig</i> , FN4 <i>SetDefaults</i> , any function that affects variables	CONFIGURE, OPERATIONAL	ACK(err = Wrong MasterId)
10	FN9 <i>ClearInterrupt</i>	CONFIGURE	ACK(err = Slave State Error)
11	Function not implemented	Any	ACK(err = Invalid Request)
12 (lowest)	FN32...FN255	CONFIGURE	ACK(err = Slave State Error)

4.5.15.2.19 Invalid Message Reception

Acknowledge messages received by a DSFI2 slave are ignored.

4.5.15.3 DSFI2 I/O Controller

4.5.15.3.1 Address Space

The address space of a DSFI2 I/O controller is as follows:

Table 70. Address space of DSFI2 I/O controller

Contained in variable	Address	Register		Description
<i>OpVariables</i>	0x00..0x7F	I/O controller specific address range		Reserved
<i>ConfigVariables</i>	0x80..0xEF	I/O controller specific address range		Reserved
	0xF0..0xF3	<i>INTERRUPT_MASK[31..0]</i>		<i>InterruptMask</i>
	0xF4..0xFF	Reserved		Reserved
<i>StatusVariables</i>	0x100..0x16F	I/O controller specific address range		Reserved
	0x170..0x173	<i>INTERRUPT_STATUS[31..0]</i>		<i>InterruptStatus</i>
	0x174..0x177	Reserved		Reserved
	0x178..0x179	<i>Reserved[15..10]</i>	<i>DEVICE[9..0]</i>	<i>DeviceId: Device Number</i>
	0x17A	<i>Reserved[7..6]</i>	<i>VARIANT[5..0]</i>	<i>DeviceId: Variant</i>
	0x17B	Reserved		Reserved
	0x17C..0x17D	<i>REV_MAJ[7..0]</i>	<i>REV_MIN[7..0]</i>	Revision
	0x17E..0x17F	Reserved		Reserved
	0x180	Operational		Parameters valid
	0x181..0x1FF	Reserved		Reserved

Note: DEVICE = 0x3F indicates that no device is present for this instance.

DEVICE describes one individual device and identifies its programming interface (register map). If several functions have the same programming model, the DEVICE is equal, but the VARIANT is different. DEVICE is read only.

DEVICE[9..0] = last three digits of the MEN product number 16Z<DEVICE> (e.g. 0x02c for 16Z**044_DISP**).

Each device can be implemented as a specialized variant (e.g. with different behavior or features), but must support always the same device-specific programming interface.

The VARIANT[5..0] number is used to differentiate between these specialized devices (e.g. GPIO and GPIO via SPI). VARIANT is read only.

The REV_MAJ[7..0] belongs to the major revision number of the I/O controller. The REV_MIN[7..0] belongs to the minor revision number of the I/O controller. REV_MAJ and REV_MIN are read only.

The INTERRUPT_STATUS register indicates pending interrupts according to the following table:

Table 71. INTERRUPT_STATUS states

	<i>INTERRUPT_STATUS[x]</i>	Description
Read	0	No interrupt pending
	1	Interrupt pending
Write	0	-
	1	Clear interrupt

The *INTERRUPT_MASK* register provides the following functions:

Table 72. INTERRUPT_MASK states

	<i>INTERRUPT_MASK[x]</i>	Description
Read	0	Interrupt enabled
	1	Interrupt disabled
Write	0	Enable interrupt
	1	Disable interrupt

Default: 0xFFFF_FFFF

4.5.15.3.2 Interrupt Generation

I/O controllers support interrupt generation. *INTERRUPT_STATUS[x]* is only set to *0b1* if the corresponding *INTERRUPT_MASK[x]* bit is *0b0*. Otherwise *INTERRUPT_STATUS[x]* is *0b0*. An external interrupt signal is set to *0b1* if any bit in *INTERRUPT_STATUS* is active.

The interrupt status can be cleared by a write access to the interrupt register in the I/O controller.

Pending interrupts are indicated by an external interrupt signal. The interrupt signal is set to *0b1* if any bit in *INTERRUPT_STATUS* is *0b1*. Otherwise the signal is set to *0b0*.

In case a Wishbone timeout occurs the error is indicated in the ERR field of the corresponding ACK according to [Table 50, ACK message header field ERR on page 66](#).

I/O controllers implement the parameter_valid register and forward and use the values of the I/O controller addresses *0x000..0x17F* as follows:

Table 73. parameter_valid register

parameter_valid	Description
0	No update of internal registers
1	Update internal registers

4.5.16 DSFI2 Handler Unit

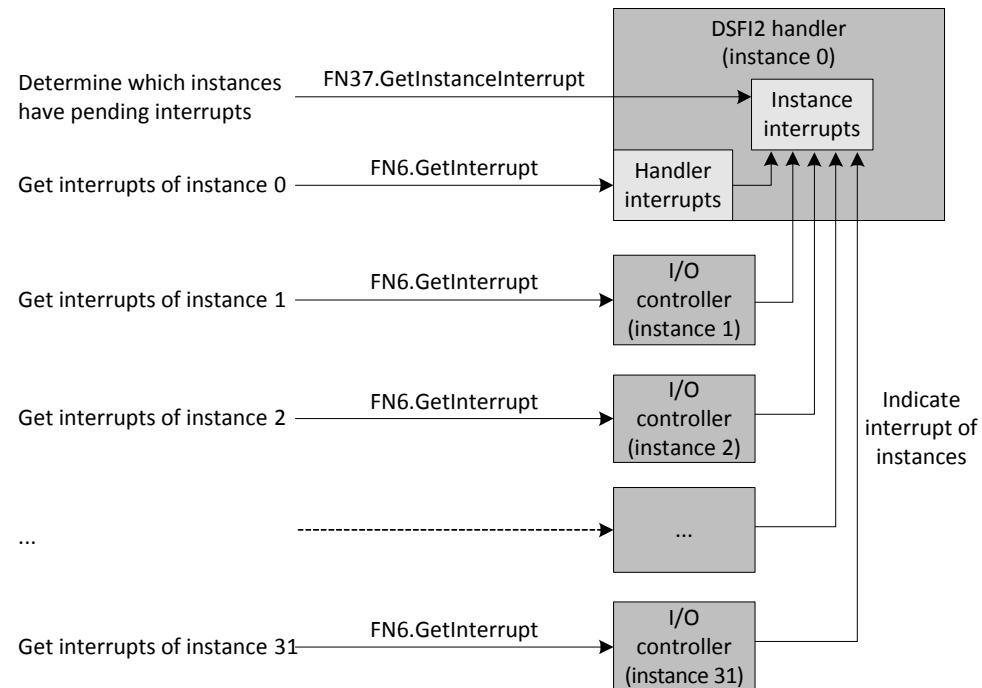
The DSFI2 handler unit provides access to the I/O controllers of the DSFI2 subsystem. It is a special DSFI2 slave which is derived from DSFI2 slave base class.

See [Chapter 4.5.15.2 DSFI2 Slave Base Class on page 68](#) for more information.

The DSFI2 handler unit is always implemented and is implemented as DSFI2 instance 0. It contains general board and status information.

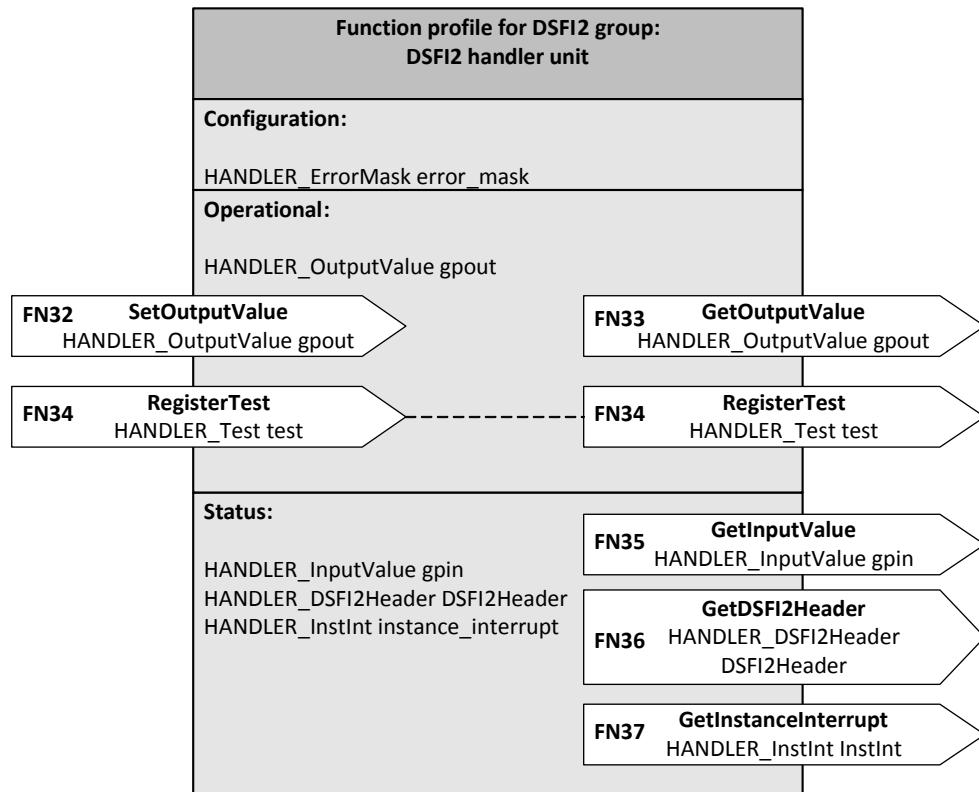
The DSFI2 handler provides interrupt information via the DSFI2 protocol as follows:

Figure 23. Interrupt information of handler unit



The DSFI2 slave of the handler unit function group supports the following functions:

Figure 24. DSFI2 handler function profile



The address space of a DSFI2 handler unit is as follows:

Table 74. DSFI2 handler unit address space

Contained in variable	Address	Register	Description
OpVariables	0x00..0x03	GPIOUT[31..0]	General purpose output
	0x04..0x07	Test[31..0]	Test register
	0x08..0x7F	Reserved	Reserved
ConfigVariables	0x80..0xEF	Reserved	Reserved
	0xF0..0xF3	INTERRUPT_MASK[31..0]	InterruptMask
	0xF4..0xFF	Reserved	Reserved
StatusVariables	0x100..0x102	0xABD	DSFI2 Header
	0x103	DSFI2ProtocolType	
	0x104..0x10F	Fpgald[95..0]	
	0x110	FpgaRevMin[7..0]	
	0x111	FpgaRevMaj[7..0]	
	0x112..0x113	Reserved	
	0x114..0x117	Instances[31..0]	
	0x118..0x11F	Reserved	Reserved
	0x120..0x123	GPIIN[31..0]	General purpose input
	0x124..0x13F	Reserved	Reserved
	0x140..0x143	reserved[31..16]	counter_0[15..0]
	0x144..0x147	reserved[31..16]	counter_1[15..0]
	0x148..0x14B	reserved[31..16]	counter_2[15..0]
	0x14C..0x14F	reserved[31..16]	counter_3[15..0]
	0x150..0x153	reserved[31..16]	counter_4[15..0]
	0x154..0x157	reserved[31..16]	counter_5[15..0]
	0x158..0x15B	reserved[31..16]	counter_6[15..0]
	0x15C..0x15F	reserved[31..16]	counter_7[15..0]
	0x160..0x163	reserved[31..6]	ga[5..0]
	0x164..0x167	codetest_id[31..0]	Codetest
	0x168..0x16B	codetest_val[31..0]	Codetest debug value
	0x16C..0x16F	INSTANCE_INTERRUPT	Instance interrupt
	0x170..0x173	INTERRUPT_STATUS[31..0]	InterruptStatus
	0x174..0x177	Reserved	Reserved
	0x178..0x179	reserved[15..10]	DEVICE[9..0]
	0x17A	reserved[7..6]	VARIANT[5..0]
	0x17B	Reserved	Reserved
	0x17C..0x17D	REV_MAJ[7..0]	REV_MIN[7..0]
	0x17E..0x17F	Reserved	Reserved
	0x180	reserved[7..1]	Operational
	0x181..0x1FF	Reserved	Reserved

Note: DEVICE = 0x3F indicates that no device is present for this instance.

The DSFI2 handler unit implements the GPOUT variable of the *HANDLER_OutputValue* data type to control general purpose outputs. Changes on GPOUT are assigned to outputs only if the Operational register is *0b1*. The *HANDLER_OutputValue* data type is 32 bit. *GPOUT[x]* is *0b0* as default. General purpose *output[x]* is set to *GPOUT[x]*.

The GPIN variable of the *HANDLER_InputValue* data type reads general purpose inputs. The *HANDLER_InputValue* data type is 32 bit. *GPIN[x]* is *0b0* as default. *GPIN[x]* reflects the status of general purpose *input[x]*. The Test variable of *HANDLER_Test* data type is writable and readable for debug purposes. The *HANDLER_Test* data type is 32 bit.

The Counter0..7 counter variables are 16 bit each and are writable. The counters are incremented/reset depending on the external connection.

The *GA[5..0]* variable represents the geographical address of the DSFI2 handler unit and the device, if available.

The *codetest_id[31..0]* and *codetest_val[31..0]* variables are used for debug purposes only. Significant parts of sources can be identified by writing unique values to the *codetest_id* variable. Values used for debug purposes can be written to *codetest_val*.

The DSFI2 handler unit provides the *INSTANCE_INTERRUPT* variable to provide the interrupt status of DSFI2 I/O controllers. The *INSTANCE_INTERRUPT* variable is mapped as follows:

Table 75. INSTANCE_INTERRUPT mapping

INSTANCE_INTERRUPT [x]	31..1	0
Interrupt source	DSFI2 instance 31..1	DSFI2 instance 0 (DSFI2 handler unit)

Table 76. INSTANCE_INTERRUPT states

InterruptStatus [x]	Description
0	No interrupt pending
1	Interrupt pending

The DSFI2 handler unit is implemented as DSFI2 instance 0. The DSFI2 handler unit implements the instances variable of the *HANDLER_Instances* data type. Instances are read only. The *HANDLER_Instances* data type is 32 bit. *Instances[x]* are *0b1* if instance number x is implemented. Otherwise *Instances[x]* is *0b0*.

The DSFI2Header variable of the *HANDLER_DSFI2Header* data type is read only. The *HANDLER_DSFI2Header* data type is 24 byte. The *DSFI2Header* has the following mapping:

Table 77. DSFI2Header

DSFI2Header[byte x]	Variable	Description
0x00..0x02	0xABD	Magic Word
0x03	DSFI2ProtocolType	DSFI2 Protocol type 0x00 - DSFI2 version 0
0x04..0x0F	Fpgald[95..0]	ASCII string of FPGA programming file name (e.g. "F403-001C001")
0x10	FpgaRevMin[7..0]	FPGA minor revision
0x11	FpgaRevMaj[7..0]	FPGA major revision
0x12..0x13	0x0000	Reserved
0x14..0x17	Instances[31..0]	Implemented instances as defined above

4.5.16.1 ***SetOutputValue***

The *SetOutputValue* function updates the *OpVariables.GpOut* variable with the content of *OutputValueIn*.

Table 78. SetOutputValue function

Write variables	<i>OpVariables.OutputValue</i>
Read variables	-
Request payload	<i>HANDLER_OutputValue OutputValueIn</i>
Acknowledge payload	None

4.5.16.2 ***GetOutputValue***

The *GetOutputValue* function returns the *OpVariables.GpOut* variable in *OutputValueOut*.

Table 79. GetOutputValue function

Write variables	-
Read variables	<i>OpVariables.OutputValue</i>
Request payload	None
Acknowledge payload	<i>HANDLER_OutputValue OutputValueOut</i>

4.5.16.3 ***RegisterTest***

The *RegisterTest* function returns the contents of *TestIn*.

Table 80. RegisterTest function

Write variables	-
Read variables	-
Request payload	<i>HANDLER_Test TestIn</i>
Acknowledge payload	<i>HANDLER_Test TestOut</i>

4.5.16.4 ***GetInputValue***

The *GetInputValue* function returns the *StatusVariables.GpIn* variable in *InputValueOut*.

Table 81. GetInputValue function

Write variables	-
Read variables	<i>StatusVariables.GpIn</i>
Request payload	None
Acknowledge payload	<i>HANDLER_InputValue InputValueOut</i>

4.5.16.5 **GetDSFI2Header**

The *GetDSFI2Header* function returns the *StatusVariables.DSFI2Header* variable in *DSFI2HeaderOut*.

Table 82. GetDSFI2Header function

Write variables	-
Read variables	<i>StatusVariables.DSFI2Header</i>
Request payload	None
Acknowledge payload	<i>HANDLER_DSFI2Header DSFI2HeaderOut</i>

4.5.16.6 **GetInstanceInterrupt**

The *GetInstanceInterrupt* function returns the *StatusVariables.Instance_Interrupt* variable in *InstInt*.

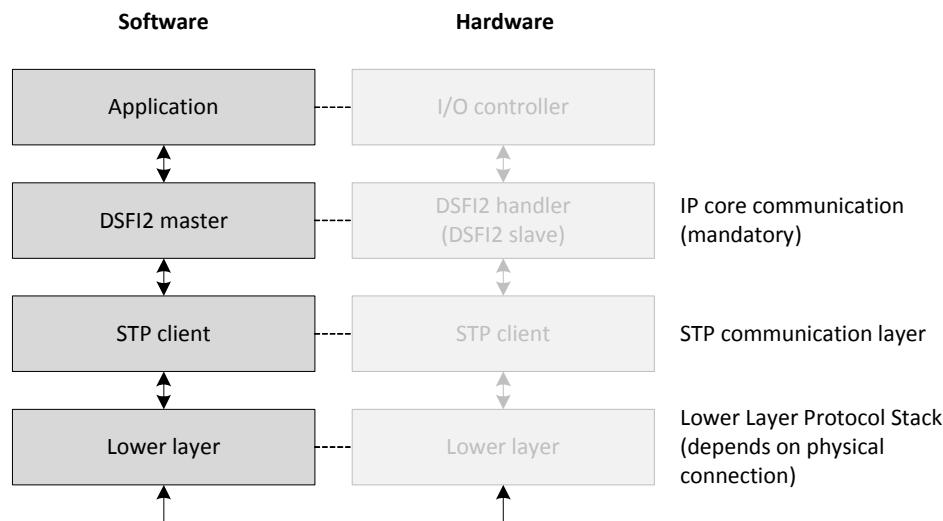
Table 83. GetInstanceInterrupt function

Write variables	-
Read variables	<i>StatusVariables.InstanceInterrupt</i>
Request payload	None
Acknowledge payload	<i>HANDLER_InstInt InstInt</i>

4.6 STP Layer

The Serial Transmission Protocol (STP) is a byte-based transmission protocol used for point-to-point connections. Data is transmitted in STP telegrams. As it is not required for point-to-point connections, the STP does not support addressing. The STP provides framing of data octets of any length. However, in case of the F403 the length is limited by internal buffers.

Figure 25. STP layer communication



The STP layer receives data from the upper layer. The received data is assembled to a STP telegram and is sent to the lower layer.

The STP layer receives data from the lower layer. The received STP telegram is sent to the upper layer.

4.6.1 Framing and Bit Stuffing

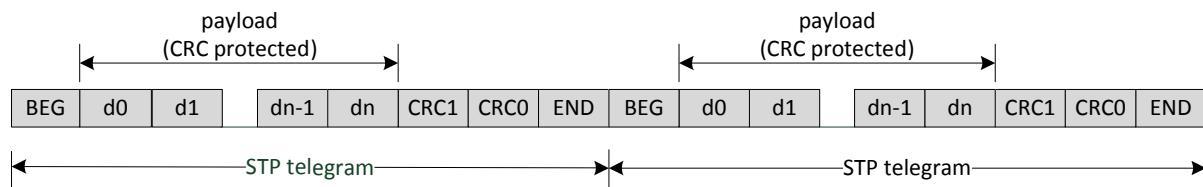
STP defines three special characters: END (0xC0), ESC (0xDB), BEG (0xC5) and RST (0xA5).

- If a data byte is the same code as END, a two byte sequence of ESC and 0xDC is sent instead.
- If a data byte is the same code as ESC, a two byte sequence of ESC and 0xDD is sent instead.
- If a data byte is the same code as RST, a two byte sequence of ESC and 0xDE is sent instead.
- If a data byte is the same code as BEG, a two byte sequence of ESC and 0xDF is sent instead.

BEG is transmitted before the first byte in the telegram and END is transmitted after the last byte in the telegram.

A 16-bit CRC is calculated over STP payload data and is appended to STP data. The CRC is appended before the END character is inserted. Received STP telegrams are discarded.

Figure 26. STP overview



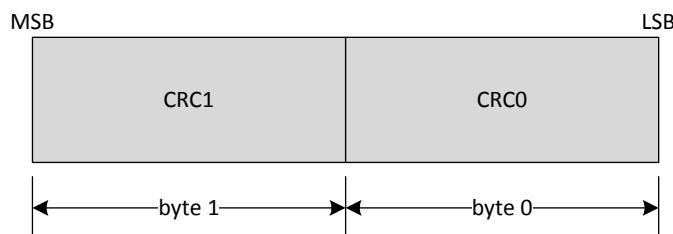
4.6.1.1 Reset

If the RST character is received, the communication channel is reset.

4.6.1.2 CRC

A 16-bit CRC is used to ensure integrity of the STP payload. Received STP frames are checked for CRC errors. The polynomial 0x139B7 is used for calculating the CRC. The initial value 0xFFFF is used for CRC calculation.

Figure 27. STP CRC

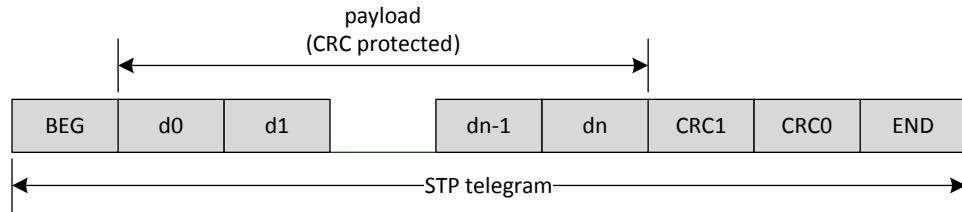


Received STP telegrams are checked for CRC errors. Erroneous telegrams are discarded and ignored.

4.6.1.3 STP Telegram Layout

An STP telegram has the following layout:

Figure 28. STP telegram



4.6.1.4 STP Special Characters

The following characters are not used in payload and are replaced as follows:

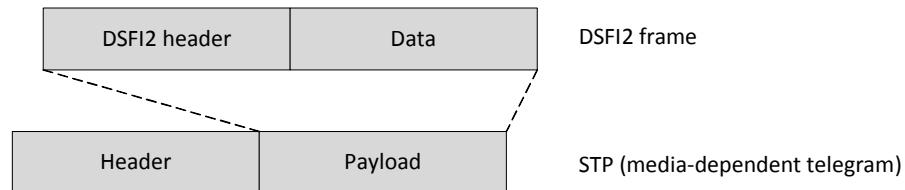
Table 84. STP special characters

Character		Replaced by	Description
Name	Value		
BEG	0xC5	ESC + 0xDF	Beginning of telegram
END	0xC0	ESC + 0xDC	End of telegram
RST	0xA5	ESC + 0xDE	Reset command
ESC	0xDB	ESC + 0xDD	Escape character

4.7 Telegram Layout

The following message layout is applied to DSFI2 frames.

Figure 29. Request layout



5 Maintenance

5.1 Updating Firmware in the FPGA

MEN's FPGA update tool – *fpga_load* – offers the possibility to load data in binary format to Flash memory. The tool is supposed to be used for firmware updates only.



See the MEN [website](#) for the FPGA update tool and the detailed user manual.

Some bus architectures swap the bytes within a word or the words within a long word. The latest update of *fpga_load* recognizes automatically if swapping is necessary or not. You should use the latest version of the update tool. If you use an older version, please refer to the tool's inline documentation for hints on byte swapping.

5.1.1 Installing the Update Tool

5.1.1.1 Windows

MEN's article number of the Windows update tool is 13Y018-70.



Download the ZIP file from the MEN website:
www.men.de/software/13y018-70/

- » Unpack it.
- » Execute the *fpga_load.exe* file.

5.1.1.2 Linux

MEN's article number of the Linux update tool is 13Z100-91.



Download the ZIP file from the MEN website:
www.men.de/software/13z100-91/

- » Unpack it.
- » The tool is now present as an executable binary file (*fpga_load*).

5.1.1.3 VxWorks

MEN's article number of the VxWorks update tool is 13Z100-60.



Download the ZIP file from the MEN website:
www.men.de/software/13z100-60/

- » Unpack it.

- » Integrate *program.mak* into an existing MDIS make file. Modify the paths of the *.mak* files if needed:

```
ALL_COM_TOOLS := \
    MDIS_API/M_MOD_ID/COM/program.mak \
    MDIS_API/M_REV_ID/COM/program.mak \
    ../../COM/TOOLS/FPGA_LOAD/COM/program.mak \
```

- » Execute *make*.

5.1.2 Using the Update Tool

You can call the help function simply by entering *fpga_load*. This outputs a list of possible parameters for *fpga_load*.

If you use the tool under VxWorks, you need to put any parameter of *fpga_load* in quotation marks, e.g. *fpga_load "s"*. This is not necessary under Windows or Linux.

The following shows how to use *fpga_load* using some examples under Windows.

5.1.3 Using the Tool with the F403

You can look up the Vendor ID, Device ID and Subsystem Vendor ID of the F403 to pass it to the tool. You can do this using the *fpga_load* parameter *-s*:

```
C:\mysystem\13z100-70\13z10070\NT\OBJ\EXE\MEN\I386\FREE>fpga_load -s
```

Nr.	bus	dev	fun	Ven ID	Dev ID	SubVen ID
0	0	0	0	0x8086	0x2a40	0x8086
1	0	1	0	0x8086	0x2a41	0x0000
2	0	2	0	0x8086	0x2a42	0x8086
3	0	2	1	0x8086	0x2a43	0x8086
4	0	26	0	0x8086	0x2937	0x8086
5	0	26	1	0x8086	0x2938	0x8086
6	0	26	7	0x8086	0x293c	0x8086
7	0	28	0	0x8086	0x2940	0x0000
8	0	28	4	0x8086	0x2948	0x0000
9	0	28	5	0x8086	0x294a	0x0000
10	0	29	0	0x8086	0x2934	0x8086
11	0	29	1	0x8086	0x2935	0x8086
12	0	29	2	0x8086	0x2936	0x8086
13	0	29	3	0x8086	0x2939	0x8086
14	0	29	7	0x8086	0x293a	0x8086
15	0	30	0	0x8086	0x2448	0x0000
16	0	31	0	0x8086	0x2917	0x8086
17	0	31	2	0x8086	0x2928	0x8086
18	0	31	3	0x8086	0x2930	0x8086
19	0	31	5	0x8086	0x292d	0x8086
20	0	31	6	0x8086	0x2932	0x8086
21	3	0	0	0x8086	0x10d3	0x8086
22	4	0	0	0x8086	0x10d3	0x8086
23	5	10	0	0x1a88	0x4d45	0x00bc

You also need the offset address from the beginning of the FPGA configuration memory space. To find it out, you need to have a look at the Chameleon table. To do this you can again use MENMON or the *fpga_load* tool itself, through parameter *-t*. The following selects vendor ID **0x1A88**, device ID **0x4D45**, subsystem vendor ID **0xBC**, instance 0:

```
C:\mysystem\13z100-70\13z10070\NT\OBJ\EXE\MEN\I386\FREE>fpga_load 1A88 4D45 BC 0 -t
Cham_Info for device 5/10/0 (bus/dev/fun):
chaRev: 2; busId: 0; tableNbr 1;
unitNbr: 4; bridgeNbr: 0; cpuNbr 0;
BAR0: 0x93500000; size: 0x00000000, mapType: MEM;
BAR1: 0x00001001; size: 0x00000000, mapType: IO;
BAR2: 0x00000000; size: 0x00000000, mapType: unused;
BAR3: 0x00000000; size: 0x00000000, mapType: unused;
BAR4: 0x00000000; size: 0x00000000, mapType: unused;
BAR5: 0x00000000; size: 0x00000000, mapType: unused;
CHAMELEONV2_HEADER # 0:
  busType=0x00, busId=0, model=0, revision=0x02,
  file=F403-00IC001, magicWord=0xabce
CHAMELEONV2_UNIT:

Idx DevId Module          Grp Inst Var Rev IRQ BAR Offset      Address
--- -----
 0 0x0018 16Z024_SRAM        0    0   1  13  63  0 0x00000000 0x93500000
 1 0x007e 16Z126_SERFLASH   0    0   0   8  63  0 0x00000200 0x93500200
 2 0x0063 ?                  0    0   0   1  63  0 0x00000280 0x93500280
 3 0x0063 ?                  0    0   0   1   4  1 0x00000000 0x1001
```

Now you can write e.g. your binary file to Flash. Parameter *-u* followed by the binary file name and the FPGA configuration number (0 for the F403) will update the FPGA with the offset address in Flash being read from the Flash header. This way you can avoid accidental overwriting of the fallback data.

```
C:\mysystem\13z100-70\13z10070\NT\OBJ\EXE\MEN\I386\FREE>fpga_load 1172 4D45 BC 0 -u F403-00IC001A1_02_00.bin 0
```

This is how you can update the factory image:

```
C:\mysystem\13z100-70\13z10070\NT\OBJ\EXE\MEN\I386\FREE>fpga_load 1a88 4d45 bc 0 -z -f -w
F403-00IC001A1_02_00.rbf 0 -v
Z100_PCIInit

Update device:
Nr.|bus|dev|fun|   BAR0 |   BAR1 |   BAR2 |   BAR3 |   BAR4 |   BAR5
 0 5 10 0 93500000 00001001 00000000 00000000 00000000 00000000
Cham_Info for device 5/10/0 (bus/dev/fun):
chaRev: 2; busId: 0; tableNbr 1;
unitNbr: 4; bridgeNbr: 0; cpuNbr 0;
BAR0: 0x93500000; size: 0x00000000, mapType: MEM;
BAR1: 0x00001001; size: 0x00000000, mapType: IO;
BAR2: 0x00000000; size: 0x00000000, mapType: unused;
BAR3: 0x00000000; size: 0x00000000, mapType: unused;
BAR4: 0x00000000; size: 0x00000000, mapType: unused;
BAR5: 0x00000000; size: 0x00000000, mapType: unused;
CHAMELEONV2_HEADER # 0:
  busType=0x00, busId=0, model=0, revision=0x01,
  file=F403-00IC001, magicWord=0xabce
CHAMELEONV2_UNIT:

Idx DevId Module          Grp Inst Var Rev IRQ BAR Offset     Address
--- -----
 0 0x0018 16Z024_SRAM      0    0   1  13  63  0 0x00000000 0x93500000
 1 0x007e 16Z126_SERFLASH 0    0   0   8  63  0 0x00000200 0x93500200

Init_Flash
FLASH::stm25p32::Identify mnf=0015 dev=0020
FLASH::stm25p32::CheckId
  Found FLASH device STMICROM25P32 4M
Write_block
ERASE Sectors: Please wait
Unlocking sector
Erasing sector 1 of 64, size = 0x010000
  Erase Sector ----> OK
Erasing sector 2 of 64, size = 0x010000
  Erase Sector ----> OK
Erasing sector 3 of 64, size = 0x010000
  Erase Sector ----> OK
  ----> OK

PROGRAM offset 0x00000000, size 0x0002eba3: Please wait ...
Unlocking sector
Current write status: 131072 bytes written from 191395 bytes !
VERIFY: Please Wait ...Write and Verify OK
```

This is how you can update the user image:

```
C:\mysystem\13z100-70\13z10070\NT\OBJ\EXE\MEN\I386\FREE>fpga_load 1a88 4d45 bc 0 -u -w
F403-00IC001A1_02_00.bin 80000 -v
Z100_PCIInit

Update device:
Nr.|bus|dev|fun|   BAR0 |   BAR1 |   BAR2 |   BAR3 |   BAR4 |   BAR5
  0 5 10 0 93500000 00001001 00000000 00000000 00000000 00000000
Cham_Info for device 5/10/0 (bus/dev/fun):
chaRev: 2; busId: 0; tableNbr 1;
unitNbr: 4; bridgeNbr: 0; cpuNbr 0;
BAR0: 0x93500000; size: 0x00000000, mapType: MEM;
BAR1: 0x00001001; size: 0x00000000, mapType: IO;
BAR2: 0x00000000; size: 0x00000000, mapType: unused;
BAR3: 0x00000000; size: 0x00000000, mapType: unused;
BAR4: 0x00000000; size: 0x00000000, mapType: unused;
BAR5: 0x00000000; size: 0x00000000, mapType: unused;
CHAMELEONV2_HEADER # 0:
  busType=0x00, busId=0, model=0, revision=0x02,
  file=F403-00IC001, magicWord=0xabce
CHAMELEONV2_UNIT:

Idx DevId Module          Grp Inst Var Rev IRQ BAR Offset      Address
--- -----
  0 0x0018 16Z024_SRAM           0    0   1 13 63 0 0x00000000 0x93500000
  1 0x007e 16Z126_SERFLASH       0    0   0   8 63 0 0x00000200 0x93500200

Init_Flash
FLASH::stm25p32::Identify mnf=0015 dev=0020
FLASH::stm25p32::CheckId
  Found FLASH device STMICROM25P32 4M
Write_block
ERASE Sectors: Please wait
Unlocking sector
Erasing sector 9 of 64, size = 0x010000
  Erase Sector ----> OK
Erasing sector 10 of 64, size = 0x010000
  Erase Sector ----> OK
Erasing sector 11 of 64, size = 0x010000
  Erase Sector ----> OK
  ----> OK

PROGRAM offset 0x00080000, size 0x0002eca4: Please wait ...
Unlocking sector
Current write status: 131072 bytes written from 191652 bytes !
VERIFY: Please Wait ...Write and Verify OK
```