# EASY3445 - M13FX Evaluation System

## M13 Multiplexer and DS3 Framer

## PEB3445


Datacom

Infineon technologies

Never stop thinking.

**Edition 2000-07-15**

**Published by Infineon Technologies AG,**
**St.-Martin-Strasse 53,**
**D-81541 München, Germany**



**Attention please!**

The information herein is given to describe certain components and shall not be considered as warranted characteristics.

Terms of delivery and rights to technical change reserved.

We hereby disclaim any and all warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

Infineon Technologies is an approved CECC manufacturer.

**Information**

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office in Germany or our Infineon Technologies Representatives worldwide (see address list).

**Warnings**

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

# EASY3445 - M13FX Evaluation System

# M13 Multiplexer and DS3 Framer

# PEB3445 Version 1.1

Datacom

Never stop thinking.

**Revision History:** **2000-07-15** DS1

Previous Version:

| Page | Subjects (major changes since last revision) |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

For questions on technology, delivery and prices please contact the Infineon Technologies Offices in Germany or the Infineon Technologies Companies and Representatives worldwide: see our webpage at http://www.infineon.com

# 1 Overview

The EASY3445 evaluation system for the M13FX consists of modular hardware and software optimized to demonstrate the performance of the M13FX (PEB3445) device in a real communication environment.

## 1.1 Hardware

The hardware of the communication system is optimized for evaluation purposes of the M13FX device. The M13FX is controlled by a 16 bit microcontroller, and interfaces with a T3 LIU and two four-port T1/E1 LIU devices. This allows a very fast evaluation of the device in a real application where the hardware can be connected to standard T3 and T1/E1 test equipment. In addition, it provides connectors to access all interfaces of the device and two RS232 interfaces for code download, and control via a PC.

## 1.2 Software

The EASY3445 evaluation system software consists of the following main components:

- the Evaluation Software WinEasy, running on any Windows 95/98/NT PC, can be used as the main user interface to the hardware. It is used for code download and also to exchange messages via the RS232 serial interface with the Target System Software running on the EASY3445 hardware.
- the Target System Software runs on the EASY3445 hardware and consists of two main sub-modules: the Device Driver System (DDS) is a minimal operating system and the Device Driver Modules for M13FX and QuadLIU are the device-specific low level driver modules

All software is available in ANSI C source code and is supported by Infineon's application engineers. A particular attention has been taken to have the low level drivers (Device Driver Module) support multiple operating systems and multiple CPU platforms, so that the integration in a customer's application should require only very little modifications.

The low level driver provides all necessary functions that cover the functionality offered by the M13FX device (configuration, alarms, loopbacks, test unit, tributary mapper, error counters, FEAC and MDL channel).

## 1.3 Document Organization

Chapter **"Hardware Description"** provides the description of all components of the EASY3445, as well as a description of all test connectors. The schematics are provided in a separate file.

Chapter "**Target System Software Description**" describes in more details the architecture of the target system software with the M13FX device driver environment and the QuadLIU device driver environment.

Chapter “**DDS system**” describes in details the minimal kernel running the target software.

Chapter “**OPI concepts**” describes the OPI architecture which allows the M3FX device driver to be OS and platform independent.

Chapter “**M13FX C/I interface**” describes the Wineasy C/I message interface for the M13FX.

Chapter “**QuadLIU C/I interface**” describes the Wineasy C/I message interface for the QuadLIU devices.

The chapters “**M13FX Low-level interface**” and “**M13FX OPI interface**” describe the Application Programming Interface a customer could use to interface his application to the Infineon M13FX low level driver.

For more details on the Evaluation Software WinEasy running on the PC, please refer to the separate document “**Graphical Evaluation Software for EASY Tools Wineasy V2.4**”.

## 1.4 Applications

The EASY3445 evaluation system is best suited for the evaluation of the M13FX device. The software delivered with the EASY3445 package provides functions to evaluate all blocks of the M13FX (refer to chapter “**Target System Software Description**” chapter for details). The WinEasy application also allows for direct read or write access to any registers of the M13FX or QuadLIU devices.

The EASY3445 could also be used as a prototype development platform, in order to develop customer-specific software or to connect the EASY3445 hardware to existing customer’s hardware using the multiple connectors provided on-board.

A CD-ROM provides all necessary informations (data sheet, schematics, source code of the device driver software,...) about the M13FX and the EASY3445 evaluation system.

## 1.5 File Organization of the CD-ROM

The following description explains the organization of the several files on the delivered CD-ROM.

The EASY3445 software package contains the Device Driver Modules for the INFINEON Communication IC M13FX. It contains all files, which are necessary to build up the device driver software and run the PC evaluation software PEB3445.

The EASY3445 software package is organized in 5 main subdirectories which contains the following topics:

- **APPLICATION**, contains all the files necessary to run the WinEASY application with the script files for the M13FX and Quadliu devices. The subdirectory **WinEASY** contains the main files to run the WinEASY software. All command and indication

messages are stored in the EASY3445.CIM file which is automatically loaded when the Single Step Window is opened. The PEB3445.PDF file is the complete M13FX data sheet and is needed by the help utility (use the <F1> key to invoke). The subdirectory **SCRIPT files** contains a complete set of script files (*.trk) demonstrating the functionality of the M13FX device (subdirectory **m13fx**) and a set of script files in order to configure the Quadliu devices (subdirectory **quadliu**). The script files are loaded through the FILE menu of the WinEASY application.

- **DDS**, contains all the files necessary to build the DDS system. The subdirectory **SRC** contains the source C files, the subdirectory **INC** contains the header files (*.h). The subdirectory OBJ contains the result of the DDS compilation (*.obj, *.err). The makefile **DDSmake** builds the DDS subsystem.

- **DRV**, contains all files necessary to build the device driver modules embedded in the target (M13FX and Quadliu). The subdirectory **SRC** contains the source C files, the subdirectory **INC** contains the header files (*.h). The subdirectory **OBJ** contains the result of the DRV compilation (*.obj, *.err). The makefile **DRVmake** builds the DRV subsystem.

- **OPI**, contains all files necessary to build the OPI environment (Appli, OPI low-level interface and Debug for M13FX, SSAL for DDS system). The subdirectory **SRC** contains the source C files, the subdirectory **INC** contains the header files (*.h). The subdirectory **OBJ** contains the result of the OPI compilation (*.obj, *.err). The makefile **OPImake** builds the OPI subsystem.

- **GEN**, contains all files necessary to build the M13FX evaluation board software. The subdirectory **TOOLS** contains the GNU make utility (GMAKE.EXE) which uses the make files in order to compile the source files, link the object files and create the **M13FXV1** hex file. This hex file will be downloaded to the Evaluation System. The **LIB** subdirectory contains the firmware library object files which are linked with the other subsystem in order to build the target software. The makefile **m13make** generates the the downloadable M13FXV1.hex file according to the linker command file **m13lnk** which specifies the object files to link and the address mapping. The m13fxv1.m66 (map file), m13fxv1.o66 (absolute file) and m13fxv1.hex files are generated into the **GEN** directory.

**Figure 1** illustrates the file structure of the EASY3445 software package.

Table 1 provides a list of the DRV directory source files.

Table 2 provides a list of the OPI directory source files.

Table 3 provides a list of the DDS directory source files.

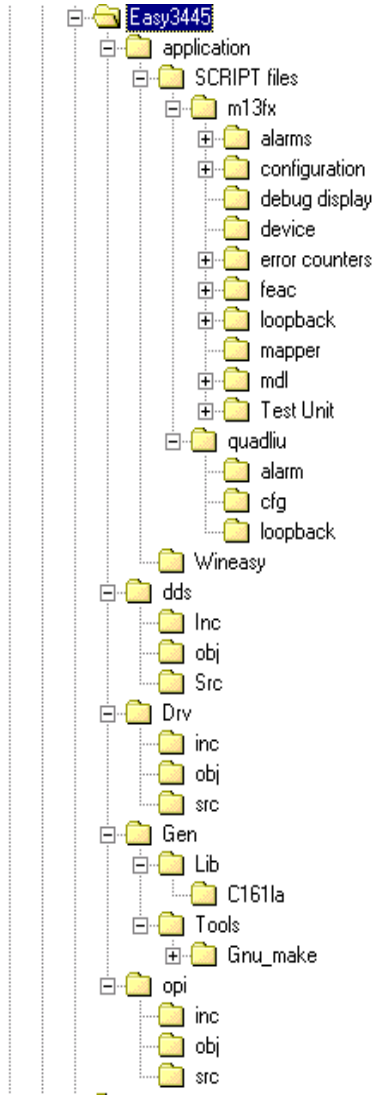



**Figure 1 PEB3445 software file structure**

**Table 1 DRV directory source files**

| File name | Description |
|---|---|
| ***DRV\SRC directory*** | |
| m13fx.c | M13FX initialization, configuration and Interrupts Service Routine module. |
| m13alrm.c | M13FX alarms module. |
| m13errc.c | M13FX Error Counters module. |
| m13feac.c | M13FX Feac channel module. |
| m13lbk.c | M13FX Loopback module. |
| m13map.c | M13FX Tributary Mapper module. |
| m13mdl.c | M13FX Maintenance Data link module. |
| m13tu.c | M13FX Test Unit module. |
| Quadliu.c | QUADLIU Device Driver Module. |
| ***DRV\INC directory*** | |
| m13fx.h | M13FX local header file. |
| m13fxdx.h | M13FX export header file. |
| quadliu.h | QuadLIU header file. |

**Table 2 OPI directory source files**

| File name | Description |
|---|---|
| ***OPI\SRC directory*** | |
| m13appl.c | APPLI for M13FX module. |
| m13opi.c | M13FX OPI Low-level interface module. |
| m13ssal.c | SSAL for DDS module. |
| m13dbg.c | DEBUG for M13FX module. |
| ***OPI\INC directory*** | |
| m13ssalx.h | SSAL for DDS export header file. |
| m13ssal.h | SSAL for DDS local header file. |
| m13opix.h | M13FX OPI interface export header file. |
| m13opi.h | M13FX OPI interface local header file. |
| opidrv.h | OPI low-level interface local header file. |
| opiext.h | OPI low-level interface export header file. |

**Table 2 OPI directory source files**

| File name | Description |
|---|---|
| m13appl.h | APPLI for M13FX header file. |
| m13dbg.h | DEBUG for M13FX header file. |

**Table 3 DDS directory source files**

| File name | Description |
|---|---|
| ***DDS\SRC directory*** | |
| DDSboard.c | DDS- Device Driver Module initialization. |
| DDSD.c | DDS - Debug module. |
| DDSH.c | DDS- Interrupt service module. |
| DDSI.c | DDS- integration module. |
| DDSK.c | DDS- Kernel module. |
| DDSM.c | DDS- Memory module. |
| DDSU.c | DDS- User interface module. |
| ***DDS\INC directory*** | |
| C161.h | C161 register access header file. |
| c161_fa.h | C161 firmware header file. |
| tgtboard.h | target board header file. |
| ddsboard.h | DDS - Device Driver Module Initialization header file. |
| ddsd.h | DDS - Debug header file. |
| ddsg.h | DDS - Message header file. |
| ddsh.h | DDS - Interrupt header file. |
| ddsi.h | DDS - integration header file. |
| ddsk.h | DDS- kernel header file. |
| ddsm.h | DDS- Memory header file. |
| dxi.h | DDS - data exchange header file. |
| dxisio.h | DDS - Serial Port header file. |
| interrpt.h | Interrupts header file. |
| msg.h | DDS - module identification header file. |
| compdef.h | DDS - general definition header file. |
| assert.h | customized header file. |

**Table 3 DDS directory source files**

| File name | Description |
|---|---|
| time.h | customized header file. |
| timer.h | customized header file. |
| stdio.h | customized header file. |

# 2 Hardware Description

## 2.1 General Hardware Architecture

The M13FX is connected to a T3 Line Interface Unit (LIU) in order to provide connectivity to standard T3 test equipment via two BNC connectors, and to two Infineon QuadLIU devices (PEB22504) in order to provide access to up to 8 DS1 or E1 interfaces and to connect to standard T1/E1 test equipment via RJ45 connectors. In this respect the EASY3445 evaluation board provides the complete functionality of a DS1/E1-DS3 multiplexer.

In addition the Hardware of the EASY3445 evaluation system allows access to all interfaces of the M13FX via connectors for measurement purposes and for connecting to other T3 LIU or T1/E1 LIU devices.

The hardware contains the Infineon C161-O 16 bit CPU, the system memory, the flash-eprom, the Infineon M13FX device, two QuadLIU devices, one TDK T3 Line Interface Unit, three clock oscillators (for the QuadLIU, the CPU and the T3 LIU) and two serial interfaces used for code download and debug terminal.

The hardware board is powered with a standard regulated AC/DC 5V power supply that is included with the EASY3445 package. All the voltage dividers required by the M13FX and QuadLIU devices are on-board.

The following figure shows a block diagram of the EASY3445 hardware highlighting the default jumper configuration. The following chapters provide a more detailed description of all jumpers and access connectors.

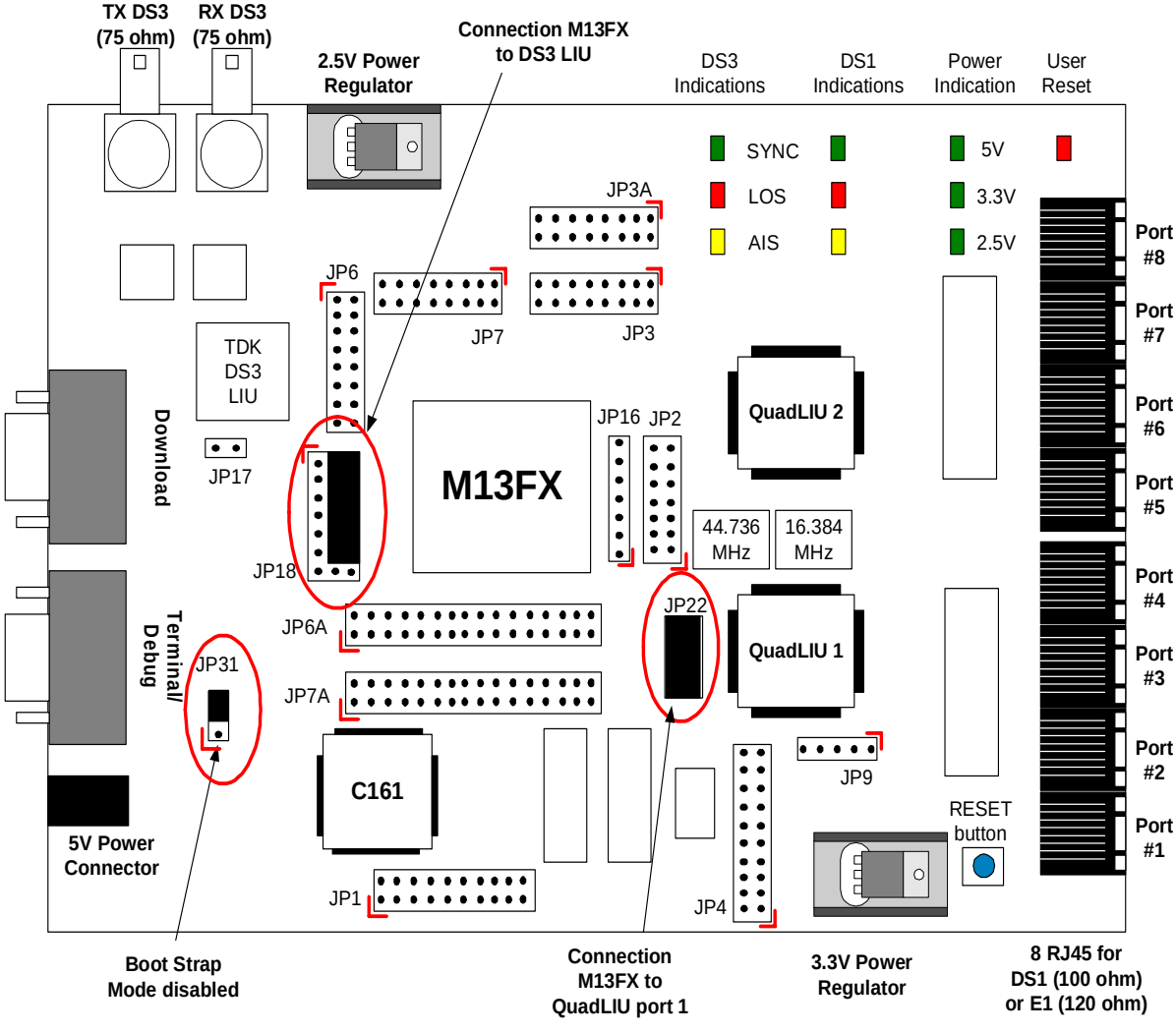



**Figure 2 EASY3445 Block Diagram and Default Jumper Setting**

## 2.2 The C161 Microcontroller System

The Processor System provided on the EASY3445 board consists of all components required for the evaluation of the M13FX with the standard delivered software, as well as the development of additional customer application specific software supporting debugging capabilities.

The key features are as follows:

- Infineon C161-O CPU operating with 16MHz CPU-Speed and 2kB internal RAM
- 1 MByte SRAM System Memory
- 512kB Flash-Eprom for Firmware and User-Application Software
- Standard serial RS232 Interface to the PC (e.g. for code download and control)
- Serial Interface with 19.2kBaud for Debug output
- Bootstrap-Loader (for software debug mode)

The FLASH-EPROM, SRAM, M13FX and the QuadLIU devices can be directly accessed by software via different chip-select signals (refer also to the schematic). The configuration of the micro-controller allows 1MB address-range with each chip-select line. The mapping is as follows:

**Table 4 Chip Select Signals, Memory Map**

| Pin Name | Controlled Device | Address Range |
|---|---|---|
| $\overline{CS0}$ | Flash-Eprom | 0x0C00000 - 0x0C7FFFF |
| $\overline{CS1}$ | SRAM | 0x0000000 - 0x00FFFFF |
| $\overline{CS2}$ | QuadLIU0 (see Note) | 0x0400000 - 0x040007F |
| | QuadLIU1 (see Note) | 0x0410000 - 0x041007F |
| $\overline{CS3}$ | M13FX | 0x0800000 - 0x08000FF |

*Note: $\overline{CS2}$ is used to control both QuadLIU devices via a GAL that uses the address pin A16 to distinguish between the address ranges.*

The C161 Microcontroller also controls the interrupts originated by the M13FX and both QuadLIU devices. The interrupt line of each communication device is routed directly to the C161.

The interrupt vector mapping is as follows:

**Table 5 Interrupt Vector Assignment**

| Controlled Device | Interrupt Vectors |
|---|---|
| QuadLIU0 | 0x1A |
| QuadLIU1 | 0x1B |
| M13FX | 0x19 |

In order to provide for easy connection of the microcontroller bus to a logic analyzer, the access connectors JP1 and JP4 provide the complete address and data bus, as well as read and write signals.

**Table 6 Pin Description of the Access Connector ‘JP1’**

| Signal | JP1 Pin Number |
|---|---|
| D0 | 2 |
| D1 | 3 |
| D2 | 4 |
| D3 | 5 |
| D4 | 6 |
| D5 | 7 |
| D6 | 8 |
| D7 | 9 |
| D8 | 10 |
| D9 | 11 |
| D10 | 12 |
| D11 | 13 |
| D12 | 14 |
| D13 | 15 |
| D14 | 16 |
| D15 | 17 |
| RDQ (Read) | 18 |
| WRQ (Write) | 19 |

**Table 7 Pin Description of the Access Connector ‘JP4’**

| Signal | JP4 Pin Number |
|---|---|
| A0 | 1 |
| A1 | 2 |
| A2 | 3 |
| A3 | 4 |
| A4 | 5 |
| A5 | 6 |
| A6 | 7 |
| A7 | 8 |
| A8 | 9 |
| A9 | 10 |
| A10 | 11 |
| A11 | 12 |
| A12 | 13 |
| A13 | 14 |
| A14 | 15 |
| A15 | 16 |
| A16 | 17 |
| A17 | 18 |
| A18 | 19 |
| A19 | 20 |

### 2.2.1 Jumper Settings of the Microcontroller

Only two jumpers have to be set for correct microcontroller operation (please also refer to **Chapter 2.2.2**):

- JP25: Always ON; it allows hardware reset via serial interface pin DTR (serial port marked DOWNLOAD).
- JP31: Default is 2-3 and selects the control of bootstrap mode via serial interface pin RTS (serial port marked DOWNLOAD).

If jumper JP31 is set in position 1-2, bootstrap mode is selected (refer to **Chapter 2.2.3**).

### 2.2.2 Power-Up Configuration of the Microcontroller

The configuration of the C161 microcontroller is programmed with weak Pull-Down resistors on Data-lines during power-up. The bus type is selected with D6 and D7 as 16Bit demultiplexed Data-Bus. The Address-Space is programmed with D11 and D12 to 1MB. With the weak Pull-Down resistor on D15 the internal clock divider is programmed to 1. So the operating speed of the C161 results to 16MHz.

### 2.2.3 The Bootstrap Loader

In order to use the bootstrap mode (e.g. for using the debugger environment), configure jumper JP31 in position 1-2 and use a serial DB9 cable with no connection for pins DTR and RTS (only connect pins 2, 3 and 5 of the serial connector).

**Table 8 Pin Description of the DB9 bootstrap cable connectors**

| Signal | DB9 Pin Number |
|---|---|
| TXD (Transmit Data) | 2 |
| RXD (Receive Data) | 3 |
| GND | 5 |

Refer to the C161 data sheet and to Keil Microvision documentation for more information about this mode.

## 2.3 The M13 Multiplexer and DS3 Framer (M13FX)

The M13FX integrates a DS3 framer, with a M13 multiplexer (that multiplexes 28 DS1 lines or 21 E1 lines into one DS3 line) and a tributary interchanger or line selector between the 28 multiplexed lines and 32 external serial line interfaces.

The DS3 framer block interfaces directly to a T3 LIU from TDK Semiconductors (refer to **Chapter 2.5**). It is also possible to disable the connection to the T3 LIU and have access to all DS3 interface signals using jumper JP18.

**Table 9 Pin Description of the Access Connector JP18**

| Signal | M13FX Pin Description | 'JP18' Pin Number |
|---|---|---|
| RCLK44 | DS3 Receive Clock Input | 3 |
| RD44N | DS3 Receive Negative Pulse (Input) | 6 |
| RD44P | DS3 Receive Positive Pulse (Input) | 9 |
| TCLK44 | DS3 Transmit Clock Output | 12 |
| TD44N | DS3 Transmit Negative Pulse (Output) | 15 |
| TD44P | DS3 Transmit Positive Pulse (Output) | 18 |

By default the connection to the TDK LIU requires the following jumpers to be set on jumper JP18: 2-3, 5-6, 8-9, 11-12, 14-15, 17-18.

In case the M13FX is connected to an other T3 LIU (e.g. PUCCINI), these jumpers should be removed. It is possible to connect very easily the M13FX evaluation board to the PUCCINI evaluation board by connecting the first row of pins of JP18 on the M13FX board to the corresponding signals on the PUCCINI evaluation board. There should be also a common GND between the two boards (GND header pins are available on each board).

The DS1/E1 serial line interfaces of the M13FX are all accessible on the connectors JP3, JP3a, JP6, JP6a, JP7 and JP7a. In addition, the DS1/E1 line interface of ports 1 to 8 is also connected to two T1/E1 QuadLIU devices (refer to **Chapter 2.4**). The following tables show the pinout of the DS1/E1 line interface connectors.

**Table 10 Pin Description of the Access Connector JP3**

| Signal | M13FX Pin Description | 'JP3' Pin Number |
|---|---|---|
| TTC1 | Tributary Transmit Clock 1 | 1 |
| TTC2 | Tributary Transmit Clock 2 | 3 |
| TTC3 | Tributary Transmit Clock 3 | 5 |
| TTC4 | Tributary Transmit Clock 4 | 7 |
| TTC5 | Tributary Transmit Clock 5 | 9 |
| TTC6 | Tributary Transmit Clock 6 | 11 |

**Table 10 Pin Description of the Access Connector JP3** (cont'd)

| Signal | M13FX Pin Description | 'JP3' Pin Number |
|---|---|---|
| TTC7 | Tributary Transmit Clock 7 | 13 |
| TTC8 | Tributary Transmit Clock 8 | 15 |
| RTC1 | Tributary Receive Clock 1 | 2 |
| RTC2 | Tributary Receive Clock 2 | 4 |
| RTC3 | Tributary Receive Clock 3 | 6 |
| RTC4 | Tributary Receive Clock 4 | 8 |
| RTC5 | Tributary Receive Clock 5 | 10 |
| RTC6 | Tributary Receive Clock 6 | 12 |
| RTC7 | Tributary Receive Clock 7 | 14 |
| RTC8 | Tributary Receive Clock 8 | 16 |

**Table 11 Pin Description of the Access Connector JP3a**

| Signal | M13FX Pin Description | 'JP3a' Pin Number |
|---|---|---|
| TTD1 | Tributary Transmit Data 1 | 1 |
| TTD2 | Tributary Transmit Data 2 | 3 |
| TTD3 | Tributary Transmit Data 3 | 5 |
| TTD4 | Tributary Transmit Data 4 | 7 |
| TTD5 | Tributary Transmit Data 5 | 9 |
| TTD6 | Tributary Transmit Data 6 | 11 |
| TTD7 | Tributary Transmit Data 7 | 13 |
| TTD8 | Tributary Transmit Data 8 | 15 |
| RTD1 | Tributary Receive Data 1 | 2 |
| RTD2 | Tributary Receive Data 2 | 4 |
| RTD3 | Tributary Receive Data 3 | 6 |
| RTD4 | Tributary Receive Data 4 | 8 |
| RTD5 | Tributary Receive Data 5 | 10 |
| RTD6 | Tributary Receive Data 6 | 12 |
| RTD7 | Tributary Receive Data 7 | 14 |
| RTD8 | Tributary Receive Data 8 | 16 |

**Table 12 Pin Description of the Access Connector JP6**

| Signal | M13FX Pin Description | 'JP3' Pin Number |
|---|---|---|
| TTC9 | Tributary Transmit Clock 9 | 1 |
| TTC10 | Tributary Transmit Clock 10 | 3 |
| TTC11 | Tributary Transmit Clock 11 | 5 |
| TTC12 | Tributary Transmit Clock 12 | 7 |
| TTC13 | Tributary Transmit Clock 13 | 9 |
| TTC14 | Tributary Transmit Clock 14 | 11 |
| TTC15 | Tributary Transmit Clock 15 | 13 |
| TTC16 | Tributary Transmit Clock 16 | 15 |
| RTC9 | Tributary Receive Clock 9 | 2 |
| RTC10 | Tributary Receive Clock 10 | 4 |
| RTC11 | Tributary Receive Clock 11 | 6 |
| RTC12 | Tributary Receive Clock 12 | 8 |
| RTC13 | Tributary Receive Clock 13 | 10 |
| RTC14 | Tributary Receive Clock 14 | 12 |
| RTC15 | Tributary Receive Clock 15 | 14 |
| RTC16 | Tributary Receive Clock 16 | 16 |

**Table 13 Pin Description of the Access Connector JP7**

| Signal | M13FX Pin Description | 'JP3' Pin Number |
|---|---|---|
| TTD9 | Tributary Transmit Clock 9 | 1 |
| TTD10 | Tributary Transmit Clock 10 | 3 |
| TTD11 | Tributary Transmit Clock 11 | 5 |
| TTD12 | Tributary Transmit Clock 12 | 7 |
| TTD13 | Tributary Transmit Clock 13 | 9 |
| TTD14 | Tributary Transmit Clock 14 | 11 |
| TTD15 | Tributary Transmit Clock 15 | 13 |
| TTD16 | Tributary Transmit Clock 16 | 15 |
| RTD9 | Tributary Receive Clock 9 | 2 |
| RTD10 | Tributary Receive Clock 10 | 4 |
| RTD11 | Tributary Receive Clock 11 | 6 |

**Table 13 Pin Description of the Access Connector JP7** (cont'd)

| Signal | M13FX Pin Description | 'JP3' Pin Number |
|---|---|---|
| RTD12 | Tributary Receive Clock 12 | 8 |
| RTD13 | Tributary Receive Clock 13 | 10 |
| RTD14 | Tributary Receive Clock 14 | 12 |
| RTD15 | Tributary Receive Clock 15 | 14 |
| RTD16 | Tributary Receive Clock 16 | 16 |

**Table 14 Pin Description of the Access Connector JP6a**

| Signal | M13FX Pin Description | 'JP3' Pin Number |
|---|---|---|
| TTC17 | Tributary Transmit Clock 17 | 1 |
| TTC18 | Tributary Transmit Clock 18 | 3 |
| TTC19 | Tributary Transmit Clock 19 | 5 |
| TTC20 | Tributary Transmit Clock 20 | 7 |
| TTC21 | Tributary Transmit Clock 21 | 9 |
| TTC22 | Tributary Transmit Clock 22 | 11 |
| TTC23 | Tributary Transmit Clock 23 | 13 |
| TTC24 | Tributary Transmit Clock 24 | 15 |
| TTC25 | Tributary Transmit Clock 25 | 17 |
| TTC26 | Tributary Transmit Clock 26 | 19 |
| TTC27 | Tributary Transmit Clock 27 | 21 |
| TTC28 | Tributary Transmit Clock 28 | 23 |
| TTC29 | Tributary Transmit Clock 29 | 25 |
| TTC30 | Tributary Transmit Clock 30 | 27 |
| TTC31 | Tributary Transmit Clock 31 | 29 |
| TTC32 | Tributary Transmit Clock 32 | 31 |
| RTC17 | Tributary Receive Clock 17 | 2 |
| RTC18 | Tributary Receive Clock 18 | 4 |
| RTC19 | Tributary Receive Clock 19 | 6 |
| RTC20 | Tributary Receive Clock 20 | 8 |
| RTC21 | Tributary Receive Clock 21 | 10 |
| RTC22 | Tributary Receive Clock 22 | 12 |

**Table 14 Pin Description of the Access Connector JP6a** (cont'd)

| Signal | M13FX Pin Description | 'JP3' Pin Number |
|---|---|---|
| RTC23 | Tributary Receive Clock 23 | 14 |
| RTC24 | Tributary Receive Clock 24 | 16 |
| RTC25 | Tributary Receive Clock 25 | 18 |
| RTC26 | Tributary Receive Clock 26 | 20 |
| RTC27 | Tributary Receive Clock 27 | 22 |
| RTC28 | Tributary Receive Clock 28 | 24 |
| RTC29 | Tributary Receive Clock 29 | 26 |
| RTC30 | Tributary Receive Clock 30 | 28 |
| RTC31 | Tributary Receive Clock 31 | 30 |
| RTC32 | Tributary Receive Clock 32 | 32 |

**Table 15 Pin Description of the Access Connector JP7a**

| Signal | M13FX Pin Description | 'JP3' Pin Number |
|---|---|---|
| TTD17 | Tributary Transmit Data 17 | 1 |
| TTD18 | Tributary Transmit Data 18 | 3 |
| TTD19 | Tributary Transmit Data 19 | 5 |
| TTD20 | Tributary Transmit Data 20 | 7 |
| TTD21 | Tributary Transmit Data 21 | 9 |
| TTD22 | Tributary Transmit Data 22 | 11 |
| TTD23 | Tributary Transmit Data 23 | 13 |
| TTD24 | Tributary Transmit Data 24 | 15 |
| TTD25 | Tributary Transmit Data 25 | 17 |
| TTD26 | Tributary Transmit Data 26 | 19 |
| TTD27 | Tributary Transmit Data 27 | 21 |
| TTD28 | Tributary Transmit Data 28 | 23 |
| TTD29 | Tributary Transmit Data 29 | 25 |
| TTD30 | Tributary Transmit Data 30 | 27 |
| TTD31 | Tributary Transmit Data 31 | 29 |
| TTD32 | Tributary Transmit Data 32 | 31 |
| RTD17 | Tributary Receive Data 17 | 2 |

**Table 15 Pin Description of the Access Connector JP7a** (cont'd)

| Signal | M13FX Pin Description | 'JP3' Pin Number |
|---|---|---|
| RTD18 | Tributary Receive Data 18 | 4 |
| RTD19 | Tributary Receive Data 19 | 6 |
| RTD20 | Tributary Receive Data 20 | 8 |
| RTD21 | Tributary Receive Data 21 | 10 |
| RTD22 | Tributary Receive Data 22 | 12 |
| RTD23 | Tributary Receive Data 23 | 14 |
| RTD24 | Tributary Receive Data 24 | 16 |
| RTD25 | Tributary Receive Data 25 | 18 |
| RTD26 | Tributary Receive Data 26 | 20 |
| RTD27 | Tributary Receive Data 27 | 22 |
| RTD28 | Tributary Receive Data 28 | 24 |
| RTD29 | Tributary Receive Data 29 | 26 |
| RTD30 | Tributary Receive Data 30 | 28 |
| RTD31 | Tributary Receive Data 31 | 30 |
| RTD32 | Tributary Receive Data 32 | 32 |

The M13FX also provides an unchannelized T3 mode, where the M13 multiplexer is bypassed and the framed DS3 signal is provided on port 1. For this application port 1 is disconnected from the QuadLIU device and the port interface signals are available on the access connector JP22.

By default the connection to the T1/E1 LIU port 1 is closed with the following jumpers set on connector JP22: 1-2, 3-4, 5-6, 7-8. In order to connect an external T3 LIU these jumpers need to be removed and access to the framed DS3 signal is provided on JP22 connector.

**Table 16 Pin Description of the Access Connector JP22**

| Signal | M13FX Pin Description | 'JP22' Pin Number |
|---|---|---|
| TTC1 | DS3 Transmit Payload Clock (Output) | 1 |
| TTD1 | DS3 Transmit Payload Data (Input) | 3 |
| RTD1 | DS3 Receive Payload Data (Output) | 5 |
| RTC1 | DS3 Receive Payload Clock (Output) | 7 |

The M13FX device is controlled via the C161 microcontroller and connected via the microprocessor interface to the 8-bit address bus (A0-A7), to the 16-bit data bus and to

the control signals Chip Select, Read, Write, Byte High Enable, Reset and Interrupt. Pin IM is connected via a pull down resistance (1 kohm) to GND in order to select Intel Interface Mode. Pin ALE is connected via a pull up resistance (1 kohm) to V33 in order to select demultiplexed bus structure. Pin DBW is connected via a pull up resistance (1 kohm) to V33 in order to select 16-bit data bus mode.

In order to provide faster data transfer to and from the internal FIFOs for the C-bit parity Path Maintenance Data Link Channel, the M13FX supports additionally DMA handshake signals.

Request signals for transmit and receive direction (DRT and DRR) indicate free respectively available channel data. The RMC (Receive Message Complete) signal and the TXME (Transmit Message End) signal indicate the end of a message. These signals are unused on the EASY3445 hardware but can be accessed on connector JP21.

**Table 17 Pin Description of the Access Connector JP21**

| Signal | M13FX Pin Description | 'JP21' Pin Number |
|---|---|---|
| DRR | Data Request Receive | 1 |
| RMC | Receive Message Complete | 2 |
| DRT | Data Request Transmit | 3 |
| TXME | Transmit Message End | 4 |

It is also possible to have access to the DS3 overhead interface of the M13FX device. This interface allows to monitor or insert the DS3 overhead bits and DS3 stuffing bits of the DS3 frame. This interface is only used for evaluation purposes or if the application requires to modify the overhead or stuffing bits. Access to this interface is provided on connector JP2. In order to disable external overhead insertion, the TOVHDEN pin should be pulled down or directly connected to GND (default configuration on EASY3445).

**Table 18 Pin Description of the Access Connector JP2**

| Signal | M13FX Pin Description | 'JP2' Pin Number |
|---|---|---|
| TOVCHK | Transmit Overhead Bit Clock (Output) | 1 |
| TOVHD | Transmit Overhead Data (Input) | 2 |
| TOVHDEN | Transmit Overhead Data Enable (Input) | 3 |
| TOVHSYN | Transmit Overhead Synchronization (Input/ Output) | 4 |
| ROVCHK | Receive Overhead Bit Clock (Output) | 6 |
| ROVHD | Receive Overhead Data (Output) | 7 |
| ROVHSYN | Receive Overhead Synchronization (Output) | 8 |
| TSBCK | Transmit Stuff Bit Clock (Output) | 9 |

**Table 18 Pin Description of the Access Connector JP2**

| Signal | M13FX Pin Description | 'JP2' Pin Number |
|---|---|---|
| TSBD | Transmit Stuff Bit Data (Input) | 5 |
| RSBCK | Receive Stuff Bit Clock (Output) | 10 |
| RSBD | Receive Stuff Bit (Output) | 11 |
| DS3_CLK | Local DS3 clock - Transmit DS3 Clock Input | 12 |

In order to disable the external overhead insertion (default operation), the pin TOVHDEN should be tight to GND.

The M13FX also provides a boundary scan interface accessible on connector JP16.

**Table 19 Pin Description of the Access Connector JP16**

| Signal | M13FX Pin Description | 'JP16' Pin Number |
|---|---|---|
| TCK | Test Clock for JTAG Boundary Scan | 1 |
| TMS | Test Mode Select for JTAG Boundary Scan | 2 |
| TDI | Test Data Input for JTAG Boundary Scan | 3 |
| $\overline{\text{TRST}}$ | Test Reset Input for JTAG Boundary Scan | 4 |
| TDO | Test Data Output for JTAG Boundary Scan | 5 |
| SCANEN | Full Scan Path Test Enable | 6 |

## 2.4 The Quad Line Interface Unit for T1/E1

The Infineon QuadLIU device provides a very integrated four ports Line Interface Unit that can be configured per-port in T1 or E1 mode for long haul or short haul applications. The digital interface connects directly to the DS1/E1 serial interface of the M13FX. It performs clock and data recovery, as well as jitter attenuation of the gapped clock output of the M13FX, and signal encoding/decoding. The analog front end is common for E1 120 ohm and T1 100 ohm without resistor change.

The T1/E1 signal is available on 8 RJ45 connectors, that can be connected to standard T1/E1 equipment (e.g. TTC Fireberd).

**Table 20 Pin Description of the RJ45 connectors**

| Signal | QuadFALC Pin Description and Number | RJ45 Pin Number |
|---|---|---|
| RL2 | Receive Line 2 (Pin 4) | 1 |
| RL1 | Receive Line 1 (Pin 2) | 2 |
| XL2 | Transmit Line 2 (Pin 13) | 4 |
| XL1 | Transmit Line 1 (Pin 15) | 5 |

The Evaluation System EASY3445 uses the APC82112 transformer by APC. Other transformers (from Halo and Pulse) have been successfully tested with the QuadLIU device.

## 2.5 The T3 Line Interface Unit

The T3 line interface unit is provided by the TDK 78P7200 device, connected via a voltage converter to the M13FX. It performs data encoding and decoding as well as clock and data recovery. The analog front end is provided with the transformer PE65968 from Pulse Engineering, and a few resistors and capacitors. The T3 signal is available on two BNC connectors (for Transmit and Receive data) and can be connected to any standard T3 test equipment (e.g. TTC T-Berd).

The TDK LIU also provides a line built out option to accommodate the transmit pulse shape in transmit direction depending on the cable length. JP17 selects the Line Built Out mode.

**Table 21 DS3 Line Built Out Selection**

| Cable Length | Line Built Out (LBO) | ‘JP17’ Position |
|---|---|---|
| < 225 feet | High | ON |
| > 225 feet | Low | OFF |

## 2.6 Clock Generation

Three clock oscillators provide a reference clock respectively to the M13FX, to the QuadLIU and to C161 microcontroller:

- the 44.736MHz oscillator provides a reference clock for the transmit DS3 clock of the M13FX for external timing. The M13FX also supports looped timing where the transmit data is sampled with at the same frequency than the receive clock coming from the T3 LIU.
- the 16.384MHz oscillator provides a reference clock for both T1/E1 QuadLIU devices. This clock is used by the clocking unit of the QuadLIU to generate all frequencies required for the jitter attenuation, and clock and data recovery. The same frequency can be used for T1 and E1 (refer to the QuadLIU data sheet and to the “Master Reference Clock Calculator” program provided on the CD-ROM).
- the 16MHz oscillator provides the clock to the C161 microcontroller.

## 2.7 Schematic and Board Layout

The schematic and the board layout of the EASY3445 are available on the CD-ROM of the EASY3445 Evaluation System Design Kit.

## 2.8 Bill of Material

The following table shows the bill of material list of the EASY3445 hardware.

**Table 22 Bill of Material for the EASY3445**

| Item Number | Quantity | Part Type | Manufacturer | Part Designator |
|---|---|---|---|---|
| 1 | 1 | 22µF, 16V, SMTKC | Epcos | C1 |
| 2 | 45 | 100n, SM0805 | Epcos | C2-C14, C19-C23, C27- C53 |
| 3 | 2 | 10µF, 16V, SMTKC | Epcos | C15, C16 |
| 4 | 1 | 82pF, 5%, CAP5-R4x2.5 | Epcos | C17 |
| 5 | 1 | 1000pF, 5%, CAP5-R4x2.5 | Epcos | C18 |
| 6 | 1 | 10pF, 5%, CAP5-R4x2.5 | Epcos | C24 |
| 7 | 1 | 0.022µF, 5%, CAP5-R7x2 | Epcos | C25 |

**Table 22 Bill of Material for the EASY3445** (cont'd)

| Item Number | Quantity | Part Type | Manufacturer | Part Designator |
|---|---|---|---|---|
| 8 | 1 | 0.22µF, 5%, CAP5-R7x2 | Epcos | C26 |
| 9 | 5 | LED,GREEN, SOT23 | Infineon | D1, D2, D6, D41, D44 |
| 10 | 16 | BAV70, SOT23 | | D3, D4, D10, D12, D15, D16, D19, D20, D21, D22, D25, D26, D31, D32, D35, D36 |
| 11 | 3 | LED,RED, SOT23 | Infineon | D5, D40, D43 |
| 12 | 16 | BAW56, SOT23 | | D7, D8, D13, D14, D17, D18, D23, D24, D27, D28, D29, D30, D33, D34, D37, D38 |
| 13 | 2 | LL4148, SMSOD80 | | D9, D11 |
| 14 | 2 | LED,YELLOW, SOT23 | Infineon | D39, D42 |
| 15 | 16 | SIDACTOR, DO214AA | Teccor | F1-F16 |
| 16 | 1 | SAB-C161O-L16M, FP080G | Infineon | I1 |
| 17 | 1 | MAX707CSA, SO08S | MAXIM | I2 |
| 18 | 1 | AM29F040B-90JC, PC32R | AMD | I3 |
| 19 | 3 | 74FCT164245TPV, SO48IU | IDT | I4, I5, I14 |
| 20 | 1 | LT1086CT, TO220LK | Linear Technology | I6 |
| 21 | 1 | GAL22V10-7, PC28GAL | Lattice | I7 |
| 22 | 1 | PEB3445E V1.1, BGA272 | Infineon | I8 |
| 23 | 2 | SRAM512Kx8-70, SO32SRAM | | I9, I10 |

**Table 22 Bill of Material for the EASY3445** (cont'd)

| Item Number | Quantity | Part Type | Manufacturer | Part Designator |
|---|---|---|---|---|
| 24 | 1 | LT1085CT, TO220S | Linear Technology | I11 |
| 25 | 1 | SP232ACT, SO16U | Sipex, Maxim | I12 |
| 26 | 1 | TDK-78P7200-IH, PLCC28 | TDK | I13 |
| 27 | 2 | PEB22504 V1.1, FP100E | Infineon | I15, I16 |
| 28 | 1 | 74HC04, SO14S | | I17 |
| 29 | 2 | JMP2X10, JMP2X10 | Y-S Electronic | JP1, JP4 |
| 30 | 1 | JMP2X7, JMP2X7 | Y-S Electronic | JP2 |
| 31 | 4 | JMP2X8, JMP2X8 | Y-S Electronic | JP3, JP3a, JP6, JP7 |
| 32 | 2 | JMP2x16, JMP2X16 | Y-S Electronic | JP6a, JP7a |
| 33 | 3 | JMP1X2, JMP1X2 | Y-S Electronic | JP10, JP15, JP17 |
| 34 | 1 | JMP1X7, JMP1X7 | Y-S Electronic | JP16 |
| 35 | 1 | JMP3x7, JMP3X7 | Y-S Electronic | JP18 |
| 36 | 2 | JMP1X5, JMP1X5 | Y-S Electronic | JP19, JP20 |
| 37 | 1 | JMP1X4, JMP1X4 | Y-S Electronic | JP21 |
| 38 | 1 | JMP2x4, JMP2X4 | Y-S Electronic | JP22 |
| 39 | 1 | JMP1X3, JMP1X3 | Y-S Electronic | JP31 |
| 40 | 2 | PE65968, SMTR05 | Pulse | L1, L2 |
| 41 | 1 | 6.8µH 10%, SM-IND02 | Epcos | L3 |
| 42 | 3 | 4µ7, SM1210 | Epcos | L4, L7, L8 |
| 43 | 2 | APC82112, SMTR03 | APC | L5, L6 |
| 44 | 1 | 0.47 µH 5%, SM1210 | Epcos | L9 |
| 45 | 2 | V25, JMP1X1 | Y-S Electronic | MP1, MP2 |
| 46 | 8 | GND, JMP1X1 | Y-S Electronic | MP3-MP10 |
| 47 | 1 | 16.000MHz, SMTOSC | Geyer electronics | O1 |

**Table 22 Bill of Material for the EASY3445** (cont'd)

| Item Number | Quantity | Part Type | Manufacturer | Part Designator |
|---|---|---|---|---|
| 48 | 1 | 16.384MHz, 3.3V, SMTOSC1 | Vectron | O2 |
| 49 | 1 | 44.736MHz 3,3V | Ecliptek | O3 |
| 50 | 10 | 100R, SM0805 | Epcos | R1, R2, R51, R52, R59, R66, R73, R80, R163, R174 |
| 51 | 1 | 120R, SM0805 | Epcos | R3 |
| 52 | 1 | 1k5, SM0805 | Epcos | R4 |
| 53 | 1 | 180R, SM0805 | Epcos | R5 |
| 54 | 1 | 68R, SM0805 | Epcos | R6 |
| 55 | 1 | 220R, SM0805 | Epcos | R7 |
| 56 | 3 | 1k, SM0805 | Epcos | R8, R11, R12 |
| 57 | 2 | 470R, SM0805 | Epcos | R9, R37 |
| 58 | 2 | 27R, SM0805 | Epcos | R10, R133 |
| 59 | 3 | 100k, SM0805 | Epcos | R13, R17, R38 |
| 60 | 4 | 10k, SM0805 | Epcos | R14, R16, R114, R117 |
| 61 | 2 | 330R, SM0805 | Epcos | R15, R33 |
| 62 | 18 | 3k3, SM0805 | Epcos | R18, R19, R34, R42, R46, R112, R113, R115, R116, R118-R126 |
| 63 | 2 | 75R, SM0805 | Epcos | R20, R21 |
| 64 | 1 | 12R, SM0805 | Epcos | R22 |
| 65 | 2 | 820R, SM0805 | Epcos | R23, R30 |
| 66 | 6 | 8k2, SM0805 | Epcos | R24-R29 |
| 67 | 2 | 5k6, SM0805 | Epcos | R31, R35 |
| 68 | 1 | 82k, SM0805 | Epcos | R32 |
| 69 | 1 | 6k8, SM0805 | Epcos | R36 |

**Table 22 Bill of Material for the EASY3445** (cont'd)

| Item Number | Quantity | Part Type | Manufacturer | Part Designator |
|---|---|---|---|---|
| 70 | 12 | 33R, SM0805 | Epcos | R39-R41, R43-R45, R109-R111, R158-R160 |
| 71 | 16 | 2R2, SM0603 | Epcos | R47, R48, R53, R56, R60, R63, R67, R70, R74, R77, R81, R84, R167, R170, R175, R178 |
| 72 | 16 | 10R, SM0603 | Epcos | R49, R50, R57, R58, R64, R65, R71, R72, R78, R79, R164, R171-R173, R179, R180 |
| 73 | 16 | 22R, SM0603 | Epcos | R54, R55, R61, R62, R68, R69, R75, R76, R82, R83, R165, R166, R168, R169, R176, R177 |
| 74 | 48 | 10k, SM0603 | Epcos | R85-R108, R134-R157 |
| 75 | 6 | 680R, SM0805 | Epcos | R127-R132 |
| 76 | 4 | 0R, SM0805 | Epcos | R161, R162, R181, R182 |
| 77 | 1 | MINBUTT1, PBUTKLBL | | RESET |
| 78 | 2 | BC847, SOT23 | | T1, T2 |
| 79 | 1 | CONPSA-3, PWRCON2 | Lumberg | X1 |
| 80 | 2 | BNC-75R, BNC-BU1 | Radiall | X2, X5 |
| 81 | 2 | V.24 Con, DB9F | Y-S Electronic | X3, X4 |
| 82 | 2 | WEST8P*4, RJ45-ARRAY | Framatome | X6, X7 |

**Table 22 Bill of Material for the EASY3445** (cont'd)

| Item Number | Quantity | Part Type | Manufacturer | Part Designator |
|---|---|---|---|---|
| 83 | 1 | SK409 25mm 8,2K/W, SK409 | Fischer | Y1 |
| 84 | 1 | SK13 17K/W, SK13 | Fischer | Y2 |
| 85 | 1 | Power Supply 5V, 4A | Phihong | |
| 86 | 1 | Power cable | Volex | |
| 87 | 1 | Power cable US | Volex | |
| 88 | 1 | Adapter DB9<->DB25 | EkrisCable | |
| 89 | 1 | RS232 cable | EkrisCable | |

# 3 Target System Software Description

The EASY3445 software is designed in order to support legacy device driver modules (DDM) compliant to the DDS specification and new device drivers compliant to the OPI concept.

The EASY3445 software provides a complete environment in order to drive the PEB3445E (M13FX) and PEB22504 (QuadLIU) Infineon ICs used in the target board. No software is required to drive the DS3 Line Interface Unit IC which runs in hardware mode.

The M13FX device driver is designed according to the OPI concepts which allows a high portability through any RTOS and HW platform. The portability is achieved by exporting in a adaptation module the dependence to the platform and the RTOS. The M13FX device driver has only to be recompiled with the target development environment tools.

The EASY3445 software provides the M13FX Device Driver Environment (M13FX DDE) which integrates the M13FX device driver into the DDS system.

The Quadliu device driver is designed as a Device Driver Module (DDM) according to the DDS environment. The device driver is DDS dependent.

The EASY3445 software provides the QuadLIU Device Driver Environment (QuadLIU DDE) which integrates the Quadliu device driver into the DDS system.

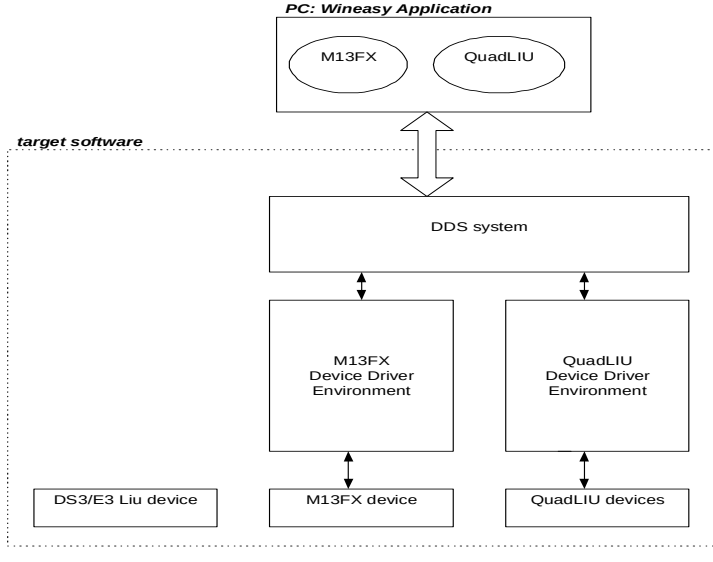



**Figure 3 M13FX environment architecture**

## 3.1 Architecture

The EASY3445 software is composed by the M13FX and QuadLIU Device Driver Environments running on the DDS system. Both environments are independent.

The M13FX DDE contains the following modules:

- **SSAL for DDS**, which makes the adaptation of the OPI environment to the EASY3445 target (DDS and Platform). It provides to the system an initialization entry-point to call at the system startup time. This function initializes the M13FX device driver environment which activates the M13FX device.
- **APPLI for M13FX**, which makes the interface adaptation between the Wineasy C/I messages and the M13FX OPI low-level messages. It handles the M13FX device driver through the OPI low-level interface.
- **M13FX Device Driver**, which drives the M13FX device. It supports all the device functionality and presents an interface application-oriented that makes the device driver easy and ready to use for the application. This module is totally RTOS and platform independent.

The QuadLIU DDE contains the following modules:

- **DDSBoard**, which initializes the QuadLIU environment. It provides to the system an initialization entry-point to call at the system startup time. This function initializes the QuadLIU device driver which activates both QuadLIU ICs.
- **QuadLIU Device Driver**, which drives the QuadLIU devices. It provides an interface conforming to the DDS specification.

The following figure shows the software architecture including the modules in both Device Driver Environments.

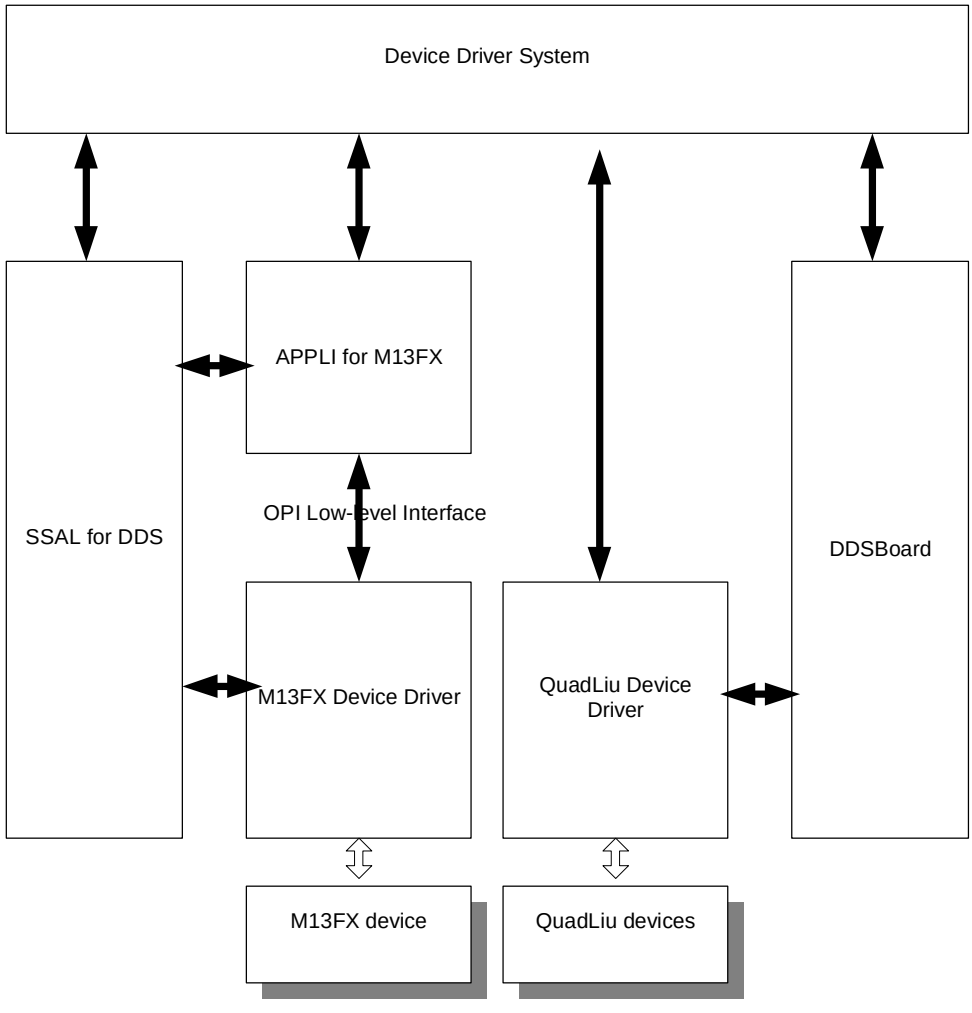



**Figure 4 M13FX target software architecture**

## 3.2 M13FX Device Driver Environment

### 3.2.1 Initialization

The M13FX DDE is initialized through the “SSAL for DDS” module. The SSAL module provides to the target system an initialization entry-point which must be issued at the system startup.

The entry-point is SSAL_init() which requires no parameters and returns an execution code. The file “opiext.h” must be included in order to access to the function and the return code values.

The entry-point is called in the DDSI module at the DDS system startup.

The execution return code must be OPI_SUCESS which means that the M13FX DDE is correctly initialized an operational.

The code included in the DDSI module is :

```
#include "opiext.h"
if (SSAL_Init() != OPI_SUCCESS)
    printf ("OPI: SSAL module initialization FAILED\n\r");
```

### 3.2.2 SSAL for DDS module

The SSAL for DDS module makes the adaptation of the OPI environment to the target system which is DDS. This module provides the services necessary to the modules running under the OPI environment, making the DDS target system transparent to them.

The SSAL performs the following actions:

- OPI environment initialization which makes the environment ready to work (chip select, device driver,...),
- Interrupt management,
- Memory management,
- Control Bloc management,
- Link List management.

The SSAL provides to the OPI users (application, device driver) the following services:

- Memory,
- Control Bloc,
- Link List.

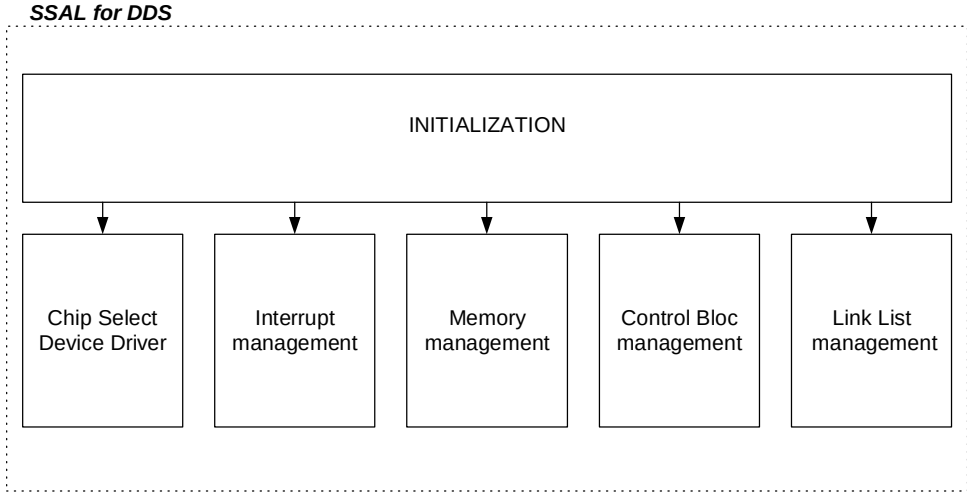



**Figure 5 SSAL for DDS Module Structure**

### 3.2.2.1 Initialization

The OPI environment is initialized by the SSAL_init function. This function performs the following actions:

- Control Bloc initialization which will be used for the memory and control bloc management,
- M13FX device Chip Select setup according to the EASY3445 hardware specifications,
- M13FX device driver initialization, passing the device base address,
- Installation of the M13FX interrupt handler.

### 3.2.2.2 Interrupt management

The detection of the hardware interrupt is performed in the SSAL module which calls the M13FX interrupt handler passing the device number in the function. The SSAL module performs the prologue and the epilogue of the interrupt handling. These two phases are system dependent. In the EASY3445 target these two phases are not required. The SSAL uses the target firmware functions in order to install and enable the M13FX interrupt handler.

```
DdshSetHWIntVect    (M13FX_DEV0_INT,&SSAL_M13fx_Interrupt_dev0);
```

```
DdshIntCtrl(M13FX_DEV0_INT,TRUE);


void SSAL_M13fx_Interrupt_dev0(void)
{
 M13FX_Interrupt_Handler ((WORD32)M13FX_DEV0);
}
```

### 3.2.2.3 Control Bloc Management

The OPI environment uses Control Blocs in order to avoid the memory copy between the OPI environment and the DDS system. These controls blocs carry the DDS data buffers used inside the OPI environment and can be linked in private link lists using the link list service.

The Control Bloc management initializes a memory pool with the Control Blocs and uses the Link List service in order to manage the Control Blocs. The Control Blocs are put in a private link list.

The Control Blocs objects are accessed through a service composed by the following functions:

- **GetHandle** which provides a Control Bloc. The Control Bloc is extract from the link list, marked as busy and the address CB is returned to the caller. A NIL address is returned when there is no available CB.
- **FreeHandle** which returns a Control Bloc to the system. The Control Bloc is inserted in the link list and marked as free.

The internal data structures used for the CB management are:

- **CB_table,** Control Bloc memory pool. It is a table containing MAX_CB_NUMBER Control Blocs.
- **LinkLst_CB**, internal Link List where the control blocs are queued in order to be accessed through the CB service.

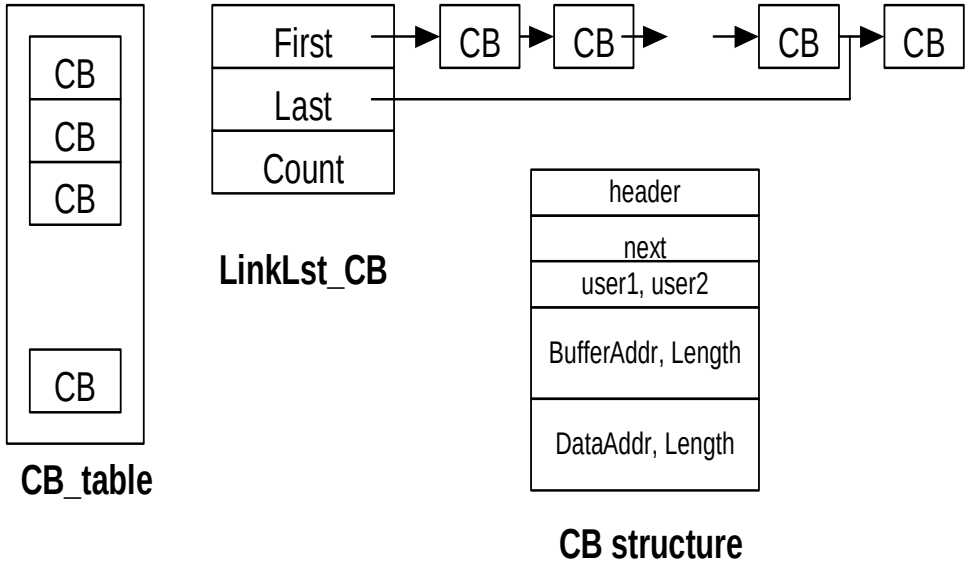



**Figure 6 CB data structure**

### 3.2.2.4 Memory management

Device drivers compliant to the OPI specifications use the OPI memory management interface. Buffers are manipulated through handles.

Therefore receiving or transmitting data buffers through the OPI interface are done via handles and not via pointers. The handle is a Control Bloc which contains all the information related to the data buffer.

The data buffers are accessed through the memory service composed by the following functions:

- **BufferAlloc**, allocates a data buffer with the specified data size. The function returns a handle which permits to access to the data buffer. The handle is a CB requested to the CB management, the data buffer is requested with the DDS buffer allocation function "`DdsmMsgBufAlloc`".
- **BufferFree**, releases a previous data buffer allocated with BufferAlloc. The function frees the data buffer referenced by the handle with the DDS buffer release function "DdsmMsgBufFree" and returns the CB supporting the handle to the CB management.

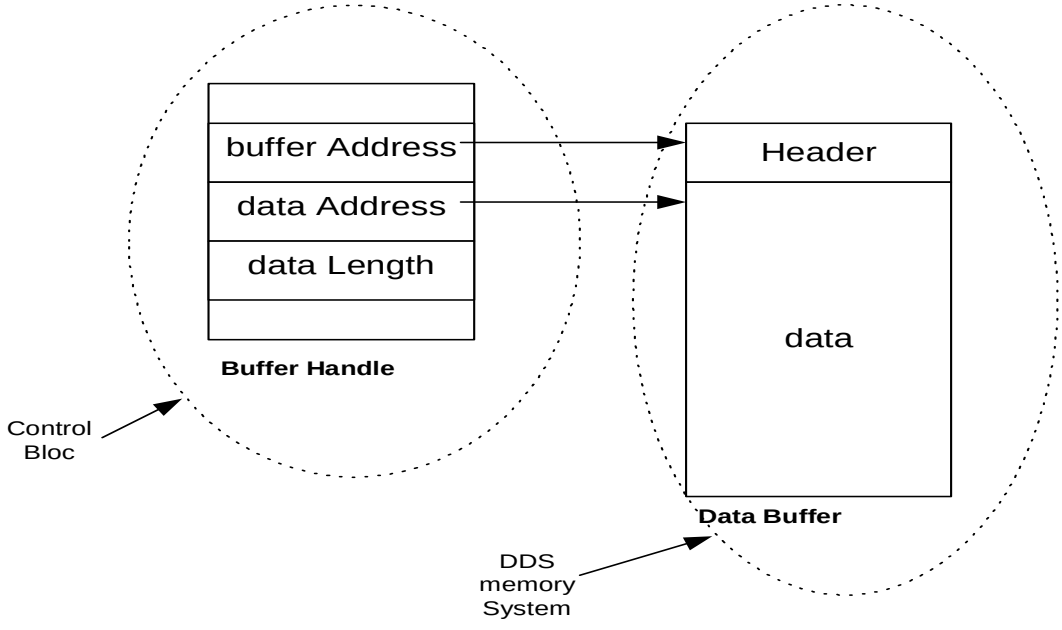



**Figure 7 SSAL memory management**

### 3.2.2.5 Link List management

The link list management is used in order to provide a queue list facility to the applications. The objects linked are CB objects. The link list management does not require any memory resource. The resources are provided by the user of the link list service which provides the link list address.

The link list facility is accessed through the following functions:

- **LL_init**, create a link list by initializing the link list field,
- **LL_reset**, reset a link list. The function releases the remaining control blocs in the link list,
- **LL_put**, insert at the end of the Link List the Control Bloc,
- **LL_nblock**, read out the current number of CB in the link list,
- **LL_get**, extract the CB in the first position in the link list,
- **LL_read**, read out the address of the first CB in the link list,
- **LL_nread**, read out the address of the next CB in the link list.

### 3.2.3 APPLI for M13FX module

The APPLI for M13FX module is the user application for the M13FX device driver. It makes the message conversion between the Wineasy M13FX C/I messages and the OPI M13FX low-level messages.

APPLI for M13FX module supports both environments DDS and OPI:

- **DDS**, the module provides the initialization and the messages processing entry-points. It is declared to the DDS system as APPLI module for the messages processing (scheduled by the DDS). It sends to the USER module the Wineasy indication messages and receives from the USER module the Wineasy command messages.
- **OPI**, the module uses the OPI mechanisms in order to communicate with the M13FX device driver. The module binds to the device driver for gaining and establishing the interface access to the device driver (callbacks exchange, user identification allocation) and grants the device resource by opening the corresponding port. The module uses the SSAL services for the memory conversion.

APPLI for M13FX makes the adaptation of the following M13FX groups commands:

- Device Activation/Deactivation,
- Device Configuration,
- Device Alarms,
- Device Loopbacks,
- Device Feac Channel,
- Device MDL Channel,
- Device Error Counters,
- Device Mapper,
- Device Test Unit.

In order to avoid the byte ordering problem when messages cross different environments, the M13FX message interface in the Wineasy application does not use the M13FX message structures defined in the device driver interface. The APPLI for M13FX module makes this conversion in both directions:

- **PC ->target**, the M13FX structure is updated with the relevant data fields of the wineasy command message.
- **target -> PC**, the data fields of the Wineasy indication message are filled with the M13FX data structure relevant fields.

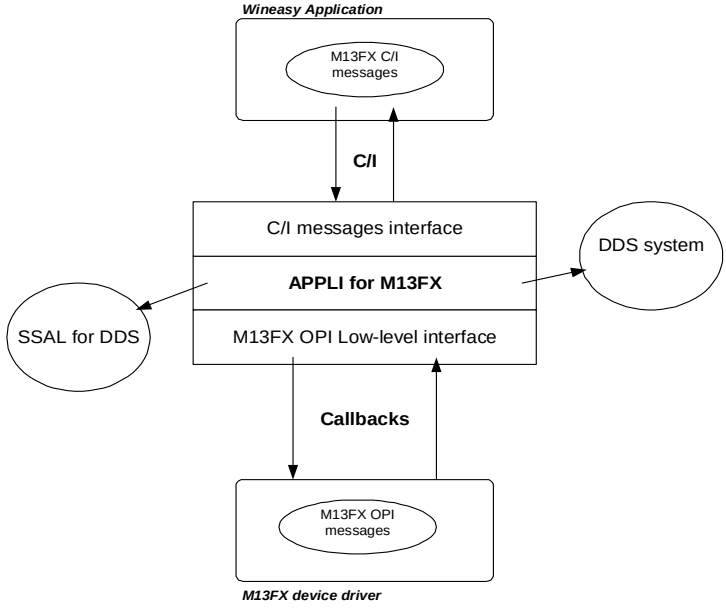



**Figure 8 APPLI for M13FX architecture**

### 3.2.3.1 Initialization

The module is initialized by calling the “vApplInit()” function which performs the following actions:

- Sets the message entry-point to the DDS system for the message processing. The module is declared as APPL_MODULE for the data path.
- Fill-out the OPI bind structure with its application callbacks that the M13FX device driver has to call in order to communicate with the application. its passes it in the OPI bind function.
- The device driver returns the bind structure updated with its callback functions that the application has to use in order to communicate with the device driver and the unique User Identification which must be used in further interactions.
- Activates the M13FX device by opening the port.

The APPLI module initialization is performed once the OPI environment is initialized and ready to be used.

### 3.2.3.2 Message processing

The messages coming from the DDS USER module are processed by the "vApplMsgEntry" function which is called by the DDS scheduler when a message's destination is the APPLI for the M13FX module. The message is decoded, encoded into the OPI format and sent to the M13FX device driver.

The messages coming from the M13FX device driver are processed by the application callbacks. The messages are converted into the C/I format and sent to Wineasy application through the USER module.

For FEAC and MDL channel data transfer, data are not copied from one to other environment. the DDS messages are transferred to the device driver by the means of handles that reference the data area of a DDS message. The device driver manipulates only handles which contains the data area informations.

when the Application module receives from the WinEASY a data buffer to send in the FEAC or MDL channel, the module allocates an handle with the function "Get_Handle", updates the handle fields with the data buffer information and passes it to the device driver.

when the Application receives a data buffer referenced by an handle from the device driver, it releases the handle with the function "Free_handle" and send the data buffer to the Wineasy application.

### 3.2.3.3 Application callbacks

The module provides to the M13FX device driver 3 callbacks according to the OPI concept which are:

- **APPL_StatusInd**, this function notifies to the application any event detected by the M13FX device. The events detected are alarms (DS3 and DS2 level), loopback codes detection (DS2 and DS1 level), Error counters (One second interrupt). An indication is sent to the wineasy application.
- **APPL_TxBufConfirm**, this function notifies to the application that a previous data buffer has been sent into the FEAC or MDL channel. The transmitted data buffer is sent to the Wineasy application.
- **APPL_RxBufInd**, this function notifies that a data buffer is received from the FEAC or MDL channel. The received data buffer is sent to the Wineasy application.

### 3.2.3.4 C/I - M13FX OPI low-level messages adaptation

The APPLI for DDS module applies the same principle for C/I - M13FX OPI Low-level messages adaptation. The following parameters correspondence is performed between both formats:

- The Wineasy "**Entity**" field corresponds to the M13FX OPI "**port**",
- The Wineasy "**ID**" field corresponds to the M13FX OPI function to call,

- The Wineasy “**Dptr**” field contains the command parameters to map to the structure parameters of the M13FX OPI function.

The following table lists the wineasy messages supported and the correspondence with the M13FX OPI level interface.

**Table 23 M13FX C/I / OPI commands correspondence**

| Wineasy command | OPI function call | Device function call |
|---|---|---|
| ***Device activation*** | | |
| DEV_STATUS | DRV_M13FXPortCfg | M13FX_Device_Status |
| DEV_ACTIVATION | DRV_M13FXPortOpen | M13FX_Device_Activate |
| DEV_DEACTIVATION | DRV_M13FXPortClose | M13FX_Device_Deactivate |
| ***Device configuration*** | | |
| CFG_PARAMETERS_DS3 | DRV_M13FXPortCfgReq | M13FX_DS3_Cfg_Parameters |
| CFG_PARAMETERS_DS2 | DRV_M13FXPortCfgReq | M13FX_DS2_Cfg_Parameters |
| CFG_PARAMETERS_DS1 | DRV_M13FXPortCfgReq | M13FX_DS1_Cfg_Parameters |
| CFG_PARAMETERS_FEAC | DRV_M13FXPortCfgReq | M13FX_Feac_Cfg_Parameters |
| CFG_PARAMETERS_MDL | DRV_M13FXPortCfgReq | M13FX_Mdl_Cfg_Parameters |
| CFG_PARAMETERS_MISC | DRV_M13FXPortCfgReq | M13FX_Misc_Cfg_Parameters |
| CFG_PARAMETERS_DS3_GET | DRV_M13FXPortCfgReq | M13FX_DS3_Cfg_Get_Parameters |
| CFG_PARAMETERS_DS2_GET | DRV_M13FXPortCfgReq | M13FX_DS2_Cfg_Get_Parameters |
| CFG_PARAMETERS_DS1_GET | DRV_M13FXPortCfgReq | M13FX_DS1_Cfg_Get_Parameters |
| CFG_PARAMETERS_FEAC_GET | DRV_M13FXPortCfgReq | M13FX_Feac_Cfg_Get_Parameters |
| CFG_PARAMETERS_MDL_GET | DRV_M13FXPortCfgReq | M13FX_Mdl_Cfg_Get_Parameters |
| CFG_PARAMETERS_MISC_GET | DRV_M13FXPortCfgReq | M13FX_Misc_Cfg_Get_Parameters |
| CFG_PARAMETERS_SET | DRV_M13FXPortCfgReq | M13FX_Set_Cfg_Parameters |
| CFG_PARAMETERS_RESET | DRV_M13FXPortCfgReq | M13FX_Reset_Cfg_Parameters |
| ***Device DS3 alarms*** | | |
| ALARM_DS3_AIS_SET | DRV_M13FXPortCfgReq | M13FX_DS3_Alarm |
| ALARM_DS3_AIS_RESET | DRV_M13FXPortCfgReq | M13FX_DS3_Alarm |
| ALARM_DS3_RA_SET | DRV_M13FXPortCfgReq | M13FX_DS3_Alarm |
| ALARM_DS3_RA_RESET | DRV_M13FXPortCfgReq | M13FX_DS3_Alarm |
| ALARM_DS3_IDLE_SET | DRV_M13FXPortCfgReq | M13FX_DS3_Alarm |

**Table 23 M13FX C/I / OPI commands correspondence**

| Wineasy command | OPI function call | Device function call |
|---|---|---|
| ALARM_DS3_IDLE_RESET | DRV_M13FXPortCfgReq | M13FX_DS3_Alarm |
| ALARM_DS3_STATUS | DRV_M13FXPortCfgReq | M13FX_DS3_Alarm |
| ***Device DS2 alarms*** | | |
| ALARM_DS2_AIS_SET | DRV_M13FXPortCfgReq | M13FX_DS2_Alarm |
| ALARM_DS2_AIS_RESET | DRV_M13FXPortCfgReq | M13FX_DS2_Alarm |
| ALARM_DS2_RA_SET | DRV_M13FXPortCfgReq | M13FX_DS2_Alarm |
| ALARM_DS2_RA_RESET | DRV_M13FXPortCfgReq | M13FX_DS2_Alarm |
| ALARM_DS2_STATUS | DRV_M13FXPortCfgReq | M13FX_DS2_Alarm |
| ***Device DS1 alarms*** | | |
| ALARM_DS1_T1AIS_SET | DRV_M13FXPortCfgReq | M13FX_DS1_Alarm |
| ALARM_DS1_T1AIS_RESET | DRV_M13FXPortCfgReq | M13FX_DS1_Alarm |
| ALARM_DS1_T13AIS_SET | DRV_M13FXPortCfgReq | M13FX_DS1_Alarm |
| ALARM_DS1_T13AIS_RESET | DRV_M13FXPortCfgReq | M13FX_DS1_Alarm |
| ALARM_DS1_STATUS | DRV_M13FXPortCfgReq | M13FX_DS1_Alarm |
| ***Device DS3 loopbacks*** | | |
| LOOPBACK_DS3_REMOTE_SET | DRV_M13FXPortCfgReq | M13FX_DS3_Loopback |
| LOOPBACK_DS3_REMOTE_RESET | DRV_M13FXPortCfgReq | M13FX_DS3_Loopback |
| LOOPBACK_DS3_LOCAL_SET | DRV_M13FXPortCfgReq | M13FX_DS3_Loopback |
| LOOPBACK_DS3_LOCAL_RESET | DRV_M13FXPortCfgReq | M13FX_DS3_Loopback |
| LOOPBACK_DS3_STATUS | DRV_M13FXPortCfgReq | M13FX_DS3_Loopback |
| ***Device DS2 loopbacks*** | | |
| LOOPBACK_DS2_REMOTE_SET | DRV_M13FXPortCfgReq | M13FX_DS2_Loopback |
| LOOPBACK_DS2_REMOTE_RESET | DRV_M13FXPortCfgReq | M13FX_DS2_Loopback |
| LOOPBACK_DS2_LPCREQ_SET | DRV_M13FXPortCfgReq | M13FX_DS2_Loopback |
| LOOPBACK_DS2_LPCREQ_RESET | DRV_M13FXPortCfgReq | M13FX_DS2_Loopback |
| LOOPBACK_DS2_STATUS | DRV_M13FXPortCfgReq | M13FX_DS2_Loopback |
| ***Device DS1 loopbacks*** | | |
| LOOPBACK_DS1_REMOTE_SET | DRV_M13FXPortCfgReq | M13FX_DS1_Loopback |
| LOOPBACK_DS1_REMOTE_RESET | DRV_M13FXPortCfgReq | M13FX_DS1_Loopback |
| LOOPBACK_DS1_LOCAL_SET | DRV_M13FXPortCfgReq | M13FX_DS1_Loopback |

**Table 23 M13FX C/I / OPI commands correspondence**

| Wineasy command | OPI function call | Device function call |
|---|---|---|
| LOOPBACK_DS1_LOCAL_RESET | DRV_M13FXPortCfgReq | M13FX_DS1_Loopback |
| LOOPBACK_DS1_LPCREQ_SET | DRV_M13FXPortCfgReq | M13FX_DS1_Loopback |
| LOOPBACK_DS1_LPCREQ_RESET | DRV_M13FXPortCfgReq | M13FX_DS1_Loopback |
| LOOPBACK_DS1_STATUS | DRV_M13FXPortCfgReq | M13FX_DS1_Loopback |
| ***Device FEAC channel*** | | |
| FEAC_STATUS | DRV_M13FXPortCfgReq | M13FX_Feac_Status |
| FEAC_ACTIVATION | DRV_M13FXChOpen | M13FX_Feac_Open |
| FEAC_DEACTIVATION | DRV_M13FXChClose | M13FX_Feac_Close |
| FEAC_SEND | DRV_M13FXChTxReq | M13FX_Feac_Send |
| FEAC_RSEND | DRV_M13FXPortCfgReq | M13FX_Feac_RSend |
| ***Device MDL channel*** | | |
| MDL_STATUS | DRV_M13FXPortCfgReq | M13FX_Mdl_Status |
| MDL_ACTIVATION | DRV_M13FXChOpen | M13FX_Mdl_Open |
| MDL_DEACTIVATION | DRV_M13FXChClose | M13FX_Mdl_Close |
| MDL_SEND | DRV_M13FXChTxReq | M13FX_Mdl_Send |
| ***Device Tributary Mapper*** | | |
| MAP_CFG_GET | DRV_M13FXPortCfgReq | M13FX_Map_Get_Configuration |
| MAP_CFG_SET | DRV_M13FXPortCfgReq | M13FX_Map_Set_Configuration |
| MAP_CFG_RESET | DRV_M13FXPortCfgReq | M13FX_Map_Reset_Configuration |
| ***Device Test Unit*** | | |
| TU_GENERATOR_SET | DRV_M13FXPortCfgReq | M13FX_TU_Set_Generation |
| TU_GENERATOR_START | DRV_M13FXPortCfgReq | M13FX_TU_Start_Generation |
| TU_GENERATOR_STOP | DRV_M13FXPortCfgReq | M13FX_TU_Stop_Generation |
| TU_GENERATOR_TEST_POINT | DRV_M13FXPortCfgReq | M13FX_TU_Test_Point |
| TU_GENERATOR_ERROR_INSERTION | DRV_M13FXPortCfgReq | M13FX_TU_Error_Insertion |
| TU_GENERATOR_RECEIVE_PATTERN | DRV_M13FXPortCfgReq | M13FX_TU_Receive_Pattern |
| TU_GENERATOR_RECEIVE_COUNTER | DRV_M13FXPortCfgReq | M13FX_TU_Receive_Counter |
| TU_FRAMER_SET | DRV_M13FXPortCfgReq | M13FX_TU_Set_Framer |
| TU_FRAMER_START | DRV_M13FXPortCfgReq | M13FX_TU_Start_Framer |
| TU_FRAMER_STOP | DRV_M13FXPortCfgReq | M13FX_TU_Stop_Framer |

**Table 23 M13FX C/I / OPI commands correspondence**

| Wineasy command | OPI function call | Device function call |
|---|---|---|
| TU_FRAMER_ERROR_INSERTION | DRV_M13FXPortCfgReq | M13FX_TU_Error_Framer |
| TU_FRAMER_RECEIVE_COUNTER | DRV_M13FXPortCfgReq | M13FX_TU_Receive_counter_Framer |

### 3.2.4 M13FX Device Driver module

The M13FX Device Driver Module is the device driver software for the M13FX device. It is designed in order to be flexible in term of interfaces and configurations offered to the target applications.

The device driver module is RTOS and platform independent speeding up the porting to any customer platform.

The device driver supports all the M13FX device functionality and presents them to the applications through an oriented-application interface: M13FX OPI Low-Level Interface. This makes the device driver easy and ready to use by the target applications.

The device driver complies with the OPI concepts which are:

- Multi-applications support, different applications can be bound dynamically to the device driver,
- Multi-devices support, the applications can access to any m13fx device declared at the device driver initialization,
- Application-oriented, the device is hidden to the applications which only manipulates port and channel for accessing to the device resources,
- RTOS and platform independent, the device driver module needs only to be recompiled with an ANSI C compiler.
- Generic interface to the application with callbacks principle.

The device driver module is composed with 2 parts:

- **M13FX OPI level**, which is at the top level of the module and provides the OPI Low-level interface to the applications. It accesses to the device resources through the M13FX low-level interface provided by the M13FX device level.
- **M13FX device level**, which is at the bottom of the device driver module and provides the access to the device resources through the M13FX Low-level interface.

The following figures shows the M13FX device driver architecture.

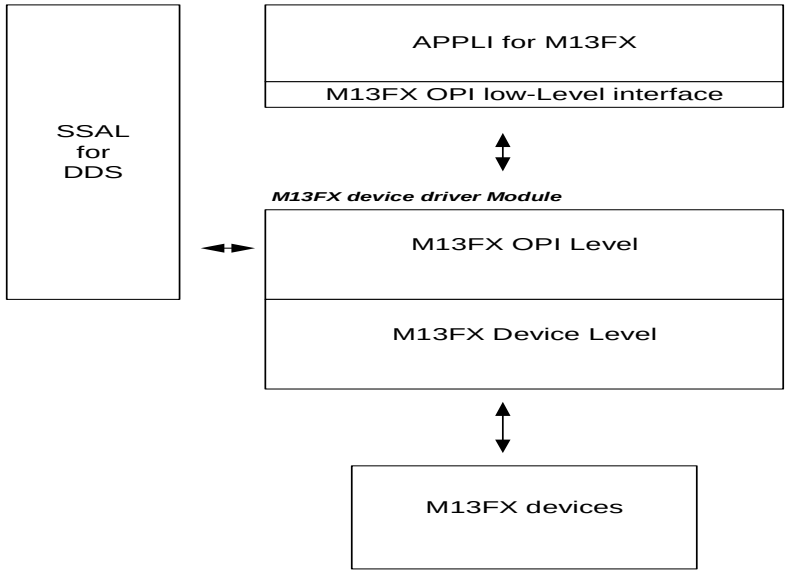



**Figure 9 M13FX device driver structure**

The M13FX device driver module is modular and can be used in different configurations which are:

- OPI environment, the target applications communicate through the OPI Low-Level interface,
- without OPI environment, the target applications communicate with the M13FX low-level interface. The M13FX OPI level module is not required and then can be removed.

The following figures shows the possible interface configurations.

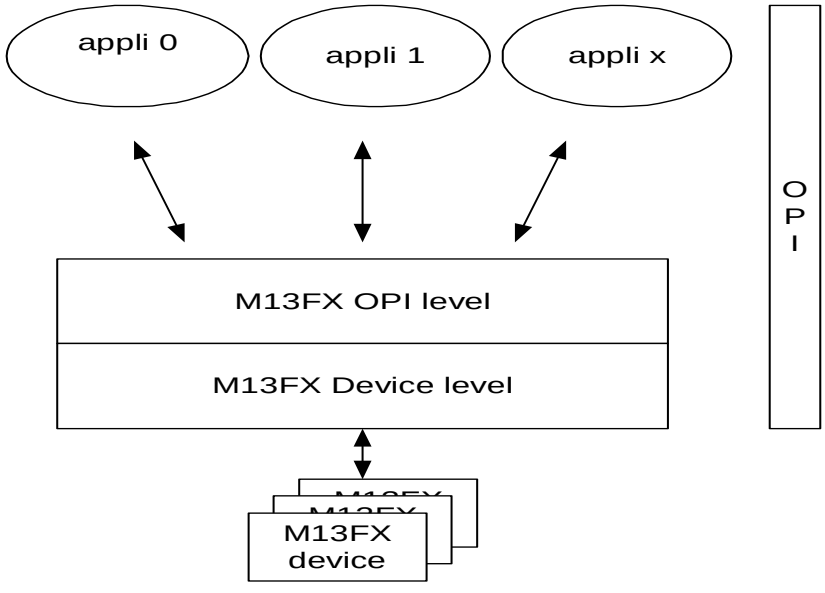



**Figure 10 M13FX device driver configuration - OPI level**

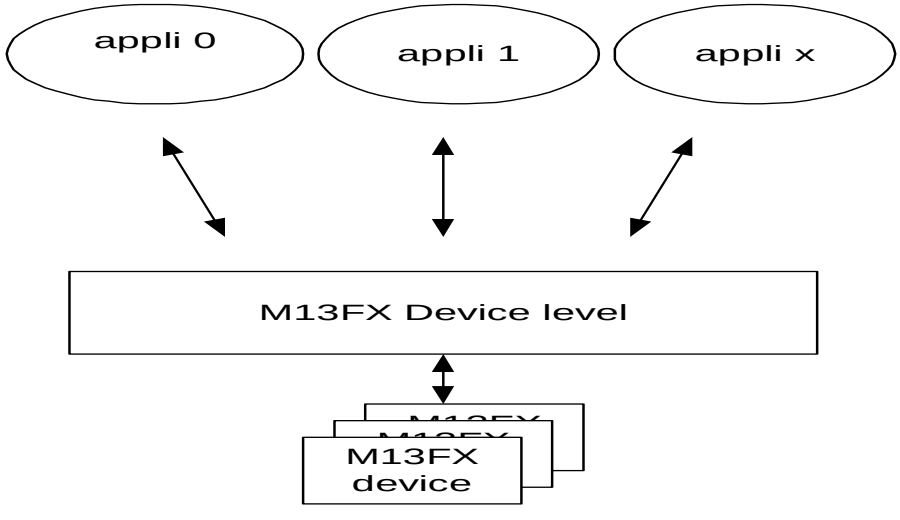



**Figure 11 M13FX device driver configuration - device level**

### 3.2.4.1 Module Initialization

The SSAL for DDS module calls the initialization function “DRV_M13FXInit” which initializes the M13FX device driver module at the OPI level. A device parameters structure is passed in the function defining the number of devices to support and assigning to each device the base address.

The device driver initializes the devices with the default configuration parameters. The devices are then ready to be accessed by the applications.

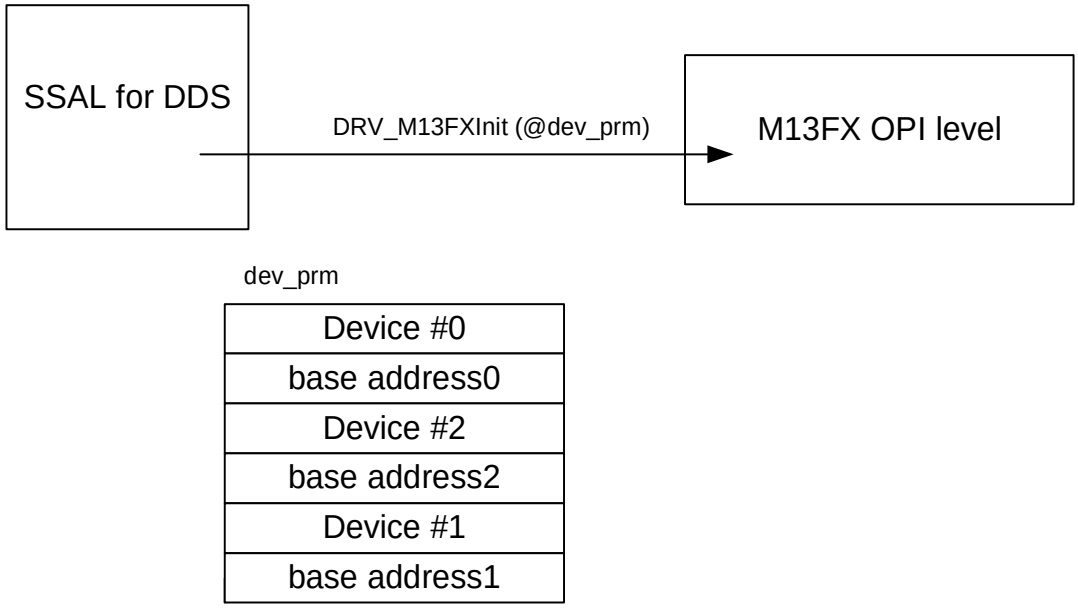



**Figure 12 M13FX device driver initialization**

### 3.2.4.2 M13FX OPI level

This module provides the M13FX OPI low-level interface to the application using the OPI environment. The module communicates with the M13FX device level in order to access to the devices resources.

The module uses internal contexts (device, application, port and channel) in order to provide the required OPI services to the applications. The context memory allocation is dynamic and uses the target memory system allocation (DDS) through the SSAL memory management (BufferAlloc and BufferFree). The memory blocs are manipulated through handles which hide the target dependencies.

The module uses for each context type a table where the contexts are memorized and accessed. The table is static and the size is defined by compilation.

The context tables are:

- DRV_dctx_table, device context table,
- DRV_actx_table, application context table,
- DRV_pctx_table, port context table,
- DRV_cctx_table, channel context table.

A table element contains a pointer to the memory bloc handle supporting the context data area and a pointer to the context. The handle is used in order to release the memory bloc when the context is no longer used.

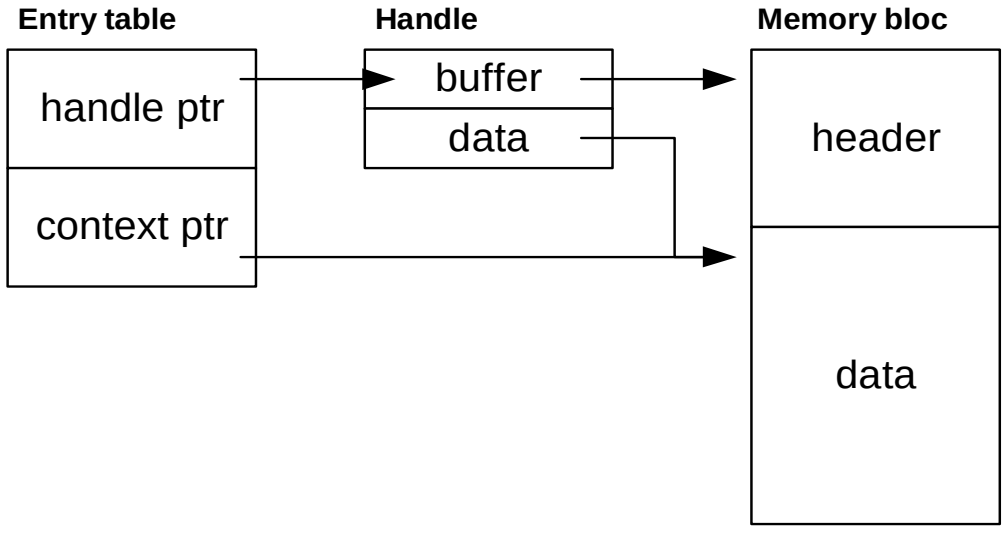



**Figure 13 M13FX OPI Level - context addressing**

Device context:

the following figure shows the device context structure allocated at the device initialization

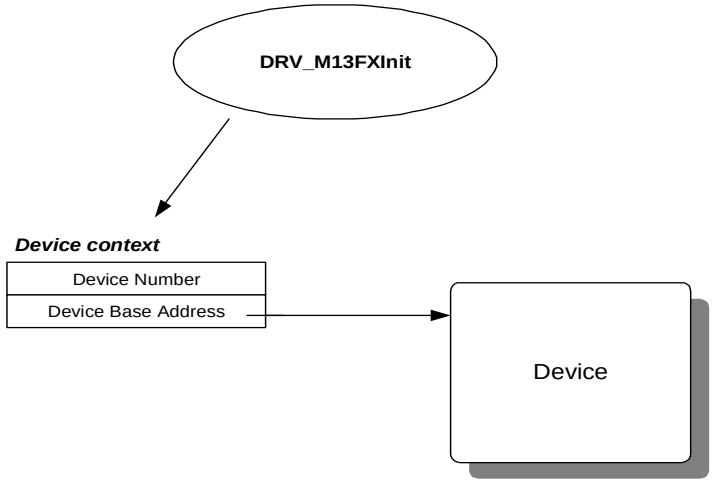



**Figure 14 M13FX OPI Level - device context**

Application context:

the following figure shows the application context structure allocated when an application binds to the device driver

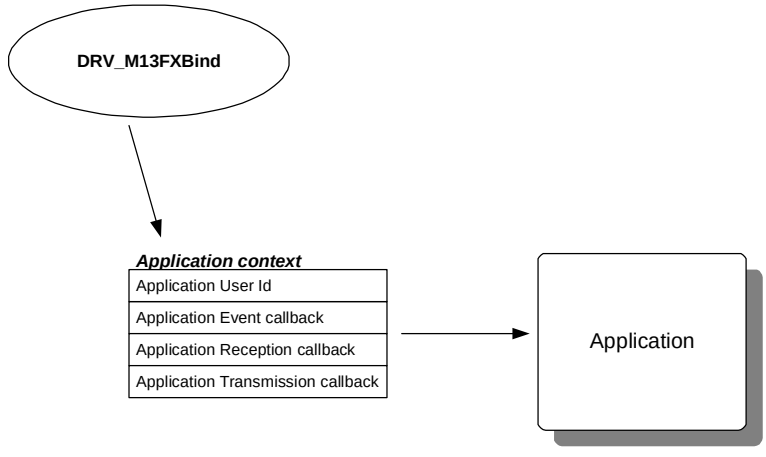



**Figure 15 M13FX OPI Level - application context.**

Port context:

the following figure shows the port context structure allocated when an application open a port.

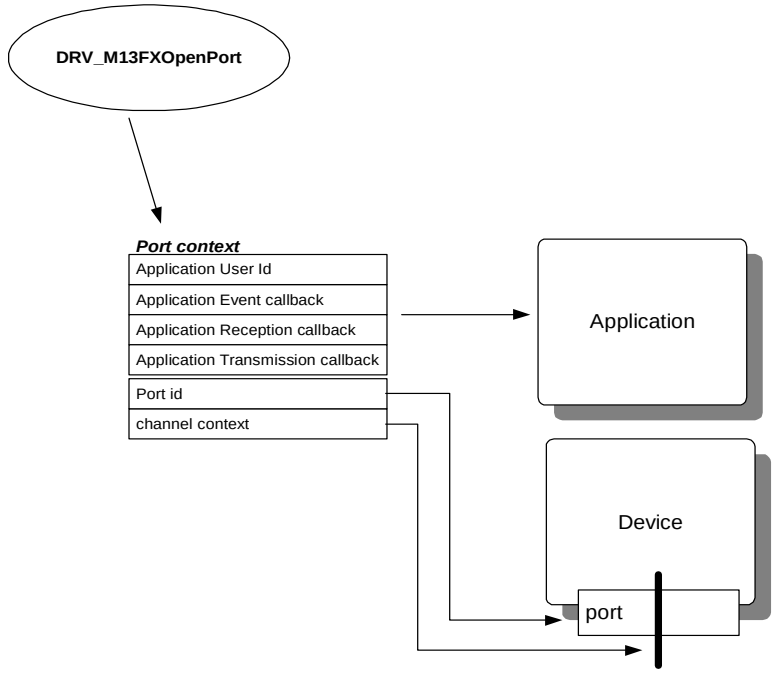



**Figure 16 M13FX OPI Level - port context**

Channel context:

the following figure shows the application context structure allocated when an application opens a channel on the port.

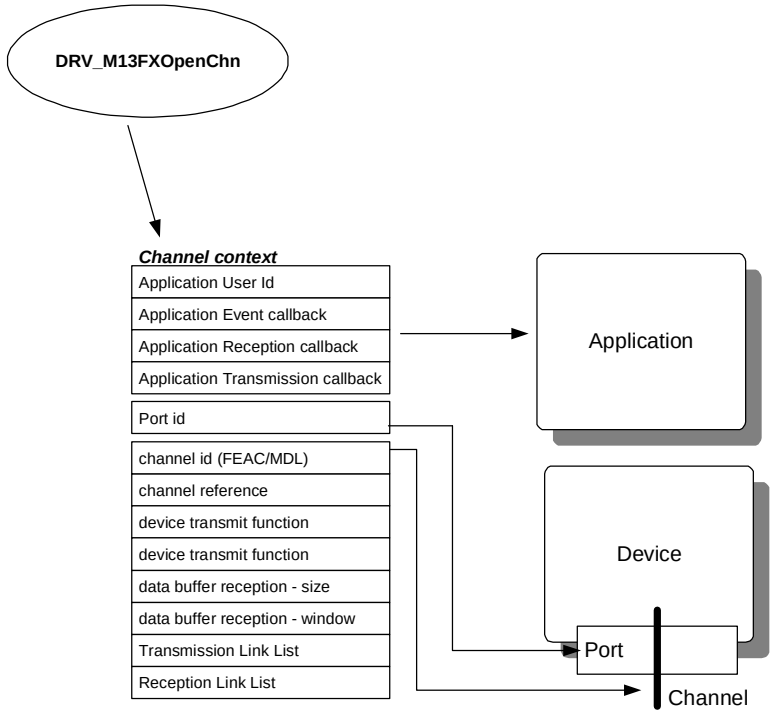



**Figure 17 M13FX OPI Level - channel context**

## M13FX OPI Level Architecture

The following figures summarizes the main parts involved in the OPI level module and their interactions inside and outside at the interface level.

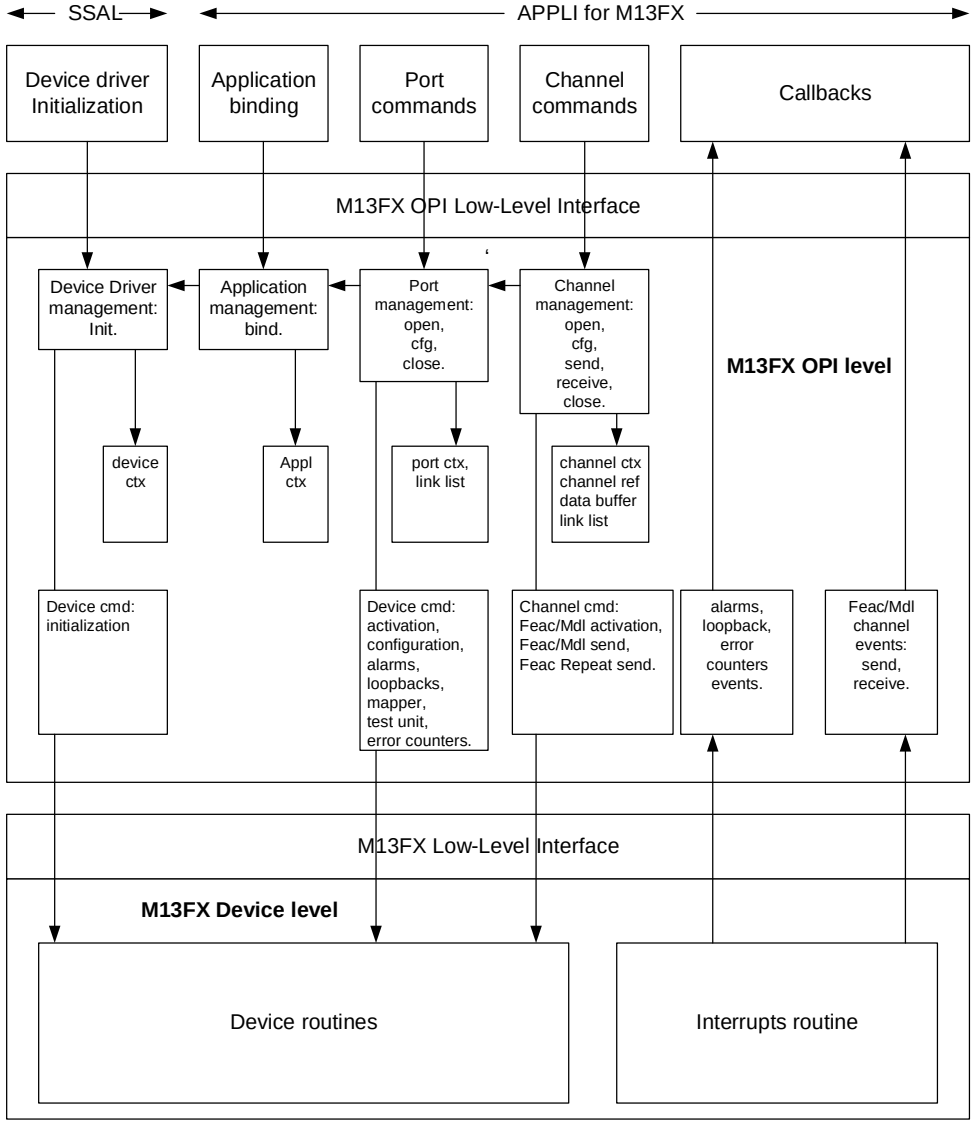



**Figure 18 M13FX OPI Level - architecture**

**DEVICE management:**

The device management bloc is responsible for the device initialization declared in the system (SSAL). The device initialization is done at the system start-up by calling the DRV_M13FXInit function.

For each device, it allocates a device context and passes to the device level module through the “M13FX_init” function, the base address.

**APPLICATION management:**

Applications call “DRV_M13FXBind” in order to access to the M13FX device driver. During this procedure the application and the device driver exchange their callbacks. The device driver assigns a unique user identification for each application, that defines the interface between the device driver and the application (set of callbacks). This procedure must be done by the application after the device driver initialization.

The Application management block is responsible for the application binding to the device driver. The Application management checks that almost one device is initialized and then allocates an application context where it memorizes the application callbacks, updates the application bind structure with the device driver callbacks and allocates a unique user id to the application. The binding procedure has no interactions with the device level.

**PORT management:**

The Port management block is responsible for the activation, the control and the deactivation of a device through the “DRV_M13FXPortxxx” functions called by the applications. The Port management handles the allocation of the device resources to the applications. A M13FX device is considered like a port and therefore the device resources can be granted by an application at one time. Once the port is opened, the application can issue the available device control commands (configuration, alarms, loopbacks, test unit, mapper, error counters) and activates a channel (FEAC, MDL).

The Port management checks that the application user identification is valid which means that the application is already bound to the device driver and that the port is not granted by another application.

If these conditions are met, the Port management allocates a port context where it memorizes the application user id, the application callbacks and then activates the device according to the port number.

Once the port is opened, the application device control commands passed through “DRV_M13FXPortCfg” are switched directly to the device level block.

**CHANNEL management:**

The CHANNEL management block is responsible for the activation of the FEAC and MDL channels and data transfer on the FEAC and MDL channel in a prior opened port. The channel resource is available to all the applications bound to the device driver (application port owner and other applications). The channel resource is allocated to the application on first-served basis.

The CHANNEL management checks that the application is bound to the device driver, the port is opened and that the channel is available.

If these conditions are met, a channel context is allocated and updated with the application relevant informations (user identification, callbacks, port number). The module allocates to the application a unique reference channel which will be used as channel direct access during all the channel life.

The CHANNEL management initializes the reception link list with the amount of data buffers (buffer window) and data length (buffer size) specified at the compilation time (BUFFER_SIZE, BUFFER_W) in m13ssalx.h and reset the transmit link list.

The CHANNEL management activates the device channel by calling the “M13FX_Feac_Open” or “M13FX_Mdl_Open” device functions. It passes the data buffer reception handle.

The CHANNEL management provides to the device level, reception functions in order to gain a new data buffer reception or send back a received message.

The CHANNEL management uses a transmission link list for the data transmission line. The data buffers received from the application are linked in the transmission link list. The CHANNEL management initializes the transmission when it receives a request from the application and that the transmission link list is empty otherwise the initialization is done automatically by the device level. The channel transmission is initialized by calling the “M13FX_Feac_Send” or “M13FX_Mdl_Send” device function with the data buffer to transmit. The Data buffer stays linked in the transmission Link List till the device driver notifies that the data has been sent in the line (w/o errors), then the data buffer is extracted from the transmission link list and sent back to the application (application transmission callback).

The Channel provides to the device level, transmission functions in order to notify the data transmission completion and gain a new data buffer to transmit.

Data buffers are manipulated through handles avoiding data copy process between the device driver and the application .

The CHANNEL management deactivates the device channel by calling the “M13FX_Feac_Close” or “M13FX_Mdl_Close” device functions.

**Application callbacks**

The application callbacks are called by the OPI level when a interrupt event is notified by the device level. The device level does not call directly the application callbacks but functions provided by the OPI level which are:

- “DRV_M13FX_Alarm_Event” for the DS3 , DS2 alarm and loopcode interrupt events,
- “DRV_M13FX_Sent_TxBuf” for the FEAC or MDL channel transmission interrupt,
- “DRV_M13FX_Rcv_RxBuf’ for the FEAC or MDL channel reception interrupt,
- “DRV_M13FX_Error_Counters_Event” for the One second Error counters interrupt.

**Internal OPI level routines**

The M13FX OPI level module contains a set of internal routines used by the device, application, port and channel management functions.

We have the following routines:

- DRV_Get_Device_Ctx, device context allocation,
- DRV_Alloc_Identifier, application user identifier allocation,
- DRV_Get_Application_Ctx, application context allocation,
- DRV_Sel_Application_Ctx, application context selection,
- DRV_Get_Port_Ctx, port context allocation,
- DRV_Put_port_Ctx, port context release,
- DRV_Sel_Port_Ctx, port context selection,
- DRV_Get_channel_Ref, channel reference allocation,
- DRV_Get_Channel_Ctx, channel context allocation,
- DRV_Sel_Channel_Ctx, channel context selection,
- DRV_Put_Channel_Ctx, channel context release,
- DRV_Init_Channel_LList, channel link list initialization.

### 3.2.4.3 M13FX Device level

The M13FX Device Level module provides the access to the M13FX device resources through a low-level interface: M13FX Low-Level Interface.

The module supports all the device functionality which are presented to the upper layer in an application-oriented approach, hidden the device complexity. Device users has only to think in term of functionality and not in term of device registers.

The M13FX Device Level is a static module which does not provides any management. It is a collection of device routines which constitutes the M13FX Low-Level interface. The module is multi device and provides at the device level the following groups of routines:

- Device Initialization,
- Device Configuration,

- Device Activation,
- Device Alarms,
- Device Loopbacks,
- Device Mapper,
- Device Test Unit,
- Device Feac Channel,
- Device Mdl Channel,
- Device Error Counters,
- Device Interrupt handler.

The following figure shows the module architecture.

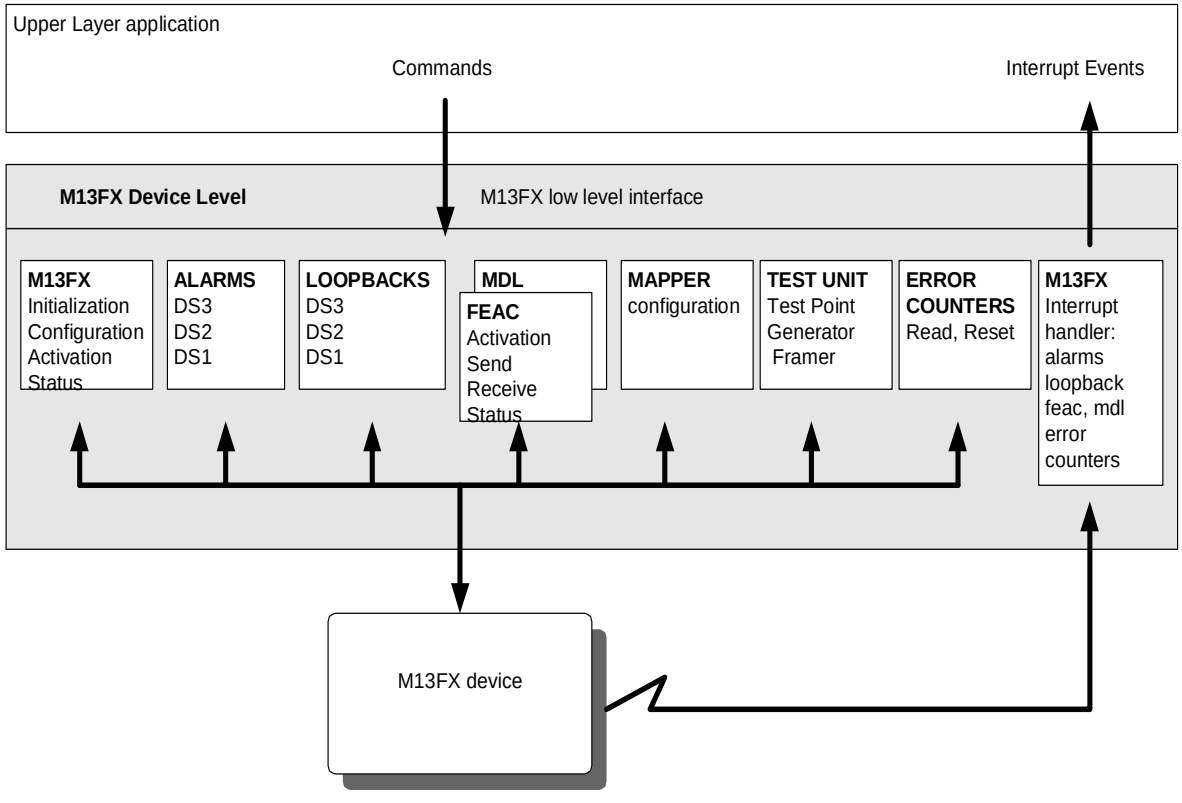



**Figure 19 M13FX Device Level - architecture**

The module uses for each device an internal context in order to deal with the configuration setup , interrupts and channel data transmission. The upper layer provides at the device initialization step, the device number and the base address associated to the device. The module memorizes both informations in the context and will use the base address in order to access to the device according to the device number.

The device contexts are memorized in a global table “**device_ctx_table**” and selected with the device number (index table) provided by the upper layer. The table is allocated statically and its size is defined at the compilation time.

The following figure shows the context structure.

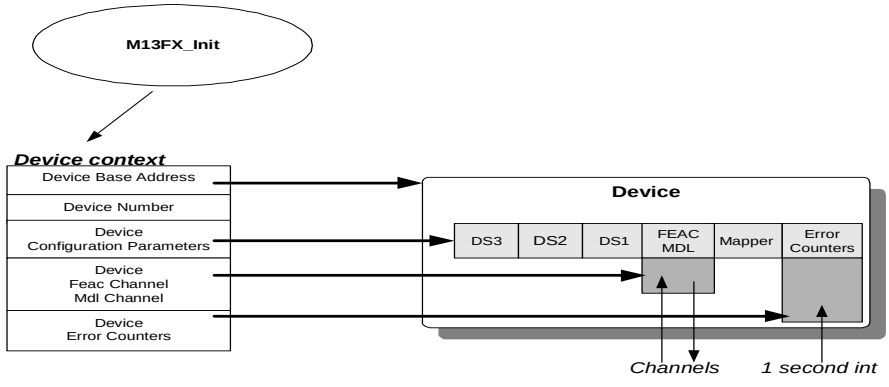



**Figure 20 M13FX Device Level - device context**

The module handles two states for the device handling:

- inactive, the device is programmed with the current configuration setup but it is not active (interrupts are turned off, AIS alarm generated toward the DS3 interface),
- active, the interrupts are turned on and the AIS alarm generation disabled.

## INITIALIZATION

A device is initialized by calling the “M13FX_Init” function. The module selects the device context in the table with the device number provided by the caller. It memorizes the base address and performs the following actions:

- put the the device in the inactive state,
- Reset the DS3 framer,
- turns off the device interrupts,
- Generates the AIS DS3 alarm,
- Programs the device with default parameter values (ds3, ds2, ds1, misc, feac, mdl and the tributary mapper).

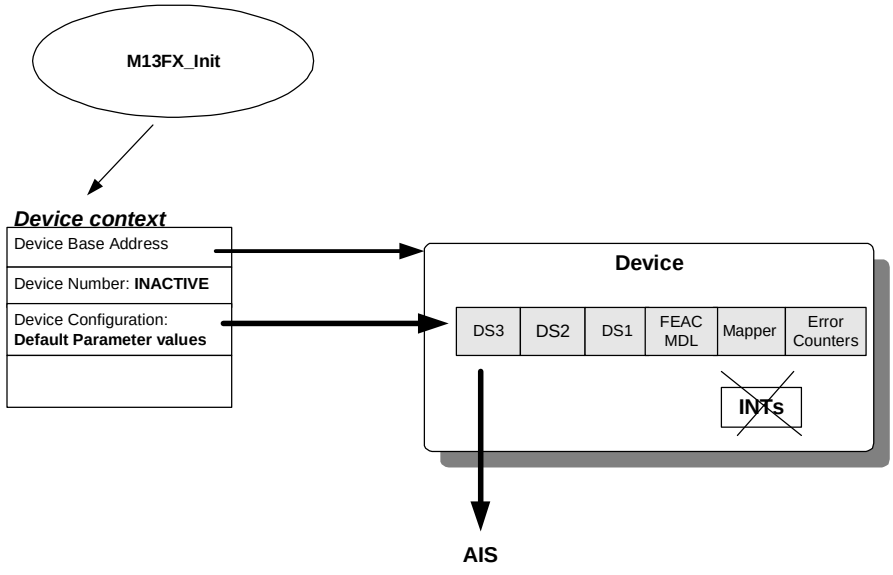



**Figure 21 M13FX Device Level - device initialization**

## CONFIGURATION

At the initialization the device is configured with the default parameters values (defined in the “M13FX_Init_Set_Default_Parameters” function).

The device level module offers a set of functions in order to modify the device configuration. The configuration is divided in several parameters groups (ds3, ds2, ds1, feac, test unit, mdl, misc) which are processed independently.

The device configuration is performed in two steps:

- **Parameters group modification**, this can be done at any time. The upper layer through the “M13FX_xxx_Cfg_Parameters” functions sends the parameters groups to update . The device driver computes the device register values for each group to update and **memorizes** them in the device context. The device is not programmed.
- **Parameters group validation**, this can be done only when the device is in the “inactive” state. The upper layer validates the memorized configuration by calling the “M13FX_Set_Cfg_parameters”. The device driver programmed the device with the register values memorized in the device context.

The default parameters can be restored by calling the “M13FX_Reset_Cfg_parameters” functions in the inactive state.

The following table summarizes the configuration functions:

**Table 24 device level - configuration functions**

| Function name | Description |
|---|---|
| M13FX_Init_Set_Default_Parameters | ds3, ds2, ds1, feac, mdl and misc parameter groups default values. |
| **M13FX_DS3_Cfg_Parameters** | ds3 parameter group modification. |
| **M13FX_DS2_Cfg_Parameters** | ds2 parameter group modification. |
| **M13FX_DS1_Cfg_Parameters** | ds1 parameter group modification. |
| **M13FX_Feac_Cfg_Parameters** | feac parameter group modification. |
| **M13FX_Mdl_Cfg_Parameters** | mdl parameter group modification. |
| **M13FX_Misc_Cfg_Parameters** | misc parameter group modification. |
| **M13FX_Set_Cfg_Parameters** | ds3, ds2, ds1, misc, parameter groups validation. mdl and feac parameter groups are validated at the channel activation time. |
| **M13FX_Reset_Cfg_Parameters** | ds3, ds2, ds1, feac, mdl and misc parameter groups reset. |
| **M13FX_DS3_Cfg_Get_Parameters** | ds3 parameter group configuration. |
| **M13FX_DS2_Cfg_Get_Parameters** | ds2 parameter group configuration. |
| **M13FX_DS1_Cfg_Get_Parameters** | ds1 parameter group configuration. |
| **M13FX_Feac_Cfg_Get_Parameters** | feac parameter group configuration. |
| **M13FX_Mdl_Cfg_Get_Parameters** | mdl parameter group configuration. |
| **M13FX_Misc_Cfg_Get_Parameters** | misc parameter group configuration. |

The following figures shows the configuration mechanisms.

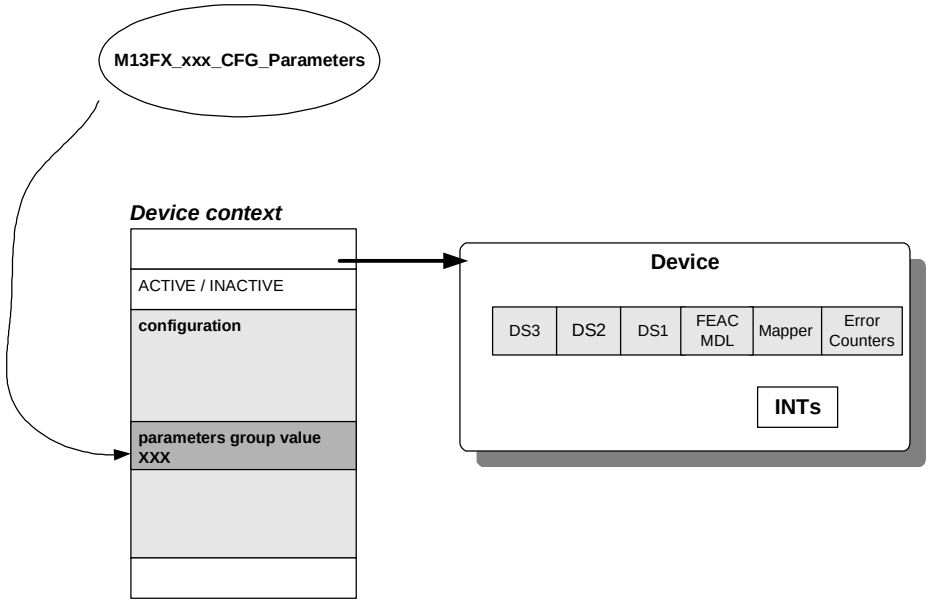



**Figure 22 M13FX Device Level - configuration modification**

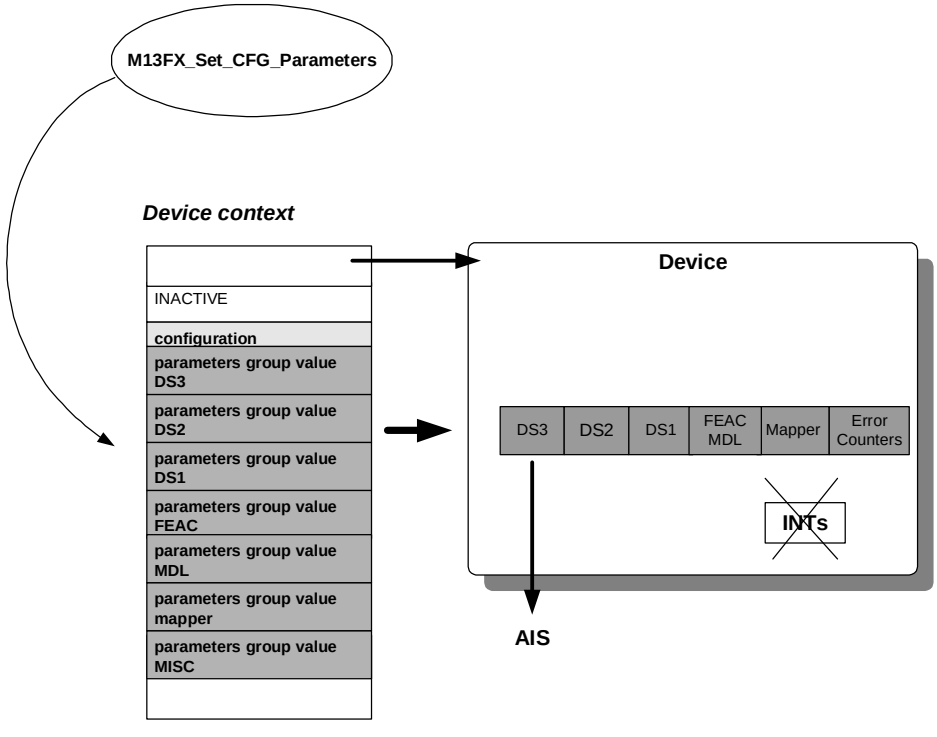



**Figure 23 M13FX Device Level - configuration validation**

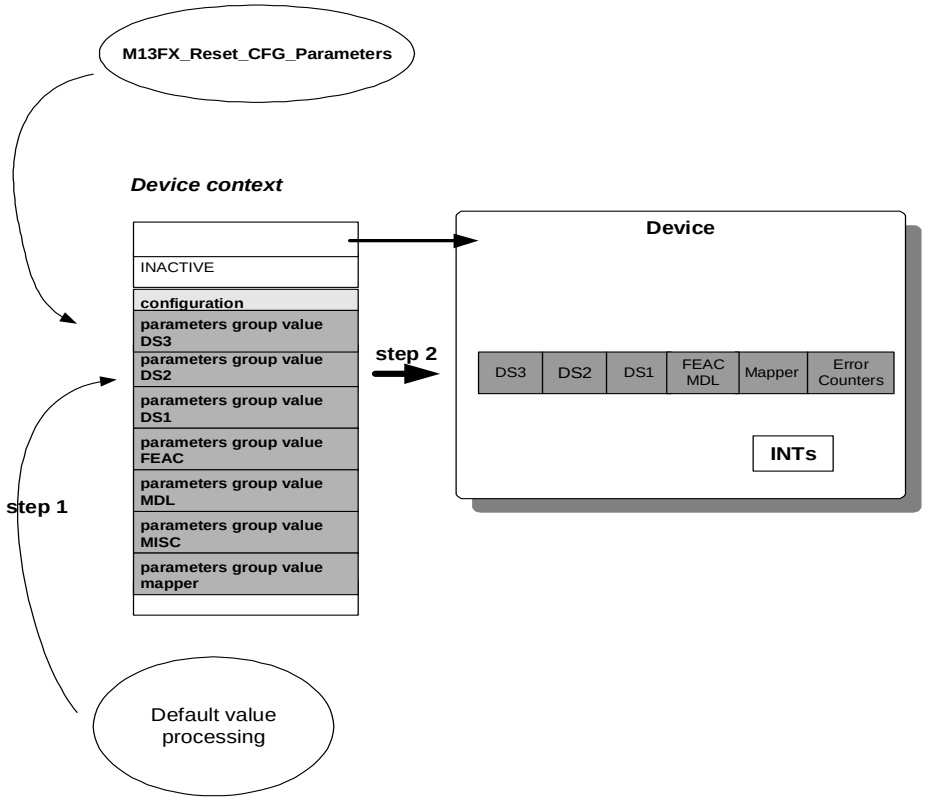



**Figure 24 M13FX Device Level - configuration reset**

## ACTIVATION

The device is controlled through the activation functions which enable/disable the device interrupts and the DS3 line interface (AIS alarm generation). The device must be in the active state prior to send any device control (alarm, loopback, test unit, feac channel) command to the device.

The **activation** command enables the device interrupts and the DS3 line interface. The device is operational and all the M13FX device functionality are accessible by the upper layer.

The **deactivation** command disables the device interrupts and the DS3 line interface (AIS alarm generation). The device is programmed with the default parameters values. The device driver is in the inactive state.

**Table 25 device level - Activation functions**

| Function name | Description |
|---|---|
| **M13FX_Device_Status** | Device status |

**Table 25 device level - Activation functions**

| Function name | Description |
|---|---|
| **M13FX_Device_Activate** | Device activation |
| **M13FX_Device_Deactivate** | Device deactivation |

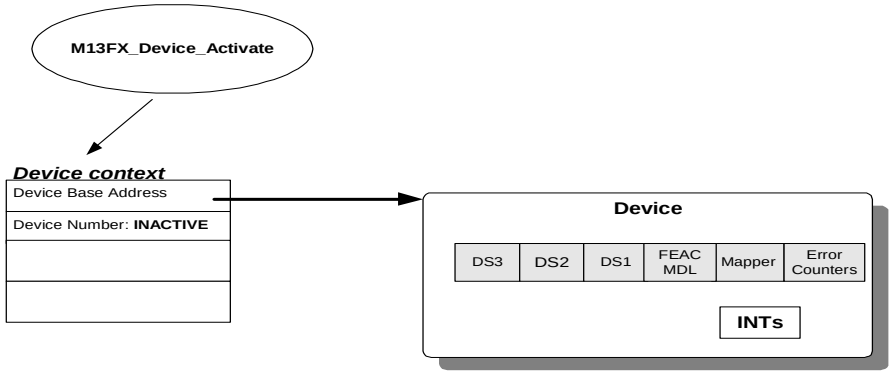



**Figure 25 M13FX Device Level - device activation**

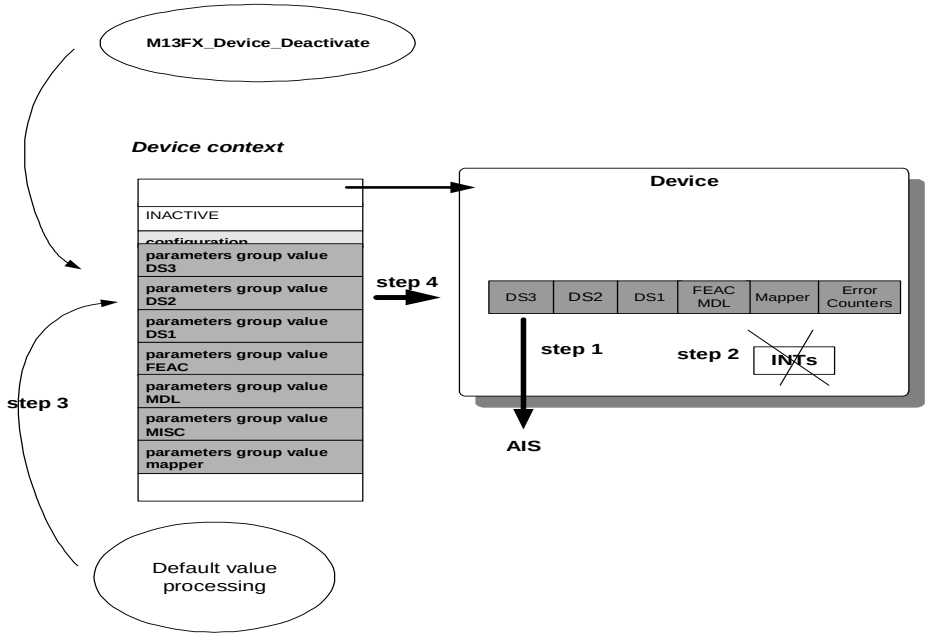



**Figure 26 M13FX Device Level - device deactivation**

## ALARMS

The ALARM block diagram provides at set of functions in order to generates and monitor the DS3, DS2 and DS1 device alarms (SET, RESET and STATUS). These functions are available in the device activation state.

The STATUS command provides the current status of the generation and detection alarms. The ALARM module handles the following alarms:

- DS3 alarms, AIS, RA (Yellow Alarm) and IDLE can be generated on the DS3 line interface; LOS, OOF, FAS and AIS detected from the DS3 line interface are reported in the STATUS command,
- DS2 alarms, AIS, and RA (Yellow Alarm) can be generated on each DS2 tributary toward the DS3 line interface; OOF, AIS, RA, LOS, OOF detected from each DS2 tributary are reported in the STATUS command,
- DS1 alarms, AIS toward the DS3 line interface, AIS toward the DS1 line interface on each 28 DS1 channels. The module does not report the alarm status coming from the DS1 line interface which is the responsibility of the QuadLiu device driver.

**Table 26 device level - alarm functions**

| Function name | Description |
|---|---|
| **M13FX_DS3_Alarm** | Ds3 alarms generation and monitor |
| **M13FX_DS2_Alarm** | Ds2 alarms generation and monitor |
| **M13FX_DS1_Alarm** | Ds1 alarms generation and monitor |

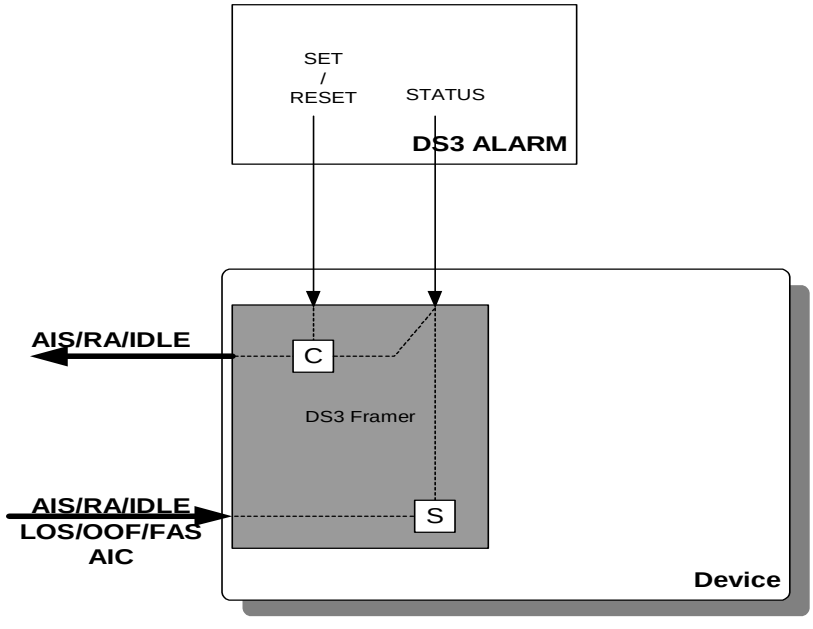



**Figure 27 M13FX Device Level - DS3 alarms**

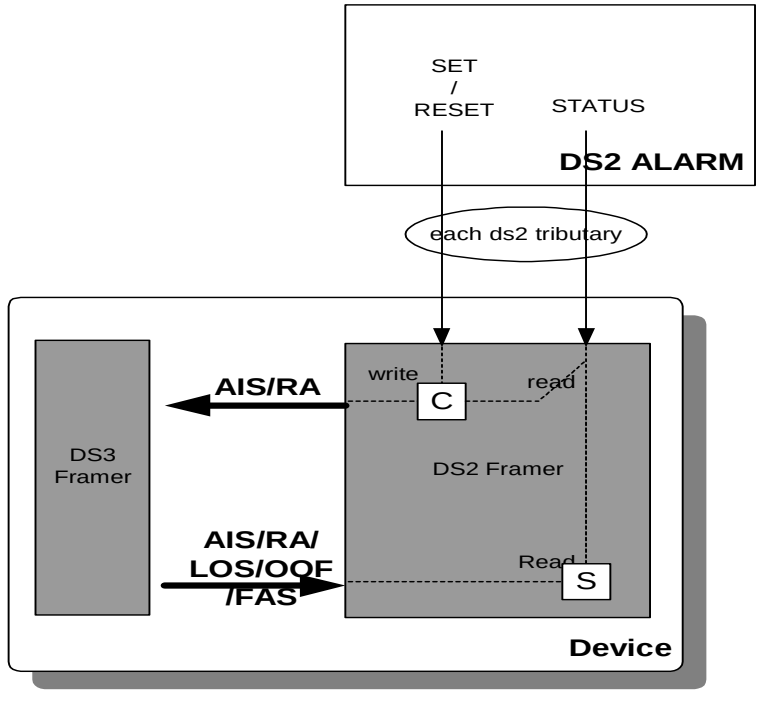



**Figure 28 M13FX Device Level - DS2 alarms**

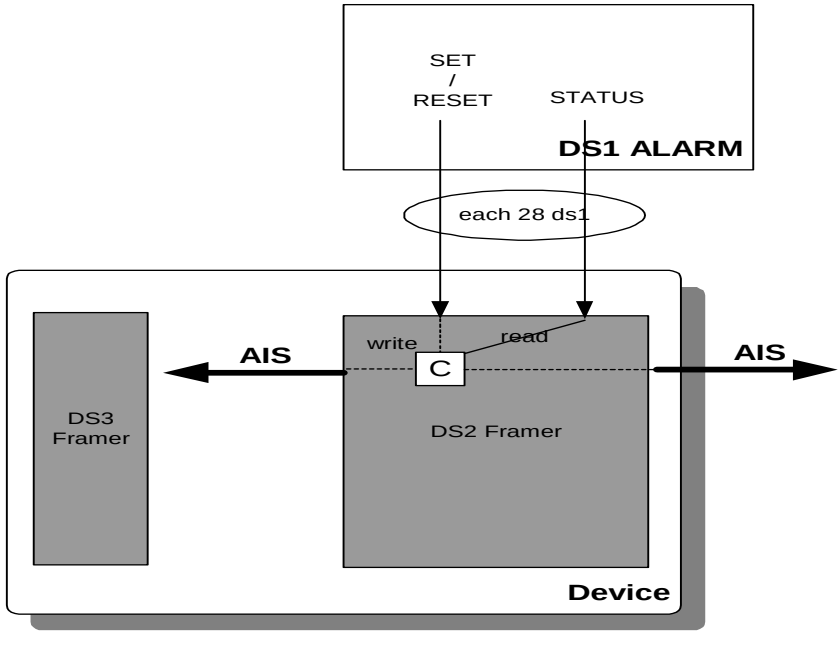



**Figure 29 M13FX Device Level - DS1 alarms**

## LOOPBACKS

The LOOPBACK block function provides a set of functions in order to set and monitor the DS3, DS2 and DS1 device loopbacks (SET, RESET and STATUS). This functions are available at the device activation state.

The STATUS command provides the current status of the loopback setting. The LOOPBACK module handles the following loopback:

- DS3 loopback, Remote and Local at the DS3 Line Interface,
- DS2 loopback, Remote and Loopcode for each DS2 tributary,
- DS1 alarms, Remote , Local and loopcode for each 28DS1/21E1 DS1 tributary.

**Table 27 device level - Loopback functions**

| Function name | Description |
|---|---|
| **M13FX_DS3_Loopback** | DS3 loopback setting and monitor |
| **M13FX_DS2_Loopback** | DS2 loopback setting and monitor |
| **M13FX_DS1_Loopback** | DS1 loopback setting and monitor |

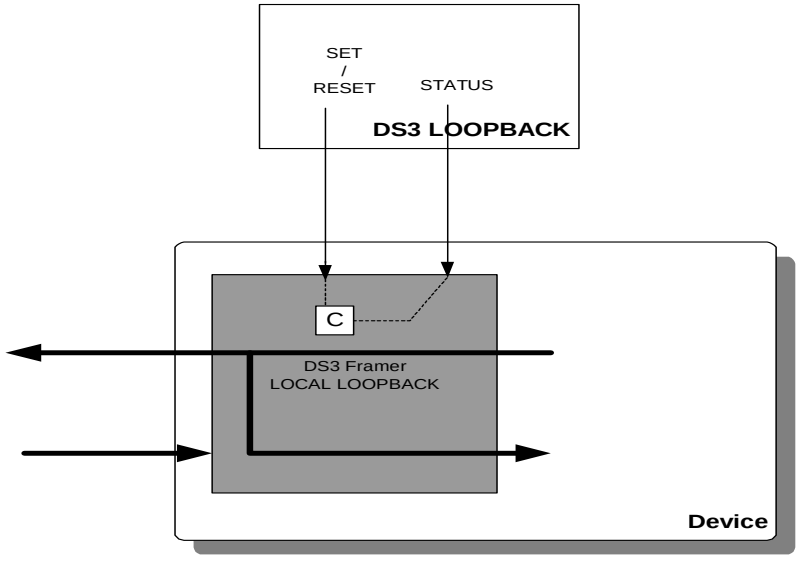



**Figure 30 M13FX Device Level - DS3 local loopback**

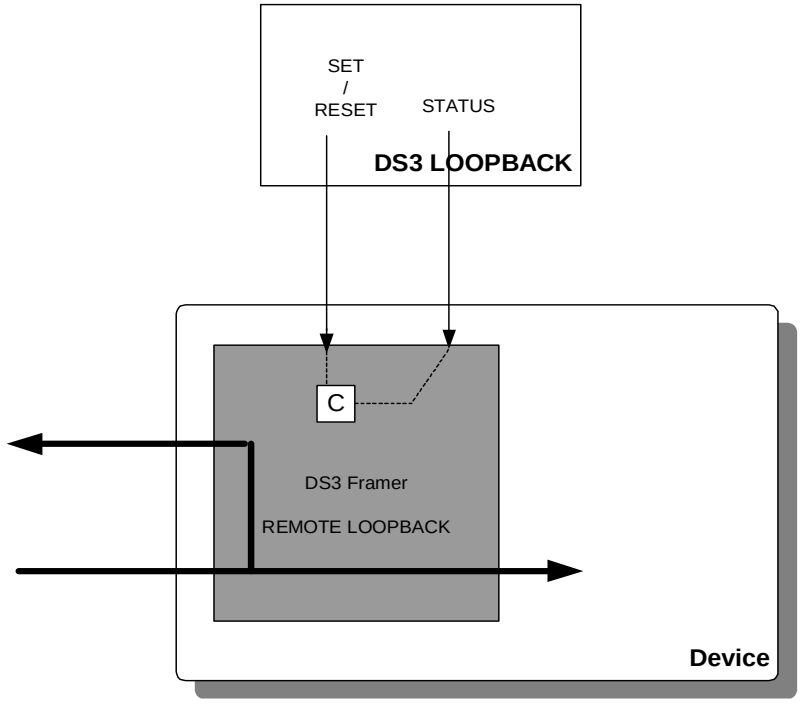



**Figure 31 M13FX Device Level - DS3 remote loopback**

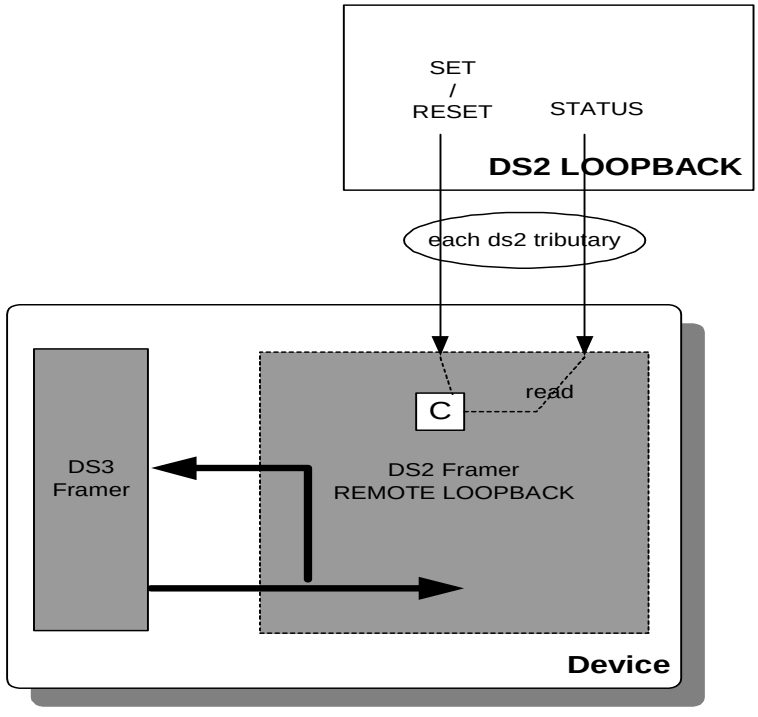



**Figure 32 M13FX Device Level - DS2 remote loopback**

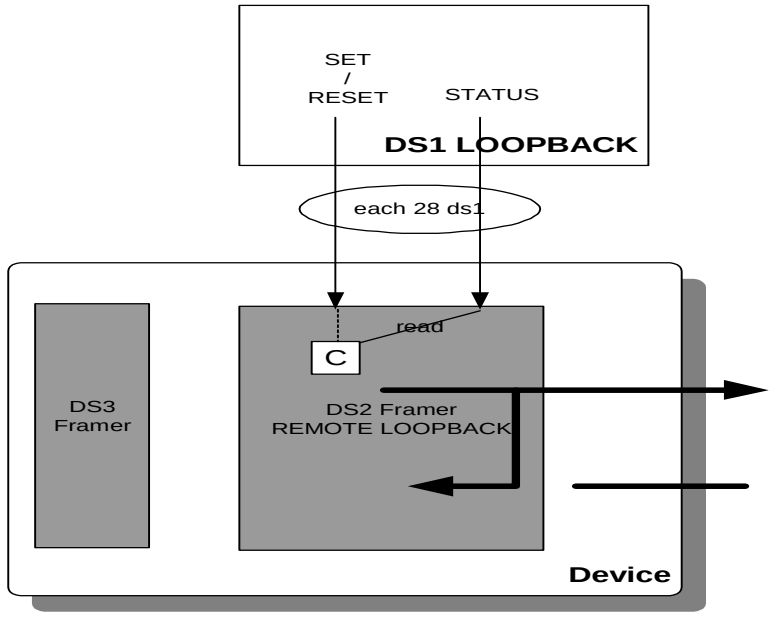



**Figure 33 M13FX Device Level - DS1 remote loopback**

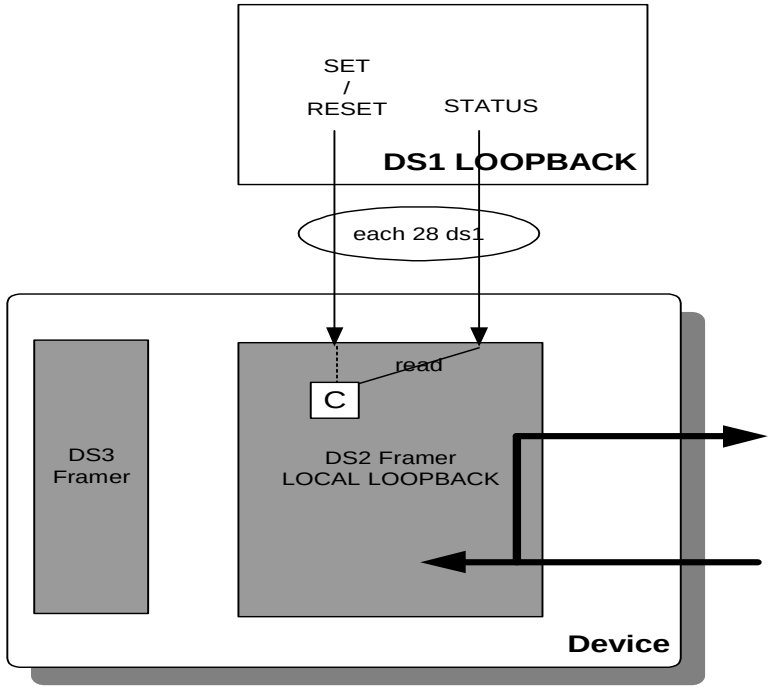



**Figure 34 M13FX Device Level - DS1 local loopback**

## FEAC CHANNEL

The FEAC channel provides a set of functions which allows to control the channel and to send / receive BOM code to/from the DS3 line interface.

The channel modes (fifo threshold, reception filter mode, transmission repeat mode) are configured through the configuration functions.

The channel is activated and operational by calling the "**M13FX_Feac_Open**" function which initializes the reception with the data buffer passed in the function (handle), turn on the Feac interrupts and enables the receiver and transmitter controller.

The channel is deactivated by calling the "**M13FX_Feac_Close**" function which disables the reception, turn off the Feac interrupts and disables the receiver and transmitter controller. The current data buffer reception is returned to the caller in the return function call.

The channel must be open in order to be able to receive or transmit BOM code. A BOM message is a codeword composed of a synchronization byte (0xFF) and a BOM code (0xxxxxxx0b) which carry the alarm and loopback commands.

The upper layer always receives or sends the BOM code part of the BOM message. The module insert for each BOM code to send the synchronization byte, in reception the

device synchronizes with the BOM synchronization byte and acquires the BOM code which is stored in the receiver FIFO.

The **transmission** is started by calling the “M13FX_Feac_Send” function. The upper layer passes the data buffer (handle) with the BOM codes to transmit. At the transmission completion (with or without error) the data buffer is passed up through the upper layer transmit complete callback with a transmission status report. The module requests to the upper layer a new data buffer to transmit by calling the upper layer transmit callback and initializes the transmission with the new data buffer otherwise the transmission will be initialized by the upper layer at the next available data buffer. The upper layer can issue a transmit request only when there is no transmission pending therefore it has to implement a transmission requests buffering function.

The **reception** is initialized at the channel opening with the data buffer provided by the upper layer. When a BOM message is received, the module prior to pass up the data buffer to upper layer (trough the receive callback) requests a new data buffer to the upper layer in order to initialize the reception of the next BOM message. If there is no data buffer reception available, the module initializes the reception with the current data buffer reception (the received BOM message is ignored) and a error counter is updated in the channel context, otherwise the reception is initialized with the requested data buffer and the received BOM code is passed up to the upper layer. There is no receive function available to upper layer, the reception completion is done in the interrupt.

The module provides to the device interrupt handler the functions to call when an channel event occurs.

**Table 28 device level - FEAC channel functions and callbacks**

| Function name | Description |
|---|---|
| **M13FX_Feac_Status** | Feac channel status. |
| **M13FX_Feac_Open** | Feac channel activation. |
| **M13FX_Feac_Close** | Feac channel deactivation. |
| **M13FX_Feac_Send** | Feac channel transmission. |
| **DRV_M13FX_Rcv_RxBuf** | Feac receive complete callback. |
| **DRV_M13FX_Sent_TxBuf** | Feac transmit completion callback. |
| **DRV_M13FX_Get_RxBuf** | Feac reception buffer allocation callback. |
| **M13FX_Feac_ALLS_Int** | Feac transmission completion interrupt. |
| **M13FX_Feac_XDU_Int** | Feac reception underrun interruption. |
| **M13FX_Feac_XPR_Int** | Feac transmit Pool Ready interrupt. |
| **M13FX_Feac_RPF_Int** | Feac Receive Pool Full interrupt. |

**Table 28 device level - FEAC channel functions and callbacks**

| Function name | Description |
|---|---|
| **M13FX_Feac_RME_Int** | Feac reception Completion interrupt. |
| **M13FX_Feac_RMI_Int** | Feac reception Message Idle interrupt. |

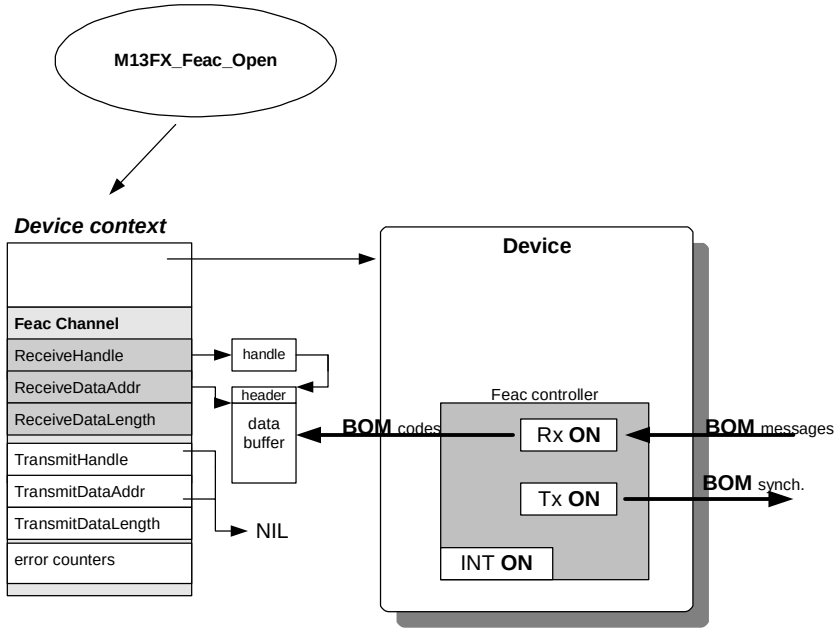



**Figure 35 M13FX Device Level - Feac Channel Open**

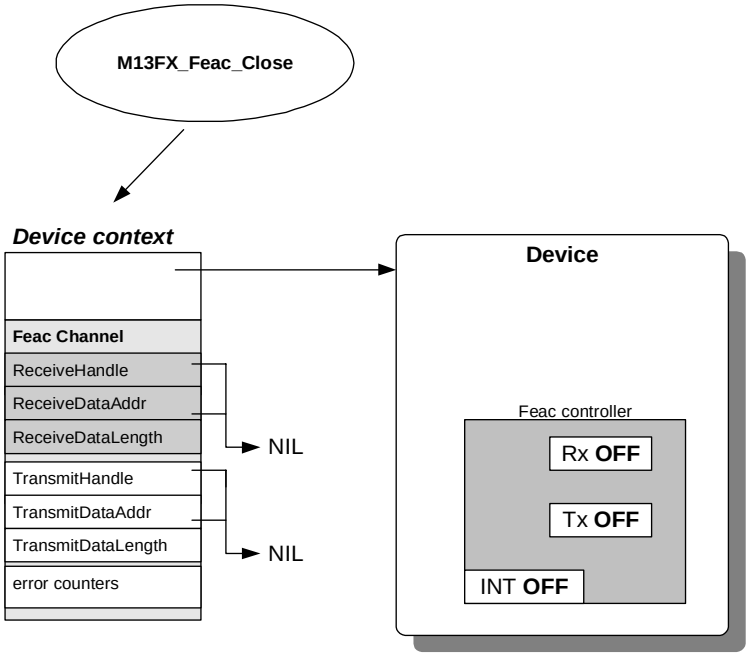



**Figure 36 M13FX Device Level - Feac Channel Close**

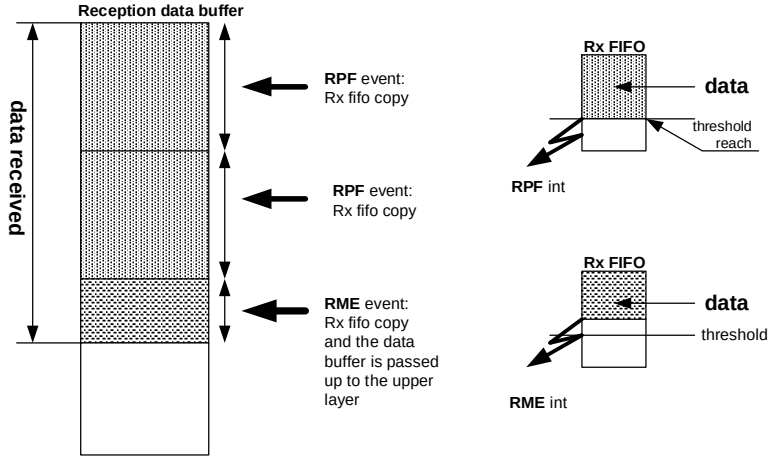



**Figure 37 M13FX Device Level - Feac Channel Reception (concept)**

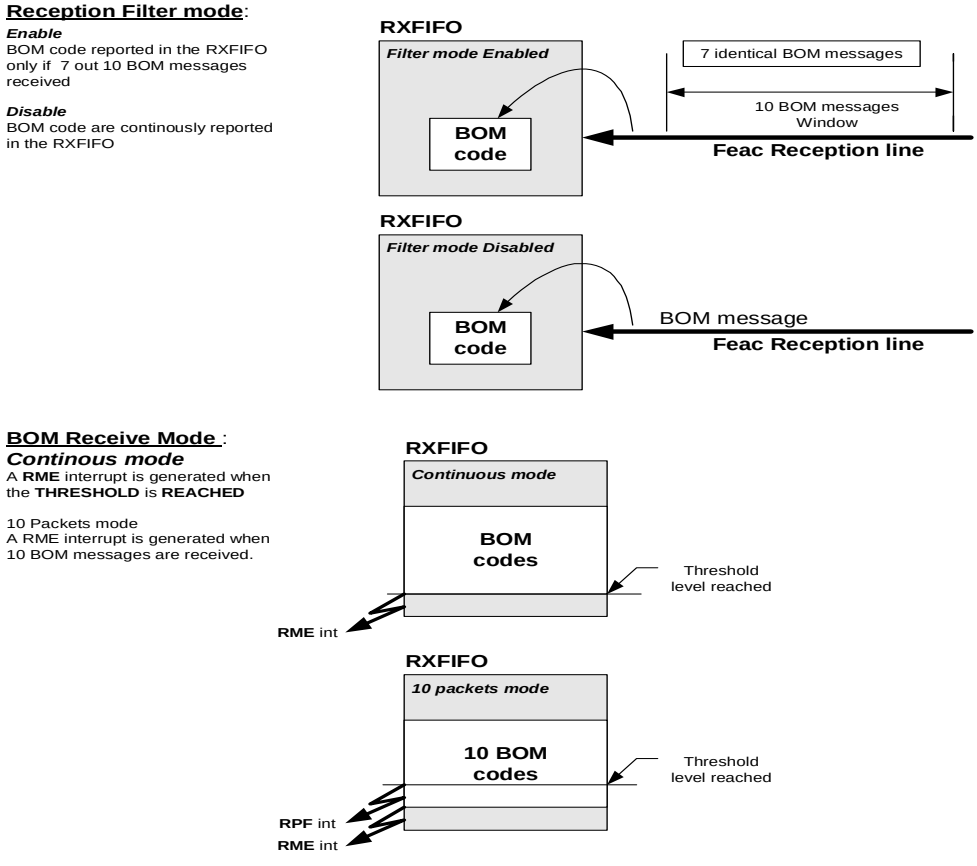
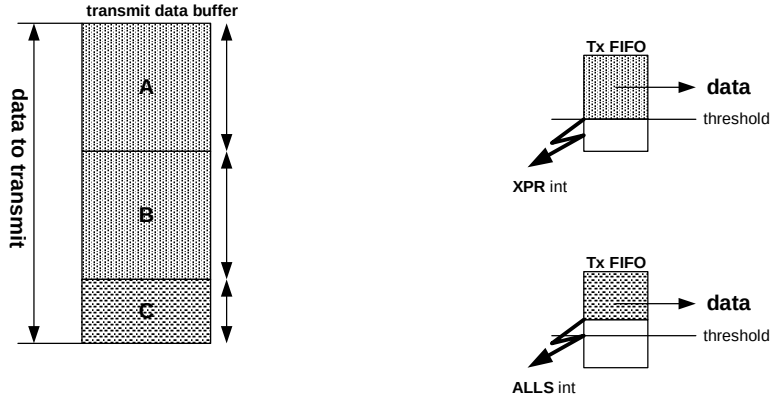
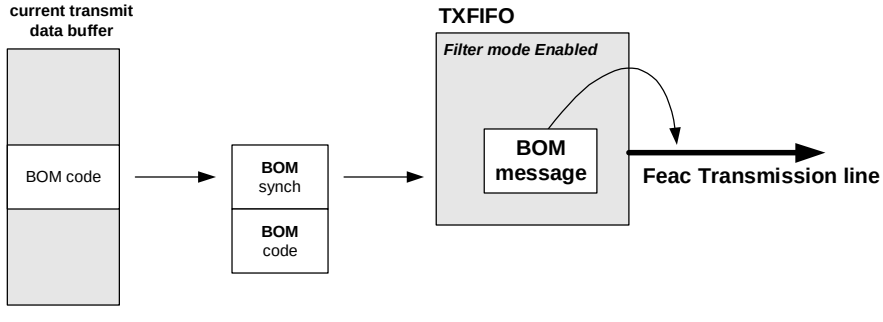



**Figure 38 M13FX Device Level - Feac Channel Reception (BOM)**




1/ data block A :
this bloc has the size of the Tx FIFO, the block is copied into the TXFIFO and the XTF command is issued. The transmitter sends the TXFIFO data in to the transmit line. The transmitter generates an XPR interrupt when the transmit is complete.
2/data block B:
same process as 1/
3/ data block C:
this is the last bloc of the message to transmit, the block is copied into the TXFIFO and the XTF and XME commands are issued. the transmitter generates an ALLS interrupt when the transmit is completed.
4/ transmit data buffer passed up to the upper layer

**Figure 39 M13FX Device Level - Feac Channel Transmission**




The BOM data transmission follows the transmission concept except that for EACH data byte a BOM synch code is INSERTED in order to compose a BOM message. The transmitter does not perform the insertion

**Figure 40 M13FX Device Level - Feac Channel Transmission**

## MDL CHANNEL

The MDL channel provides a set of functions which allows to control the channel and to send / receive HDLC-based frames to/from the DS3 line interface.

The channel modes (Rx fifo threshold level, Rx CRC check and transfer, Tx CRC generation) are configured through the configuration functions and validated at the channel activation.

The channel is activated and operational by calling the "**M13FX_Mdl_Open**" function which initializes the reception with the data buffer passed in the function (handle), turns on the Mdl interrupts and enables the receiver and transmitter controller.

The channel is deactivated by calling the "**M13FX_Mdl_Close**" function which disables the reception, turns off the Mdl interrupts and disables the receiver and transmitter controllers.

The channel must be open in order to be able to receive or transmit HDLC-based messages.

The **transmission** is started by calling the "M13FX_Mdl_Send" function. The upper layer passes the data buffer (handle) with the HDLC data frame to transmit. At the transmission completion (with or without error) the data buffer is passed up through the upper layer transmit complete callback with a transmission status report. The module requests to the upper layer a new data buffer to transmit by calling the upper layer transmit callback and initializes the transmission with the new data buffer otherwise the transmission will be initialized by the upper layer at the next available data buffer. The upper layer can issue a transmit request only when there is no transmission pending therefore it has to implement a transmission requests buffering function.

The **reception** is initialized at the channel opening with the data buffer provided by the upper layer. When a HDLC frame is received, the module prior to pass up the data buffer to upper layer (trough the receive callback) requests a new data buffer to the upper layer in order to initialize the reception of the next HDLC frame. If there is no data buffer reception available, the module initializes the reception with the current data buffer reception (the received frame message is ignored) and a error counter is updated in the channel context, otherwise the reception is initialized with the requested data buffer and the received frame is passed up to the upper layer. There is no receive function available to upper layer, the reception completion is done in the interrupt.

The module provides to the device interrupt handler the functions to call when an channel event occurs.

**Table 29 device level - MDL channel functions and callbacks**

| Function name | Description |
|---|---|
| **M13FX_Mdl_status** | Mdl channel status. |
| **M13FX_Mdl_Open** | Mdl channel activation. |

**Table 29 device level - MDL channel functions and callbacks**

| Function name | Description |
|---|---|
| **M13FX_Mdl_Close** | Mdl channel deactivation. |
| **M13FX_Mdl_Send** | Mdl channel transmission. |
| **DRV_M13FX_Rcv_RxBuf** | Receive complete callback. |
| **DRV_M13FX_Sent_TxBuf** | Transmit completion callback. |
| **DRV_M13FX_Get_RxBuf** | Reception buffer allocation callback. |
| **M13FX_Mdl_ALLS_Int** | Mdl transmission completion interrupt. |
| **M13FX_Mdl_XDU_Int** | Mdl reception underrun interruption. |
| **M13FX_Mdl_XPR_Int** | Mdl transmit Pool Ready interrupt. |
| **M13FX_Mdl_RPF_Int** | Mdl receive Pool Full interrupt. |
| **M13FX_Mdl_RME_Int** | Mdl reception Completion interrupt. |

The MDL channel follows the same principle processing that the FEAC channel. Refer to the FEAC channel figures.

**ERROR COUNTERS:**

The Error counters module is responsible for handling the DS3 and DS2 error counters. The error counters are incremented and updated according to the following Error counters mode which can be set through the MISC configuration functions:

- **One Second interrupt** mode, the counter values are copied to the foreground error counter registers and read out to the device context, in a one second interval. At the same time the background error counter registers are reset to zero. This operation is synchronous with the periodic one second interrupt which alerts the module to read out the error counters. In this mode, the One Second interrupt is enabled. The module provides an entry-point to call "**M13FX_Interrupt_1S"** when the One second interrupt occurs. An error counters "**DRV_M13FX_Error_Counters_Event**" indication is sent to the upper layer if only there is almost ONE error counter with a NON zero value.
- **Polling** mode, the background error counter registers are copied to the foreground registers when a copy command (with or without clearing the error counters) is issued by the application. In this mode the application has to poll periodically the target in order to get the error counter updates. The module provides to application a function "**M13FX_Error_Counters**" which allows to get the error counter updates.

The following table shows the error counters module functions.

**Table 30 device level - ERROR COUNTERS functions**

| Function name | Description |
|---|---|
| **M13FX_Error_Counters** | Error counters commands. |
| **M13FX_Interrupt_1S** | Error counter One Second Interrupt. |
| **DRV_M13FX_Error_Counters_Event** | Error counter indication. |

**TEST UNIT:**

The Test Unit module provides a set of functions in order to access to the pattern GENERATOR and the FRAMER. The GENERATOR and FRAMER set of functions are independent in order to allow framed or unframed pattern generations. The test unit generates an indication to the upper layer when a event occurs detected by the interrupt service routine.

The GENERATOR and FRAMER sequence programing follow the same scheme which is:

- Set the Generator/Framer configuration. Automatically stops the generator/framer.
- Set the Test Point Insertion for the generator. Automatically stops the generator.
- Start the Generator/Framer. Run the generator/framer with the current setting.
- Error insertion into the generator/framer.
- Stop the Generator/Framer. The current setting is not changed. The error insertion is disabled.
- At any time the generator receive fixed-pattern and counters can be read.
- At any time the framer error counters can be read.

The following table shows the test unit module functions.

**Table 31 device level - Test unit functions**

| Function name | Description |
|---|---|
| **M13FX_TU_Set_Generation** | Configure the generator. |
| **M13FX_TU_Start_Generation** | Start the generator. |
| **M13FX_TU_Stop_Generation** | Stop the generator. |
| **M13FX_TU_Test_Point** | Set the generator Test Point Insertion. |
| **M13FX_TU_Error_Insertion** | Generator error insertion command. |
| **M13FX_TU_Receive_Pattern** | Read the generator fixed-pattern. |

**Table 31 device level - Test unit functions**

| Function name | Description |
|---|---|
| **M13FX_TU_Receive_Counter** | Read the generator bit and error counters. |
| **M13FX_TU_Set_Framer** | Configure the framer. |
| **M13FX_TU_Start_Framer** | Start the framer. |
| **M13FX_TU_Stop_Framer** | Stop the framer. |
| **M13FX_TU_Error_Framer** | Framer error insertion command. |
| **M13FX_TU_Receiver_Counter_Framer** | Read the framer error counters. |
| **DRV_M13FX_Alarm_Event** | Test Unit indication when an interrupt occurs. |

## TRIBUTARY MAPPER:

The Tributary MAPPER module provides a set of functions in order to configure the tributary interchanger.The module maintains a structure which indicates for each 32 tributaries (28 tributaries + 4 spares) which serial interface is assigned. The data structure can be configured, read or reset to the default configuration setting.

The following table shows the test unit module functions.

**Table 32 device level - Test unit functions**

| Function name | Description |
|---|---|
| **M13FX_Map_Get_Configuration** | Read the tributary mapper configuration. |
| **M13FX_Map_Set_Configuration** | Configure the tributary mapper. |
| **M13FX_Map_Reset_Configuration** | Reset the tributary mapper configuration. |

## 3.3 QuadLIU Device Driver Environment

### 3.3.1 Initialization

The QuadLIU DDE is initialized through the “DDS Board” module. The DDS Board module provides to the target system an initialization entry-point which must be issued at the system startup.

The entry-point is DDSBoard_Init() which requires no parameter. The file “ddsboard.h” must be included in order to access to the function. The entry-point is called in the DDSI module at the DDS system startup.

The code included in the DDSI module is :

```
#include "ddsboard.h"

DDSBoard_Init();
```

### 3.3.2 DDS Board module

The DDS Board module provides the initialization of DDS device drivers. After the initialization, all the DDS device drivers are initialized and ready to be accessed by the relevant application.

The DDS board module performs the following actions:

- QuadLIU chip setup according to the HW specifications,
- QuadLIU Device driver initialization for 2 QuadLIU devices,
- QuadLIU Interrupts Routine installation,
- 8 DS1 lines initialized in T1 mode.

### 3.3.3 QuadLIU Device Driver Module

The description of the Quadliu Device Driver Module is out of scope of this document which is focused in the M13FX device. The device driver is compliant with the DDS driver driver specifications and can be accessed through the C/I message interface. The device driver is declared to the DDS system with the QuadLIU_MODULE (4) identifier module. A set of C/I messages allows to:

- Configure the line in T1 or E1 mode,
- Set or reset a DS1/E1 loopback (local or remote),
- Generate a DS1/E1 AIS alarm.

### 3.3.4 QuadLIU C/I interface

The QuadLIU C/I interface is a message-based interface according to the DDS software specifications. The message description is provided in **Chapter 7**.

# 4 Device Driver System (DDS)

The Device Driver System (DDS) is a minimal Operating System that supports the transfer of Command/Indication (C/I) Messages, with a single standardized format, as the basic communication mechanism for the exchange of information between the device driver modules (in this case M13FX and QuadLIU) and the user application (in this case WinEASY).

For more details, please refer to the Device Driver System for EASY Tools Description (provided on the CD-ROM).

# 5 OPI concept

The OPI (Open Protocol Interface) concept is a software architecture which provides the following interfaces:

- A unique interface to the application for the Infineon Device Drivers (OPI upper layer),
- A generic interface to the Infineon Device Drivers families (OPI lower layer),
- An adaptation module which adapts the Device Driver to the target OS and platform. (System Service Adaptation Layer).

The following figure shows the OPI architecture.

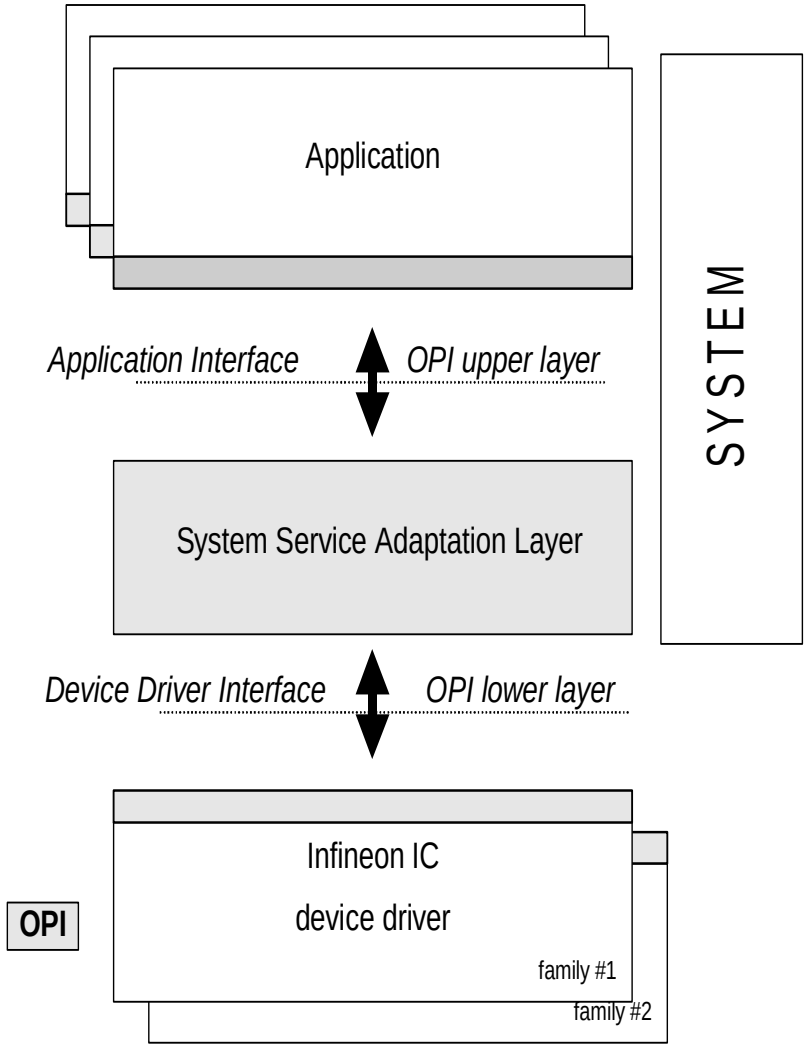



**Figure 41 OPI - architecture**

## 5.1 OPI characteristics.

The OPI concept has the following characteristics,

- Generic interface for the application and for the device driver,
- Choice of the interface level (application level or device driver level),
- Connectivity to the device driver is dynamic,
- Multi-application access to the device driver,
- Multi-device access,
- Adaptation to the RTOS is done in the SSAL module,
- Transparent Buffer management by using handles which hide the header-system,
- Port and Channel concept is generic enough to be applied to different types of devices,
- The notion of device is hidden to the application which only knows port and channel,
- Support different system configurations differentiated by the SSAL module,
- The device driver is totally OS and platform independent.

## 5.2 Different OPI Configurations

The following figures show the different configuration availables with the OPI architecture.

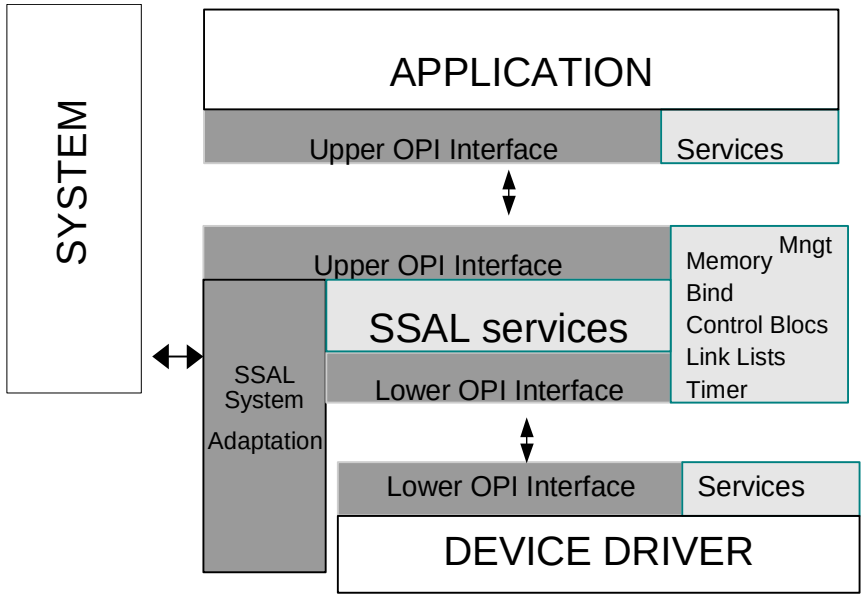



**Figure 42 OPI configuration - Upper OPI level**

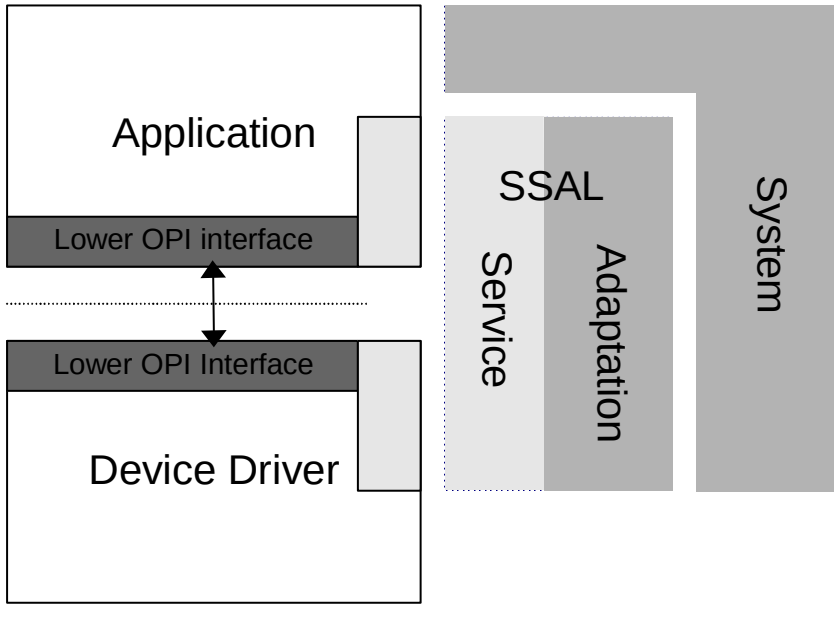



**Figure 43 OPI configuration - Lower OPI level**

## 5.3 OS and platform independent

The OPI concept preserves the OS and Platform independence for the target device driver. Each configuration (OS, platform) is defined by the SSAL module. The target device driver does not require any modification.

The system dependent functions are ALWAYS and ONLY located in the SSAL system adaptation module. The system adaptation module varies with the target system and provides the following services:

- Interrupt handling,
- Device driver Initialization,
- Memory adaptation,
- link list management.

the following figure shows that only the SSAL module varies according to the target system.

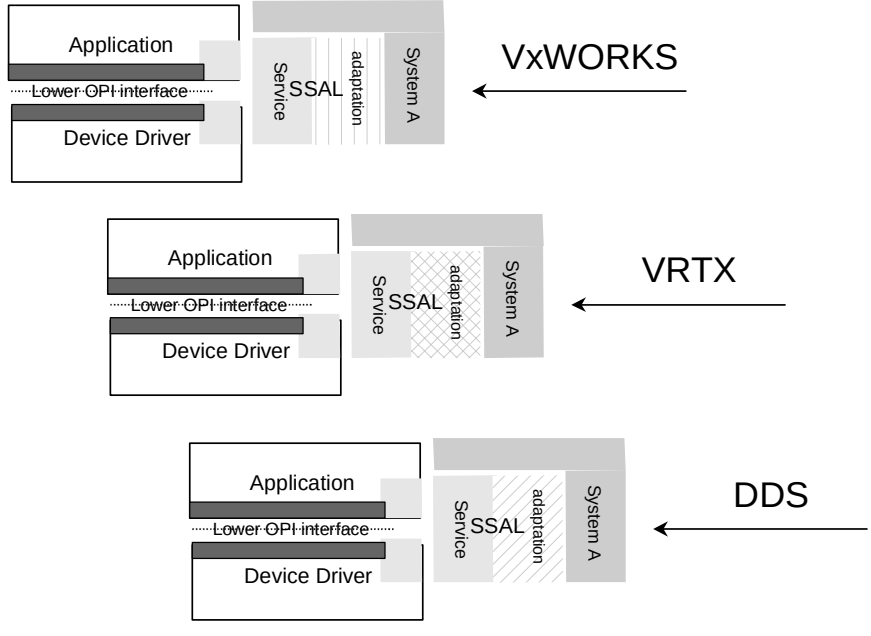



**Figure 44 OPI - OS and platform independent**

## 5.4 Multi-application and multi-devices support

The OPI architecture supports the multi-application and multi-device access. Any application can be bound to any device driver.

The following figure shows an example of different applications/devices configurations that can be used.

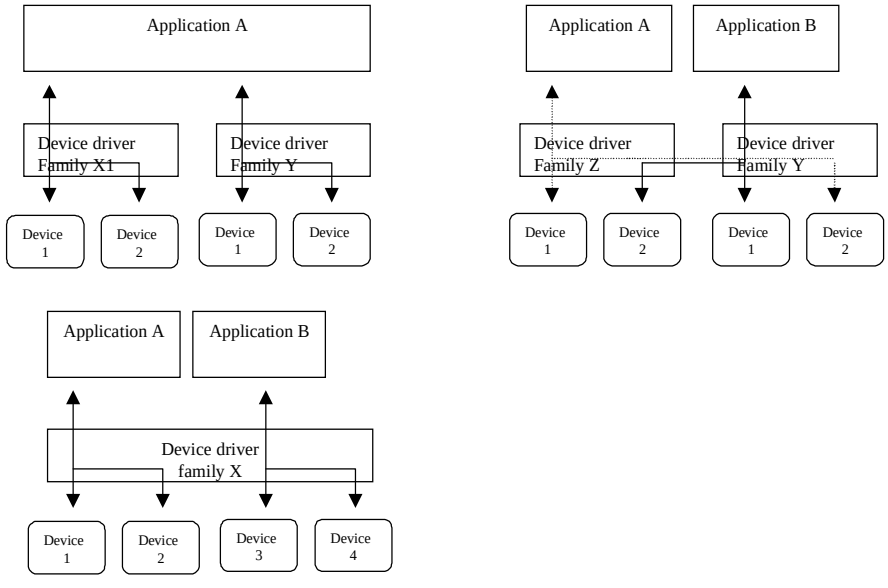



**Figure 45 OPI multi-application / multi-device access**

## 5.5 Device driver dynamic link

The application is linked dynamically to the target device driver with the bind procedure provided by the OPI interface.

The following figure shows the bind procedure which links the application to the device driver.

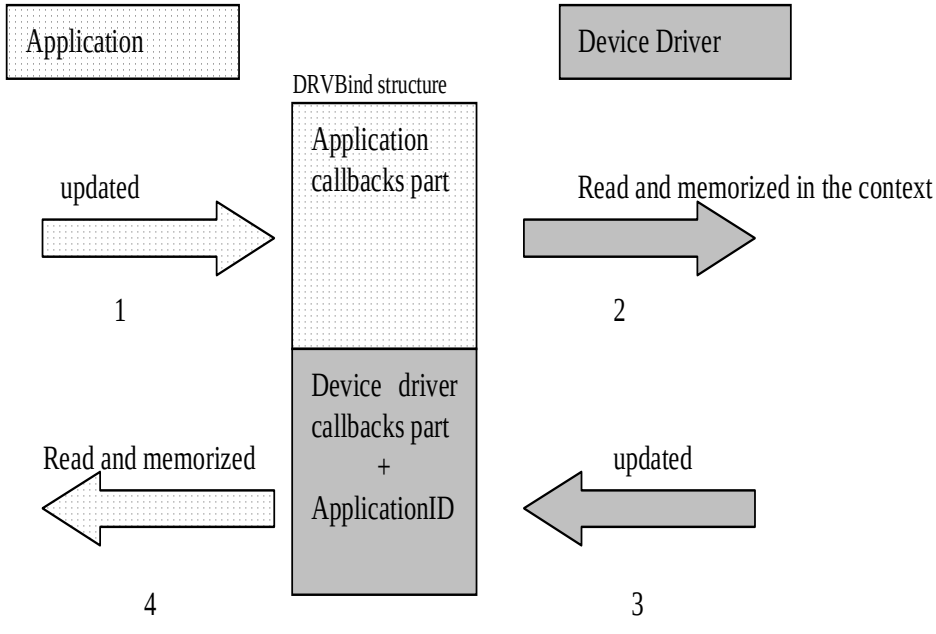



**Figure 46 OPI - Bind procedure**

## 5.6 Dialogue principles

Application and Device Driver communicate through a set of callbacks which have been exchanged during the bind procedure. The application get access to the device driver by using the device driver callbacks. The device driver sends the events to the application by calling the application callbacks. The interface dialogue between the application and the device driver is defined by the unique Application ID allocated by the device driver to the application at the bind procedure.

The following figures summarize the dialogue principle at the Application/device driver interface.

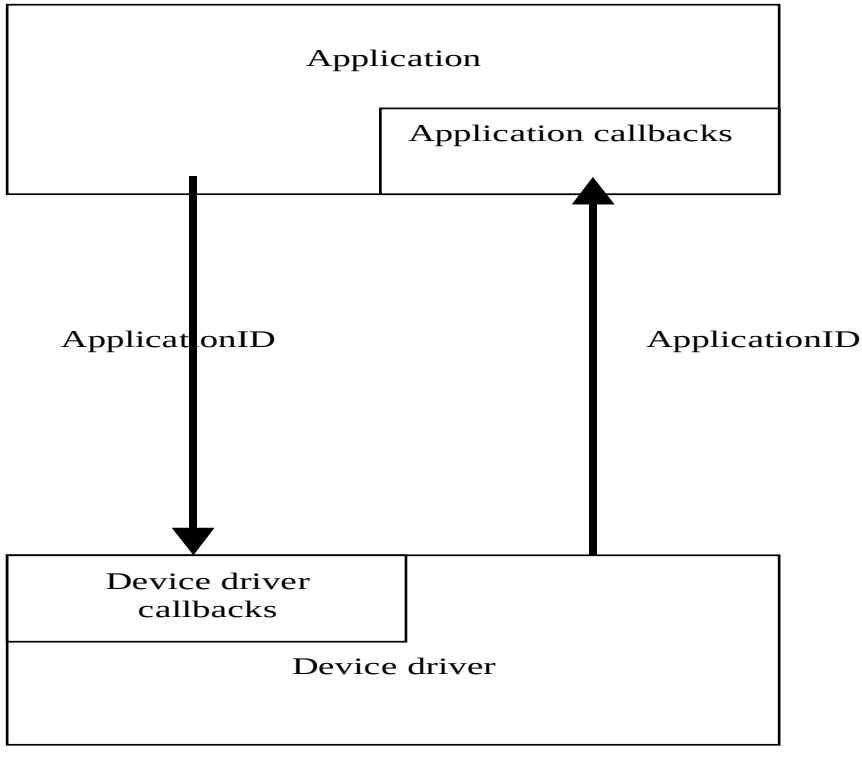



**Figure 47 OPI - Dialogue principle**

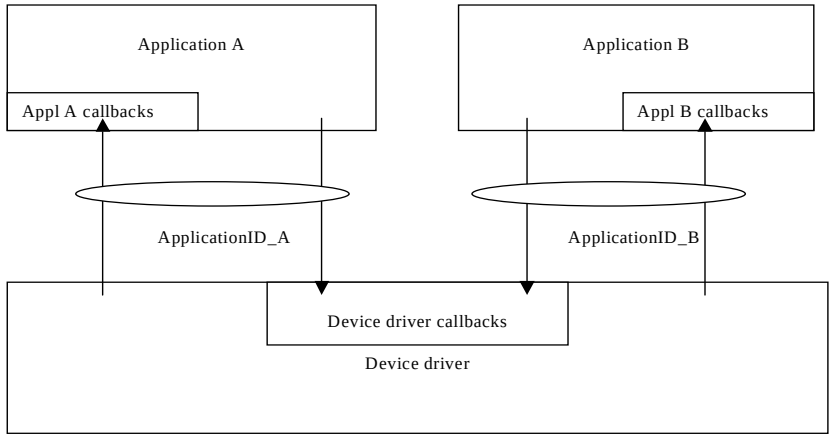



**Figure 48 OPI - Application ID**

## 5.7 Port and channel

At the OPI interface the device is accessed through the generic port and channel notions. The port and channel has to be mapped according to the device characteristics.

The following figures shows how the applications are mapped with port and channel.

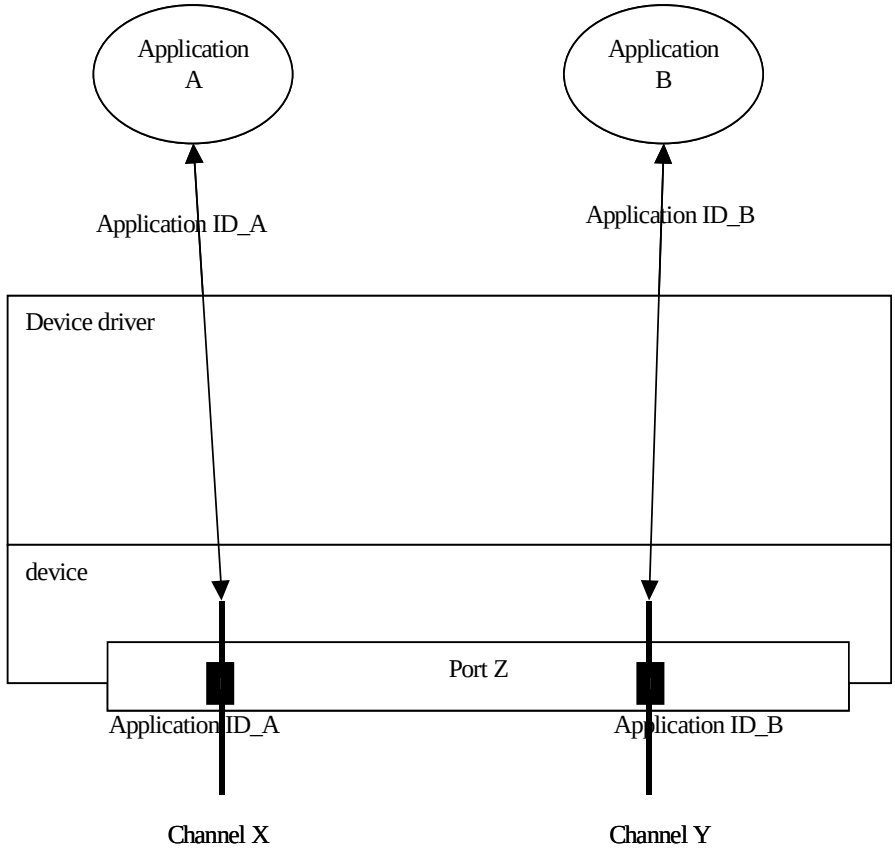



**Figure 49 OPI - Application Mapping**

## 5.8 OPI Low Level Interface

The following figures shows the OPI low-level interface.

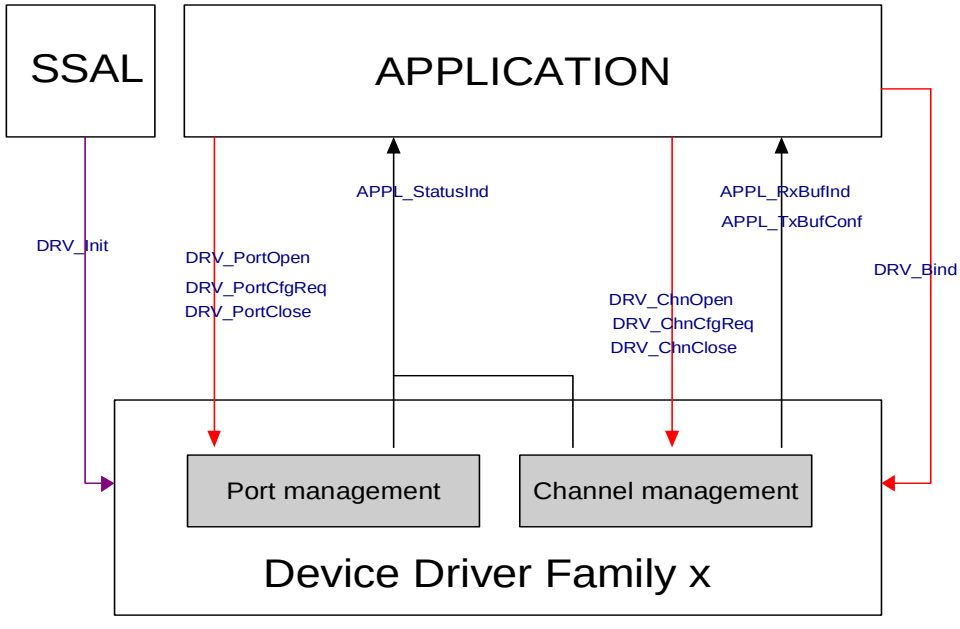



**Figure 50 OPI Low-level interface**

# 6 WinEASY for EASY3445

WinEASY is the graphical user interface running on the PC, that is used to download the code on the target board, and to exchange Command/Indication messages between the PC and the target board.

For more details, please refer to the WinEASY V2.4 Tool Description (provided on the CD-ROM).

# 7 QuadLIU C/I interface

The QuadLIU C/I interface is a message-based interface according to the DDS software specifications.

The **Source**, **Destination** and **Entity** IDs of the C/I message must be updated according to the values listed in the following table.

**Table 33 QuadLIU C/I header message parameters**

| C/I header parameters | Source | Destination | Entity |
|---|---|---|---|
| Command message (C) | USER (0) | QLIU (4) | Line number (0 to 7) |
| Indication message (I) | QLIU (4) | USER (0) | Line number (0 to 7) |

The command and indication messages listed in the following table can be used from any external PC application program or from any internal application program module in order to handle the QUADLIU device in the scope of the EASY3445 board.

**Table 34 QuadLIU C/I messages**

| Message | Direction | Message ID |
|---|---|---|
| ***device activation*** | | |
| QLIU_PARAMETERS | C | 20 |
| QLIU_PARAMETERS_SET | C | 21 |
| QLIU_ALARM_SET | C | 27 |
| QLIU_LOOPBACK_SET | C | 29 |
| QLIU_ERROR | I | 9 |

## 7.1 QLIU_PARAMETERS command

**Table 35 QLIU_PARAMETERS command**

| | |
|---|---|
| **Dest** | QLIU |
| **Src** | USER |
| **Entity** | Line number |
| **ID** | **20** |
| **DSize** | 8 |
| **DPtr** | |

| | |
|---|---|
| 0 | voltage |
| 1 | mode |
| 2 | line_coding |
| 3 | clock |
| 4 | jitter_attenuation |
| 5 | line_build_out |
| 6 | prbs_monitor |
| 7 | prbs_generator |

The **QLIU_PARAMETERS** message updates the QuadLIU device configuration parameters according to the parameter values passed in the data area. The configuration will be active when the **QLIU_PARAMETERS_SET** command will be issued. An indication error message **QLIU_ERROR** is generated by the target if the command is erroneous.

The following table summarizes the meaning of each quadliu parameters.

**Table 36 QuadLIU configuration parameters**

| Field | value | Description |
|---|---|---|
| voltage | 0,1 | Select the power supply voltage.<br>0: 3.3 volts<br>1: 5.0 volts |
| mode | 0,1,2 | Select the channel mode.<br>0: T1 mode<br>1: E1 mode<br>2: J1 mode |
| line_coding | 0,1,2 | Select the line coding.<br>0: AMI coding<br>1: B8ZS coding<br>2: HDB3 coding |
| clock | 0,1 | Master mode selection:<br>0: slave mode.<br>1: master mode. the DCO circuitry is frequency synchronized with the clock (2.048 MHz, 1.544 MHz or 8 kHz) supplied by SYNC. If this pin is connected to VSS or VDD the DCO circuitry is centered and no receive jitter attenuation is performed. The generated clocks are stable. |
| jitter_attenuation | 0,1,2 | Jitter attenuation selection:<br>0: No jitter attenuation. The elastic buffer is disabled.<br>1: Reception Jitter attenuation. The elastic buffer is placed on the receive path.<br>2: Transmit Jitter attenuation. The elastic buffer is placed on the transmit path. |
| line_build_out | 0,1,2,3 | Line build-out selection:<br>0: 0 db attenuation.<br>1: -7 db attenuation.<br>2: -15 db attenuation.<br>3: -22.5 db attenuation. |
| prbs_monitor | 0 | Reserved.<br>must be set to 0. |
| prbs_generator | 0 | Reserved.<br>must be set to 0. |

## 7.2 QLIU_PARAMATERS_SET command

**Table 37 QLIU_PARAMETERS_SET command**

| | |
|---|---|
| **Dest** | QLIU |
| **Src** | USER |
| **Entity** | line number |
| **ID** | **21** |
| **DSize** | 0 |
| **DPtr** | |

The **QLIU_CFG_PARAMETERS_SET** message validates the QUADLIU device configuration parameters according to the configuration parameter set passed with **QLIU_CFG_PARAMETERS** message.

## 7.3 QLIU_ALARM_SET command

**Table 38 QLIU_ALARM_SET command**

| | |
|---|---|
| **Dest** | QLIU |
| **Src** | USER |
| **Entity** | Line number |
| **ID** | **27** |
| **DSize** | 1 |
| **DPtr** | |

| | |
|---|---|
| 0 | ais |

The **QLIU_ALARM_SET** message allows to start/stop the DS1 AIS alarm. An indication error message **QLIU_ERROR** is generated by the target if the command is erroneous.

The following table summarizes the description of the alarm parameters.

**Table 39 Alarm command parameters.**

| Field | value | Description |
|---|---|---|
| ais | 0,1 | AIS alarm generation:<br>0: stop alarm generation.<br>1: start alarm generation. |

## 7.4 QLIU_LOOPBACK_SET command

**Table 40 QLIU_LOOPBACK_SET command**

| | |
|---|---|
| **Dest** | QLIU |
| **Src** | USER |
| **Entity** | Line number |
| **ID** | **27** |
| **DSize** | 3 |
| **DPtr** | |

| | |
|---|---|
| 0 | remote |
| 1 | local |
| 2 | digital |

The **QLIU_LOOPBACK_SET** message allows to set/reset a loopback. An indication error message **QLIU_ERROR** is generated by the target if the command is erroneous.

The following table summarizes the description of the loopback parameters.

**Table 41 Alarm command parameters.**

| Field | value | Description |
|---|---|---|
| remote | 0,1 | Remote loopback selection:<br>0: Disable remote loopback.<br>1: Enable remote loopback. The remote loopback mode disconnects the transmit data received at XDIP/N from the transmitter. Received data at pins RL1/2 are looped back to the line interface with or without jitter attenuation. The decoder and encoder are ignored. |
| local | 0,1 | Local loopback selection:<br>0: Disable local loopback.<br>1: Enable local loopback. Data received at ports XDIP/N are looped back through the analog receiver to pins RDOP/N. |
| digital | 0,1 | Digital loopback selection:<br>0: Disable digital loopback.<br>1: Enable digital loopback. Data received at ports XDIP/N are looped back to pins RDOP/N. |

## 7.5 QLIU_ERROR indication

**Table 42 QLIU_ERROR indication.**

| | |
|---|---|
| **Dest** | USER |
| **Src** | QLIU |
| **Entity** | line number |
| **ID** | **9** |
| **DSize** | 0 |
| **DPtr** | |

The **QLIU_ERR indication** message is sent to the application when an erroneous command is received from the application. The erroneous command is not processed by the target.

# 8 M13FX C/I interface

The M13FX C/I interface is a message-based interface according to the DDS software specifications.

The **Source**, **Destination** and **Entity** IDs of the C/I message must be updated according to the values listed in the following table.

**Table 43 M13FX C/I header message parameters**

| C/I header parameters | Source | Destination | Entity |
|---|---|---|---|
| Command message (C) | USER (0) | APPLI (5) | 0 |
| Indication message (I) | APPLI (5) | USER (0) | 0 |

The command and indication messages listed in the following table can be used from any external PC application program or from any internal application program module in order to handle the M13FX device.

**Table 44 M13FX C/I messages**

| Message | Direction | Message ID |
|---|---|---|
| ***device activation*** | | |
| DEV_STATUS | C-I | 10 |
| DEV_ACTIVATION | C - I | 11 |
| DEV_DEACTIVATION | C - I | 12 |
| ***device configuration*** | | |
| CFG_PARAMETERS_SET | C - I | 20 |
| CFG_PARAMETERS_RESET | C - I | 21 |
| CFG_PARAMETERS_DS3 | C - I | 22 |
| CFG_PARAMETERS_DS2 | C - I | 23 |
| CFG_PARAMETERS_DS1 | C - I | 24 |
| CFG_PARAMETERS_FEAC | C - I | 25 |
| CFG_PARAMETERS_MDL | C - I | 26 |
| CFG_PARAMETERS_MISC | C - I | 27 |
| CFG_PARAMETERS_DS3_GET | C - I | 28 |
| CFG_PARAMETERS_DS2_GET | C - I | 29 |

**Table 44 M13FX C/I messages**

| Message | Direction | Message ID |
|---|---|---|
| CFG_PARAMETERS_DS1_GET | C - I | 30 |
| CFG_PARAMETERS_FEAC_GET | C - I | 31 |
| CFG_PARAMETERS_MDL_GET | C - I | 32 |
| CFG_PARAMETERS_MISC_GET | C - I | 33 |
| ***Feac Channel*** | | |
| FEAC_STATUS | C - I | 60 |
| FEAC_RECEIVE_INDICATION | I | 61 |
| FEAC_ACTIVATION | C - I | 62 |
| FEAC_DEACTIVATION | C - I | 63 |
| FEAC_SEND | C - I | 64 |
| FEAC_RSEND | C - I | 65 |
| ***Mdl Channel*** | | |
| MDL_STATUS | C - I | 70 |
| MDL_RECEIVE_INDICATION | I | 71 |
| MDL_ACTIVATION | C - I | 72 |
| MDL_DEACTIVATION | C - I | 73 |
| MDL_SEND | C - I | 74 |
| ***DS3 Alarms*** | | |
| ALARM_DS3_STATUS | C - I | 80 |
| ALARM_DS3_INDICATION | I | 81 |
| ALARM_DS3_AIS_SET | C - I | 82 |
| ALARM_DS3_AIS_RESET | C - I | 83 |
| ALARM_DS3_RA_SET | C - I | 84 |
| ALARM_DS3_RA_RESET | C - I | 85 |
| ALARM_DS3_IDLE_SET | C - I | 86 |
| ALARM_DS3_IDLE_RESET | C - I | 87 |
| ***DS2 Alarms*** | | |
| ALARM_DS2_STATUS | C - I | 90 |
| ALARM_DS2_INDICATION | I | 91 |
| ALARM_DS2_AIS_SET | C - I | 92 |

**Table 44 M13FX C/I messages**

| Message | Direction | Message ID |
|---|---|---|
| ALARM_DS2_AIS_RESET | C - I | 93 |
| ALARM_DS2_RA_SET | C - I | 94 |
| ALARM_DS2_RA_RESET | C - I | 95 |
| ***DS1 Alarms*** | | |
| ALARM_DS1_STATUS | C - I | 100 |
| ALARM_DS1_T1AIS_SET | C - I | 101 |
| ALARM_DS1_T1AIS_RESET | C - I | 102 |
| ALARM_DS1_T13AIS_SET | C - I | 103 |
| ALARM_DS1_T13AIS_RESET | C - I | 104 |
| ***DS3 Loopbacks*** | | |
| LOOPBACK_DS3_STATUS | C - I | 110 |
| LOOPBACK_DS3_REMOTE_SET | C - I | 111 |
| LOOPBACK_DS3_REMOTE_RESET | C - I | 112 |
| LOOPBACK_DS3_LOCAL_SET | C - I | 113 |
| LOOPBACK_DS3_LOCAL_RESET | C - I | 114 |
| ***DS2 Loopbacks*** | | |
| LOOPBACK_DS2_STATUS | C - I | 120 |
| LOOPBACK_DS2_INDICATION | I | 121 |
| LOOPBACK_DS2_REMOTE_SET | C - I | 122 |
| LOOPBACK_DS2_REMOTE_RESET | C - I | 123 |
| LOOPBACK_DS2_LPCREQ_SET | C - I | 124 |
| LOOPBACK_DS2_LPCREQ_RESET | C - I | 125 |
| ***DS1 Loopbacks*** | | |
| LOOPBACK_DS1_STATUS | C - I | 130 |
| LOOPBACK_DS1_REMOTE_SET | C - I | 131 |
| LOOPBACK_DS1_REMOTE_RESET | C - I | 132 |
| LOOPBACK_DS1_LOCAL_SET | C - I | 133 |
| LOOPBACK_DS1_LOCAL_RESET | C - I | 134 |
| LOOPBACK_DS1_LPCREQ_SET | C - I | 135 |
| LOOPBACK_DS1_LPCREQ_RESET | C - I | 136 |

**Table 44 M13FX C/I messages**

| Message | Direction | Message ID |
|---|---|---|
| LOOPBACK_DS1_INDICATION | I | 138 |
| ***Error Counters*** | | |
| ERROR_COUNTERS_CMD | C - I | 160 |
| ERROR_COUNTERS_INDICATION | I | 161 |
| ***Test Unit*** | | |
| TU_INDICATION | I | 201 |
| TU_GENERATOR_SET | C - I | 202 |
| TU_GENERATOR_START | C - I | 203 |
| TU_GENERATOR_STOP | C - I | 204 |
| TU_GENERATOR_TEST_POINT | C - I | 205 |
| TU_GENERATOR_ERROR_INSERTION | C - I | 206 |
| TU_GENERATOR_RECEIVE_PATTERN | C - I | 207 |
| TU_GENERATOR_RECEIVE_COUNTER | C - I | 208 |
| TU_FRAMER_SET | C - I | 209 |
| TU_FRAMER_START | C - I | 210 |
| TU_FRAMER_STOP | C - I | 211 |
| TU_FRAMER_ERROR_INSERTION | C - I | 212 |
| TU_FRAMER_RECEIVE_COUNTER | C - I | 213 |
| ***Mapper*** | | |
| MAP_CFG_SET | C - I | 300 |
| MAP_CFG_RESET | C - I | 301 |
| MAP_CFG_GET | C - I | 302 |
| ***Debug message display*** | | |
| DBG_DISPLAY | C- I | 500 |
| ***Error indication*** | | |
| ERROR_INDICATION | I | 1000 |

## 8.1 M13FX device commands

The M13FX device command allows the application to activate and access to the M13FX device. At the startup system, the target software activates automatically the M13FX device with the default parameters. Once the device is activated, the M13FX functions are then accessible (configuration, mapper, test unit, feac and mdl channel, loopbacks, alarms,...).

The device commands permit:

- to get the current device status,
- to activate the device. The M13FX is then accessible and the DS3 line interface is active.
- to deactivate the device. The M13FX is no longer accessible and the DS3 line interface is not operational (Blue alarm generation).

### 8.1.1 M13FX device status

**Table 45 DEV_STATUS command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **10** |
| **DSize** | 0 |
| **DPtr** | |

This command reads the current device status and returns it in the **DEV_STATUS** message indication sent back to the application. This command provides an overview of the device status.

**Table 46 DEV_STATUS indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **10** |
| **DSize** | 26 |
| **DPtr** | |

**Table 46 DEV_STATUS indication**

| | |
|---|---|
| 0-3 | port |
| 4-7 | base |
| 8 | ds3_clock |
| 9 | ds3_error_counters |
| 10 | ds3_framing |
| 11 | ds3_multiplexer |
| 12 | ds3_aic |
| 13 | ds3_feac |
| 14 | ds3_mdl |
| 15 | ds3_alarms |
| 16 | ds3_loopbacks |
| 17 | ds2_mode |
| 18 | ds2_alarms |
| 19 | ds2_loopbacks |
| 20 | ds1_alarms |
| 21 | ds1_loopbacks |
| 22 | tu.generator |
| 23 | tu.framer |
| 24 | map_rx |
| 25 | map_tx |

This message indication is sent to the PC-based application in response of the **DEV_STATUS** command. It contains the current device status.

The following table shows the meaning of each field in the device status indication message.

**Table 47 DEVICE STATUS indication field**

| Field | value | Description |
|---|---|---|
| port | 0 to 65535 | device number |
| base | 32 bits | M13FX base address |

**Table 47 DEVICE STATUS indication field**

| Field | value | Description |
|---|---|---|
| ds3_clock | 0,1 | DS3 clock mode<br>0, Normal mode<br>1, loop timing mode |
| ds3_error_counters | 0,1 | Error counters mode<br>0, Polling mode<br>1, One second interrupt mode |
| ds3_framing | 0,1 | DS3 framing format<br>0, M13 format<br>1, C-bit parity format |
| ds3_multiplexer | 2,3 | DS3 Multiplexer mode<br>2, channelized<br>3, full payload |
| ds3_aic | bitmap<br>bit 0: generation<br>bit 1:detection | DS3 AIC-bit generation and detection<br>bit set to 1: detection /generation |
| ds3_feac | 0,1 | DS3 FEAC channel status<br>0, channel OFF<br>1, channel ON |
| ds3_mdl | 0,1 | DS3 MDL channel status<br>0, channel OFF<br>1, channel ON |
| ds3_alarms | bitmap<br>bit 0: generation<br>bit 1: detection | DS3 alarms generation and detection<br>bit set to 1: detection /generation |
| ds3_loopbacks | bitmap<br>bit 0: remote loopback<br>bit 1: local loopback | DS3 loopback status<br>bit set to 1: SET |
| ds2_mode | bitmap<br>bit 0: DS2 tributary 1<br>bit 6: DS2 tributary 7 | DS2 tributary mode<br>bit set to 0: T1 mode<br>bit set to 1: E1 mode |
| ds2_alarms | bitmap<br>bit 0: generation<br>bit 1: detection | DS2 tributary alarm generation and detection (all DS2s)<br>bit set to 1: detection /generation |

**Table 47 DEVICE STATUS indication field**

| Field | value | Description |
|---|---|---|
| ds2_loopbacks | bitmap<br>bit 0: remote loopback<br>bit 1: not used<br>bit 2: loopcode generation<br>bit 3: loopcode detection | DS2 tributary loopback status (all DS2s)<br>bit set to 1: SET loopback - loopcode generation/detection |
| ds1_alarms | bitmap<br>bit 0: generation toward DS3<br>bit 1: detection toward DS1 | DS1 tributary AIS alarms generation (all DS1s)<br>bit set to 1: detection /generation |
| ds1_loopbacks | bitmap<br>bit 0: remote loopback<br>bit 1: local loopback<br>bit 2: loopcode generation<br>bit 3: loopcode detection | DS1 tributary loopback status (all DS1s)<br>bit set to 1: SET loopback - loopcode generation/detection |
| tu_generator | 0,1,2 | Test Unit generator status<br>0, generator OFF<br>1, generator in FIXED-PATTERN<br>2, generator in PRBS |
| tu_framer | 0,1,2 | Test Unit framer status<br>0, framer OFF<br>1, framer in E1 mode<br>2, framer in T1 mode |
| map_rx | 0,1 | Mapper receive side status<br>0, straight line<br>1, almost 1 line is mapped |
| map_tx | 0,1 | mapper transmit side status<br>0, straight line<br>1, almost 1 line is mapped |

### 8.1.2 M13FX device activation

**Table 48 DEV_ACTIVATION command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **11** |
| **DSize** | 0 |
| **DPtr** | |

This message activates the M13FX device with the current configuration parameters set. The interrupts are turned on and the Blue Alarm is released at the DS3 interface. The device is then operational. An indication message **DEV_ACTIVATION** is sent back to the PC-based application which acknowledges the command.

### 8.1.3 M13FX device deactivation

**Table 49 DEV_DEACTIVATION command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **12** |
| **DSize** | 0 |
| **DPtr** | |

This function deactivates the M13FX device. The interrupts are turn off and the Blue Alarm is generated at the DS3 interface. The device is non operational.
An indication message **DEV_DEACTIVATION** is sent back to the PC-based application that acknowledges the command.

## 8.2 M13FX Device configuration

The M13FX Device configuration parameters set messages are divided in the following groups:

- DS3 parameters,
- DS2 parameters,
- DS1 parameters,
- FEAC parameters,
- MDL parameters,
- MISC parameters.

For each commands group the following functions are available:

- Update parameters group (**CFG_PARAMETERS_xxx**),
- Read parameters group (**CFG_PARAMETERS_xxx_GET**).

For all parameters groups, the following functions are available:

- Validate parameters (**CFG_PARAMETERS_SET**) except for MDL and FEAC parameters set groups which are validated at the channel activation.
- Re initialize parameters (**CFG_PARAMETERS_RESET**).

### 8.2.1 Update DS3 Configuration Parameters

**Table 50 CFG_PARAMETERS_DS3 command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **22** |
| **DSize** | 21 |
| **DPtr** | |

| | |
|---|---|
| 0 | c_bit_parity |
| 1 | full_payload |
| 2 | loop_timing_mode |

**Table 50 CFG_PARAMETERS_DS3 command**

| | |
|---|---|
| 3 | invert_clock_rx |
| 4 | invert_clock_tx |
| 5 | invert_data_rx |
| 6 | invert_data_tx |
| 7 | unipolar_mode_rx |
| 8 | unipolar_mode_tx |
| 9 | tovhsyn_input |
| 10 | b3zs_code_polarity |
| 11 | interrupt_open_drain |
| 12 | interrupt_active_high |
| 13 | febe_error_parity |
| 14 | check_x_bit |
| 15 | multiframe_framing_two_m_bits |
| 16 | multiframe_reframing |
| 17 | f_framing_sixteen_f_bits |
| 18 | ais_detection_eight_err_per_multiframe |
| 19 | ais_framed |
| 20 | alarm_cv |

This message updates the M13FX device DS3 configuration parameters according to the parameter choice (YES/NO) passed in the parameter choice structure. This configuration will be active when the SetConfiguration command will be issued.

The following table summarizes the meaning of each ds3 parameters.

**Table 51 DS3 configuration parameters**

| parameter | YES | NO |
|---|---|---|
| **c_bit_parity:** Enable the C-bit parity or M13 asynchronous mode | **C-bit parity** mode | **M13** mode |

**Table 51 DS3 configuration parameters**

| | | |
|---|---|---|
| **full_payload:** enables the M23 multiplex operation or the full payload rate format. | **Full payload** rate format. The payload is one single, high speed data stream without stuffing available in DS2 tributary number 0. | **M23** multiplex operation. Payload is formed by interleaving 7 asynchronous DS2 tributaries. |
| **loop_timing_mode:** DS3 looped timing where the transmitter uses the receiver DS3 input clock. | enable loop timing | disable loop timing |
| **invert_clock_rx:** Sets the clock edge for data sampling. | Update data on the falling edge of RTC(x). | Update data on the rising edge of RTC(x). |
| **invert_clock_tx:** selects the clock edge for data transmission. | sample transmit data on the falling edge of transmit clock. | sample transmit data on the rising edge of transmit clock. |
| **invert_data_rx:** enables inversion of receive data. | Invert data provided via RTD(x). | No inversion of data provided via RTD(x). |
| **invert_data_tx:** enables inversion of transmit data. | Transmit data is logic low (inverted). | Transmit data is logic high (not inverted). |
| **unipolar_mode_rx:** sets the port mode to dual-rail mode or unipolar mode for the receive data | Unipolar mode (single rail data input). | B3ZS (dual rail data input). |
| **unipolar_mode_tx:** sets the port mode to dual-rail mode or unipolar mode for the transmit data | Unipolar mode (single rail data input). | B3ZS (dual rail data input). |

**Table 51 DS3 configuration parameters**

| | | |
|---|---|---|
| **tovhsyn_input:** switches between input mode and output mode of the signal pin TOVHSYN. If TOVHSYN is operated in input mode it marks the position of the X-bit. Therefore the outgoing DS3 frame is aligned to TOVHSYN. If TOVHSYN is switched to output mode TOVHSYN is asserted when the X-bit needs to be inserted via the transmit overhead interface. | TOVHSYN switched to input. | TOVHSYN switched to output. |
| **b3zs_code_polarity:** selects the B3ZS violations alternate polarity to maintain line balance (“00V” or “10V” Acceptance Condition). | Convert codeword only if alternate violation polarity rule is satisfied. | Convert all B3ZS codeword patterns to “000” regardless of polarity. |
| **interrupt_open_drain:** selects the operating mode of the interrupt pin. | Open Drain. | Push-Pull. |
| **interrupt_active_high:** selects the active level of the interrupt pin. | Active High. | Active Low. |
| **febe_error_parity:** selects the event which leads to FEBE indication. It is available in C-bit parity mode only. | Receive multiframe parity error or framing error. | Receive multiframe parity error. |
| **check_x_bit:** Enable or disable checking of the X-bit for AIS and idle detection. | Check X-bit. | Disable check of X-bit. |
| **multiframe_framing_two_m_bits:** Selects the M-bit error condition which triggers the DS3 framer to start a new frame search. | Start new F-frame search if M-bit errors are detected in two out of four consecutive M-frames. | Start new F-frame search if M-bit errors are detected in three out of four consecutive M-frames. |

**Table 51 DS3 configuration parameters**

| | | |
|---|---|---|
| **multiframe_reframing:** Enable or disable the reframing due to M-bit errors. | Enable reframe due to M-bit errors. | Disable reframe due to M-bit errors. |
| **f_framing_sixteen_f_bits:** selects the F-bit error condition which triggers the DS3 framer to start a new frame search. | A new frame search is started when 3 out of 16 contiguous F-bits are in error. | A new frame search is started when 3 out of 8 contiguous F-bits are in error. |
| **ais_detection_eight_err_per_multiframe:** select the error rate for AIS detection. Declaration of AIS depends on value defined in bit field CV. | AIS is recognized when the alarm indication signal is received with less than 8 errors per multiframe. | AIS is recognized when the alarm indication signal is received with less than 15 errors per multiframe. |
| **ais_framed:** sets the AIS code. | Set AIS to ’1010... ’ between overhead bits, C-bits all ‘0’s. | Set AIS to unframed all ‘1’s (non-standard). |
| **alarm counter resolution:** Specifies the number of frames when the M13FX declares AIS, RED or Idle. | 0 to 63. | - |

## 8.2.2 Update DS2 Configuration Parameters

**Table 52 CFG_PARAMETERS_DS2 command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **23** |
| **DSize** | 9 |
| **DPtr** | |

**Table 52 CFG_PARAMETERS_DS2 command**

| | |
|---|---|
| 0 | tn |
| 1 | t1_mode |
| 2 | e1_reserved_bit_to_one |
| 3 | automatic_ais_insertion |
| 4 | multiframe_framing_three_m_bits_errors |
| 5 | f_framing_five_f_bits_errors |
| 6 | ais_detection_nine_err_per_multi frame |
| 7 | counter_mode_per_multiframes |
| 8 | alarm_cv |

This message updates the M13FX device DS2 configuration parameters according to the parameter choice (YES/NO) passed in the parameter choice structure. This configuration will be active when the SetConfiguration command will be issued.

The following table summarizes the meaning of each ds2 parameters.

**Table 53 DS2 configuration parameters**

| parameter | YES | NO |
|---|---|---|
| **tn:**<br>DS2 tributary number where the parameter set is applied<br>1 to 7 | | |
| **t1_mode:**<br>selects the operation mode of the low speed multiplexer. | M12 mode (4 DS1 into DS2). T1 mode. | ITU-T G.747 mode (3 E1 into DS2). E1 mode |
| **e1_reserved_bit_to_one:**<br>sets the value to be transmitted in the reserved bit of<br>ITU-T G.747 format. | Transmit reserved bit as ’1’. | Transmit reserved bit as ’0’. |

**Table 53 DS2 configuration parameters**

| | | |
|---|---|---|
| **automatic_ais_insertion:** enables automatic insertion of AIS in downstream direction if DS2 framer OR DS3 framer is out of frame. | Enable automatic insertion of AIS. | Disable automatic insertion of AIS. |
| **multiframe_framing_three_m_bits_errors:** selects the M-bit error condition which triggers the DS2 framer to start a new frame search. It is valid in DS1 mode only. | F-frame search started if 3 contiguous multiframes have M-bit errors. | Inhibit new F-frame search due to M-bit errors. |
| **f_framing_five_f_bits_errors:** selects the F-bit error condition which triggers the DS2 framer to start a new frame search. | A new frame search is started when 2 out of 5 contiguous F-bits are in error. | A new frame search is started when 2 out of 4 contiguous F-bits are in error. |
| **ais_detection_nine_err_per_multiframe:** sets the error rate for AIS detection. Declaration of AIS is specified by bits CM and CV. | AIS condition is recognized when the alarm indication signal is received with less than 9 errors in each of 2 consecutive multiframes. | AIS condition is recognized when the alarm indication signal is received with less than 5 errors in each of 2 consecutive multiframes. |

**Table 53 DS2 configuration parameters**

| | | |
|---|---|---|
| **counter_mode_per_multiframes:** selects the alarm timer mode.If the counter mode is set to multiframes (‘0’) the value in CV determines the number of multi-frames after which the M13FX declares AIS or RED. When the counter mode is set to ‘½ milliseconds’ (‘1’) the value in CV determines the time in CV x 0.5 ms after which AIS or RED is declared. | ½ Milliseconds. | Multiframes. |
| **alarm_cv:** specifies the number of frames or the time in multiples of 0.5 milliseconds when AIS or RED is declared. | 0 to 63. | - |

## 8.2.3 Update DS1 Configuration Parameters

**Table 54 CFG_PARAMETERS_DS1 Command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **24** |
| **DSize** | 2 |
| **DPtr** | |

| | |
|---|---|
| 0 | invert_tributary_clock |
| 1 | invert_tributary_data |

This message updates the M13FX device DS1 configuration parameters according to the parameter choice (YES/NO) passed in the parameter choice structure. This configuration will be active when the SetConfiguration command will be issued.

The following table summarizes the meaning of each DS1 parameters

**Table 55 DS1 configuration parameters**

| parameter | YES | NO |
|---|---|---|
| **invert_tributary_clock:** sets the clock edge for data update on the low speed tributaries. | Sample data on the rising edge of TTC(x). Update data on the falling edge of RTC(x). | Sample data on the falling edge of TTC(x). Update data on the rising edge of RTC(x). |
| **invert_tributary_data:** enables inversion of sampled tributary data. | Invert data sampled on TTD(x). Invert data provided via RTD(x). | No inversion of data sampled on TTD(x). No inversion of data provided via RTD(x). |

## 8.2.4 Update FEAC Configuration Parameters

**Table 56 CFG_PARAMETERS_FEAC command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **25** |
| **DSize** | 3 |
| **DPtr** | |

↓

| | |
|---|---|
| 0 | bom_rx_threshold |

**Table 56 CFG_PARAMETERS_FEAC command**

| | |
|---|---|
| 1 | bom_rx_filter_mode |
| 2 | bom_rx_continuous_mode |

This message updates the M13FX device FEAC configuration parameters according to the parameter choice (YES/NO) passed in the parameter choice structure. This configuration is active when the ***FEAC channel will be activated***.
The following table summarizes the meaning of each FEAC parameters.

**Table 57 FEAC configuration parameters**

| parameter | YES | NO |
|---|---|---|
| **bom_rx_threshold:** sets the threshold of the receive FIFO and is applied to both pages of the receive FIFO. A 'Receive Pool Full' interrupt vector will be generated, when the programmed threshold is reached. | 2, two bytes<br>4, four bytes<br>16, sixteen bytes<br>32, thirty two bytes | |
| **bom_rx_filter_mode:** selects that the byte oriented messages have to be filtered. The BOM is reported only if 7 out 10 data is received. | Enable BOM filter mode. | Disable BOM filter mode. |
| **bom_rx_continuous_mode:** switches between continuous and 10 byte packet reception of the receive signalling controller. In 10 byte packet mode a receive FIFO full interrupt is generated after 10 bytes. In continuous reception mode a receive message interrupt is generated when the receive FIFO threshold level is reached. | Enable continuous reception mode. | Enable 10 byte packets mode. |

### 8.2.5 Update MDL Configuration Parameters

**Table 58 CFG_PARAMETERS_MDL command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **26** |
| **DSize** | 9 |
| **DPtr** | |

| | |
|---|---|
| 0 | rx_threshold |
| 1 | rx_check_crc |
| 2 | rx_transfer_crc |
| 3 | tx_gen_crc |
| 4 | tx_share_flags |
| 5 | rx_interframe_filling_status |
| 6 | polarity_active_high |
| 7 | invert_data |
| 8 | dma_mode |

This message updates the M13FX device MDL configuration parameters according to the parameter choice (YES/NO) passed in the parameter choice structure. This configuration is active when the ***MDL channel will be activated***.
The following table summarizes the meaning of each MDL parameters.

**Table 59 MDL configuration parameters**

| parameter | YES | NO |
|---|---|---|
| **rx_threshold:** sets the threshold of the receive FIFO and is applied to both pages of the receive FIFO. A 'Receive Pool Full' interrupt vector will be generated, when the programmed threshold is reached. | 2, two bytes<br>4, four bytes<br>16, sixteen bytes<br>32, thirty two bytes | |
| **rx_check_crc:** enables or disables the CRC check of incoming data packets. | Enable CRC check. | Disable CRC check. |
| **rx_transfer_crc:** selects that CRC of incoming data packets shall be transferred to the receive FIFO or not. | Transfer of CRC to RFIFO. | No transfer of CRC to RFIFO. |
| **tx_gen_crc:** enables CRC generation and transmission on transmission of HDLC packets. | Enable CRC generation. | Disable CRC generation. |
| **tx_share_flags:** enables transmission of protocol data with shared flags. | Enable shared flags. | Disable shared flags. |
| **rx_interframe_filling_status:** selects, that interframe time-fill changes should be reported. | Enable IFF status messages. | Disable IFF status messages. |
| **polarity_active_high:** sets the polarity of $\overline{\text{RMC}}$,$\overline{\text{DRR}}$, $\overline{\text{TXME}}$ and $\overline{\text{DRT}}$ signals. | polarity to active high. | polarity to active low. |

**Table 59 MDL configuration parameters**

| | | |
|---|---|---|
| **invert_data:**<br>Invert data input/output from Receive/Transmit Framer. | Enable data inversion. | Disable data Inversion. |
| **dma_mode:**<br>enables the DMA functionality of the C-bit parity Path Maintenance Data Link transmitter.<br>Reception:<br>While DMA is active new data is indicated by an asserted DRR signal. The end of a message is indicated by an asserted RMC signal.<br>Transmission:<br>While DMA is active a data request is indicated by an asserted DRT signal. The end of a message is indicated by the user with an asserted TXME signal. | Enable DMA. | Disable DMA. |

## 8.2.6 Update MISC Configuration Parameters

**Table 60 CFG_PARAMETERS_MISC command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **27** |
| **DSize** | 3 |
| **DPtr** | |

| | |
|---|---|
| 0 | error_counter_one_second |

**Table 60 CFG_PARAMETERS_MISC command**

| | |
|---|---|
| 1 | ds2_loopcode |
| 2 | ds1_loopcode |

This message updates the M13FX device MISC configuration parameters according to the parameter choice (YES/NO) passed in the parameter choice structure. This configuration will be active when the SetConfiguration command will be issued.

The following table summarizes the meaning of each MISC parameters.

**Table 61 MISC configuration parameters**

| parameter | YES | NO |
|---|---|---|
| **error_counter_one_second:** Select the error counters mode (polling or one second interrupt) | One second interrupt mode. The error counter values are copied to the foreground register in one second intervals. At the same time the background registers are reset to zero. This operation is synchronous with the periodic one second interrupt which alerts software to read the register. | Polling mode. The error counter values are copied to foreground when copy command is executed. The one second interrupt is disabled. |

**Table 61 MISC configuration parameters**

| | | |
|---|---|---|
| **ds2_loopcode:**<br>Select the C-bit which will be inverted when loopback requests are transmitted in the DS3 framing in order to request a DS2 payload loopback. | **0**: invert 1st C-bit.<br>**1**: invert 2nd C-bit.<br>**2**: invert 3rd C-bit. | - |
| **ds1_loopcode:**<br>Select the C-bit which will be inverted when loopback requests are transmitted in the DS2 framing in order to request a DS1 payload loopback. | **0**: invert 1st C-bit.<br>**1**: invert 2nd C-bit.<br>**2**: invert 3rd C-bit. | - |

### 8.2.7 Read DS3 Configuration Parameters

**Table 62 CFG_PARAMETERS_DS3_GET command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **28** |
| **DSize** | 0 |
| **DPtr** | |

This command reads the current DS3 parameters configuration of the M13FX and returns the parameters in the **CFG_PARAMETERS_DS3_GET** message indication sent back to the PC-based application.

**Table 63 CFG_PARAMETERS_DS3_GET indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |

**Table 63 CFG_PARAMETERS_DS3_GET indication**

| | |
|---|---|
| **Entity** | 0 |
| **ID** | **28** |
| **DSize** | 21 |
| **DPtr** | |

| | |
|---|---|
| 0 | c_bit_parity |
| 1 | full_payload |
| 2 | loop_timing_mode |
| 3 | invert_clock_rx |
| 4 | invert_clock_tx |
| 5 | invert_data_rx |
| 6 | invert_data_tx |
| 7 | unipolar_mode_rx |
| 8 | unipolar_mode_tx |
| 9 | tovhsyn_input |
| 10 | b3zs_code_polarity |
| 11 | interrupt_open_drain |
| 12 | interrupt_active_high |
| 13 | febe_error_parity |
| 14 | check_x_bit |
| 15 | multiframe_framing_two_m_bits |
| 16 | multiframe_reframing |
| 17 | f_framing_sixteen_f_bits |
| 18 | ais_detection_eight_err_per_multiframe |
| 19 | ais_framed |
| 20 | alarm_cv |

This indication message is sent to the PC-application in response of the **CFG_PARAMETERS_DS3_GET** command. It contains the current DS3 configuration parameters.

### 8.2.8 Read DS2 Configuration Parameters

**Table 64 CFG_PARAMETERS_DS2_GET command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **29** |
| **DSize** | 0 |
| **DPtr** | |

This command reads the current DS2 parameters configuration of the M13FX device and returns the parameters in the **CFG_PARAMETERS_DS2_GET** message indication sent back to the application.

**Table 65 CFG_PARAMETERS_DS2_GET indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **29** |
| **DSize** | 9 |
| **DPtr** | |

| | |
|---|---|
| 0 | tn |
| 1 | t1_mode |
| 2 | e1_reserved_bit_to_one |
| 3 | automatic_ais_insertion |
| 4 | multiframe_framing_three_m_bits_errors |

**Table 65 CFG_PARAMETERS_DS2_GET indication**

| | |
|---|---|
| 5 | f_framing_five_f_bits_errors |
| 6 | ais_detection_nine_err_per_multi frame |
| 7 | counter_mode_per_multiframes |
| 8 | alarm_cv |

This indication message is sent to the PC-based application in response of the **CFG_PARAMETERS_DS2_GET** command. It contains the current DS2 configuration parameters.

### 8.2.9 Update DS1 Configuration Parameters

**Table 66 CFG_PARAMETERS_DS1_GET command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **30** |
| **DSize** | 0 |
| **DPtr** | |

This command reads the current DS1 parameters configuration of the M13FX device and returns the parameters in the **CFG_PARAMETERS_DS1_GET** message indication sent back to the application.

**Table 67 CFG_PARAMETERS_DS1_GET indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **30** |

**Table 67 CFG_PARAMETERS_DS1_GET indication**

| | |
|---|---|
| **DSize** | 2 |
| **DPtr** | |

| | |
|---|---|
| 0 | invert_tributary_clock |
| 1 | invert_tributary_data |

This indication message is sent to the PC-based application in response of the **CFG_PARAMETERS_DS1_GET** command. It contains the current DS1 configuration parameters.

### 8.2.10 Read FEAC Configuration Parameters

**Table 68 CFG_PARAMETERS_FEAC_GET command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **31** |
| **DSize** | 0 |
| **DPtr** | |

This command reads the current FEAC parameters configuration of the M13FX device and returns the parameters in the **CFG_PARAMETERS_FEAC_GET** message indication sent back to the application.

**Table 69 CFG_PARAMETERS_FEAC_GET indication**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |

**Table 69 CFG_PARAMETERS_FEAC_GET indication**

| | |
|---|---|
| **ID** | **31** |
| **DSize** | 3 |
| **DPtr** | |

| | |
|---|---|
| 0 | bom_rx_threshold |
| 1 | bom_rx_filter_mode |
| 2 | bom_rx_continuous_mode |

This message indication is sent to the PC-based application in response of the **CFG_PARAMETERS_FEAC_GET** command. It contains the current FEAC configuration parameters.

### 8.2.11 Read MDL Configuration Parameters

**Table 70 CFG_PARAMETERS_MDL_GET command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **32** |
| **DSize** | 0 |
| **DPtr** | |

This command reads the current MDL parameters configuration of the M13FX device and returns the parameters in the **CFG_PARAMETERS_MDL_GET** message indication sent back to the application.

**Table 71 CFG_PARAMETERS_MDL_GET indication**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **32** |
| **DSize** | 9 |
| **DPtr** | |

| | |
|---|---|
| 0 | rx_threshold |
| 1 | rx_check_crc |
| 2 | rx_transfer_crc |
| 3 | tx_gen_crc |
| 4 | tx_share_flags |
| 5 | rx_interframe_filling_status |
| 6 | polarity_active_high |
| 7 | invert_data |
| 8 | dma_mode |

This message indication is sent to the PC-based application in response of the **CFG_PARAMETERS_MDL_GET** command. It contains the current MDL configuration parameters.

### 8.2.12 Read MISC Configuration Parameters

**Table 72 CFG_PARAMETERS_MISC_GET command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **33** |
| **DSize** | 0 |

**Table 72 CFG_PARAMETERS_MISC_GET command**

| DPtr |
|---|

This command reads the current MISC parameters configuration of the M13FX device and returns the parameters in the **CFG_PARAMETERS_MISC_GET** message indication sent back to the application.

**Table 73 CFG_PARAMETERS_MISC_GET indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **33** |
| **DSize** | 3 |
| **DPtr** | |

| | |
|---|---|
| 0 | error_counter_one_second |
| 1 | ds2_loopcode |
| 2 | ds1_loopcode |

This message indication is sent to the PC-based application in response of the **CFG_PARAMETERS_MISC_GET** command. It contains the current MISC configuration parameters.

### 8.2.13 Configuration Parameters Validation

**Table 74 CFG_PARAMETERS_SET command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |

**Table 74 CFG_PARAMETERS_SET command**

| | |
|---|---|
| **Entity** | 0 |
| **ID** | **20** |
| **DSize** | 0 |
| **DPtr** | |

This command validates the current configuration parameters groups updates (**CFG_PARAMETERS_xxx**). The M13FX device is programmed with the parameters set. The **CFG_PARAMETERS_SET** indication message is sent back to the PC-based application acknowledging the command.

The FEAC and MDL parameters are validated when the respective channel is activated.

### 8.2.14 Configuration Parameters Reinitialization

**Table 75 CFG_PARAMETERS_RESET command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **21** |
| **DSize** | 0 |
| **DPtr** | |

This command initializes the M13FX device configuration with the default parameters values. The M13FX device is programmed with the default parameters set. The **CFG_PARAMETERS_RESET** indication message is sent back to the PC-based application acknowledging the command.

The following table shows the default parameters values for each configuration group when RESET is applied or when the device driver is initialized..

**Table 76 M13FX configuration default values**

| parameter | value | mode |
|---|---|---|
| ***DS3 parameters:*** | | |
| **c_bit_parity:** Enable the C-bit parity or M13 asynchronous mode | **YES** | **C-bit parity** mode. |
| **full_payload:** enables the M23 multiplex operation or the full payload rate format. | **NO** | **M23** multiplex operation. |
| **loop_timing_mode:** DS3 looped timing where the transmitter uses the receiver DS3 input clock. | **NO** | **disable** loop timing. |
| **invert_clock_rx:** Sets the clock edge for data sampling. | **NO** | Update data on the **rising edge** of RTC(x). |
| **invert_clock_tx:** selects the clock edge for data transmission. | **NO** | Sample transmit data on the **rising edge** of transmit clock. |
| **invert_data_rx:** enables inversion of receive data. | **NO** | **No inversion** of data provided via RTD(x). |
| **invert_data_tx:** enables inversion of transmit data. | **NO** | Transmit data is **logic high** (not inverted). |
| **unipolar_mode_rx:** sets the port mode to dual-rail mode or unipolar mode for the receive data | **NO** | **B3ZS** (dual rail data input). |
| **unipolar_mode_tx:** sets the port mode to dual-rail mode or unipolar mode for the transmit data | **NO** | **B3ZS** (dual rail data input). |

**Table 76 M13FX configuration default values**

| | | |
|---|---|---|
| **tovhsyn_input:** switches between input mode and output mode of the signal pin TOVHSYN. If TOVHSYN is operated in input mode it marks the position of the X-bit. Therefore the outgoing DS3 frame is aligned to TOVHSYN. If TOVHSYN is switched to output mode TOVHSYN is asserted when the X-bit needs to be inserted via the transmit overhead interface. | **NO** | TOVHSYN switched to **output**. |
| **b3zs_code_polarity:** selects the B3ZS violations alternate polarity to maintain line balance (“00V” or “10V” Acceptance Condition). | **YES** | Convert codeword only if **alternate violation polarity** rule is satisfied. |
| **interrupt_open_drain:** selects the operating mode of the interrupt pin. | **YES** | **Open Drain**. |
| **interrupt_active_high:** selects the active level of the interrupt pin. | **NO** | **Active Low**. |
| **febe_error_parity:** selects the event which leads to FEBE indication. It is available in C-bit parity mode only. | **YES** | Receive multiframe **parity error** or **framing error**. |
| **check_x_bit:** Enable or disable checking of the X-bit for AIS and idle detection. | **YES** | **Check** X-bit. |
| **multiframe_framing_two_m_bits:** Selects the M-bit error condition which triggers the DS3 framer to start a new frame search. | **YES** | Start new F-frame search if M-bit errors are detected in **two out of four consecutive M-frames**. |

**Table 76 M13FX configuration default values**

| | | |
|---|---|---|
| **multiframe_reframing:** Enable or disable the reframing due to M-bit errors. | **YES** | **Enable** reframe due to M-bit errors. |
| **f_framing_sixteen_f_bits:** selects the F-bit error condition which triggers the DS3 framer to start a new frame search. | **YES** | A new frame search is started when **3 out of 16 contiguous F-bits** are in error. |
| **ais_detection_eight_err_per_multiframe:** select the error rate for AIS detection. Declaration of AIS depends on value defined in bit field CV. | **YES** | AIS is recognized when the alarm indication signal is received with l**ess than 8 errors per multiframe**. |
| **ais_framed:** sets the AIS code. | **YES** | Set **AIS to '1010... '** between overhead bits, C-bits all '0's. |
| **alarm counter resolution:** Specifies the number of frames when the M13FX declares AIS, RED or Idle. | **0** | **One** frame . |
| ***DS2 parameters:*** | | |
| **t1_mode:** selects the operation mode of the low speed multiplexer. | **YES** | M12 mode (4 DS1 into DS2). **T1** mode. |
| **e1_reserved_bit_to_one:** sets the value to be transmitted in the reserved bit of ITU-T G.747 format. | **YES** | not relevant in T1 mode |
| **automatic_ais_insertion:** enables automatic insertion of AIS in downstream direction if DS2 framer OR DS3 framer is out of frame. | **YES** | **Enable** automatic insertion of AIS. |

**Table 76 M13FX configuration default values**

| | | |
|---|---|---|
| **multiframe_framing_three_m_bits_errors:** selects the M-bit error condition which triggers the DS2 framer to start a new frame search. It is valid in DS1 mode only. | **YES** | F-frame search started if **3 contiguous multi-frames have M-bit errors**. |
| **f_framing_five_f_bits_errors:** selects the F-bit error condition which triggers the DS2 framer to start a new frame search. | **YES** | A new frame search is started when **2 out of 5 contiguous F-bits are in error**. |
| **ais_detection_nine_err_per_multiframe:** sets the error rate for AIS detection. Declaration of AIS is specified by bits CM and CV. | **YES** | AIS condition is recognized when the alarm indication signal is received with **less than 9 errors in each of 2 consecutive multi-frames**. |
| **counter_mode_per_multiframes:** selects the alarm timer mode.If the counter mode is set to multiframes (‘0’) the value in CV determines the number of multi-frames after which the M13FX declares AIS or RED. When the counter mode is set to ‘½ milliseconds’ (‘1’) the value in CV determines the time in CV x 0.5 ms after which AIS or RED is declared. | **YES** | **½ Milliseconds**. |
| **alarm_cv:** specifies the number of frames or the time in multiples of 0.5 milliseconds when AIS or RED is declared. | **0** | **One** frame. |
| ***DS1 parameters:*** | | |

**Table 76 M13FX configuration default values**

| | | |
|---|---|---|
| **invert_tributary_clock:** sets the clock edge for data update on the low speed tributaries. | **NO** | Sample data on the **falling edge** of TTC(x). Update data on the rising edge of RTC(x). |
| **invert_tributary_data:** enables inversion of sampled tributary data. | **NO** | **No inversion** of data sampled on TTD(x). **No inversion** of data provided via RTD(x). |
| ***FEAC channel parameters:*** | | |
| **bom_rx_threshold:** sets the threshold of the receive FIFO and is applied to both pages of the receive FIFO. A 'Receive Pool Full' interrupt vector will be generated, when the programmed threshold is reached. | **32** | **thirty two** bytes. |
| **bom_rx_filter_mode:** selects that the byte oriented messages have to be filtered. The BOM is reported only if 7 out 10 data is received. | **YES** | **Enable** BOM filter mode. |
| **bom_rx_continuous_mode:** switches between continuous and 10 byte packet reception of the receive signalling controller. In 10 byte packet mode a receive FIFO full interrupt is generated after 10 bytes. In continuous reception mode a receive message interrupt is generated when the receive FIFO threshold level is reached. | **YES** | **Enable** continuous reception mode. |
| ***MDL channel parameters:*** | | |

**Table 76 M13FX configuration default values**

| | | |
|---|---|---|
| **rx_threshold:** sets the threshold of the receive FIFO and is applied to both pages of the receive FIFO. A 'Receive Pool Full' interrupt vector will be generated, when the programmed threshold is reached. | **32** | **Thirty- two** bytes. |
| **rx_check_crc:** enables or disables the CRC check of incoming data packets. | **YES** | **Enable** CRC check. |
| **rx_transfer_crc:** selects that CRC of incoming data packets shall be transferred to the receive FIFO or not. | **NO** | **No transfer of CRC** to RFIFO. |
| **tx_gen_crc:** enables CRC generation and transmission on transmission of HDLC packets. | **YES** | **Enable** CRC generation. |
| **tx_share_flags:** enables transmission of protocol data with shared flags. | **NO** | **Disable** shared flags. |
| **rx_interframe_filling_status:** selects, that interframe time-fill changes should be reported. | **NO** | **Disable** IFF status messages. |
| **polarity_active_high:** sets the polarity of $\overline{\text{RMC}}$,$\overline{\text{DRR}}$, $\overline{\text{TXME}}$ and $\overline{\text{DRT}}$ signals. | **YES** | Polarity to active **high**. |
| **invert_data:** Invert data input/output from Receive/Transmit Framer. | **YES** | **Enable** data inversion. |

**Table 76 M13FX configuration default values**

| | | |
|---|---|---|
| **dma_mode:**<br>enables the DMA functionality of the C-bit parity Path Maintenance Data Link transmitter.<br>Reception:<br>While DMA is active new data is indicated by an asserted $\overline{\text{DRR}}$ signal. The end of a message is indicated by an asserted $\overline{\text{RMC}}$ signal.<br>Transmission:<br>While DMA is active a data request is indicated by an asserted $\overline{\text{DRT}}$ signal. The end of a message is indicated by the user with an asserted $\overline{\text{TXME}}$ signal. | **NO** | **Disable** DMA. |
| ***MISC parameters:*** | | |
| **error_counter_one_second:**<br>Select the error counters mode (polling or one second interrupt). | **YES** | **One second interrupt** mode. |
| **ds2_loopcode:**<br>Select the C-bit which will be inverted when loopback requests are transmitted in the DS3 framing in order to request a DS2 payload loopback. | **0** | **Invert 1st** C-bit. |
| **ds1_loopcode:**<br>Select the C-bit which will be inverted when loopback requests are transmitted in the DS2 framing in order to request a DS1 payload loopback. | **0** | **Invert 1st** C-bit. |

## 8.3 M13FX DS3 ALARMS commands

The DS3 alarms set of commands allows to handle the M13FX alarms at the DS3 line interface which permits:

- to generate by setting or resetting the alarms at the DS3 interface (local alarms),
- to detect the remote DS3 alarms status changes (remote alarms),
- to monitor at any time the current status of the DS3 alarms (local and remote alarms).

Each **SET** or **RESET** command is acknowledged by the target by sending an indication to the PC-based application which has the same message code.

For the **STATUS** command, the target sends back to the application a status indication providing the complete status of the DS3 alarms.

The target sends to the application a DS3 alarm indication when a remote DS3 alarm status change is detected from the DS3 line interface. The alarm status is a bit map and the application has to memorize the status changes in order to recover the status changes.

### 8.3.1 DS3 SET/RESET commands

The following tables shows the message format for the SET / RESET DS3 alarms commands. These command do not need any parameters, the target acknowledges sending back an indication with the same ID code.

**Table 77 ALARM_DS3_xxx_xxx command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | ds3 alarm code |
| **DSize** | 0 |
| **DPtr** | |

The target sets or resets the requested local ds3 alarm generation (AIS, RA and IDLE). An indication message is sent back to the application when the command is performed.

**Table 78 ALARM_DS3_xxx_xxx indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | ds3 set/reset alarm code |
| **DSize** | 0 |
| **DPtr** | |

The target sends to the application the set/reset ds3 alarm indication message in answer to the command message. The application can check with the status command the result of the command execution.

The following table lists the ds3 set/reset available codes.

**Table 79 DS3 set/reset alarm codes**

| ds3 alarm code | ID | description |
|---|---|---|
| ALARM_DS3_AIS_SET | 82 | Start the AIS alarm generation |
| ALARM_DS3_AIS_RESET | 83 | Stop the AIS alarm generation |
| ALARM_DS3_RA_SET | 84 | Start the Remote Alarm generation |
| ALARM_DS3_RA_RESET | 85 | Stop the Remote Alarm generation |
| ALARM_DS3_IDLE_SET | 86 | Start the IDLE generation |
| ALARM_DS3_IDLE_RESET | 87 | Stop the IDLE generation |

### 8.3.2 DS3 STATUS command

The DS3 STATUS command is used in order to get at any time the current status of the local and remote ds3 alarms. It can be issued at any time as long as the M13FX device is active.

The DS3 STATUS command does not need any parameter. The target send back to the application the STATUS indication message with the current DS3 status alarms.

The following tables shows the message format for the STATUS DS3 alarms command and indication.

**Table 80 ALARM_DS3_STATUS command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **80** |
| **DSize** | 0 |
| **DPtr** | |

The target collects the current ds3 status alarms and sent it back to the application trough the ds3 status indication message.

**Table 81 ALARM_DS3_STATUS indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **80** |
| **DSize** | **15** |
| **DPtr** | |

| | |
|---|---|
| 0 | bmstat |
| 2 | ais |
| 3 | rai |
| 4 | oof |
| 5 | los |
| 6 | lfa |
| 7 | cofa |

**Table 81 ALARM_DS3_STATUS indication**

| | |
|---|---|
| 8 | idle |
| 9 | aic |
| 10 | lais |
| 11 | lrai |
| 12 | lidle |
| 13 | lxbit |
| 14 | laic |

This indication is sent to the application in answer to the ds3 status command issued by the application. It contains the current status of the remote and local ds3 alarms.

The remote alarms are presented in a bit-map format and in a ON/OFF format for each signal.

The local alarms are presented in a ON/OFF format only.

The following table shows the meaning of each field in the STATUS indication message.

**Table 82 DS3 status indication fields**

| Field | value | Description |
|---|---|---|
| bmstat | bit map | current ds3 alarm status (Remote) |
| ais | ON / OFF | ds3 AIS alarm status (Remote) |
| rai | ON / OFF | ds3 Remote Alarm alarm status (Remote) |
| oof | ON / OFF | ds3 Out Of Frame Alarm status (Remote) |
| los | ON / OFF | ds3 Loss Of Signal alarm status (Remote) |
| lfa | ON / OFF | ds3 Loss of Frame Alignment status (Remote) |
| cofa | ON / OFF | ds3 Change Of Frame Alignment status (Remote) |
| idle | ON / OFF | ds3 IDLE alarm status (Remote) |
| aic | ON / OFF | AIC-bit image (Remote) |
| lais | ON / OFF | ds3 AIS alarm generation status (local) |
| lrai | ON / OFF | ds3 RAI alarm generation status (local) |
| lidle | ON / OFF | ds3 IDLE alarm generation status (local) |
| lxbit | ON / OFF | X-bit generation status (local) |
| laic | ON / OFF | AIC-bit generation status (local) |

### 8.3.3 DS3 ALARM INDICATION message

The DS3 Alarm indication message is sent to the application each time the target detects a DS3 alarm status change from the DS3 line interface. The status is bit-map coded and presents the current state of each DS3 remote alarm.

The following table show the format of the DS3 alarm indication message.

**Table 83 ALARM_DS3_INDICATION Indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **81** |
| **DSize** | 1 |
| **DPtr** | |

| | |
|---|---|
| 0 | bmstat |

The following table shows the meaning to the DS3 bit-map status. When the bit is set to 0 the status is no or no longer existing. When the bit is set to 1 the status is existing for the concerned ds3 signal.

**Table 84 DS3 bit-map alarm status**

| Alarm Signal | Bit Number | Description |
|---|---|---|
| FAS | 0 | Frame Alignment status |
| COFA | 1 | Change Of Frame Alignment status |
| LOS | 2 | Los Of Signal signal status |
| RED | 3 | Red Alarm signal status |
| AIS | 4 | AIS signal Status |
| IDLE | 5 | Idle signal Status |
| XBIT | 6 | X-bit image |

**Table 84 DS3 bit-map alarm status**

| | | |
|---|---|---|
| AIC | 7 | AIC-bit image |
| Reserved | 8-15 | |

## 8.4 M13FX DS2 ALARMS commands

The DS2 alarms set of commands allows to handle the M13FX alarms at the DS2 framer interface for each tributary (1 to 7) which permits:

- to generate by setting or resetting the alarms at the DS2 tributary interface (local alarms),
- to detect the remote DS2 tributary alarms status changes (remote alarms),
- to monitor at any time the current status of the DS2 tributary alarms (local and remote alarms).

Each SET or RESET command is acknowledged by the target by sending an indication to the PC-based application which has the same message code.

For the STATUS command, the target send back to the application a status indication providing the complete status of the DS2 tributary alarms.

The target sends to the application a DS2 alarm indication when a remote DS2 alarm status change is detected from the DS2 tributary interface. The alarm status is a bit map and the application has to memorize the status changes in order to recover the status change.

### 8.4.1 DS2 SET/RESET commands

The following tables shows the message format for the SET / RESET DS2 alarms commands. The command needs the ds2 tributary number where the command is applied, the target acknowledges sending back an indication with the same ID code.

**Table 85 ALARM_DS2_xxx_xxx command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | ds2 alarm code |
| **DSize** | 1 |
| **DPtr** | |

| | |
|---|---|
| 0 | tn (1 to 7) |

The target sets or resets the requested local ds2 alarm generation (**AIS** and **RA**) in the selected ds2 tributary. An indication message is sent back to the application when the command is performed.

**Table 86 ALARM_DS2_xxx_xxx indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | ds2 alarm code |
| **DSize** | 1 |
| **DPtr** | |

| | |
|---|---|
| 0 | tn (1 to 7) |

The target sends to the application the set/reset ds2 alarm indication message in answer to the command message. The application can checks with the status command the result of the command performance.

The following table lists the ds2 set/reset available codes.

**Table 87 DS2 set/reset alarm codes**

| ds2 alarm code | ID | description |
|---|---|---|
| ALARM_DS2_AIS_SET | 92 | Start the **AIS** alarm generation |
| ALARM_DS2_AIS_RESET | 93 | Stop the **AIS** alarm generation |
| ALARM_DS2_RA_SET | 94 | Start the **Remote Alarm** generation |
| ALARM_DS2_RA_RESET | 95 | Stop the **Remote Alarm** generation |

### 8.4.2 DS2 STATUS command

The DS2 STATUS command is used in order to get at any time the current status of the local and remote DS2 tributary alarms. It can be issued at any time as long as the M13FX device is active.

The DS2 STATUS command needs the tributary number parameter. The target sends back to the application the STATUS indication message with the current DS2 status alarms for the requested ds2 tributary.

The following tables shows the message format for the STATUS DS2 alarms command and indication.

**Table 88 ALARM_DS2_STATUS command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **90** |
| **DSize** | 1 |
| **DPtr** | |

| | |
|---|---|
| 0 | tn (1 to 7) |

The target collects the current ds2 tributary status alarms and sent it back to the application trough the ds2 status indication message.

**Table 89 ALARM_DS2_STATUS indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **90** |
| **DSize** | 11 |
| **DPtr** | |

**Table 89 ALARM_DS2_STATUS indication**

| | |
|---|---|
| 0 | tn (1 to 7) |
| 1 | bmstat |
| 2 | ais |
| 3 | rai |
| 4 | oof |
| 5 | los |
| 6 | lfa |
| 7 | cofa |
| 8 | lais |
| 9 | lrai |
| 10 | mode |

This indication is sent to the application in answer to the ds2 status command issued by the application. It contains the current status of the remote and local ds2 tributary alarms. For convenience it provides the DS2 tributary framer mode (E1/T1).

The remote alarms are presented in a bit-map format and in a ON/OFF format for each signal. The local alarms are presented in a ON/OFF format only.

The following table shows the meaning of each field in the STATUS indication message.

**Table 90 DS2 status indication fields**

| Field | value | Description |
|---|---|---|
| tn | 1.7 | ds2 tributary number |
| bmstat | bit map | ds2 alarms status (Remote) |
| ais | ON / OFF | ds2 AIS alarm status (Remote) |
| rai | ON / OFF | ds2 Remote Alarm alarm status (Remote) |
| oof | ON / OFF | ds2 Out Of Frame Alarm status (Remote) |
| los | ON / OFF | ds2 Loss Of Signal alarm status (Remote) |
| lfa | ON / OFF | ds2 Loss of Frame Alignment status (Remote) |
| cofa | ON / OFF | ds2 Change Of Frame Alignment status (Remote) |
| lais | ON / OFF | ds2 AIS alarm generation status (local) |
| lrai | ON / OFF | ds2 **RAI** alarm generation status (local) |
| mode | E1/T1 | ds2 tributary current mode |

### 8.4.3 DS2 ALARM INDICATION message

The DS2 Alarm indication message is sent to the application each time the target detects a DS2 tributary alarm status change from the DS2 tributary interface. The status is bit-map coded and presents the current state of each DS2 tributary remote alarm.

The following table show the format of the DS2 alarm indication message.

**Table 91 ALARM_DS2_INDICATION indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **91** |
| **DSize** | 2 |
| **DPtr** | |

| | |
|---|---|
| 0 | tn (1 to 7) |
| 1 | bmstat |

The following table shows the meaning of the DS2 bit-map status. When the bit is set to 0 the status is no or no longer existing. When the bit is set to 1 the status is existing for the concerned ds2 tributary signal.

**Table 92 DS2 tributary bit-map alarm status**

| Alarm Signal | Bit Number | Description |
|---|---|---|
| FAS | 0 | Frame Alignment status |
| COFA | 1 | Change Of Frame Alignment status |
| RAS | 2 | Remote Alarm status |
| RES | 3 | Reserved bit status (E1 mode) |
| RED | 4 | Red Alarm signal status |

**Table 92 DS2 tributary bit-map alarm status**

| | | |
|---|---|---|
| AIS | 5 | AIS signal Status |
| not used | 6- 7 | |

## 8.5 M13FX DS1 ALARMS commands

The DS1 ALARMS commands set allows to generate at the DS2 framer interface the T1 AIS alarm on each 28 DS1 channels multiplexed on the DS3 interface. The application can generate toward the DS3 line interface or toward the DS1 line interface using the SET/RESET commands. The application can monitor the current setting by issuing the STATUS command.

### 8.5.1 DS1 SET/RESET commands

The following tables show the message format for the SET / RESET DS1 alarms commands. The command needs the DS1channel number where the command is applied, the target acknowledges sending back an indication with the same ID code.

**Table 93 ALARM_DS1_xxx_xxx command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | DS1 alarm code |
| **DSize** | 1 |
| **DPtr** | |

| | |
|---|---|
| 0 | DS1 channel number (1 to 28) |

The target sets or resets the requested local DS1 alarm generation (T1 AIS toward DS1 interface or T1 AIS toward DS3 interface) in the selected DS1 channel number. An indication message is sent back to the application when the command is performed.

**Table 94 ALARM_DS1_xxx_xxx indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | DS1 alarm code |
| **DSize** | 1 |

**Table 94 ALARM_DS1_xxx_xxx indication**

| DPtr |
|---|

| | |
|---|---|
| 0 | DS1 channel number (1 to 28) |

The target sends to the application the set/reset DS1 alarm indication message in answer to the command message. The application can checks with the status command the result of the command performance.

The following table lists the DS1 set/reset available codes.

**Table 95 DS1 set/reset alarm codes**

| DS1 alarm code | ID | description |
|---|---|---|
| ALARM_DS1_T1AIS_SET | 101 | Start the **T1 AIS** alarm generation toward the **DS1** line interface |
| ALARM_DS1_T1AIS_RESET | 102 | Stop the **T1 AIS** alarm generation toward the **DS1** line interface |
| ALARM_DS1_T13AIS_SET | 103 | Start the **T1 AIS** alarm generation toward the **DS3** line interface |
| ALARM_DS1_T13AIS_RESET | 104 | Stop the **T1 AIS** alarm generation toward the **DS3** line interface |

## 8.5.2 DS1 STATUS command

The DS1 STATUS command is used in order to get at any time the current status of the local DS1channel alarms. It can be issued at any time as long as the **M13FX** device is active.

The DS1 STATUS command does not need parameter. The target returns the current setting for the T1 AIS alarms on each 28 DS1channels in a bit-map format.

**Table 96 ALARM_DS1_STATUS command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **100** |

**Table 96 ALARM_DS1_STATUS command**

| | |
|---|---|
| **DSize** | 0 |
| **DPtr** | |

The target collects the current DS1 channel setting and sent it back to the application through the DS1 status indication message.

**Table 97 ALARM_DS1_STATUS indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **100** |
| **DSize** | 8 |
| **DPtr** | |

| | |
|---|---|
| 0-1 | lh1ais |
| 2-3 | ll1ais |
| 4-5 | lh3ais |
| 6-7 | ll3ais |

This indication is sent to the application in answer to the DS1 status command issued by the application. It contains the current setting of the local DS1 AIS alarms for both line interfaces DS1 and DS3.

The AIS alarms status are presented in a bit-map format

The following table shows the meaning of each field in the **STATUS** indication message.

**Table 98 DS1 status indication fields**

| Field | value | Description |
|---|---|---|
| lh1ais | bitmap<br>bit 0: channel 17<br>bit 11: channel 28 | DS1 channel 28 - 17 (toward DS1 interface) |
| ll1ais | bit map<br>bit 0: channel 1<br>bit 15: channel 16 | DS1 channel 16 - 1 (toward DS1 interface) |
| lh3ais | bitmap<br>bit 0: channel 17<br>bit 11: channel 28 | DS1 channel 28 - 17 (toward DS3 interface) |
| ll3ais | bitmap<br>bit 0: channel 1<br>bit 15: channel 16 | DS1 channel 16 - 1 (toward DS3 interface) |

## 8.6 M13FX DS3 LOOPBACK commands

The DS3 LOOPBACK commands allow the application to handle the **M13FX** loopback at the DS3 line interface which permit:

- to set or reset a loopback at the DS3 interface (remote or local loopback),
- to get at any time the current status of the DS3 loopbacks (local and remote).

Each SET or RESET command is acknowledged by the target by sending an indication to the PC-based application which has the same message code.

For the STATUS command, the target send back to the application a status indication providing the complete status of the DS3 loopbacks.

### 8.6.1 DS3 LOOPBACK SET/RESET commands

The following tables show the message format for the **SET** / **RESET** DS3 loopback commands. These commands do not need any parameters, the target acknowledges sending back an indication with the same ID code.

**Table 99 LOOPBACK_DS3_xxx_xxx command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | ds3 set/reset loopback code |
| **DSize** | 0 |
| **DPtr** | |

The target sets or resets the requested ds3 loopback (REMOTE, LOCAL). An indication message is sent back to the application when the command is performed.

**Table 100 LOOPBACK_DS3_xxx_xxx indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | ds3 set/reset loopback code |
| **DSize** | 0 |
| **DPtr** | |

The target sends to the application the set/reset ds3 loopback indication message in answer to the command message. The application can checks with the status command the result of the command execution.

The following table lists the ds3 set/reset available codes.

**Table 101 DS3 set/reset loopback codes**

| ds3 loopback code | ID | description |
|---|---|---|
| LOOPBACK_DS3_REMOTE_SET | 111 | Set Remote ds3 loopback |
| LOOPBACK_DS3_REMOTE_RESET | 112 | Reset Remote ds3 loopback |
| LOOPBACK_DS3_LOCAL_SET | 113 | Set Local ds3 loopback |
| LOOPBACK_DS3_LOCAL_RESET | 114 | Reset Local ds3 loopback |

### 8.6.2 DS3 LOOPBACK STATUS command

The DS3 STATUS command is used in order to get at any time the current status of the local and remote ds3 loopback. It can be issued at any time as long as the M13FX device is active.

The DS3 STATUS command does not need any parameter. The target send back to the application the STATUS indication message with the current DS3 status loopbacks.

The following tables shows the message format for the **STATUS** DS3 loopback command and indication.

**Table 102 LOOPBACK_DS3_STATUS command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **110** |
| **DSize** | 0 |
| **DPtr** | |

The target collects the current ds3 status alarms and sent it back to the application trough the ds3 status indication message.

**Table 103 LOOPBACK_DS3_STATUS indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **110** |
| **DSize** | **2** |
| **DPtr** | |

| | |
|---|---|
| 0 | remote |
| 1 | local |

This indication is sent to the application in answer to the ds3 status command issued by the application. It contains the current status of the remote and local ds3 loopback.

The loopback status are presented in a ON/OFF format.

The following table shows the meaning of each field in the STATUS indication message.

**Table 104 DS3 loopback status indication fields**

| Field | value | Description |
|---|---|---|
| remote | ON / OFF | ds3 remote loopback status |
| local | ON / OFF | ds3 local loopback status |

## 8.7 M13FX DS2 LOOPBACK commands

The DS2 LOOPBACK commands allow the application to handle the **M13FX** loopback at the DS2 tributary interface which permit:

- to set or reset a loopback at the DS2 tributary interface (remote loopback only),
- to set or reset a loopcode by manipulating the C-bit in the DS3 frame (M13 mode only) in order to request ds2 payload loopback.
- to get at any time the current status of the DS2 loopbacks (remote loopback, loopcode generation and detection).

Each **SET** or **RESET** command is acknowledged by the target by sending an indication to the PC-based application which has the same message code.

For the **STATUS** command, the target send back to the application a status indication providing the complete status of the DS2 loopback for each ds2 tributary (bitmap).

### 8.7.1 DS2 LOOPBACK SET/RESET commands

The following tables show the message format for the **SET** / **RESET** DS2 loopback commands. The command needs the ds2 tributary number where the command is applied, the target acknowledges sending back an indication with the same **ID** code.

**Table 105 LOOPBACK_DS2_xxx_xxx command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | ds2 loopback code |
| **DSize** | 1 |
| **DPtr** | |

| | |
|---|---|
| 0 | tn (1 to 7) |

The target sets or resets the requested ds2 tributary remote loopback in the selected ds2 tributary. An indication message is sent back to the application when the command is performed.

**Table 106 LOOPBACK_DS2_xxx_xxx indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | ds2 loopback code |
| **DSize** | 1 |
| **DPtr** | |

| | |
|---|---|
| 0 | tn (1 to 7) |

The target sends to the application the set/reset ds2 loopback indication message in answer to the command message. The application can checks with the status command the result of the command execution.

The following table lists the ds2 set/reset available codes.

**Table 107 DS2 set/reset loopback codes**

| ds2 alarm code | ID | description |
|---|---|---|
| LOOPBACK_DS2_REMOTE_SET | 122 | Set a DS2 tributary remote loopback |
| LOOPBACK_DS2_REMOTE_RESET | 123 | Reset a DS2 tributary remote loopback |
| LOOPBACK_DS2_LPCREQ_SET | 124 | Set a DS2 tributary loopcode request |
| LOOPBACK_DS2_LPCREQ_RESET | 125 | Reset a DS2 tributary loopcode request |

## 8.7.2 DS2 LOOPBACK STATUS command

The DS2 STATUS command is used in order to get at any time the current status of the remote ds2 tributary loopback. It can be issued at any time as long as the M13FX device is active.

The DS2 **STATUS** command needs the tributary number parameter. The target sends back to the application the **STATUS** indication message with the current DS2 status loopbacks for the requested ds2 tributary.

The following tables shows the message format for the **STATUS** DS2 alarms command and indication.

The target collects the current ds2 tributaries status loopback and sent it back to the application through the ds2 status indication message.

**Table 108 LOOPBACK_DS2_STATUS command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **120** |
| **DSize** | 0 |
| **DPtr** | |

**Table 109 LOOPBACK_DS2_STATUS indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **120** |
| **DSize** | 3 |
| **DPtr** | |

| | |
|---|---|
| 0 | remote |
| 1 | rloopc |
| 2 | lloopc |

This indication is sent to the application in answer to the ds2 status command issued by the application. It contains the current status of the remote ds2 tributaries loopback and the local and remote loopcode requests.

The remote loopback, remote and local loopcode are presented in a bit-map format.

The following table shows the meaning of each field in the STATUS indication message.

**Table 110 DS2 loopback status indication field**

| Field | value | Description |
|---|---|---|
| remote | bitmap<br>bit 0: tributary 1<br>bit 6: tributary 7 | ds2 tributaries remote loopback status<br>bit set to 0: reset<br>bit set to 1: set |
| rloopc | bitmap<br>bit 0: tributary 1<br>bit 6: tributary 7 | ds2 tributaries remote loopcode request detection status<br>bit set to 0: request not detected<br>bit set to 1: request detected |
| lloopc | bitmap<br>bit 0: tributary 1<br>bit 6: tributary 7 | ds2 tributaries local loopcode request generation status<br>bit set to 0: request not generated<br>bit set to 1: request generated |

### 8.7.3 DS2 LOOPBACK indication message

The DS2 Loopback indication message is sent to the application each time the target detects a DS2 tributary loopcode status change from the DS2 tributary interface. The status is bit-map coded and presents the current state of each DS2 tributary for the remote loop-code.

The following table show the format of the DS2 loopback indication message.

**Table 111 LOOPBACK_DS2_INDICATION indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **91** |
| **DSize** | 2 |
| **DPtr** | |

↓

| | |
|---|---|
| 1 | rloopc |

the parameter is described in the table “**DS2 loopback status indication field**” .

## 8.8 M13FX DS1 LOOPBACK commands

The DS1 LOOPBACK commands allow the application to handle the **M13FX** loopback at the DS1 tributary interface which permit:

- to set or reset a loopback at the DS1 tributary interface (remote and local),
- to set or reset a loopcode by manipulating the C-bit in the DS2 frame in order to request a DS1 payload loopback.
- to get at any time the current status of the DS1 loopbacks (remote loopback, loopcode generation and detection).

Each **SET** or **RESET** command is acknowledged by the target by sending an indication to the PC-based application which has the same message code.

For the **STATUS** command, the target send back to the application a status indication providing the complete status of the DS1 loopback for each DS1 tributary (bitmap).

### 8.8.1 DS1 LOOPBACK SET/RESET commands

The following tables shows the message format for the SET / RESET DS1 loopback commands. The command needs the DS1channel number where the command is applied, the target acknowledges sending back an indication with the same ID code.

**Table 112 LOOPBACK_DS1_xxx_xxx command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | DS1 loopback code |
| **DSize** | 1 |
| **DPtr** | |

| | |
|---|---|
| 0 | DS1 channel number (1 to 28) |

The target sets or resets the requested DS1 loopback (remote or local) in the selected DS1 channel number. An indication message is sent back to the application when the command is performed.

**Table 113 LOOPBACK_DS1_xxx_xxx indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | DS1 loopback code |
| **DSize** | 1 |
| **DPtr** | |

| | |
|---|---|
| 0 | DS1 channel number (1 to 28) |

The target sends to the application the set/reset DS1 loopback indication message in answer to the command message. The application can checks with the status command the result of the command execution.

The following table lists the DS1 set/reset available codes.

**Table 114 DS1 set/reset loopback codes**

| ds2 alarm code | ID | description |
|---|---|---|
| LOOPBACK_DS1_REMOTE_SET | 131 | set a DS1/E1 remote loopback |
| LOOPBACK_DS1_REMOTE_RESET | 132 | reset a DS1/E1 remote loopback |
| LOOPBACK_DS1_LOCAL_SET | 133 | set a DS1/E1 local loopback |
| LOOPBACK_DS1_LOCAL_RESET | 134 | reset a DS1/E1 local loopback |
| LOOPBACK_DS1_LPCREQ_SET | 135 | set a DS1 tributary loopcode request |
| LOOPBACK_DS1_LPCREQ_RESET | 136 | reset a DS1 tributary loopcode request |

### 8.8.2 DS1 LOOPBACK STATUS command

The DS1 STATUS command is used in order to get at any time the current status of the DS1/E1channel loopbacks. It can be issued at any time as long as the M13FX device is active.

The DS1 STATUS command does not need parameter. The target returns the current setting for the remote and local loopback on each 28 DS1/21E1channels in a bit-map format.

**Table 115 LOOPBACK_DS1_STATUS command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **130** |
| **DSize** | 0 |
| **DPtr** | |

The target collects the current DS1 channel setting and sent it back to the application through the DS1 status indication message.

**Table 116 LOOPBACK_DS1_STATUS indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **130** |
| **DSize** | 16 |
| **DPtr** | |

| | |
|---|---|
| 0-1 | localh |
| 2-3 | locall |
| 4-5 | remoteh |
| 6-7 | remotel |
| 8-9 | lloopch |
| 10-11 | lloopcl |
| 12-13 | rloopch |
| 14-15 | rloopcl |

This indication is sent to the application in answer to the DS1 status command issued by the application. It contains the current setting of both remote and local loopback and both remote and local loopcode request for each 28DS1/21E1 channel multiplexed in the DS3 interface.The loopback status are presented in a bit-map format.

The following table shows the meaning of each field in the STATUS indication message.

**Table 117 DS1 loopback status indication fields**

| Field | value | Description |
|---|---|---|
| localh | bitmap<br>bit 0: channel 17<br>bit 11: channel 28 | DS1/E1local loopback status, channels 28 - 17 |
| locall | bit map<br>bit 0: channel 1<br>bit 15: channel 16 | DS1/E1local loopback status, channels 16 - 1 |
| remoteh | bitmap<br>bit 0: channel 17<br>bit 11: channel 28 | DS1/E1remote loopback status, channels 28 - 16 |
| remotel | bitmap<br>bit 0: channel 1<br>bit 15: channel 16 | DS1/E1remote loopback status, channels 16 - 1 |
| rloopch | bitmap<br>bit 0: channel 17<br>bit 11: channel 28 | DS1/E1remote loopcode request status, channels 28 - 16 |
| rloopcl | bitmap<br>bit 0: channel 1<br>bit 15: channel 16 | DS1/E1remote loopcode request status, channels 16 - 1 |
| lloopch | bitmap<br>bit 0: channel 17<br>bit 11: channel 28 | DS1/E1local loopcode request status, channels 28 - 16 |
| lloopcl | bitmap<br>bit 0: channel 1<br>bit 15: channel 16 | DS1/E1local loopcode request status, channels 16 - 1 |

### 8.8.3 DS1 LOOPBACK indication message

The DS1 Loopback indication message is sent to the application each time the target detects a DS1 tributary loopcode status change from the DS1 tributary interface. The status is bit-map coded and presents the current state of the 4 DS1tributary in the DS2 tributary where a loopcode change has occurred.

The following table show the format of the DS1 loopback indication message.

**Table 118 LOOPBACK_DS1_INDICATION indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **138** |
| **DSize** | 2 |
| **DPtr** | |

| | |
|---|---|
| 0 | tn (1 to 7) |
| 1 | rloopc |

The following table shows the meaning of each field in the LOOPBACK indication message.

**Table 119 DS1 loopback indication field**

| Field | value | Description |
|---|---|---|
| tn | 1 to 7 | ds2 tributary |
| rloopc | bitmap<br>bit 0: tributary 1<br>bit 3: tributary 4 | DS1 tributaries remote loopcode request detection status<br>bit set to 0: request not detected<br>bit set to 1: request detected |

## 8.9 M13FX FEAC commands

The FEAC commands allows the application to handle the FEAC channel of the M13FX device in order to send/ receive BOM codes to/from the DS3 line interface. The FEAC commands permit:

- to get the current FEAC channel status,
- to activate or deactivate the FEAC channel,
- to send BOM codes to the DS3 line interface,
- to send a BOM code repeatedly,
- to receive BOM codes from the DS3 line interface.

The FEAC channel is activated with the FEAC configuration parameters settled at the device configuration (default or current values). The M13FX device must be activated prior to activate the FEAC channel. Once the FEAC channel is activated, the channel is ready for transmitting or receiving from the line.

### 8.9.1 FEAC channel status

**Table 120 FEAC_STATUS command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **60** |
| **DSize** | 0 |
| **DPtr** | |

This command reads the current FEAC channel status and returns it in the **FEAC_STATUS** message indication sent back to the application.

**Table 121 FEAC_STATUS indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **60** |
| **DSize** | 8 |
| **DPtr** | |

**Table 121 FEAC_STATUS indication**

| | |
|---|---|
| 0-1 | Ref |
| 2-3 | BufferSize |
| 4 | Threshold |
| 5 | FilterMode |
| 6 | ReceiveMode |
| 7 | Rsend |

This message indication is sent to the PC-based application in response of the **FEAC_STATUS** command. It contains the current FEAC channel status.

The following table shows the meaning of each field in the FEAC_STATUS indication message.

**Table 122 FEAC STATUS indication field**

| Field | value | Description |
|---|---|---|
| Ref | 0 to 65535 | Channel reference allocated by the target at the activation:<br>(-1) channel not activated<br><> (-1) channel activated |
| BufferSize | 1 to 32 | length of the data buffer reception. it is defined by compilation FEAC_BUFFER_SIZE |
| Threshold | 2,4,16,32 | receive FIFO threshold level |
| FilterMode | 0,1 | Reception filter mode<br>BOM_NORMAL_MODE (0)<br>BOM_FILTER_MODE (1) |
| ReceiveMode | 2,3 | Reception mode<br>BOM_CONTINUOUS_MODE (2)<br>BOM_10PACKETS_MODE (3) |

### 8.9.2 FEAC channel activation

**Table 123 FEAC_ACTIVATION command**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **62** |
| **DSize** | 0 |
| **DPtr** | |

This message activates the FEAC channel with the current FEAC configuration parameters settled at the device configuration. The command is accepted if the M13FX device is activated and the FEAC channel is not yet activated.

A FEAC_ACTIVATION indication message with the current channel status is sent to the application once the channel is activated and ready for sending or receiving BOMs to/ from the line interface.

**Table 124 FEAC_ACTIVATION indication**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **62** |
| **DSize** | 8 |
| **DPtr** | |

| | |
|---|---|
| 0-1 | Ref |
| 2-3 | BufferSize |
| 4 | Threshold |
| 5 | FilterMode |
| 6 | ReceiveMode |
| 7 | Rsend |

This message indication is sent to the application when the FEAC channel is activated.

It acknowledges the feac activation command and provides the current channel settings.

### 8.9.3 M13FX FEAC channel deactivation

**Table 125 FEAC_DEACTIVATION command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **63** |
| **DSize** | 0 |
| **DPtr** | |

This command deactivates the FEAC channel. The underlaying resources are released An indication message FEAC_DEACTIVATION is sent back to the PC-based application that acknowledges the command.

**Table 126 FEAC_DEACTIVATION indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **63** |
| **DSize** | 0 |
| **DPtr** | |

This message indication is sent to the application when the FEAC channel is deactivated. It acknowledges the feac deactivation command.

### 8.9.4 M13FX FEAC transmission command

The FEAC transmission command allows the application to send BOM codes in the DS3 FEAC channel. The application provides the BOM codes to transmit, the target inserts for each BOM code to transmit the BOM synchronization.

The target sends back to the application a transmission indication message once the transmission is completed (with or without errors).

The BOM code must have the following format **0xxxxxx0** otherwise it will be discarded by the BOM transmitter.

The following tables show the transmission command and indication messages.

**Table 127 FEAC_SEND command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **64** |
| **DSize** | data length |
| **DPtr** | |

| | |
|---|---|
| 0 | BOM code |
| 1 | BOM code |
| | |
| data length - 1 | BOM code |

**Table 128 FEAC_SEND Indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **64** |
| **DSize** | data length |
| **DPtr** | |

| | |
|---|---|
| 0 | BOM code |
| 1 | BOM code |
| | |
| data length - 1 | BOM code |

### 8.9.5 M13FX FEAC Repeat transmit command

The FEAC repeat transmit command allows the application to send a BOM code repeatedly in the DS3 FEAC channel until the application stop the transmission or continues with an other BOM code.

The BOM code must have the following format **0xxxxxx0** otherwise it will be discarded by the BOM transmitter.

The following tables show the repeat transmit command and indication messages.

**Table 129 FEAC_RSEND command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **65** |
| **DSize** | 2 |
| **DPtr** | |

| | |
|---|---|
| 0 | cmde |
| 1 | BOM code |

The following table shows the meaning of each field in the FEAC_RSEND command message.

**Table 130 FEAC_RSEND parameters**

| Field | value | Description |
|---|---|---|
| cmde | 0,1 | repeat transmission command<br>0 stop the BOM code transmission<br>1 start the BOM code transmission |
| bom | 0xxxxxx0 | BOM code to transmit repeatedly |

A FEAC_RSEND indication message is sent back to the application, which acknowledges the command otherwise an ERROR_INDICATION message is generated if the transmitter is busy.

The following table shows the FEAC_RSEND indication message.

**Table 131 FEAC_RSEND Indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **65** |
| **DSize** | 0 |
| **DPtr** | |

### 8.9.6 M13FX FEAC Receive indication

The target sends back to the application the BOM codes received from the DS3 Feac channel. The reception is performed according to the current FEAC configuration parameters set.

The following table shows the FEAC receive indication message sent to the application.

**Table 132 FEAC_RECEIVE Indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **61** |
| **DSize** | data length |
| **DPtr** | |

| | |
|---|---|
| 0 | BOM code |
| 1 | BOM code |
| | |
| data length - 1 | BOM code |

## 8.10 M13FX MDL commands

The MDL commands allows the application to handle the MDL channel of the M13FX device in order to send/ receive HDLC-based messages to/from the DS3 line interface. The MDL commands permit:

- to get the current MDL channel status,
- to activate or deactivate the MDL channel,
- to send HDLC-based messages to the DS3 line interface,
- to receive HDLC-based messages from the DS3 line interface.

The MDL channel is activated with the MDL configuration parameters settled at the device configuration (default or current values). The M13FX device must be activated prior to activate the MDL channel. Once the MDL channel is activated, the channel is ready for transmitting or receiving HDLC frames from the line.

### 8.10.1 MDL channel status

This command reads the current MDL channel status and returns it in the **MDL_STATUS** message indication sent back to the application.

**Table 133 MDL_STATUS command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **70** |
| **DSize** | 0 |
| **DPtr** | |

This message indication is sent to the PC-based application in response of the **MDL_STATUS** command. It contains the current MDL channel status.

**Table 134 MDL_STATUS indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **70** |
| **DSize** | 10 |
| **DPtr** | |

**Table 134 MDL_STATUS indication**

| | |
|---|---|
| 0-1 | Ref |
| 2-3 | BufferSize |
| 4 | Threshold |
| 5 | CheckCrc |
| 6 | TransferCrc |
| 7 | GenCrc |
| 8 | ShareFlag |
| 9 | Riftf |

The following table shows the meaning of each field in the MDL_STATUS indication message.

**Table 135 MDL STATUS indication field**

| Field | value | Description |
|---|---|---|
| Ref | 0 to 65535 | Channel reference allocated by the target at the activation:<br>(-1) channel not activated<br><> (-1) channel activated |
| BufferSize | 1 to 32 | length of the data buffer reception. it is defined by compilation MDL_BUFFER_SIZE |
| Threshold | 2,4,16,32 | receive FIFO threshold level |
| CheckCrc | 0,1 | Reception check CRC<br>DISABLE (0)<br>ENABLE (1) |
| TransferCrc | 0,1 | Reception transfer CRC<br>DISABLE (0)<br>ENABLE (1) |
| GenCrc | 0,1 | Transmit generation CRC<br>DISABLE (0)<br>ENABLE (1) |

**Table 135 MDL STATUS indication field**

| Field | value | Description |
|---|---|---|
| ShareFlag | 0,1 | Reception share flags<br>DISABLE (0)<br>ENABLE (1) |
| Riftf | 0,1 | Report the Interframe Time-Fill Change<br>DISABLE (0)<br>ENABLE (1) |

### 8.10.2 MDL channel activation

**Table 136 MDL_ACTIVATION command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **72** |
| **DSize** | 0 |
| **DPtr** | |

This message activates the MDL channel with the current MDL configuration parameters settled at the device configuration. The command is accepted if the M13FX device is activated and the MDL channel is not yet activated.

A MDL_ACTIVATION indication message with the current channel status is sent to the application once the channel is activated and ready for sending or receiving HDLC-based messages to/from the line interface.

**Table 137 MDL_ACTIVATION indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **72** |
| **DSize** | 10 |
| **DPtr** | |

**Table 137 MDL_ACTIVATION indication**

| | |
|---|---|
| 0-1 | Ref |
| 2-3 | BufferSize |
| 4 | Threshold |
| 5 | CheckCrc |
| 6 | TransferCrc |
| 7 | GenCrc |
| 8 | ShareFlag |
| 9 | Riftf |

This message indication is sent to the application when the MDL channel is activated. It acknowledges the MDL activation command.

### 8.10.3 M13FX MDL channel deactivation

**Table 138 MDL_DEACTIVATION command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **73** |
| **DSize** | 0 |
| **DPtr** | |

This command deactivates the MDL channel. The underlaying resources are released An indication message MDL_DEACTIVATION is sent back to the PC-based application that acknowledges the command.

**Table 139 MDL_DEACTIVATION indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **73** |

**Table 139 MDL_DEACTIVATION indication**

| | |
|---|---|
| **DSize** | 0 |
| **DPtr** | |

This message indication is sent to the application when the MDL channel is deactivated. It acknowledges the MDL deactivation command.

### 8.10.4 M13FX MDL transmission command

The MDL transmission command allows the application to send HDLC-based messages in the DS3 MDL channel. The application provides the data to send in the HDLC frame.

The target sends back to the application a transmission indication message once the transmission is completed (with or without errors).

The following tables show the transmission command and indication messages.

**Table 140 MDL_SEND command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **74** |
| **DSize** | data length |
| **DPtr** | |

| | |
|---|---|
| 0 | data |
| 1 | data |
| | |
| data length - 1 | data |

**Table 141 MDL_SEND Indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |

**Table 141 MDL_SEND Indication**

| | |
|---|---|
| **ID** | **74** |
| **DSize** | data length |
| **DPtr** | |

| | |
|---|---|
| 0 | data |
| 1 | data |
| | |
| data length - 1 | data |

## 8.10.5 M13FX MDL Receive indication

The target sends back to the application the HDLC-based messages received from the DS3 MDL channel. The reception is performed according to the current MDL configuration parameters set.

The following table shows the MDL receive indication message sent to the application.

**Table 142 MDL_RECEIVE Indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **71** |
| **DSize** | data length |
| **DPtr** | |

| | |
|---|---|
| 0 | data |
| 1 | data |
| | |
| data length - 1 | data |

## 8.11 M13FX ERROR COUNTERS commands

The ERROR_ COUNTERS_CMD and ERROR_COUNTERS_INDICATION messages allow the application to handle the M13FX error counters at the DS3 and DS2 level interface.

Through the ERROR_ COUNTERS_CMD the application can:

- Read and reset the error counters of the M13FX device updated in the context ,
- Read and clear the error counter of the M13FX device (Polling mode only),
- Read and not clear the error counters of the M13FX device (Polling mode only).

Through the ERROR_ COUNTERS_INDICATION message the application receives the error counters values synchronized with the one second interrupt (One second interrupt mode only).

Each command is acknowledged by the target by sending an indication to the PC-based application which has the same message code.

### 8.11.1 ERROR COUNTERS commands

The ERROR_COUNTERS_CMD command is used in order to get at any time the content of the error counters at the DS3 and DS2 level. The processing depends on the selected command. The error counter values are returned in the ERROR_COUNTERS_CMD message indication.

**Table 143 ERROR_COUNTERS_CMD command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | 160 |
| **DSize** | 1 |
| **DPtr** | |

| | |
|---|---|
| 0 | command selection |

The following table lists the error counter commands.

**Table 144 Error counters commands**

| command selection | value | description |
|---|---|---|
| READ_COUNTERS | 0 | Read the error counter context |
| RESET_COUNTERS | 1 | Reset the error counters context |
| GET_AND_CLEAR_COUNTERS | 2 | Read and not clear the M13FX error counters and update the context. |
| GET_AND_NOT_CLEAR_COUNTERS | 3 | Read and clear the M13FX error counters and update the context. |

The **ERROR_COUNTERS_CMD** indication provides the contents of the error counters at the DS3 and DS2 level.

**Table 145 ERROR_COUNTERS_CMD indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | 160 |
| **DSize** | 38 |
| **DPtr** | |

| | |
|---|---|
| 0-1 | rcve3 |
| 2-3 | rexz3 |
| 4-5 | rpec3 |
| 6-7 | rcpec3 |
| 8-9 | rfebec3 |
| 10-11 | rfec2 tn 1 |
| 12-13 | rpec2 tn 1 |
| 14-15 | rfec2 tn 2 |
| 16-17 | rpec2 tn 2 |
| 18-19 | rfec2 tn 3 |
| 20-21 | rpec2 tn 3 |

**Table 145 ERROR_COUNTERS_CMD indication**

| | |
|---|---|
| 22-23 | rfec2 tn 4 |
| 24-25 | rpec2 tn 4 |
| 26-27 | rfec2 tn 5 |
| 28-29 | rpec2 tn 5 |
| 30-31 | rfec2 tn 6 |
| 32-33 | rpec2 tn 6 |
| 34-35 | rfec2 tn 7 |
| 36-37 | rpec2 tn 7 |

The following table shows the meaning of each field in the **ERROR_COUNTERS_CMD** indication message.

**Table 146 ERROR_COUNTERS indication field**

| Field | value | Error Counter description |
|---|---|---|
| rcve3 | 0 to 65535 | DS3 B3ZS line code violations |
| rexz3 | 0 to 65535 | DS3 Excessive Zeroes |
| rfec3 | 0 to 65535 | DS3 Framing bit - F-bit and M-bit |
| rpec3 | 0 to 65535 | DS3 Parity Error - P-bit |
| rcpec3 | 0 to 65535 | DS3 Path Parity Error - CP-bit |
| rfebec3 | 0 to 65535 | DS3 FEBE event errors |
| rfec2 tn xx | 0 to 65535 | DS2 Framing bit - F-bit and M-bit.<br>tributary xx (1 to 7) |
| rpec2 tn xx | 0 to 65535 | DS2 Parity Error.<br>tributary xx (1 to 7) |

### 8.11.2 ERROR_COUNTERS indication message

The ERROR_COUNTER_INDICATION indication message is sent to the application when a change in the error counters occurs synchronously with the one second interrupt. The indication message is generated when the One second interrupt mode is active.

The following table shows the format of the **ERROR_COUNTERS_INDICATION** indication message. The parameters description is identical to **ERROR_COUNTERS_CMD** command message.

**Table 147 ERROR_COUNTERS_INDICATION message**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **161** |
| **DSize** | 38 |
| **DPtr** | |

| | |
|---|---|
| 0-37 | ds3 and ds2 error counters |

## 8.12 M13FX MAPPER commands

The MAPPER commands allow the application to configure the M13FX MAPPER module which maps the 28 DS1/E1plus 4 spare tributaries in the M12 stage of the DS2 framer to the 32 DS1/E1 serial interface of the T1/E1 line interface.

The following configuration commands are available:

- Set a configuration,
- Reset to the default configuration,
- Get the current configuration.

### 8.12.1 MAPPER configuration validation commands

The following tables show the message format for the SET MAP commands. the target acknowledges sending back an indication with the same ID code.

**Table 148 MAP_CFG_SET command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **300** |
| **DSize** | 4 |
| **DPtr** | |

| | |
|---|---|
| 0 | mode |
| 1 | tn |
| 2 | rsin |
| 3 | tsin |

The following table shows the meaning of each field in the STATUS indication message.

**Table 149 MAPPER Configuration parameters**

| Parameters | value | Description |
|---|---|---|
| mode | 0,<br>0xff | mapper configuration mode:<br>**0**, one mapper entry (tn,rsin,tsin) is configured with:<br>tn, the DS1/E1 tributary number<br>rsin, the receive side of the t1/e1 serial interface to map to the selected tributary (tn) in the transmit.<br>tsin, the transmit side of the t1/e1 serial interface to map to the selected tributary (tn) in the receive side.<br>**0xff**, all the 32 mapper entries (tn,rsin) are configured with:<br>tn, the starting DS1/E1 tributary number (the last will be automatically tn+28 module 28).<br>rsin, the starting t1/e1 serial interface to map to the starting tributary number (the last will be automatically rsin+28 modulo 32) in both direction receive an transmit.<br>tn is not used. |
| tn | 1 - 32 | DS1/E1 tributary number |
| rsin | 1 - 32 | Receive DS1/E1 serial interface |
| tsin | 1 - 32 | transmit DS1/E1 serial interface |

**Table 150 MAP_CFG_SET indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **300** |
| **DSize** | 0 |
| **DPtr** | |

The MAP_CFG_SET indication message is sent back to the application as an acknowledgment of the MAP_CFG_SET command message.

### 8.12.2 MAPPER configuration re initialization command

The following tables show the message format for the RESET MAP commands. the target acknowledges sending back an indication with the same ID code.

**Table 151 MAP_CFG_RESET command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **301** |
| **DSize** | 0 |
| **DPtr** | |

The MAP_CFG_RESET command re initializes the mapper module with the default values settled at the device initialization.

**Table 152 MAP_CFG_RESET indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **301** |
| **DSize** | 0 |
| **DPtr** | |

The MAP_CFG_RESET indication message is sent back to the application as an acknowledgment of the MAP_CFG_RESET command message.

### 8.12.3 MAPPER read configuration commands

The following tables shows the message format for the GET MAP command. the target provides in the indication message the mapper current configuration.

**Table 153 MAP_CFG_GET command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **302** |
| **DSize** | 0 |
| **DPtr** | |

The MAP_CFG_GET command provides the current configuration of the mapper which is sent back to the application in the MAP_CFG_GET indication message.

**Table 154 MAP_CFG_GET indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **302** |
| **DSize** | 64 |
| **DPtr** | |

| | |
|---|---|
| 0 | rsin1 |
| 1 | rsin2 |
| . | . |
| 31 | rsin32 |
| 32 | tsin1 |
| ... | ... |
| 63 | tsin32 |

with,

- rsinX (x=1,32) the receive serial interface (1 to 32) assigned to the transmit tributary number X.
- tsinX (x=1,32) the receive serial interface (1 to 32) assigned to the receive tributary number X.

## 8.13 M13FX TEST UNIT commands

The TEST UNIT commands allow the application to handle the TEST UNIT embedded in the M13FX device which permit:

- to configure the pattern generator in both modes fixed or random pattern,
- to select the test point insertion where the pattern generation will be inserted,
- to start and stop the generator with the current setting mode,
- to insert single or continuous errors in the pattern stream,
- to read the generator status and the current received pattern (fixed-pattern mode only),
- to read the generator receive error counters.
- to configure the T1/E1 framer with the relevant framing formats,
- to start and stop the framer,
- to insert framing errors,
- to read the framer receive error counters.

### 8.13.1 TEST UNIT GENERATOR configuration messages

The following tables shows the message format for the **TU_GENERATOR_SET** messages.

**Table 155 TU_GENERATOR_SET command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **202** |
| **DSize** | 8 |
| **DPtr** | |

| | |
|---|---|
| 0 | mode |
| 1 | len |
| 2 | zs_enable |
| 3 | padding |
| 4-5 | patternH |
| 6-7 | patternL |

The **TU_GENERATOR_SET** message command configures the Test unit pattern generator according to the parameter set passed in the message. The generator is configured but NOT activated. The **TU_GENERATOR_TEST_POINT** must be issued in order to select the test insertion point where the pattern will be inserted when the generation will start with the command **TU_GENERATOR_START**.

The following table shows the meaning of each field in the STATUS indication message.

**Table 156 GENERATOR Configuration parameters**

| Parameters | value | Description |
|---|---|---|
| mode | 0<br>1 | test unit pattern generator mode<br>**0**, FIXED-PATTERN generation<br>with:<br>len, fixed-pattern bit length (1 to 32 bits)<br>patternH, 16-bit pattern high<br>patternL, 16-bit pattern low<br><br>**1**, PRBS generation<br>with:<br>len, 5-bit shift register XORed with the Feedback Tap and the bit0 of the synchronizer input.<br>patternL, 5-bit feedback tap of the test unit synchronizer (receiver). The next input to the PRBS reference shift register (bit 0) is XOR of shift register bits (len) and the feedback tap.<br>zs_enable, zero suppression. |
| len | 1 - 32 | length of the fixed-pattern or 5-bit shift register (PRBS) |
| zs_enable | 0<br>1 | 0, disable the zero suppression<br>1, enable the zero suppression |
| patternH | 16 bits | 16-bit pattern HIGH |
| patternL | 16 bits | 16-bit pattern LOW in fixed- pattern mode<br>5-bit feedback TAP in PRBS mode |

The following table gives the parameter setting for the typical PRBS generation.

**Table 157 parameter values for typical PRBS patterns**

| PRBS pattern | len | zs_enable | patternL |
|---|---|---|---|
| QRSS | 0x14 | 1 | 0x0010 |
| 2^15-1 | 0x0f | 0 | 0x000d |
| 2^20-1 | 0x14 | 0 | 0x0010 |
| 2^23-1 | 0x12 | 0 | 0x0016 |

**Table 158 TU_GENERATOR_SET indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **202** |
| **DSize** | 8 |
| **DPtr** | |

| | |
|---|---|
| 0 | mode |
| 1 | len |
| 2 | zs_enable |
| 3 | void |
| 4-5 | patternH |
| 6-7 | patternL |

The **TU_GENERATOR_SET** indication message is sent back to the application as an acknowledgment to the **TU_GENERATOR_SET** command message. It returns the current generator setting.

### 8.13.2 TEST UNIT GENERATOR test point insertion messages

The following tables shows the message format for the **TU_GENERATOR_TEST_POINT** messages.

**Table 159 TU_GENERATOR_TEST_ POINT command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **205** |
| **DSize** | 2 |
| **DPtr** | |

| | |
|---|---|
| 0 | tpi |
| 1 | tn |

The **TU_GENERATOR_TEST_POINT** message command configures the Test Point Insertion according to the parameter set passed in the message. The **TU_GENERATOR_TEST_POINT** must be issued in order to select the test insertion point where the pattern will be inserted when the generation will start with the command **TU_GENERATOR_START**.

The following table shows the meaning of the parameter set.

**Table 160 TU test point insertion parameters**

| Parameters | value | Description |
|---|---|---|
| len | 1 - 32 | length of the fixed-pattern |
| zs_enable | 0<br>1 | 0, disable the zero suppression<br>1, enable the zero suppression |
| patternH | 16 bits | 16-bit pattern HIGH |
| patternL | 16 bits | 16-bit pattern LOW in fixed- pattern mode<br>5-bit feedback TAP in PRBS mode |

The following table shows the meaning of the parameter set.

**Table 161 Test Point Insertion parameters**

| parameters | value | description |
|---|---|---|
| tpi | 0, tu loopback<br>1, DS1 tributary<br>2, ds2 tributary<br>3, ds2 payload<br>4, ds3 payload | **TU LOOPBACK**: set a local loopback from the TU transmitter to the TU receiver.<br>**DS1 TRIBUTARY:** set a DS1/E1 tributary insertion point between the DS2 M12 stage and the mapper. The tributary number (tn) indicates in which DS1/E1 channel the insertion has to be applied.<br>**DS2 TRIBIUTARY:** set a DS2 tributary insertion point between the DS2 framer and the DS2 M12 stage. The tributary number (tn) indicates in which DS2 tributary the insertion has to be applied.<br>**DS2 PAYLOAD:** set a DS2 tributary payload insertion point between the DS3 M23 stage and DS2 framer. The tributary number (tn) indicates in which DS2 tributary the insertion has to be applied.<br>**DS3 PAYLOAD:** set a DS3 payload insertion point between the DS3 framer and DS3 M23 stage. |
| tn | 1-7<br>1-28 | tributary number where the generation is inserted<br>1 to 7 for DS2 tributary<br>1 to 28 for DS1/E1 channel |

**Table 162 TU_GENERATOR_TEST_ POINT indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **205** |
| **DSize** | 2 |
| **DPtr** | |

| | |
|---|---|
| 0 | tpi |
| 1 | tn |

The **TU_GENERATOR_TEST_POINT** indication message is sent back to the application as an acknowledgment to the **TU_GENERATOR_TEST_POINT** command message. It returns the current test point insertion setting.

### 8.13.3 TEST UNIT GENERATOR START/STOP messages

The following tables shows the message format for the **TU_GENERATOR_START** and **TU_GENERATOR_STOP** messages.

**Table 163 TU_GENERATOR_START command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **203** |
| **DSize** | 0 |
| **DPtr** | |

The **TU_GENERATOR_START** message command activates the pattern GENERATOR. Prior the GENARATOR has been configured with the **TU_GENERATOR_SET** command and a test point insertion selected with the **TU_GENERATOR_TEST_POINT** command. The command returns in the indication message the current setting of the GENERATOR (configuration and test point insertion).

**Table 164 TU_GENERATOR_START indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **203** |
| **DSize** | 8 |
| **DPtr** | |

| | |
|---|---|
| 0 | mode |
| 1 | len |
| 2 | zs_enable |

**Table 164 TU_GENERATOR_START indication**

| | |
|---|---|
| 3 | void |
| 4-5 | patternH |
| 6-7 | patternL |

The **TU_GENERATOR_START** indication message is sent back to the application as an acknowledgment to the **TU_GENERATOR_START** command message. It returns the current generator status.

The parameters are identical to those described in the **TU_GENERATOR_SET** command except for the “mode” parameters which indicates the current state of the generator (OFF, FIXED PATTERN or PRBS).

**Table 165 TU_GENERATOR_STOP command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **204** |
| **DSize** | 0 |
| **DPtr** | |

The **TU_GENERATOR_STOP** message command deactivates the pattern GENERATOR. The current generator setting remains unchanged.

**Table 166 TU_GENERATOR_STOP indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **204** |
| **DSize** | 0 |
| **DPtr** | |

The **TU_GENERATOR_STOP** message indication acknowledges the command message.

### 8.13.4 TEST UNIT GENERATOR error insertion messages

The following tables show the message format for the **TU_GENERATOR_ERROR_INSERTION** messages.

**Table 167 TU_GENERATOR_ERROR_INSERTION command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **206** |
| **DSize** | 1 |
| **DPtr** | |

| | |
|---|---|
| 0 | error insertion rate |

The **TU_GENERATOR_ERROR_INSERTION** message command allows to insert error at different rates.

The following table shows the meaning of the parameter set.

**Table 168 TU generator error insertion parameters**

| Parameters | value | Description |
|---|---|---|
| error_insertion_rate | 0<br>1-7<br>0xff | Error insertion rate:<br>0, no error insertion<br>1, $10^{-1}$ error insertion rate<br>2, $10^{-2}$ error insertion rate<br>3, $10^{-3}$ error insertion rate<br>4, $10^{-4}$ error insertion rate<br>5, $10^{-5}$ error insertion rate<br>6, $10^{-6}$ error insertion rate<br>7, $10^{-7}$ error insertion rate<br>0xff, single error insertion |

**Table 169 TU_GENERATOR_ERROR_INSERTION indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |

**Table 169 TU_GENERATOR_ERROR_INSERTION indication**

| | |
|---|---|
| **Entity** | 0 |
| **ID** | **206** |
| **DSize** | 1 |
| **DPtr** | |

| | |
|---|---|
| 0 | error_insertion_rate |

The **TU_GENERATOR_ERROR_INSERTION** indication message is sent back to the application as an acknowledgment to the **TU_GENERATOR_ERROR_INSERTION** command message.

### 8.13.5 TEST UNIT GENERATOR Receive Pattern messages

The following tables show the message format for the **TU_GENERATOR_RECEIVE_PATTERN** messages.

**Table 170 TU_GENERATOR_RECEIVE_PATTERN command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **207** |
| **DSize** | 0 |
| **DPtr** | |

The **TU_GENERATOR_RECEIVE_PATTERN** message requests the current fixed-pattern detected by the generator receiver (only valid in FIXED-PATTERN mode) and the current generator setting which are provided in the message indication.

**Table 171 TU_GENERATOR_RECEIVE_PATTERN indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **207** |
| **DSize** | 9 |

**Table 171 TU_GENERATOR_RECEIVE_PATTERN indication**

| DPtr |
|---|

| | |
|---|---|
| 0-1 | rx_status |
| 2-3 | patternH |
| 4-5 | patternL |
| 6 | len |
| 7 | tp |
| 8 | tn |

The **TU_GENERATOR_RECEIVE_PATTERN** indication message is sent back to the application as an acknowledgment to the **TU_GENERATOR_RECEIVE_PATTERN** command message. It returns the current detected fixed-pattern and the generator status.

The following table shows the meaning of each field in the **TU_GENERATOR_RECEIVE_PATTERN** indication message.

**Table 172 TU Generator receive pattern parameters**

| Parameters | value | Description |
|---|---|---|
| rx_status | bitmap | TU generator receiver status:<br>OFF (0xffff)<br>>< receiver status |
| len | 1 - 32 | length of the fixed-pattern detected by the receiver |
| patternH | 16 bits | 16-bit fixed-pattern HIGH detected by the receiver |
| patternL | 16 bits | 16-bit fixed-pattern LOW detected by the receiver |
| tpi | 0,1,2,3,4 | Test point insertion |
| tn | 1-7<br>1-28 | tributary number where the generation is inserted<br>1 to 7 for DS2 tributary<br>1 to 28 for DS1/E1 channel |

**Table 173 TU generator bit-map receiver status**

| Receiver status | Bit Number | Description |
|---|---|---|
| OOS | 0 | Receiver Out of synchronization |
| A0 | 1 | input all “0”s |
| A1 | 2 | input all “1”s |
| LBE | 3 | Latch Bit Error detection |
| EMI | 4 | End Of Measurement Interval |
| LOOS | 5 | Latch Out Of Synchronization |
| LA0 | 6 | Latch input all “0”s |
| LA1 | 7 | Latch input all “1”s |
| INVS | 8 | Inverted pattern |
| OFF | 0-15 set to 1 | Receiver OFF |

### 8.13.6 TEST UNIT GENERATOR Receive Counter messages

The following tables show the message format for the **TU_GENERATOR_RECEIVE_COUNTER** messages.

**Table 174 TU_GENERATOR_RECEIVE_COUNTER command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **208** |
| **DSize** | 0 |
| **DPtr** | |

The **TU_GENERATOR_RECEIVE_COUNTER** message requests the current bit count and bit error count received and the current generator setting which are provided in the message indication.

**Table 175 TU_GENERATOR_RECEIVE_COUNTER indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |

**Table 175 TU_GENERATOR_RECEIVE_COUNTER indication**

| ID | 208 |
|---|---|
| DSize | 12 |
| DPtr | |

| | |
|---|---|
| 0-1 | rx_status |
| 2-3 | errorH |
| 4-5 | errorL |
| 6-7 | bbitH |
| 8-9 | bbitL |
| 10 | mode |
| 11 | ierr |

The **TU_GENERATOR_RECEIVE_COUNTER** indication message is sent back to the application as an acknowledgment to the **TU_GENERATOR_RECEIVE_COUNTER** command message. It returns the current received bit count, bit error and the generator status.

The following table shows the meaning of each field in the **TU_GENERATOR_RECEIVE_COUNTER** indication message.

**Table 176 TU Generator receive counter parameters**

| Parameters | value | Description |
|---|---|---|
| rx_status | bitmap | TU generator receiver status:<br>OFF (0xffff)<br>>< receiver status |
| errorH | 16 bits | 32-bit receive error counter HIGH |
| errorL | 16 bits | 32-bit receive error counter LOW |
| bbitH | 16 bits | 32-bit receive bit counter HIGH |
| bbitL | 16 bits | 32-bit receive bit counter LOW |

**Table 176 TU Generator receive counter parameters**

| Parameters | value | Description |
|---|---|---|
| mode | 0<br>1 | test unit pattern generator mode<br>**0**, FIXED-PATTERN generation<br>**1**, PRBS generation |
| ierr | 0<br>1-7 | test unit error insertion status<br>**0** No error insertion<br>**1-7** error insertion rate |

## 8.13.7 TEST UNIT FRAMER configuration messages

The following tables shows the message format for the **TU_FRAMER_SET** messages.

**Table 177 TU_FRAMER_SET command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **209** |
| **DSize** | 3 |
| **DPtr** | |

| | |
|---|---|
| 0 | mode |
| 1 | format |
| 2 | rai |

The **TU_FRAMER_SET** message command configures the Test unit framer according to the parameters set passed in the message. The framer is configured but **NOT** activated. The Framer is used with the GENERATOR in order to generate framed-patterns. The framer is activated by issuing the **TU_FRAMER_START**.

The following table shows the meaning of each field in the FRAMER_SET command message.

**Table 178 FRAMER Configuration parameters**

| Parameters | value | Description |
|---|---|---|
| mode | 0<br>1 | test unit framer mode<br>**0**,T1 mode<br>with framing set to SF or ESF<br>**1**, E1 mode<br>with framing set to DF or MF |
| framing | 0,1,2,3 | framing format<br>0, Extended Super Frame format (T1 F24)<br>1, Super Frame format (T1 F12)<br>2, Double Frame Format (E1)<br>3, Multi Frame Format (E1) |
| rai | 0<br>1 | T1 Remote alarm format<br>0, FS-bit in frame 12<br>1, Bit 2=0 in every channel |

**Table 179 TU_FRAMER_SET indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **209** |
| **DSize** | 3 |
| **DPtr** | |

| | |
|---|---|
| 0 | mode |
| 1 | format |
| 2 | rai |

The **TU_FRAMER_SET** indication message is sent back to the application as an acknowledgment to the **TU_FRAMER_SET** command message. It returns the current framer setting.

### 8.13.8 TEST UNIT FRAMER START/STOP messages

The following tables show the message format for the **TU_FRAMER_START** and **TU_FRAMER_STOP** messages.

**Table 180 TU_FRAMER_START command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **210** |
| **DSize** | 0 |
| **DPtr** | |

The **TU_FRAMER_START** message command activates the framer. Prior the FRAMER has been configured with the **TU_FRAMER_SET** command. The command returns in the indication message the current setting of the FRAMER (configuration).

**Table 181 TU_FRAMER_START indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **210** |
| **DSize** | 3 |
| **DPtr** | |

| | |
|---|---|
| 0 | mode |
| 1 | format |
| 2 | rai |

The **TU_FRAMER_START** indication message is sent back to the application as an acknowledgment to the **TU_FRAMER_START** command message. It returns the current framer setting.

The parameters are identical to those described in the **TU_FRAMER_SET** command except for the “mode” parameters which indicates the current state of the framer (OFF, T1 or E1).

**Table 182 TU_FRAMER_STOP command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **211** |
| **DSize** | 0 |
| **DPtr** | |

The **TU_FRAMER_STOP** message command deactivates the framer. The current framer setting remains unchanged.

**Table 183 TU_FRAMER_STOP indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **211** |
| **DSize** | 0 |
| **DPtr** | |

The **TU_FRAMER_STOP** message indication acknowledges the command message.

### 8.13.9 TEST UNIT FRAMER error insertion messages

The following tables shows the message format for the **TU_FRAMER_ERROR_INSERTION** messages.

**Table 184 TU_FRAMER_ERROR_INSERTION command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **212** |
| **DSize** | 2 |
| **DPtr** | |

| | |
|---|---|
| 0 | action |
| 1 | error |

The **TU_FRAMER_ERROR_INSERTION** message command allows to insert different types of errors.

The following table shows the meaning of the parameter set.

**Table 185 TU framer error insertion parameters**

| Parameters | value | Description |
|---|---|---|
| action | 0,1 | Error insertionaction<br>0, RESET the error insertion<br>1, SET the error insertion |
| error | 0,1,2,3 | Error insertion selection<br>0, framing errors<br>1, CRC errors (ESF-T1 / MF-E1)<br>2, E-bit errors (E1 only)<br>3, Remote Alarm generation (T1 only) |

**Table 186 TU_FRAMER_ERROR_INSERTION indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **212** |
| **DSize** | 1 |
| **DPtr** | |

**Table 186 TU_FRAMER_ERROR_INSERTION indication**

| | |
|---|---|
| 0 | error_insertion_status |

The **TU_FRAMER_ERROR_INSERTION** indication message is sent back to the application as an acknowledgment to the **TU_FRAMER_ERROR_INSERTION** command message. It provides the current status of the error insertion.

**Table 187 TU framer bit-map error insertion status**

| error insertion status | Bit Number | Description |
|---|---|---|
| RA | 0 | Remote Alarm |
| FE | 1 | Framing Error |
| CRC | 2 | CRC error |
| EBIT | 3 | E-bit error |

## 8.13.10 TEST UNIT FRAMER Receive Counter messages

The following tables shows the message format for the **TU_FRAMER_RECEIVE_COUNTER** messages.

**Table 188 TU_FRAMER_RECEIVE_COUNTER command**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **213** |
| **DSize** | 0 |
| **DPtr** | |

The **TU_FRAMER_RECEIVE_COUNTER** message requests the current values of the error counters which are provided in the message indication.

**Table 189 TU_FRAMER_RECEIVE_COUNTER indication**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **213** |
| **DSize** | 10 |
| **DPtr** | |

| | |
|---|---|
| 0-1 | fec |
| 2-3 | crcc |
| 4-5 | ebitc |
| 6 | rx_status |
| 7 | mode |
| 8 | format |
| 9 | ierr |

The **TU_FRAMER_RECEIVE_COUNTER** indication message is sent back to the application as an acknowledgment to the **TU_FRAMER_RECEIVE_COUNTER** command message. It returns the current values of the error counters and status of the framer.

The following table shows the meaning of each field in the **TU_FRAMER_RECEIVE_COUNTER** indication message.

**Table 190 TU framer receive counter parameters**

| Parameters | value | Description |
|---|---|---|
| rx_status | bitmap | TU framer receiver status:<br>OFF (0xffff)<br>>< receiver status |
| fec | 16 bits | 16-bit framing error counter |
| crcc | 16 bits | 16-bit CRC error counter |
| ebic | 16 bits | 16-bit E-bit error counter |
| mode | 0<br>1 | Test unit framer mode<br>**0:** T1<br>**1**: E1 |

**Table 190 TU framer receive counter parameters**

| Parameters | value | Description |
|---|---|---|
| format | 0,1,2,3 | framing format<br>0, Extended Super Frame format (T1 F24)<br>1, Super Frame format (T1 F12)<br>2, Double Frame Format (E1)<br>3, Multi Frame Format (E1) |
| ierr | bitmap<br>bit 0 -3 | test unit error insertion status<br>bit 0, remote alarm<br>bit 1, framing error<br>bit 2, CRC error<br>bit 3, E-bit error |

### 8.13.11 M13FX Error indication

The ERROR_INDICATION message is generated by the target when an invalid command is issued. It returns an error code.

**Table 191 ERROR_INDICATION message**

| | |
|---|---|
| **Dest** | USER |
| **Src** | APPLI |
| **Entity** | 0 |
| **ID** | **1000** |
| **DSize** | 1 |
| **DPtr** | |

| | |
|---|---|
| 0 | error code |

the following table lists the error codes

**Table 192 M13FX error codes**

| action | value | Description |
|---|---|---|
| device driver initialization | 1 | Unable to allocate a device context |
| device driver unitization | 2 | Device number not defined |
| application binding | 3 | Device driver not initialized |
| application binding | 4 | Application not bound to the device driver |
| application binding | 5 | Unable to allocate an application id |
| application binding | 6 | Unable to allocate an application context |
| device command | 7 | Device not activated |
| device command | 8 | Command not defined |
| device activation | 9 | Device already activated |
| device deactivation | 10 | Device already deactivated |
| device deactivation | 11 | Channels remain active |
| channel activation | 12 | Device not activated |
| channel activation | 13 | Channel already activated |
| channel activation | 14 | Unable to allocate a channel ref |
| channel activation | 15 | Unable to allocate a channel context |
| channel activation | 16 | Unable to allocate a reception buffer |
| channel deactivation | 17 | Device not activated |
| channel deactivation | 18 | Channel not activated |
| channel transmission | 19 | Channel not activated |
| channel transmission | 20 | Channel busy |
| channel transmission | 21 | Unable to allocate a handle |
| status indication | 22 | Device status error |
| receive indication | 23 | Channel receive error |
| transmission indication | 24 | Channel transmission error |
| OPI function | 25 | OPI function not supported |

### 8.13.12 M13FX DEBUG DISPLAY Commands

The DBG_DISPLAY message allows to enable/disable the display of the M13FX indication messages on the debug port.

**Table 193 DBG_DISPLAY message**

| | |
|---|---|
| **Dest** | APPLI |
| **Src** | USER |
| **Entity** | 0 |
| **ID** | **500** |
| **DSize** | 1 |
| **DPtr** | |

| | |
|---|---|
| 0 | action |
| 1 | indication |

The following table shows the meaning of the parameter set.

**Table 194 DBG_DISPLAY parameters**

| Parameters | value | Description |
|---|---|---|
| action | 0,1 | display action<br>0, DISABLE the display<br>1, ENABLE the display |
| indication | 0 - 7 | select the indication<br>0, DS3 alarms indication<br>1, DS2 alarms indication<br>2, DS2 loopcode indication<br>3, DS1 loopcode indication<br>4, ERROR COUNTERS indication (1 sec.)<br>5, FEAC channel indication<br>6, MDL channel indication<br>7, TEST UNIT indication |

# 9 M13FX OPI interface

The M13FX OPI interface is a procedural interface conforms to the OPI concept.

We have the following groups of functions:

- DEVICE DRIVER INITIALIZATION and APPLICATION BINDING,
- PORT MANAGEMENT,
- CHANNEL MANAGEMENT,
- APPLICATION CALLBACKS,
- OPI FUNCTIONS NO SUPPORTED.

The file **m13opix.h** must be included in order to access to the interface.

## 9.1 DEVICE DRIVER INITIALIZATION and APPLICATION BINDING

### 9.1.1 DRV_M13FXInit

**NAME:** DRV_M13FXInit.

**DESCRIPTION:** Initialization of the device driver. It must be called once at the system startup. The structure parameter contains the list of the devices declared with the corresponding base address. Each device is initialized with the default configuration parameters but remains inactive.

**PROTOTYPE:**
```
POPI_STATUS DRV_M13FXInit (Y_S_device_prm* pdrvinit)
```

**PARAMETERS:**

| | |
|---|---|
| **pdrvinit** | Device driver initialization table address.<br>The table contains N+1 entries with N the number of devices supported by the device driver. Each entry is a device structure parameter (Y_S_device) with the following fields. The last entry is a end of table marker and marked as DEVICE_UNDEFINE in the dev_number field. |
| **.dev_number** | Device number.<br>0 to N,<br>DEVICE_UNDEFINE. |
| **.base_address** | Device base address |

**RETURN:** Error code.
OPI_SUCCESS, OPI_M13INIT_DEVCTX_ERR,OPI_M13INIT_DEVUNDEF_ERR.
The device driver is ready to be accessed by the applications which must be bound to it prior to access the device resources through the port and channel commands.

### 9.1.2 DRV_M13FXBind

**NAME:** DRV_M13FXBind.

**DESCRIPTION:** This function binds an application to the device driver. Each application must be bound to the device driver in order to access to the device resources. During this procedure, the application and the device driver exchange their entry-points through the bind structure parameters provided by the application. The device driver memorizes the application callbacks and updates the bind structure with its entry-points that the application has to use for further interactions with the device driver.The device driver assigns a unique user identifier to the application, which defines the interface between them.

**PROTOTYPE:**

```
POPI_STATUS          DRV_M13FXBind          (USER_ID
*puserid,DRV_Bind* pdrvbind)
```

**PARAMETERS:**

**puserid** Application identifier address.
It will be updated by the device driver with the allocated identifier. This identifier is used by both sides (device driver, application) for further interactions.

**pdrvbind** Bind parameters structure address.
The application provides its callbacks.
The device driver updates with its entry-points.

**RETURN:** Error code.
OPI_SUCCESS, OPI_M13BIND_DEVUNDEF_ERR, OPI_M13BIND_APPID_ERR, OPI_M13BIND_APPCTX_ERR.
The application is bound to the device driver. it can access the device driver through the port and channel command using the unique identifier assigned by the device driver.

## 9.2 PORT MANAGEMENT

### 9.2.1 DRV_M13FXPortOpen

**NAME:** DRV_M13FXPortOpen.

**DESCRIPTION:** Allows an application module to open a port. Previously the application must be bound to the device driver in order to access the desired port resource. the application module provides the User identifier allocated at the binding procedure. This permits the device driver to associate the port resource to the application. The identifier is the means for the device driver to select the application callback when an event occurs.

**PROTOTYPE:**

```
POPI_STATUS    DRV_M13FXPortOpen(USER_ID    userid,
WORD32 port, void* param)
```

**PARAMETERS:**

**userid** Application identifier.

**port** Device number.
0 to N.

**param** Device parameters structure address.
NIL.

**RETURN:** Error code.
OPI_SUCCESS, OPI_M13PORTOPEN_OPEN_ERR.
The device is activated which can be accessed through the device functions.

### 9.2.2 DRV_M13FXPortCfgReq

**NAME:** DRV_M13FXPortCfgReq.

**DESCRIPTION:** Allows the application to access to the device resources through the device functions. The device has been previously activated with the open port command.

**PROTOTYPE:**

```
POPI_STATUS   DRV_M13FXPortCfgReq(USER_ID   userid,
WORD32 port, WORD32 code, void* param)
```

**PARAMETERS:**

**userid** Application identifier.

**port** Device number.
0 to N.

**code** Device function.
The following classes of function are availables:
DS3, DS2, DS1, FEAC, MDL, MISC configuration.
DS3, DS2, DS1 Alarms.
DS3, DS2, DS1 Loopbacks.
FEAC, MDL channels.
Test Unit.
Tributary Mapper.

**param** Device function parameters structure address

The structure depends on the selected function.

**RETURN:** Error code.
OPI_SUCCESS,
OPI_M13PORTCFG_CMDE_ERR, OPI_M13PORTCFG_NOTOPEN_ERR.

### 9.2.3 DRV_M13FXPortClose

**NAME:** DRV_M13FXPortClose.

**DESCRIPTION:** This function closes a designated port previously opened. The caller must provide the user identifier used at the port opening.

**PROTOTYPE:**

```
POPI_STATUS   DRV_M13FXPortClose(USER_ID   userid,
WORD32 port)
```

**PARAMETERS:**

**userid** Application identifier.

**port** Device number.

0 to N

**RETURN:** Error code.
OPI_SUCCESS, OPI_M13PORTCLOSE_CLOSE_ERR,
OPI_M13PORTCLOSE_CHNACT_ERR.
The device is deactivated.

## 9.3 CHANNEL MANAGEMENT

### 9.3.1 DRV_M13FXChOpen

**NAME:** DRV_M13FXChOpen.

**DESCRIPTION:** This function opens a channel in the designated port. The port channel can be opened by any application bound to the device driver. The device driver allocated a reference which must be used for the data transmission.

**PROTOTYPE:**

```
POPI_STATUS DRV_M13FXChOpen(USER_ID userid, WORD32
port, WORD32 channel, void* param, Y_R_ref *pref)
```

**PARAMETERS:**

**userid** Application identifier.

**port** Device number.
0 to N.

**channel** Channel selection.
M13FX_FEAC_CHANNEL,
M13FX_MDL_CHANNEL.

**param** Channel parameter structure address.

NIL.

**pref** Reference channel address.
It will be updated by the device driver with the allocated reference. This identifier is used by both sides (device driver, application) for further channel interactions.

**RETURN:** Error code.
OPI_SUCCESS, OPI_M13CHNOPEN_PORTCLOSE_ERR, OPI_M13CHNOPEN_CHNOPEN_ERR, OPI_M13CHNOPEN_CHNREF_ERR, OPI_M13CHNOPEN_CHNCTX_ERR, OPI_M13CHNOPEN_BUFALLOC_ERR.
The selected channel is activated.

### 9.3.2 DRV_M13FXChClose

**NAME:** DRV_M13FXChClose.

**DESCRIPTION:** This function closes a designated channel previously opened.The caller must provided the user identifier used at the channel opening.

**PROTOTYPE:**
```
POPI_STATUSDRV_M13FXChClose(USER_ID userid, WORD32
port, WORD32 channel)
```

**PARAMETERS:**

**userid** Application identifier.

**port** Device number.
0 to N.

**channel** Channel selection.
M13FX_FEAC_CHANNEL,
M13FX_MDL_CHANNEL.

**RETURN:** Error code.
OPI_SUCCESS, OPI_M13CHNOPEN_PORTCLOSE_ERR,
OPI_M13CHNOPEN_CHNOPEN_ERR,OPI_M13CHNOPEN_CHNREF_ERR,
OPI_M13CHNOPEN_CHNCTX_ERR. OPI_M13CHNOPEN_BUFALLOC_ERR.
The selected channel is deactivated.

### 9.3.3 DRV_M13FXChTxReq

**NAME:** DRV_M13FXChTxReq.

**DESCRIPTION:** This function sends a data buffer in the channel referred by the reference channel.

**PROTOTYPE:**

```
POPI_STATUS   DRV_M13FXChTxReq(USER_ID   userid,
Y_R_ref ref, HANDLE* handle)
```

**PARAMETERS:**

**userid** Application identifier.

**ref** Channel reference.
Allocated by the device driver at the channel opening.

**handle** Data transmission handle.
The handle contains the address and the length of the data to transmit in the selected channel.

**RETURN:** Error code.
OPI_SUCCESS,
OPI_M13CHNSEND_CHNREF_ERR,OPI_M13CHNSEND_BUSY_ERR.
The data buffer is transmitted to the selected channel.

## 9.4 APPLICATION CALLBACKS

### 9.4.1 APPL_StatusInd

**NAME:** APPL_StatusInd.

**DESCRIPTION:** The Status Indication application callback is passed to the device driver at the bind procedure. All device driver events are passed through this callback.

**PROTOTYPE:**

```
void APPL_StatusInd (USER_ID UserId, WORD32 StatusCode, WORD32 port, WORD32 channel, HANDLE*handle)
```

**PARAMETERS:**

**UserId** Application identifier.

**StatusCode** Event code.
M13FX_ERROR_COUNTERS,
M13FX_DS3_ALARM,
M13FX_DS2_ALARM,
M13FX_DS2_LOOPBACK,
M13FX_DS1_LOOPBACK,
M13FX_TU_INDICATION.

**port** Device number.
0 to N.

**channel** Channel number.
Not used.

**handle** handle containing the related indication parameters

**RETURN:** None.

### 9.4.2 APPL_TxBufConfirm

**NAME:** APPL_TxBufConfirm.

**DESCRIPTION:** The TxBufConf application callback is passed to the device driver at the bind procedure. This callback is called by the device drive when a data buffer is sent out in the referred channel.

**PROTOTYPE:**

```
void APPL_TxBufConfirm (USER_ID UserId, Y_R_ref
ref,HANDLE* Handle,POPI_STATUS status)
```

**PARAMETERS:**

**userid** Application identifier.

**status** Transmission status.
CHANNEL_OK,
CHANNEL_TXFIFO_BUSY_ERROR,
CHANNEL_DATA_UNDERRUN_ERROR

**ref** Channel reference.
Allocated by the device driver at the channel opening.

**handle** Transmitted data handle.
The handle contains the address and the length of the transmitted data in the referred channel.

**RETURN:** None.

### 9.4.3 APPL_RxBufInd

**NAME:** APPL_RxBufInd.

**DESCRIPTION:** The RxBufInd application callback is passed to the device driver at the bind procedure. This callback is called by the device driver when a data buffer is received from the referred channel.

**PROTOTYPE:**

```
void APPL_RxBufInd (USER_IDUserId, Y_R_refref,
HANDLE*Handle,POPI_STATUS status)
```

**PARAMETERS:**

**userid** Application identifier.

**status** Reception status.
CHANNEL_OK.

**ref** Channel reference.
Allocated by the device driver at the channel opening.

**handle** Received data handle.
The handle contains the address and the length of the received data in the referred channel.

**RETURN:** None.

## 9.5 OPI FUNCTIONS NO SUPPORTED

The following OPI functions are not supported for the M13FX device driver. An error code ( OPI_M13FUNCT_NOT_SUPPORTED_ERR) is the returned to the caller:

- DRV_M13FXChCfgReq,
- DRV_M13FXGetStatReq,
- DRV_M13FXIOCtlReq.

## 9.6 M13FX OPI mapping

The following table shows the mapping between the M13FX OPI interface and the M13FX Low-level interface in term of function calls. For further information on the low-level interface please refer to the M13FX Low-Level interface chapter.

**Table 195 M13FX OPI interface / M13FX Low-Level Interface mapping**

| module | M13FX OPI function | M13FX Low-level function |
|---|---|---|
| SSAL | DRV_M13FXInit | M13FX_Init |
| APPL | DRV_M13FXBind | |
| APPL | DRV_M13FXPortOpen | M13FX_Device_Activate |
| APPL | DRV_M13FXPortClose | M13FX_Device_Deactivate |
| APPL | DRV_M13FXPortCfgReq | M13FX_Device_status |
| APPL | - | M13FX_DS3_Loopback |
| APPL | - | M13FX_DS2_Loopback |
| APPL | - | M13FX_DS1_Loopback |
| APPL | - | M13FX_DS3_Alarm |
| APPL | - | M13FX_DS2_Alarm |
| APPL | - | M13FX_DS1_Alarm |
| APPL | - | M13FX_Error_Counters |
| APPL | - | M13FX_TU_Set_Generation |
| APPL | - | M13FX_TU_Start_Generation |
| APPL | - | M13FX_TU_Stop_Generation |
| APPL | - | M13FX_TU_Test_Point |
| APPL | - | M13FX_TU_Error_Insertion |
| APPL | - | M13FX_TU_Receive_Pattern |
| APPL | - | M13FX_TU_Receive_Counter |
| APPL | - | M13FX_TU_Set_Framer |
| APPL | - | M13FX_TU_Start_Framer |
| APPL | - | M13FX_TU_Stop_Framer |
| APPL | - | M13FX_TU_Error_Framer |
| APPL | - | M13FX_TU_Receive_Counter_Framer |
| APPL | - | M13FX_DS3_Cfg_Parameters |
| APPL | - | M13FX_DS2_Cfg_Parameters |

**Table 195 M13FX OPI interface / M13FX Low-Level Interface mapping**

| module | M13FX OPI function | M13FX Low-level function |
|---|---|---|
| APPL | - | M13FX_DS1_Cfg_Parameters |
| APPL | - | M13FX_Feac_Cfg_Parameters |
| APPL | - | M13FX_Mdl_Cfg_Parameters |
| APPL | - | M13FX_Misc_Cfg_Parameters |
| APPL | - | M13FX_DS3_Cfg_Get_Parameters |
| APPL | - | M13FX_DS2_Cfg_Get_Parameters |
| APPL | - | M13FX_DS1_Cfg_Get_Parameters |
| APPL | - | M13FX_Feac_Cfg_Get_Parameters |
| APPL | - | M13FX_Mdl_Cfg_Get_Parameters |
| APPL | - | M13FX_Misc_Cfg_Get_Parameters |
| APPL | - | M13FX_Reset_Cfg_Parameters |
| APPL | - | M13FX_Set_Cfg_Parameters |
| APPL | - | M13FX_Map_Get_Configuration |
| APPL | - | M13FX_Map_Set_Configuration |
| APPL | - | M13FX_Map_Reset_Configuration |
| APPL | - | M13FX_Feac_Status |
| APPL | DRV_M13FXChOpen | M13FX_Feac_Send<br>M13FX_Mdl_Send |
| **module** | **M13FX Low-Level Function** | **Application callback** |
| DRV | DRV_M13FX_Alarm_Event | APPL_StatusInd |
| DRV | DRV_M13FX_Error_Counters_Event | APPL_StatusInd |
| DRV | DRV_M13FX_Sent_TxBuf | APPL_TxBufConfirm |
| DRV | DRV_M13FX_Rcv_RxBuf | APPL_RxBufInd |

with,

SSAL, M13FX SSAL module,

APPL, M13FX application module,

DRV, M13FX Low-level device driver.

## 9.7 ANNEXES

### 9.7.1 M13OPI interface header file

```
#ifndef __M13OPIX_H
#define __M13OPIX_H

#include "opiext.h"     /* OPI interface                 */
#include "m13fxdx.h"    /* M13FX device driver interface */


#define MAX_APPL_NUMBER     1                /* Max Application Number */
#define MAX_DEVICE_NUMBER    1                /* Max Device Number     */

#define MAX_PORT_NUMBER     MAX_DEVICE_NUMBER    /* Max port Number    */
#define MAX_CHANNEL_NUMBER   (2*MAX_PORT_NUMBER)  /* Max channel Number */


/*-----------------------------------------------*/
/*M13FX port commands                            */
/*  they are passed to the device driver         */
/*  through the OPI DRV_M13FXPortCfgReq.         */
/*                                               */
/*-----------------------------------------------*/
typedef enum
{
  M13FX_DEV_STATUS,                   /* Device status                 */
  M13FX_DS3_ALARM,                   /* DS3 alarms                   */
  M13FX_DS2_ALARM,                    /* DS2 alarms                    */
  M13FX_DS1_ALARM,                    /* DS1 alarms                    */

  M13FX_ERROR_COUNTERS,              /* Error counters               */
  M13FX_DS3_LOOPBACK,                 /* DS3 loopback                  */
  M13FX_DS2_LOOPBACK,                 /* DS2 loopback                  */
  M13FX_DS1_LOOPBACK,                 /* DS1 loopback                  */

  M13FX_TU_GENERATOR_NSET,            /* TU-GEN. configuration         */
  M13FX_TU_GENERATOR_START,           /* TU-GEN. start                 */
  M13FX_TU_GENERATOR_STOP,            /* TU-GEN. stop                  */
  M13FX_TU_GENERATOR_TEST_POINT,      /* TU-GEN. test point insertion  */
  M13FX_TU_GENERATOR_ERROR_INSERTION, /* TU-GEN. error insertion       */
  M13FX_TU_GENERATOR_RECEIVE_PATTERN, /* TU-GEN. Receive Fixed-pattern */
  M13FX_TU_GENERATOR_RECEIVE_COUNTER, /* TU-GEN. Receive counters      */

  M13FX_TU_FRAMER_SET,                /* TU-FRM. configuration        */
  M13FX_TU_FRAMER_START,               /* TU-FRM. start                 */
  M13FX_TU_FRAMER_STOP,                /* TU-FRM. start                 */
```

```
  M13FX_TU_FRAMER_ERROR_INSERTION,     /* TU-FRM. error insertion      */
  M13FX_TU_FRAMER_RECEIVE_COUNTERS,    /* TU-FRM. Receive counters     */
  M13FX_TU_INDICATION,                 /* TU - Indication              */

  M13FX_CFG_DS3_PARAMETERS,           /* CNF. set DS3 parameters       */
  M13FX_CFG_DS2_PARAMETERS,           /* CNF. set DS2 parameters       */
  M13FX_CFG_DS1_PARAMETERS,           /* CNF. set DS1 parameters       */
  M13FX_CFG_FEAC_PARAMETERS,          /* CNF. set FEAC parameters      */
  M13FX_CFG_MDL_PARAMETERS,           /* CNF. set MDL parameters       */
  M13FX_CFG_MISC_PARAMETERS,          /* CNF. set MISC parameters      */

  M13FX_CFG_GET_DS3_PARAMETERS,        /* CNF. get DS3 parameters      */
  M13FX_CFG_GET_DS2_PARAMETERS,        /* CNF. get DS2 parameters      */
  M13FX_CFG_GET_DS1_PARAMETERS,        /* CNF. get DS1 parameters      */
  M13FX_CFG_GET_FEAC_PARAMETERS,       /* CNF. get FEAC parameters     */
  M13FX_CFG_GET_MDL_PARAMETERS,        /* CNF. get MDL parameters      */
  M13FX_CFG_GET_MISC_PARAMETERS,       /* CNF. get MISC parameters     */

  M13FX_CFG_SET_PARAMETERS,           /* CNF. validate the parameters  */
  M13FX_CFG_RESET_PARAMETERS,         /* CNF. restore default parameters*/

  M13FX_MAP_GET_CONFIGURATION,         /* MAP. get the configuration   */
  M13FX_MAP_SET_CONFIGURATION,        /* MAP. update the configuration */
  M13FX_MAP_RESET_CONFIGURATION,       /* MAP. restore the configuration */


  M13FX_FEAC_RSEND,                   /* FEAC Send repeat Code        */
  M13FX_FEAC_STATUS,                   /* FEAC channel status          */
  M13FX_MDL_STATUS                     /* MDL channel status           */

} Y_E_dev_cmd;


/*-----------------------------------------------*/
/* DS3 channels:                                 */
/* The MDL and FEAC channels are mapped into     */
/* the OPI channel concept.                      */
/*-----------------------------------------------*/
typedef enum
{
  M13FX_FEAC_CHANNEL,           /* Select the FEAC channel */
  M13FX_MDL_CHANNEL,            /* Select the MDL  channel */
  M13FX_MAX_CHANNEL             /* There are 2 channels    */
} Y_E_channels;
```

```
/*-------------------------------------------------------*/
/*device driver init parameter structure:                */
/*  each element of the device driver initialization  */
/*  is composed by the Y_S_device_prm, which assigns  */
/*  for each device driver the base address.          */
/*  The last element is the end of the structure where*/
/*  device number is defined as DEVICE_UNDEF.         */
/*-------------------------------------------------------*/
#define DEVICE_UNDEFINE 0xff
typedef struct
{
  WORD32       dev_number;          /* device number       */
  void*        dev_base_address;    /* device base address */
}Y_S_device_prm;



/*-------------------------------------------------------*/
/* M13FX OPI Function Interface                           */
/*-------------------------------------------------------*/
extern POPI_STATUS DRV_M13FXInit (Y_S_device_prm*);
extern POPI_STATUS DRV_M13FXBind (USER_ID*,DRV_Bind*);

extern POPI_STATUS DRV_M13FXPortCfgReq (USER_ID,WORD32,WORD32,void*);
extern POPI_STATUS DRV_M13FXPortOpen   (USER_ID,WORD32,void*);
extern POPI_STATUS DRV_M13FXPortClose  (USER_ID,WORD32);

extern POPI_STATUS DRV_M13FXChCfgReq (USER_ID, WORD32, WORD32, WORD32,
void*);
extern POPI_STATUS DRV_M13FXChOpen (USER_ID, WORD32, WORD32, void*,
Y_R_ref*);
extern POPI_STATUS DRV_M13FXChClose   (USER_ID,WORD32,WORD32);
extern POPI_STATUSDRV_M13FXChTxReq    (USER_ID,Y_R_ref,HANDLE*);

extern POPI_STATUS  DRV_M13FXGetStatReq (USER_ID,void*);
extern POPI_STATUS  DRV_M13FXIOCtlReq   (USER_ID,void*);

/*-------------------------*/
/* device driver interface */
/*-------------------------*/
extern void     DRV_M13FX_Put_RxPool  (Y_R_ref);
extern BHANDLE* DRV_M13FX_Get_RxBuf   (Y_R_ref);
extern void     DRV_M13FX_Put_RxBuf   (Y_R_ref,BHANDLE*);
extern void     DRV_M13FX_Rcv_RxBuf   (BHANDLE*);
extern void     DRV_M13FX_Sent_TxBuf  (BHANDLE*);
```

```
extern void     DRV_M13FX_Error_Counters_Event (WORD32,BHANDLE*);
extern void   DRV_M13FX_Alarm_Event (WORD32,Y_E_dev_cmd,Y_R_tn,WORD16);

#endif /* __M13OPIX_H */
```

### 9.7.2 OPI interface HEADER files

**OPIEXT.H**:

```
#ifndef __OPI_EXT
#define __OPI_EXT
#include "opidrv.h"              /* OPI device driver interface */
#include "m13ssalx.h"            /* SSAL interface              */
#endif /* __OPI_EXT */
```

**OPIDRV.H**:

```
ifndef __OPIDRV_H
#define __OPIDRV_H

/*---------------------------------------------*/
/* POPI_STATUS: OPI execution status report   */
/*                                             */
/*---------------------------------------------*/
typedef enum
{
  OPI_SUCCESS,                  /*00*/

  /*------------------------------*/
  /* Device driver initialization */
  /*------------------------------*/
  OPI_M13INIT_DEVCTX_ERR,    /*01 Unable to allocate a DRV ctx */
  OPI_M13INIT_DEVUNDEF_ERR,  /*02 invalid DRV number          */

  /*------------------------------*/
  /* Application Binding          */
  /*------------------------------*/
  OPI_M13BIND_DEVUNDEF_ERR,  /*03 DRV not initialized               */
  OPI_M13BIND_APPUNDEF_ERR,  /*04 Application not bounded to the DRV  */
  OPI_M13BIND_APPID_ERR,     /*05 Unable to allocate an Application Id */
  OPI_M13BIND_APPCTX_ERR,    /*06 Unable to allocate an Application ctx*/
```

```
/*------------------------------*/
/* Device commands              */
/*------------------------------*/
OPI_M13PORTCFG_NOTOPEN_ERR,    /*07 DRV not activated     */
OPI_M13PORTCFG_CMDE_ERR,       /*08 Invalid DRV command   */

/*------------------------------*/
/* Device activation            */
/*------------------------------*/
OPI_M13PORTOPEN_OPEN_ERR,      /*09 DRV is already activated */

/*------------------------------*/
/* Device deactivation          */
/*------------------------------*/
OPI_M13PORTCLOSE_CLOSE_ERR,     /*10 Drv is already deactivated */
OPI_M13PORTCLOSE_CHNACT_ERR,   /*11 Channels remain active    */

/*------------------------------*/
/* Channel activation           */
/*------------------------------*/
OPI_M13CHNOPEN_PORTCLOSE_ERR,  /*12 DRV not active                  */
OPI_M13CHNOPEN_CHNOPEN_ERR,    /*13 the channel is already open     */
OPI_M13CHNOPEN_CHNREF_ERR,     /*14 Unable to allocate a channel ref */
OPI_M13CHNOPEN_CHNCTX_ERR,     /*15 Unable to allocate a channel ctx */
OPI_M13CHNOPEN_BUFALLOC_ERR,   /*16 Unable to allocate a Rx buffer   */

/*------------------------------*/
/* Channel deactivation         */
/*------------------------------*/
OPI_M13CHNCLOSE_PORTCLOSE_ERR, /*17 Device is not active         */
OPI_M13CHNCLOSE_CHNREF_ERR,    /*18 Channel not active           */

/*------------------------------*/
/* Channel transmission         */
/*------------------------------*/
OPI_M13CHNSEND_CHNREF_ERR,     /*19 Channel not active           */
OPI_M13CHNSEND_BUSY_ERR,       /*20 Channel is busy              */
OPI_M13CHNSEND_HANDLEALLOC_ERR,/*21 Unable to allocate a handle  */

/*------------------------------*/
/* Device Indications           */
/*------------------------------*/
OPI_M13IND_STATUS_ERR,         /*22 Device Status indication        */
OPI_M13IND_RCV_ERR,            /*23 Channel Receive indication      */
OPI_M13IND_SEND_ERR,           /*24 Channel Transmission indication */
```

```
  /*-----------------------------*/
  /* OPI function not supported   */
  /*-----------------------------*/
  OPI_M13FUNCT_NOT_SUPPORTED_ERR /*25 not supported */
} POPI_STATUS;

/*---------------------------*/
/* USER_ID: User Identifier  */
/*---------------------------*/
typedef WORD32       USER_ID ;
#define NO_USER_ID   0xffffffff
#define DrvHandle    WORD32


/*---------------------------*/
/* Reference Channel         */
/* --------------------------*/
typedef WORD16      Y_R_ref;
#define NO_REF      0xffff


/*-----------------------------------------------------------*/
/* HANDLE: handle structure used for Link List and           */
/* data buffer management.                                   */
/*-----------------------------------------------------------*/
typedef struct _HANDLE
{
  WORD32          header;       /* handle header     */
  struct _HANDLE* next;         /* next handle       */
  WORD16          user1;        /* free for use      */
  WORD16          user2;        /* free for use      */
  WORD8*          BufferAddr;   /* buffer address    */
  WORD16          BufferLength; /* buffer length     */
  WORD8*          DataAddr;     /* Data area address */
  WORD16          DataLength;   /* data area Length  */
} HANDLE,BHANDLE;

/*-----------------------------*/
/* Link List structure         */
/*-----------------------------*/
typedef struct
{
  BHANDLE* first; /* first in the link list           */
  BHANDLE* last;  /* last in the link list            */
  WORD8    count; /* number of blocks in the link list */
} Y_S_ll;
typedef Y_S_ll LHANDLE; /* link List Handle */
```

```
/*----------------------------------------*/
/* Definitions for BIND function structure */
/*----------------------------------------*/

/*----------------------------------------*/
/*Port management:                        */
/*----------------------------------------*/
typedef POPI_STATUS (*tDRV_PortCfgReq) (USER_ID userid, WORD32 port,
WORD32 code, void* param);
typedef POPI_STATUS (*tDRV_PortOpen)(USER_ID userid, WORD32 port, void*
param);
typedef POPI_STATUS (*tDRV_PortClose) (USER_ID userid, WORD32 port);


/*----------------------------------------*/
/*Channel management:                     */
/*----------------------------------------*/
typedef POPI_STATUS (*tDRV_ChCfgReq) (USER_ID userid, WORD32 port, WORD32
channel,WORD32 code, void* param);
typedef POPI_STATUS (*tDRV_ChOpen) (USER_ID userid, WORD32 port, WORD32
channel, void* param, Y_R_ref*);
typedef POPI_STATUS (*tDRV_ChClose) (USER_ID userid, WORD32 port, WORD32
channel);
typedef POPI_STATUS (*tDRV_ChTxReq) (USER_ID userid, Y_R_ref ref,
BHANDLE* handle);
typedef void (*tDRV_ISRHandler) (USER_ID userid, WORD32 port, WORD32
channel, void* param);

/*----------------------------------------*/
/*Generic functions:                      */
/*----------------------------------------*/
typedef POPI_STATUS (*tDRV_GetStatReq) (USER_ID userid, void* param);
typedef POPI_STATUS (*tDRV_IOCtlReq) (USER_ID userid, void* param);


/*----------------------------------------*/
/*Application callback functions:         */
/*----------------------------------------*/
typedef void (*tAPPL_StatusInd)(USER_ID UserId, WORD32 Statuscode, WORD32
Port, WORD32 Channel, HANDLE* handle);
typedef void (*tAPPL_TxBufConf)(USER_ID UserId, Y_R_ref ref, BHANDLE*
Handle,POPI_STATUS status);
typedef void (*tAPPL_RxBufInd) (USER_ID UserId, Y_R_ref ref, BHANDLE*
Handle,POPI_STATUS status);
```

```
/*--------------------------------------*/
/*Application - Device Driver           */
/*         BIND structure:              */
/*--------------------------------------*/
typedef struct
{
  tAPPL_StatusInd APPL_StatusInd;
  tAPPL_TxBufConf APPL_TxBufConf;
  tAPPL_RxBufInd  APPL_RxBufInd;

  tDRV_PortCfgReq DRV_PortCfgReq;
  tDRV_PortOpen   DRV_PortOpen;
  tDRV_PortClose  DRV_PortClose;

  tDRV_ChCfgReq   DRV_ChCfgReq;
  tDRV_ChOpen     DRV_ChOpen;
  tDRV_ChClose    DRV_ChClose;
  tDRV_ChTxReq    DRV_ChTxReq;
  tDRV_GetStatReq DRV_GetStatReq;
  tDRV_IOCtlReq   DRV_IOCtlReq;
} DRV_Bind;

#endif /* __OPIDRV_H */
```

**M13SSALX.H**:

```
#ifndef __M13SSALX_H
#define __M13SSALX_H

/*----------------------------*/
/* EVENT handle configuration */
/*----------------------------*/
#define MAX_HANDLE_EVENT       5 /* Mas 5 event handles in the pool */
#define POOL_EVENT_BUFFER_SIZE 8 /* Align to type 0 DDS message     */

/*-----------------------------------------------*/
/* FEAC and MDL channel configuration parameters */
/*-----------------------------------------------*/
#define POOL_THRESHOLD   2  /* Min 2 Rx handles in the channel Rx pool */
#define FEAC_BUFFER_SIZE 30 /* Max FEAC Rx, Tx buffer size             */
#define FEAC_BUFFER_W    3  /* FEAC buffer window in reception         */
#define MDL_BUFFER_SIZE  81 /* Max MDL Rx, Tx buffer size              */
#define MDL_BUFFER_W     5  /* MDL buffer window in reception          */
```

```
/*-----------------------------*/
/* M13FX SSAL module interface */
/*-----------------------------*/
/* module initialization */
extern  POPI_STATUS SSAL_Init                (void);
extern  void        SSAL_M13fx_Interrupt_dev0(void);
/* memory management */
extern BHANDLE*     BufferAlloc (WORD32 w32Size);
extern  void        BufferFree  (BHANDLE* phandle);
extern  WORD16      MemPoolCount(WORD16 bsize);
/* handle management */
extern  BHANDLE*    GetHandle (void);
extern  void        FreeHandle (BHANDLE* phandle);
/* Link list management */
extern  void        LL_init  (LHANDLE* pll);
extern  void        LL_reset (LHANDLE* pll);
extern  void        LL_put   (LHANDLE* pll,BHANDLE *phandle);
extern  WORD8       LL_nblock(LHANDLE* pll);
extern  BHANDLE*    LL_get   (LHANDLE* pll);
extern  BHANDLE*    LL_read  (LHANDLE* pll);
extern  BHANDLE*    LL_nread (BHANDLE* phandle);

#endif /* __M13SSALX_H */
```

# 10 M13FX Low-level interface

The M13FX Low-level interface is a procedural interface.

We have the following groups of functions:

- INITIALIZATION and INTERRUPTS,
- CONFIGURATION,
- ACTIVATION,
- ALARMS,
- LOOPBACKS,
- FEAC,
- MDL,
- MAPPER,
- TEST UNIT,
- ERROR COUNTERS,
- USER FUNCTIONS CALLED by the LOW-LEVEL DEVICE DRIVER.

The file **m13fxdx.h** must be included in order to access to the interface.

## 10.1 INITIALIZATION and INTERRUPTS

### 10.1.1 M13FX_Init

**NAME:** M13FX_Init.

**DESCRIPTION:** This function assigns the base address to the device selected with the device number. This is the first function to call in order to access to the device. The device is programmed with the default configuration parameters set. The device is not active.

**PROTOTYPE:** `char* M13FX_Init(WORD32 port, void* pbadr)`

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**pbadr** Device base address.

**RETURN:** String version.

### 10.1.2 M13FX_Interrupt_Handler

**NAME:** M13FX_Interrupt_Handler.

**DESCRIPTION:** Device interrupt handler entry-point to call when a M13FX interrupt occurs.

**PROTOTYPE:** `void M13FX_Interrupt_Handler (WORD32 port)`

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**RETURN:** None.

## 10.2 CONFIGURATION

### 10.2.1 M13FX_DS3_Cfg_Parameters

**NAME:** M13FX_DS3_Cfg_Parameters.

**DESCRIPTION:** This function updates the device ds3 configuration parameters according to the parameter choice (YES/NO) passed in the parameter choice structure. This configuration will be active when the Set command will be issued.

**PROTOTYPE:**

```
void M13FX_DS3_Cfg_Parameters (WORD32port,
Y_S_ds3_prm_choice* param)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**param** DS3 configuration parameters address.

**RETURN:** None.

### 10.2.2 M13FX_DS2_Cfg_Parameters

**NAME:** M13FX_DS2_Cfg_Parameters.

**DESCRIPTION:** This function updates the device ds2 configuration parameters according to the parameter choice (YES/NO) passed in the parameter choice structure. This configuration will be active when the Set command will be issued.

**PROTOTYPE:**

```
void M13FX_DS2_Cfg_Parameters (WORD32 port,
Y_S_ds2_prm_choice* param)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**param** DS2 configuration parameters address.

**RETURN:** None.

### 10.2.3 M13FX_DS1_Cfg_Parameters

**NAME:** M13FX_DS1_Cfg_Parameters

**DESCRIPTION:** This function updates the device ds1 configuration parameters according to the parameter choice (YES/NO) passed in the parameter choice structure. This configuration will be active when the Set command will be issued.

**PROTOTYPE:**

```
void M13FX_DS1_Cfg_Parameters (WORD32 port,
Y_S_ds1_prm_choice* param)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**param** DS1 configuration parameters address.

**RETURN:** None.

### 10.2.4 M13FX_Feac_Cfg_Parameters

**NAME:** M13FX_Feac_Cfg_Parameters

**DESCRIPTION:** This function updates the device FEAC configuration parameters according to the parameter choice (YES/NO) passed in the parameter choice structure. This configuration will be active when the channel will be activated.

**PROTOTYPE:**

```
void M13FX_Feac_Cfg_Parameters (WORD32 port,
Y_S_feac_prm_choice* param)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**param** FEAC configuration parameters address.

**RETURN:** None.

### 10.2.5 M13FX_Mdl_Cfg_Parameters

**NAME:** M13FX_Mdl_Cfg_Parameters

**DESCRIPTION:** This function updates the device MDL configuration parameters according to the parameter choice (YES/NO) passed in the parameter choice structure. This configuration will be active when the channel will be activated.

**PROTOTYPE:**

```
void M13FX_Mdl_Cfg_Parameters (WORD32 port,
Y_S_mdl_prm_choice* param)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**param** MDL configuration parameters address.

**RETURN:** None.

### 10.2.6 M13FX_Misc_Cfg_Parameters

**NAME:** M13FX_Misc_Cfg_Parameters

**DESCRIPTION:** This function updates the device MISC configuration parameters according to the parameter choice (YES/NO) passed in the parameter choice structure. This configuration will be active when the Set command will be issued.

**PROTOTYPE:**

```
void M13FX_Misc_Cfg_Parameters (WORD32 port,
Y_S_misc_prm_choice* param)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**param** MISC configuration parameters address.

**RETURN:** None.

### 10.2.7 M13FX_DS3_Cfg_Get_Parameters

**NAME:** M13FX_DS3_Cfg_Get_Parameters

**DESCRIPTION:** This function returns the current DS3 parameters configuration of the M13FX. The function updates the configuration parameters structure provides by the caller. The values returned are those updated with the M13FX_DS3_Cfg_Parameters function.

**PROTOTYPE:**

```
void M13FX_DS3_Cfg_Get_Parameters (WORD32 port,
Y_S_ds3_prm_choice* param)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**param** DS3 configuration parameters address.

**RETURN:** Updates the DS3 configuration parameters structure with the current values.

### 10.2.8 M13FX_DS2_Cfg_Get_Parameters

**NAME:** M13FX_DS2_Cfg_Get_Parameters

**DESCRIPTION:** This function returns the current DS2 parameters configuration of the M13FX. The function updates the configuration parameters structure provides by the caller. The values returned are those updated with the M13FX_DS2_Cfg_Parameters function.

**PROTOTYPE:**

```
void M13FX_DS2_Cfg_Get_Parameters (WORD32 port,
Y_S_ds2_prm_choice* param)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**param** DS2 configuration parameters address.

**RETURN:** Updates the DS2 configuration parameters structure with the current values.

### 10.2.9 M13FX_DS1_Cfg_Get_Parameters

**NAME:** M13FX_DS1_Cfg_Get_Parameters

**DESCRIPTION:** This function returns the current DS1 parameters configuration of the M13FX. The function updates the configuration parameters structure provides by the caller. The values returned are those updated with the M13FX_DS1_Cfg_Parameters function.

**PROTOTYPE:**

```
void M13FX_DS1_Cfg_Get_Parameters (WORD32 port,
Y_S_ds1_prm_choice* param)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**param** DS1 configuration parameters address.

**RETURN:** Updates the DS1 configuration parameters structure with the current values.

### 10.2.10 M13FX_Feac_Cfg_Get_Parameters

**NAME:** M13FX_Feac_Cfg_Get_Parameters

**DESCRIPTION:** This function returns the current FEAC parameters configuration of the M13FX. The function updates the configuration parameters structure provides by the caller. The values returned are those updated with the M13FX_Feac_Cfg_Parameters function.

**PROTOTYPE:**

```
void M13FX_Feac_Cfg_Get_Parameters (WORD32 port,
Y_S_feac_prm_choice* param)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**param** FEAC configuration parameters address.

**RETURN:** Updates the FEAC configuration parameters structure with the current values.

### 10.2.11 M13FX_Mdl_Cfg_Get_Parameters

**NAME:** M13FX_Mdl_Cfg_Get_Parameters

**DESCRIPTION:** This function returns the current MDL parameters configuration of the M13FX. The function updates the configuration parameters structure provides by the caller. The values returned are those updated with the M13FX_Mdl_Cfg_Parameters function.

**PROTOTYPE:**

```
void M13FX_Mdl_Cfg_Get_Parameters (WORD32 port,
Y_S_mdl_prm_choice* param)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**param** MDL configuration parameters address.

**RETURN:** Updates the MDL configuration parameters structure with the current values.

### 10.2.12 M13FX_Misc_Cfg_Get_Parameters

**NAME:** M13FX_Misc_Cfg_Get_Parameters

**DESCRIPTION:** This function returns the current MISC parameters configuration of the M13FX. The function updates the configuration parameters structure provides by the caller. The values returned are those updated with the M13FX_MISC_Cfg_Parameters function.

**PROTOTYPE:**

```
void M13FX_Misc_Cfg_Get_Parameters (WORD32 port,
Y_S_misc_prm_choice* param)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**param** Test Unit configuration parameters address.

**RETURN:** Updates the MISC configuration parameters structure with the current values.

### 10.2.13 M13FX_Set_Cfg_Parameters

**NAME:** M13FX_Set_Cfg_Parameters

**DESCRIPTION:** This function programs the device configuration registers according to the choices selected with the cfg commands. The function must be performed when the device is deactivated.

**PROTOTYPE:** `void M13FX_Set_Cfg_Parameters (WORD32 port)`

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**RETURN:** None.

### 10.2.14 M13FX_Reset_Cfg_Parameters

**NAME:** M13FX_Reset_Cfg_Parameters

**DESCRIPTION:** This function reprograms the device configuration registers with the default parameters.The function must be performed when the device is deactivated.

**PROTOTYPE:**

```
void M13FX_Reset_Cfg_Parameters (WORD32 port)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**RETURN:** None.

## 10.3 ACTIVATION / DEACTIVATION / STATUS

### 10.3.1 M13FX_Device_Activate

**NAME:** M13FX_Device_Activate

**DESCRIPTION:** This function activates the device with the current configuration parameters set. The interrupts are turn on and the Blue Alarm is released at the DS3 interface. The device is operational.

**PROTOTYPE:**

```
void M13FX_Device_Activate (WORD32 port)
```

**PARAMETERS:**

| | |
|---|---|
| **port** | Device Number.<br>0 to MAX_DEVICE_NUMBER. |

**RETURN:** None.

### 10.3.2 M13FX_Device_Deactivate

**NAME:** M13FX_Device_Deactivate

**DESCRIPTION:** This function deactivates the device The interrupts are turn off and the Blue Alarm is generated at the DS3 interface. The device is non operational.

**PROTOTYPE:**

```
void M13FX_Device_Deactivate (WORD32 port)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**RETURN:** None.

### 10.3.3 M13FX_Device_Status

**NAME:** M13FX_Device_Status

**DESCRIPTION:** This function provides the status of the M13FX device. It provides a global view of the M13FX device at the different levels.

**PROTOTYPE:**
```
void M13FX_Device_status (WORD32 port,
Y_S_device_status* psta)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**psta** Device Status structure address.

**.ds3** DS3 interface status:
clock, error counters, framing, multiplexer mode, feac, mdl, loopbacks alarms, aic bit.

**.ds2** DS2 interface status:
framer mode, alarms, loopbacks and loopcode

**.ds1** DS1 interface status:
T1 AIS alarms, loopbacks and loopcode.

**.tu** Test Unit status:
generator, framer.

**.map** Mapper status:
receive line,
transmit line.

**RETURN:** The data structure is updated with the current device status.

### 10.3.4 ALARMS

### 10.3.5 M13FX_DS3_Alarm

**NAME:** M13FX_DS3_Alarm

**DESCRIPTION:** This function allows to generate and monitor the DS3 alarms.

**PROTOTYPE:**

```
void M13FX_DS3_Alarm(WORD32 port, Y_S_ds3_alarm* ds3_alarm)
```

**PARAMETERS:**

| | |
|---|---|
| **port** | Device Number.<br>0 to MAX_DEVICE_NUMBER. |
| **ds3_alarm** | DS3 alarm parameter address. |
| **.action** | action.<br>SET, RESET, STATUS. |
| **.alarm** | Alarm selection.<br>ALRM_AIS, ALRM_IDLE, ALRM_RA. |
| **.bmstat** | Bit map status. D3RSTAT register. |
| **.ais** | ON / OFF. |
| **.idle** | ON / OFF. |
| **.rai** | ON / OFF. |
| **.oof** | ON / OFF. |
| **.los** | ON / OFF. |
| **.lfa** | ON / OFF. |
| **.lais** | ON / OFF. |
| **.lxbit** | ON / OFF. |
| **.lidle** | ON / OFF. |

**RETURN:** The alarm structure is updated with the current DS3 alarms status for the STATUS action.

### 10.3.6 M13FX_DS2_Alarm

**NAME:** M13FX_DS2_Alarm

**DESCRIPTION:** This function allows to generate and monitor the DS2 alarms.

**PROTOTYPE:**
```
void M13FX_DS2_Alarm(WORD32 port, Y_S_ds2_alarm* ds2_alarm)
```

**PARAMETERS:**

| | |
|---|---|
| **port** | Device Number.<br>0 to MAX_DEVICE_NUMBER. |
| **ds2_alarm** | DS2 alarm parameter address |
| **.tn** | Tributary number.<br>1 to 7 |
| **.action** | Action.<br>SET, RESET, STATUS. |
| **.alarm** | Alarm selection.<br>ALRM_AIS, ALRM_IDLE, ALRM_RA. |
| **.bmstat** | Bit map status. D2RSTAT register. |
| **.mode** | mode. E1, T1. |
| **.ais** | ON / OFF. |
| **.rai** | ON / OFF. |
| **.oof** | ON / OFF. |
| **.los** | ON / OFF. |
| **.lfa** | ON / OFF. |
| **.lais** | ON / OFF. |
| **.lra** | ON / OFF. |

**RETURN:** The alarm structure is updated with the current DS2 alarms status for the STATUS action.

### 10.3.7 M13FX_DS1_Alarm

**NAME:** M13FX_DS1_Alarm

**DESCRIPTION:** This function allows to generate and monitor the DS1 alarms.

**PROTOTYPE:**
```
void M13FX_DS1_Alarm(WORD32 port, Y_S_ds1_alarm* ds1_alarm)
```

**PARAMETERS:**

| | |
|---|---|
| **port** | Device Number.<br>0 to MAX_DEVICE_NUMBER. |
| **ds1_alarm** | DS1 alarm parameter address |
| **.tn** | Tributary number.<br>1 to 28. |
| **.action** | Action.<br>SET, RESET, STATUS. |
| **.alarm** | Alarm selection.<br>ALRM_T1AIS,ALRM_T3AIS. |
| **.lh1ais** | DS1 tn 28 - 17 (toward DS1 interface). |
| **.ll1ais** | DS1 tn 16 - 1(toward DS1 interface). |
| **.lh3ais** | DS3 tn 28 - 17(toward DS3 interface). |
| **.ll3ais** | DS3 tn 16 - 1(toward DS3 interface). |

**RETURN:** The alarm structure is updated with the current DS1 alarms status for the STATUS action.

## 10.4 LOOPBACKS

### 10.4.1 M13FX_DS3_Loopback

**NAME:** M13FX_DS3_Loopback

**DESCRIPTION:** This function allows to set and monitor the DS3 loopback

**PROTOTYPE:**
```
void        M13FX_DS3_Loopback(WORD32        port,
Y_S_ds3_loopback* plbk)
```

**PARAMETERS:**

| | |
|---|---|
| **port** | Device Number.<br>0 to MAX_DEVICE_NUMBER. |
| **plbk** | DS3 loopback parameters address. |
| **.action** | Action.<br>SET, RESET, STATUS. |
| **.lback** | Loopback selection.<br>LOCAL_LBK,REMOTE_LBK. |
| **.local** | ON/OFF. |
| **.remote** | ON/OFF. |

**RETURN:** The loopback structure is updated with the current DS3 loopbacks status for the STATUS action.

### 10.4.2 M13FX_DS2_Loopback

**NAME:** M13FX_DS2_Loopback

**DESCRIPTION:** This function allows to set and monitor the DS2 loopback . Remote loopback, loopcode generation and detection.

**PROTOTYPE:**

```
void        M13FX_DS2_Loopback(WORD32        port,
Y_S_ds2_loopback* plbk)
```

**PARAMETERS:**

| | |
|---|---|
| **port** | Device Number.<br>0 to MAX_DEVICE_NUMBER. |
| **plbk** | DS2 loopback parameters address. |
| **.tn** | Tributary number.<br>1 to 7. |
| **.action** | Action.<br>SET, RESET, STATUS. |
| **.lback** | Loopback selection.<br>REMOTE_LBK, REQLPC_LBK. |
| **.remote** | ON/OFF. |
| **.lloopc** | ON/OFF. |
| **.rloopc** | ON/OFF. |

**RETURN:** The loopback structure is updated with the current DS2 loopbacks status for the STATUS action.

### 10.4.3 M13FX_DS1_Loopback

**NAME:** M13FX_DS1_Loopback

**DESCRIPTION:** This function allows to set and monitor the DS1 loopback . Remote/local loopback, loopcode generation and detection.

**PROTOTYPE:**

```
void        M13FX_DS1_Loopback(WORD32        port,
Y_S_ds1_loopback* plbk)
```

**PARAMETERS:**

| | |
|---|---|
| **port** | Device Number.<br>0 to MAX_DEVICE_NUMBER. |
| **plbk** | DS1 loopback parameters address. |
| **.tn** | Tributary number.<br>1 to 28. |
| **.action** | Action.<br>SET, RESET, STATUS. |
| **.lback** | Loopback selection.<br>REMOTE_LBK, LPCREQ_LBK. |
| **.localh** | DS1/E1 tn 27 - 16. |
| **.locall** | DS1/E1 tn 15 - 0. |
| **.remoteh** | DS1/E1 tn 27 - 1. |
| **.remotel** | DS1/E1 tn 15 - 0. |
| **.lloopch** | DS1/E1 tn 27 - 1. |
| **.lloopcl** | DS1/E1 tn 15 - 0. |
| **.rloopch** | DS1/E1 tn 27 - 1. |
| **.rloopcl** | DS1/E1 tn 15 - 0. |

**RETURN:** The loopback structure is updated with the current DS1 loopbacks status for the STATUS action.

## 10.5 FEAC CHANNEL

### 10.5.1 M13FX_Feac_Open

**NAME:** M13FX_Feac_Open

**DESCRIPTION:** This function activates the FEAC channel with the current FEAC configuration parameters set (filter mode, reception mode and reception fifo threshold level). The caller provides the handle supporting the data buffer reception and the feac status data structure in order to get the FEAC parameters activation.

**PROTOTYPE:**

```
void M13FX_Feac_Open (WORD32port, BHANDLE*handle,
Y_S_feac_status* pprm)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**handle** Data reception buffer handle.
The handle contains the address and the length of the reception buffer in the selected channel.

**pprm** FEAC status structure address.

The structure is updated with the current setting.

**RETURN:** The FEAC status structure is updated with the current activation parameters.

### 10.5.2 M13FX_Feac_Close

**NAME:** M13FX_Feac_Close

**DESCRIPTION:** This function deactivates the FEAC channel. The current resources are released prior to close the channel.

**PROTOTYPE:** `void M13FX_Feac_Close (WORD32 port)`

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**RETURN:** None.

### 10.5.3 M13FX_Feac_Status

**NAME:** M13FX_Feac_Status

**DESCRIPTION:** This function provides the current status of the FEAC channel. The caller provides the data structure which will be updated with the currents values.

**PROTOTYPE:**

```
void        M13FX_Feac_Status(WORD32        port,
Y_S_feac_status* pprm)
```

**PARAMETERS:**

| | |
|---|---|
| **port** | Device Number.<br>0 to MAX_DEVICE_NUMBER. |
| **pprm** | FEAC status structure address. |
| **.Ref** | FEAC channel reference. |
| **.BufferSize** | Reception buffer length. |
| **.Threshold** | Receive FIFO threshold level. |
| **.FilterMode** | Filter mode. |
| **.ReceiveMode** | Receive mode. |
| **.Rsend** | Repeat transmit status.<br>FF, no transmission pending,<br>0xxxxxx0, BOM code transmission pending. |

**RETURN:** The FEAC status structure is updated with the current FEAC channel status.

### 10.5.4 M13FX_Feac_Send

**NAME:** M13FX_Feac_Send

**DESCRIPTION:** This functions allows to send a data buffer containing BOM codes. The User provides the BOM code part of the FEAC codeword. The synchronization byte (0xff) is inserted automatically by the device driver. The BOM code must have the following pattern "0xxxxxxx0" otherwise the BOM code is discarded by the transmitter. parameters activation..

**PROTOTYPE:**
```
void M13FX_Feac_Send(WORD32 port, BHANDLE* handle)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**handle** Data transmission handle.
The handle contains the address and the length of the data to transmit in the selected channel.

**RETURN:** None.

### 10.5.5 M13FX_Feac_RSend

**NAME:** M13FX_Feac_RSend

**DESCRIPTION:** This functions allows to send repeatedly a BOM code. The User provides the BOM code part of the FEAC codeword. The synchronization byte (0xff) is inserted automatically by the device driver. The BOM code must have the following pattern "0xxxxxxx0" other wise the BOM code is discarded by the transmitter.

**PROTOTYPE:**

```
void M13FX_Feac_RSend(WORD32 port, Y_S_feac_rsend* param)
```

**PARAMETERS:**

| | |
|---|---|
| **port** | Device Number.<br>0 to MAX_DEVICE_NUMBER. |
| **param** | BOM code transmission structure address. |
| **.cmde** | Command.<br>ON/OFF. |
| **.code** | BOM code to transmit.<br>0xxxxxx0. |

**RETURN:** None.

## 10.6 MDL CHANNEL

### 10.6.1 M13FX_Mdl_Open

**NAME:** M13FX_Mdl_Open

**DESCRIPTION:** This function Activates the MDL channel with the current MDL configuration parameters set(filter mode, reception mode and reception fifo threshold level). The caller provides the handle supporting the data buffer reception and the Mdl status data structure in order to get the Mdl parameters activation. parameters activation.

**PROTOTYPE:**

```
void M13FX_Mdl_Open(WORD32 port, BHANDLE*handle,
Y_S_mdl_status* pprm)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**handle** Data reception buffer handle.
The handle contains the address and the length of the reception buffer in the selected channel.

**pprm** Mdl status structure address.

status structure updated with the current setting.

**RETURN:** The MDL status structure is updated with the current MDL channel activation parameters.

### 10.6.2 M13FX_Mdl_Close

**NAME:** M13FX_Mdl_Close

**DESCRIPTION:** This function deactivates the MDL channel. The current resources are released prior to close the channel.

**PROTOTYPE:**

```
void M13FX_Mdl_Close (WORD32 port)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**RETURN:** None.

### 10.6.3 M13FX_Mdl_Status

**NAME:** M13FX_Mdl_Status

**DESCRIPTION:** This function provides the current status of the MDL channel. The caller provides the data structure which will be updated with the currents values.

**PROTOTYPE:**

```
void M13FX_Mdl_Status (WORD32 port, Y_S_mdl_status* pprm)
```

**PARAMETERS:**

| | |
|---|---|
| **port** | Device Number.<br>0 to MAX_DEVICE_NUMBER. |
| **pprm** | MDL status structure address. |
| **.Ref** | MDL channel reference. |
| **.BufferSize** | Reception buffer length. |
| **.Threshold** | Receive FIFO threshold level. |
| **.CheckCrc** | Receive Check CRC. |
| **.Riftf** | Receive inter frame status change. |
| **.TransferCrc** | Receive transfer CRC. |
| **.GenCrc** | Transmit, CRC generation. |
| **.ShareFlag** | Transmit, Share Flag. |
| **.statistics** | MDL channel statistics counters. |

**RETURN:** The MDL status structure is updated with the current MDL channel activation parameters.

### 10.6.4 M13FX_Mdl_Send

**NAME:** M13FX_Mdl_Send

**DESCRIPTION:** This functions allows to send a HDLC frame on the MDL channel.

**PROTOTYPE:** `void M13FX_Mdl_Send(WORD32 port, BHANDLE*handle)`

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**handle** Data transmission handle.
The handle contains the address and the length of the data to transmit in the selected channel.

**RETURN:** None.

## 10.7 ERROR COUNTERS

### 10.7.1 M13FX_Error_Counters

**NAME:** M13FX_Error_Counters

**DESCRIPTION:** This function permits to get the current values of the error counters (DS3 and DS2).It reads the value latched by the ONE second interrupt. The caller provides the error counters structure which is updated by the function with the latest values latched by the one second interrupt.

**PROTOTYPE:**
```
void        M13FX_Error_Counters(WORD32        port,
Y_S_error_counters* pparam)
```

**PARAMETERS:**

| | |
|---|---|
| **port** | Device Number.<br>0 to MAX_DEVICE_NUMBER. |
| **pparam** | Error counters structure address. |
| **.action** | Action.<br>READ_COUNTERS,<br>RESET_COUNTERS,<br>GET_AND_CLEAR_COUNTERS,<br>GET_AND_NOT_CLEAR_COUNTERS. |
| **.counters** | DS3 and DS2 errors counters. |

**RETURN:** The ERROR COUNTERS structure is updated with the current values stored in the device context.

## 10.8 MAPPER

### 10.8.1 M13FX_Map_Get_Configuration

**NAME:** M13FX_Map_Get_Configuration

**DESCRIPTION:** This functions return the current mapper configuration. The caller provides the configuration structure which is updated with the choices memorized in the device context.

**PROTOTYPE:**

```
void M13FX_Map_Get_Configuration (WORD32 port,
Y_S_mapper_cfg* pparam)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**pparam** Mapper configuration structure address.

**.rxt [32]** Receive tributary (Serial Interface to tributary).
contains the serial interface assigned to the tributary.
0 to 31.

**.txt [32]** Transmit tributary (Tributary to serial interface).
Contains the serial interface assigned to the tributary.
0 to 31.

**RETURN:** The MAPPER structure is updated with the current mapper configuration.

### 10.8.2 M13FX_Map_Set_Configuration

**NAME:** M13FX_Map_Set_Configuration

**DESCRIPTION:** This functions updates the mapper configuration. The caller provides the mapper configuration structure.

**PROTOTYPE:**
```
void M13FX_Map_Set_Configuration (WORD32 port,
Y_S_mapper_cfg* pparam)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**pparam** Mapper configuration structure address.

**.rxt [32]** Receive tributary (Serial Interface to tributary).
contains the serial interface assigned to the tributary.
0 to 31.

**.txt [32]** Transmit tributary (Tributary to serial interface).
Contains the serial interface assigned to the tributary.
0 to 31.

**RETURN:** None.

### 10.8.3 M13FX_Map_Reset_Configuration

**NAME:** M13FX_Map_Reset_Configuration

**DESCRIPTION:** This functions updates the mapper configuration with the default parameters.

**PROTOTYPE:** `void M13FX_Map_Reset_Configuration (WORD32 port)`

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**RETURN:** None.

## 10.9 TEST UNIT

### 10.9.1 M13FX_TU_Set_Generation

**NAME:** M13FX_TU_Set_Generation

**DESCRIPTION:** This function configures the Test Unit Generator for a fixed pattern or PRBS generation. The function stops eventually the generator in both directions reception and transmission. It sets the generator according to the parameters. The generator must be started with the "start generator" command. The caller passes the generator parameters structure.

**PROTOTYPE:**

```
void        M13FX_TU_Set_Generation        (WORD32
w32Port,Y_S_tu_generator* param)
```

**PARAMETERS:**

| | |
|---|---|
| **port** | Device Number.<br>0 to MAX_DEVICE_NUMBER. |
| **param** | Generator configuration structure address. |
| **.mode** | Generator mode.<br>FIXED_PATTERN,PRBS. |
| **.pattern** | Generator pattern.<br>32-bit pattern in FIXED_PATTERN mode.<br>5-bit Feedback Tap in PRBS mode. PRBS shift register input bit 0 is XOR of shift register bits LEN and FBT. |
| **.len** | Generator Pattern length (bits).<br>1 to 32. |
| **.enable_zs** | Enable/disable zero suppression. where a '1' bit is inserted at the output if the next 14 bits in the shift register are '0'. It must be enabled for QRSS pattern generation.<br>TRUE, FALSE. |

**RETURN:** The GENERATOR structure is updated with the current setting.

### 10.9.2 M13FX_TU_Start_Generation

**NAME:** M13FX_TU_Start_Generation

**DESCRIPTION:** This function starts the test unit generator in the current generator mode and test point insertion. Prior to start the generator the test unit must be configured with the "set generation" command the test point insertion selected with the "Test Point Insertion" command. The data insertion is ENABLED.

**PROTOTYPE:**
```
void M13FX_TU_Start_Generation (WORD32 w32Port,
Y_S_tu_generator* param)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**param** Generator configuration structure address.

**RETURN:** The GENERATOR structure is updated with the current setting.

### 10.9.3 M13FX_TU_Stop_Generation

**NAME:** M13FX_TU_Stop_Generation

**DESCRIPTION:** This function stops the test unit generation. The current setting and test point insertion are kept BUT the data insertion is DISABLED

**PROTOTYPE:** `void M13FX_TU_Stop_Generation (WORD32 w32Port)`

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**RETURN:** None.

### 10.9.4 M13FX_TU_Test_Point

**NAME:** M13FX_TU_Test_Point

**DESCRIPTION:** This function programs the Insertion Test Point. This function has to be issued before to start the test unit generator. The insertion is ALWAYS done toward the DS3 interface.

**PROTOTYPE:**

```
void M13FX_TU_Test_Point(WORD32 w32Port, Y_S_tu_tp*
param)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**param** Test point insertion structure address.

**.tp** Test point insertion.
TU_LOOPBACK, TU_DS1_TRIBUTARY, TU_DS2_TRIBUTARY, TU_DS2_PAYLOAD, TU_DS3_PAYLOAD.

**.tn** Tributary number.

1 to 7 for DS2 tributary.

1 to 28 for DS1 tributary.

**RETURN:** The TEST POINT structure is updated with the current setting.

### 10.9.5 M13FX_TU_Error_Insertion

**NAME:** M13FX_TU_Error_Insertion

**DESCRIPTION:** This function allows to insert bit errors in test unit data stream.

**PROTOTYPE:**
```
void    M13FX_TU_Error_Insertion(WORD32    w32Port,
Y_E_tu_err_generator *option)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**option** Error insertion option address.
INSERT_NO_ERRORS (0),
INSERT_NO_ERRORS +1, $10^{-1}$ error rate insertion,
INSERT_NO_ERRORS +2, $10^{-2}$ error rate insertion,
to,
INSERT_NO_ERRORS +7, $10^{-7}$ error rate insertion,
INSERT_SINGLE_BIT_ERROR (0xff), single error.

**RETURN:** None.

### 10.9.6 M13FX_TU_Receive_Pattern

**NAME:** M13FX_TU_Receive_Pattern

**DESCRIPTION:** This function provides the current state of the Test Unit receiver's 32-bit fixed pattern generator.This function is only used in fixed pattern generator mode. If the receiver is synchronized, the fixed-pattern being received is provided according to the current length of the programmed fixed-pattern.

**PROTOTYPE:**

```
void        M13FX_TU_Receive_Pattern        (WORD32
w32Port,Y_S_tu_rx_pattern* pparam)
```

**PARAMETERS:**

| | |
|---|---|
| **port** | Device Number.<br>0 to MAX_DEVICE_NUMBER. |
| **pparam** | Receive pattern structure address. |
| **.pattern** | 32-bit fixed-pattern currently received from the TU receiver. |
| **.len** | length of the received pattern. |

**RETURN:** The GENERATOR RECEIVE PATTERN structure is updated with the current values.

### 10.9.7 M13FX_TU_Receive_Counter

**NAME:** M13FX_TU_Receive_Counter

**DESCRIPTION:** This function provides the current receive counters values. We have the test unit receive Bit Count which counts the received bits when the receiver is enabled and the Test Unit Receive Error Counter which counts the errors detected since the receiver was enabled.
The user provides a counters structure which will be updated with the current values of both counters.
The background counters are copied to the foreground counters and then provides to the user.

**PROTOTYPE:**

```
void M13FX_TU_Receive_Counter (WORD32 w32Port,
Y_S_tu_rcv_counters* pcounter)
```

**PARAMETERS:**

| | |
|---|---|
| **port** | Device Number.<br>0 to MAX_DEVICE_NUMBER. |
| **pcounter** | Generator counters structure address which is updated with the current counters values. |
| **.bbit** | Received bit count since the last update. |
| **.error** | Received bit error count since the last update. |

**RETURN:** The GENERATOR COUNTERS structure is updated with the current counter values.

### 10.9.8 M13FX_TU_Set_Framer

**NAME:** M13FX_TU_Set_Framer

**DESCRIPTION:** This function configures the Test Unit Framer. It can be used in T1/E1 mode with the relevant frame format. The framer is used in order to generate framed patterns (fixed or random)

**PROTOTYPE:**
```
void M13FX_TU_Set_Framer (WORD32 w32Port, Y_S_tu_framer* param)
```

**PARAMETERS:**

| | |
|---|---|
| **port** | Device Number.<br>0 to MAX_DEVICE_NUMBER. |
| **param** | Framer configuration structure address. |
| **.mode** | Framer mode.<br>TU_T1,TU_E1. |
| **.format** | Framing.<br>TU_ESF, TU_SF,<br>TU_DOUBLE_FRAME, TU_MULTI_FRAME. |
| **.rai** | T1 Remote Alarm format.<br>RAI_SF_BIT2, RAI_SF_FS. |

**RETURN:** The FRAMER structure is updated with the current setting.

### 10.9.9 M13FX_TU_Start_Framer

**NAME:** M13FX_TU_Start_Framer

**DESCRIPTION:** This function starts the test unit Framer in the current farmer mode. Prior to start, the framer must be configured with the "set_framer" command.

**PROTOTYPE:**
```
void M13FX_TU_Start_Framer (WORD32 w32Port, Y_S_tu_framer* param)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**param** Framer configuration structure address.

**RETURN:** The FRAMER structure is updated with the current setting.

### 10.9.10 M13FX_TU_Stop_Framer

**NAME:** M13FX_TU_Stop_Framer

**DESCRIPTION:** This function stops the test unit Framer. The current framer mode is kept.

**PROTOTYPE:**
```
void M13FX_TU_Stop_Framer(WORD32 w32Port)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**RETURN:** None.

### 10.9.11 M13FX_TU_Error_Framer

**NAME:** M13FX_TU_Error_Framer

**DESCRIPTION:** This functions allows to insert framing error and Remote Alarm

**PROTOTYPE:**
```
void M13FX_TU_Error_Framer (WORD32 w32Port,
Y_S_tu_err_framer* perror)
```

**PARAMETERS:**

| | |
|---|---|
| **port** | Device Number.<br>0 to MAX_DEVICE_NUMBER. |
| **perror** | Framer error insertion structure address. |
| **.action** | Action.<br>SET, RESET. |
| **.error insertion** | Error selection.<br>FRAMING_ERROR, CRC_ERROR, E_BIT_ERROR, RAI. |

**RETURN:** None.

### 10.9.12 M13FX_TU_Receiver_Counter_Framer

**NAME:** M13FX_TU_Receiver_Counter_Framer

**DESCRIPTION:** This function reads out the current error counters values. It copies the background counters to the foreground counters and read them in the structure provided by the caller.

**PROTOTYPE:**

```
void    M13FX_TU_Receive_Counter_Framer    (WORD32
w32Port, Y_S_tu_counters_framer* param)
```

**PARAMETERS:**

| | |
|---|---|
| **port** | Device Number.<br>0 to MAX_DEVICE_NUMBER. |
| **pcounter** | Counters structure address which is updated with the current counter values. |
| **.framing** | Framing Error counter. |
| **.crc** | CRC Error Counter. |
| **.ebit** | E-bit or Errored Bloc counter. |

**RETURN:** The FRAMER counters structure is updated with the current counter values.

## 10.10 USER FUNCTIONS CALLED by the LOW-LEVEL DEVICE DRIVER.

### 10.10.1 DRV_M13FX_Error_Counters_Event

**NAME:** DRV_M13FX_Error_Counters_Event

**DESCRIPTION:** This function is called by device driver when an Error Counter event occurs. It occurs each One Second when the ECM mode is selected in the configuration otherwise the application has to poll regularly the error counters status.the event handle data area contains the DS3 and DS2 error counters current values. In the ECM mode the values are latched automatically when the One second interrupt occurs. In the polling mode, the errors counters values are latched at EACH user request.

at the interrupt level.

**PROTOTYPE:**

```
void DRV_M13FX_Error_Counters_Event (WORD32 port,
BHANDLE* phandle)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**phandle** Handle supporting the error counters structure address.
Y_S_error_counters structure.

**RETURN:** The ERROR COUNTERS structure is updated with the current values stored in the device context.

### 10.10.2 DRV_M13FX_Sent_TxBuf

**NAME:** DRV_M13FX_Sent_TxBuf

**DESCRIPTION:** This function is called by the channel transmitter when the current transmit buffer is sent to the line. the handle contains the status transmission and the channel reference.

at the interrupt level.

**PROTOTYPE:**

```
void DRV_M13FX_Sent_TxBuf (BHANDLE* phandle)
```

**PARAMETERS:**

**phandle** Transmitted data handle.
The handle contains the address and the length of the transmitted data in the referred channel.

**RETURN:** None.

### 10.10.3 DRV_M13FX_Put_RxPool

**NAME:** DRV_M13FX_Put_RxPool

**DESCRIPTION:** This function allocates Rx buffer in the Channel Reception Link list. A number of data buffer must be available in the link list in order that the channel receiver is able to receive data from the line. The channel receiver prior to give the current received buffer to the application, requests for a new data buffer reception. If there is no buffer reception available, the receiver kept the current received buffer and re initialize the reception. Therefore the received message is lost, a statistic counter is updated indicating the number of data messages ignored. In the other case, the receiver is initialized with the allocated buffer reception AND then the current received buffer is sent to the application. The buffer reception pool is created when the channel is open. A link list contains the number of buffer by the windows buffer parameter. the window parameter indicates how many messages the receiver is able to receive without allocation in the Rx POOL. When the POOL_THRESHOLD parameter is reached (how many buffers under the window size), this function allocates buffer reception in order to update the amount of buffers till the window size is reached.

at the interrupt level.

**PROTOTYPE:**

```
void DRV_M13FX_Put_RxPool (Y_R_ref ref)
```

**PARAMETERS:**

**ref** Channel reference.

**RETURN:** None.

### 10.10.4 DRV_M13FX_Put_RxBuf

**NAME:** DRV_M13FX_Rcv_RxBuf

**DESCRIPTION:** This function is called by the channel Receiver when the current receive buffer is received from the line. The handle contains the status transmission and the channel reference.

at the interrupt level.

**PROTOTYPE:**

```
void DRV_M13FX_Rcv_RxBuf (BHANDLE* phandle)
```

**PARAMETERS:**

**phandle** Received data handle.
The handle contains the address and the length of the received data in the referred channel.

**RETURN:** None.

### 10.10.5 DRV_M13FX_Alarm_Event

**NAME:** DRV_M13FX_Alarm_Event

**DESCRIPTION:** This function is called by device driver when a DS3/DS2 alarm, DS2 /DS1 loopcode, test unit events occurs. The data area referenced by the handle contains the status of the event.

at the interrupt level.

**PROTOTYPE:**

```
void DRV_M13FX_Alarm_Event (WORD32 port,Y_E_dev_cmd
cmde, Y_R_tn gn,WORD16 status)
```

**PARAMETERS:**

**port** Device Number.
0 to MAX_DEVICE_NUMBER.

**event** Event code.
M13FX_DS3_ALARM,
M13FX_DS2_ALARM,
M13FX_DS2_LOOPBACK,
M13FX_DS1_LOOPBACK,
M13FX_TU_INDICATION.

**gn** Group number.
0 to 6: DS2 alarms and DS1 loopcode.
7: DS3 alarms and DS2 loopcode.

**status** Register status.
D3RINTC for DS3/DS2 alarms,
D3RINTL for DS2/DS1 loopcode,
TURSTAT for Test unit.

**RETURN:** None.

## 10.11 ANNEXES

### 10.11.1 M13FX LOW-LEVEL INTERFACE HEADER FILE (m13xdx.h)

#### 10.11.1.1 Common definitions

```
/*-------------------------*/
/* Configuration setting  */
/*-------------------------*/
typedef enum
{
  NO,          /* no  */
  YES          /* yes */
} Y_E_choice;

/*-------------------------*/
/* generic state          */
/*-------------------------*/
typedef enum
{
  OFF,          /* Signal is OFF */
  ON            /* Signal is ON  */
} Y_E_state;

/*-------------------------*/
/* command actions        */
/*-------------------------*/
typedef enum
{
  RESET,         /* Reset a signal     */
  SET,           /* Set a signal       */
  STATUS,        /* Get a status       */
  CFG            /* Configure a signal */
} Y_E_action;

/*-------------------------*/
/* Alarm selection        */
/*-------------------------*/
typedef enum
{
  ALRM_AIS,      /* Alarm Indication Signal          */
  ALRM_RA,       /* Remote Alarm                     */
  ALRM_IDLE,     /* IDLE alarm                       */
  ALRM_RAI,      /* Remote Alarm Indication          */
  ALRM_OOF,      /* Out Of Frame signal              */
  ALRM_LOS,      /* Loss Of Signal                    */
```

```
  ALRM_LFA,      /* Loss of Frame Alignment           */
  ALRM_COFA,     /* Change Of frame Alignment         */
  ALRM_T1AIS,    /* T1 AIS alarm toward ds1 interface */
  ALRM_T13AIS    /* T1 AIS alarm toward ds3 interface */
} Y_E_alarm;



/*-------------------------*/
/* loopback selection     */
/*-------------------------*/
typedef enum
{
  REMOTE_LBK,   /* Remote loopback  */
  LOCAL_LBK,    /* Local loopback   */
  LPCREQ_LBK    /* Loopcode request */
} Y_E_lback;


typedef WORD8       Y_R_tn;  /* tributary number     */
#define DS2_TN_MAX  7        /* Ds2 max tributaries  */
#define DS1_TN_MAX  28       /* Ds1 max tributaries  */

typedef WORD16      Y_R_Wcount; /* 16-bit counter    */
typedef WORD32      Y_R_count;  /* 32-bit counter    */

/*-------------------------*/
/* loopcode selection     */
/*-------------------------*/
typedef enum
{
  LOOPC_FIRST_CBIT,         /* 1st C-bit  */
  LOOPC_SECOND_CBIT,        /* 2nd C-bit  */
  LOOPC_THIRD_CBIT          /* 3rd C-bit  */
}Y_E_loopc;
```

### 10.11.1.2 CONFIGURATION structures

```
/*---------------------------------*/
/* DS3 configuration parameters set */
/*---------------------------------*/
typedef struct
{
  Y_E_choice   loop_timing_mode;
  Y_E_choice   invert_clock_rx;
```

```
  Y_E_choice   invert_clock_tx;
  Y_E_choice   invert_data_rx;
  Y_E_choice   invert_data_tx;
  Y_E_choice   unipolar_mode_rx;
  Y_E_choice   unipolar_mode_tx;
  Y_E_choice   c_bit_parity;
  Y_E_choice   full_payload;
  Y_E_choice   tovhsyn_input;
  Y_E_choice   b3zs_code_polarity;
  Y_E_choice   interrupt_open_drain;
  Y_E_choice   interrupt_active_high;
  Y_E_choice   febe_error_parity;
  Y_E_choice   check_x_bit;
  Y_E_choice   multiframe_framing_two_m_bits;
  Y_E_choice   multiframe_reframing;
  Y_E_choice   f_framing_sixteen_f_bits;
  Y_E_choice   ais_detection_eight_err_per_multiframe;
  Y_E_choice   ais_framed;
  WORD8        alarm_cv;
} Y_S_ds3_prm_choice;

/*----------------------------------*/
/* DS2 configuration parameters set */
/* per DS2 tributary                */
/*----------------------------------*/
typedef struct
{
  Y_R_tn       tn;
  Y_E_choice   t1_mode;
  Y_E_choice   e1_reserved_bit_to_one;
  Y_E_choice   automatic_ais_insertion;
  Y_E_choice   multiframe_framing_three_m_bits_errors;
  Y_E_choice   f_framing_five_f_bits_errors;
  Y_E_choice   ais_detection_nine_err_per_multiframe;
  Y_E_choice   counter_mode_per_multiframes;
  WORD8        alarm_cv;
}Y_S_ds2_prm_choice;



/*---------------------------------*/
/*DS1 configuration parameters set */
/*---------------------------------*/
typedef struct
{
  Y_E_choice   invert_tributary_clock;
  Y_E_choice   invert_tributary_data;
```

```
} Y_S_ds1_prm_choice;


/*-----------------------------------*/
/* FEAC configuration parameter set */
/*-----------------------------------*/
typedef struct
{
  WORD8        bom_rx_threshold;        /* rx threshold level       */
  Y_E_choice   bom_rx_filter_mode;      /* rx filter mode           */
  Y_E_choice   bom_rx_continuous_mode;  /* rx continuous/10 packets */
}Y_S_feac_prm_choice;

/*-----------------------------------*/
/* MDL configuration parameter set */
/*-----------------------------------*/
typedef struct
{
  WORD8       rx_threshold;               /* rx threshold level       */
  Y_E_choice   rx_check_crc;               /* rx Check CRC             */
  Y_E_choice   rx_transfer_crc;           /* rx Transfer CRC into FIFO */
  Y_E_choice   tx_gen_crc;                 /* tx CRC generation        */
  Y_E_choice   tx_share_flags;             /* tx share flags           */
  Y_E_choice   rx_interframe_filling_status;/* rx inter frame          */
  Y_E_choice   polarity_active_high;       /* signal polarity          */
  Y_E_choice   invert_data;               /* signaling data invertion  */
  Y_E_choice   dma_mode;                   /* dma mode                 */
}Y_S_mdl_prm_choice;

/*-----------------------------------*/
/* MISC configuration parameters set */
/*-----------------------------------*/
typedef struct
{
  Y_E_choice   error_counter_one_second; /* Error counters mode  */
  Y_E_loopc    ds2_loopcode;             /* DS2 Loopcode request */
  Y_E_loopc    ds1_loopcode;             /* DS1 Loopcode request */
}Y_S_misc_prm_choice;
```

### 10.11.1.3 ALARM structures

```
/*-------------------------------*/
/*DS3 Alarms:                    */
/*-------------------------------*/
typedef struct
{
```

```
  Y_E_action  action;    /* SET, RESET, STATUS          */
  Y_E_alarm   alarm ;    /* alarm selection             */

  WORD16      bmstat;    /* bit map status              */
  Y_E_state   ais;       /* ais alarm(detection)        */
  Y_E_state   idle;      /* idle alarm(detection)       */
  Y_E_state   rai;       /* rai alarm(detection)        */
  Y_E_state   oof;       /* oof alarm(detection)        */
  Y_E_state   los;       /* los alarm(detection)        */
  Y_E_state   lfa;       /* lfa alarm(detection)        */
  Y_E_state   cofa;      /* cofa alarm(detection)       */
  Y_E_state   aic;       /* aic bit(detection)          */

  Y_E_state   lais;      /* local ais alarm(generation) */
  Y_E_state   lrai;      /* local rai alarm(generation) */
  Y_E_state   lidle;     /* local idle alarm(generation)*/
  Y_E_state   lxbit;     /* local xbit(generation)      */
  Y_E_state   laic;      /* local aic bit(generation)   */
} Y_S_ds3_alarm;

/*------------------------------*/
/* DS2 Alarms:                  */
/*------------------------------*/
typedef struct
{
  Y_E_action  action;    /* SET, RESET, STATUS          */
  Y_E_alarm   alarm ;    /* alarm selection             */
  Y_R_tn      tn;        /* tributary number            */

  WORD8       bmstat;    /* bit map status              */
  WORD8       mode;      /* E1/T1 mode                  */
  Y_E_state   ais ;      /* ais alarm(detection)        */
  Y_E_state   rai ;      /* rai alarm(detection)        */
  Y_E_state   oof ;      /* oof alarm(detection)        */
  Y_E_state   los ;      /* los alarm(detection)        */
  Y_E_state   lfa ;      /* lfa alarm(detection)        */
  Y_E_state   cofa;      /* cofa alarm(detection)       */

  Y_E_state   lais;      /* local ais alarm(generation) */
  Y_E_state   lrai;      /* local rai alarm(generation) */
} Y_S_ds2_alarm;


/*------------------------------*/
/* DS1 Alarms:                  */
/*------------------------------*/
typedef struct
```

```
{
  Y_E_action  action;    /* SET, RESET, STATUS            */
  Y_E_alarm   alarm ;    /* alarm selection               */
  Y_R_tn      tn;        /* tributary number              */
  WORD16      lh1ais;    /* AIS high toward DS1 interface */
  WORD16      ll1ais;    /* AIS low toward DS1 interface  */
  WORD16      lh3ais;    /* AIS high toward DS3 interface */
  WORD16      ll3ais;    /* AIS low toward DS3 interface  */
} Y_S_ds1_alarm;
```

### 10.11.1.4 LOOPBACK structures

```
/*------------------------------*/
/* DS3 loopbacks:               */
/*------------------------------*/
typedef struct
{
  Y_E_action   action;    /* SET, RESET, STATUS    */
  Y_E_lback    lback ;    /* REMOTE / LOCAL        */
  WORD8        remote;    /* remote loopback state */
  WORD8        local;     /* local loopback state  */
} Y_S_ds3_loopback;

/*------------------------------*/
/* DS2 loopback:                */
/*------------------------------*/
typedef struct
{
  Y_E_action     action;    /* SET, RESET, STATUS, CFG        */
  Y_E_lback      lback ;    /* alarm selection: only REMOTE   */
  Y_R_tn         tn;        /* tributary number               */
  WORD8          value;     /* loopcode configuration         */
  WORD8          bmstat;    /* loopcode bit map status        */

  WORD8          remote;    /* remote loopback state          */
  WORD8          lloopc;    /* local request remote loopback  */
  WORD8          rloopc;    /* remote request remote loopback */
} Y_S_ds2_loopback;


/*------------------------------*/
/* DS1 loopback:                */
/*------------------------------*/
typedef struct
{
  Y_E_action     action;    /* SET, RESET, STATUS, CFG         */
  Y_E_lback      lback;     /*  selection: REMOTE / LOCAL      */
```

```
  Y_R_tn          tn;        /* tributary number                */
  Y_E_state       value;     /* loopcode configuration          */
  WORD8           bmstat;    /* loopcode bit map status         */

  WORD16          localh;    /* local high loopback state       */
  WORD16          locall;    /* local low loopback state        */
  WORD16          remoteh;   /* remote high oopback state       */
  WORD16          remotel;   /* remote low loopback state       */
  WORD16          lloopch;   /* local  high DS1 lbc code status */
  WORD16          lloopcl;   /* local  low  DS1 lbc code status */
  WORD16          rloopch;   /* remote high DS1 lbc code status */
  WORD16          rloopcl;   /* remote low  DS1 lbc code status */
} Y_S_ds1_loopback;
```

### 10.11.1.5 STATUS structures

```
typedef enum
{
  CLK_NORMAL,  /* Normal DS3 clock mode */
  CLK_LOOP     /* Loop timing mode      */
}Y_E_clock;

typedef enum
{
  CTR_POLLING, /* Error counters polling mode            */
  CTR_1S       /* Error counters 1 Second interrupt mode */
}Y_E_ecm;

typedef enum
{
  M13_M13,     /* DS3 framer in M13 framing format          */
  M13_CBIT     /* DS3 framer in C-bit Parity framing format */
}Y_E_m13;

typedef enum
{
  M23_MUX,      /* M23 stage in Chanelized mode   */
  M23_PAYLOAD   /* M23 stage in full payload mode */
}Y_E_m23;

typedef enum
{
  STU_NONE,     /* Test Unit OFF                    */
  STU_FIXED,    /* Test Unit in Fixed-pattern mode */
  STU_PRBS,     /* Test Unit in Pseudo-random mode */
  STU_E1,       /* Test Unit Framer in E1 mode     */
  STU_T1        /* Test Unit Framer in T1 mode     */
```

```
}Y_E_stu;

typedef enum
{
  SMAP_STRAIGHT, /* all tributary lines are in straight setup */
  SMAP_MAPPED    /* almost one tributary line is mapped       */
}Y_E_smap;

#define SALR_NONE           (0 << 0)  /* No alarm/AIC activity   */
#define SALR_GENERATION     (1 << 0)  /* Alarm/AIC generation    */
#define SALR_DETECTION      (1 << 1)  /* Alarm/AIC detection     */
#define SALR_GENERATION_DS1 (1 << 2)  /* T1 AIS alarm generation */

#define SLBK_NONE           (0 << 0)  /* No loopback activity    */
#define SLBK_REMOTE         (1 << 0)  /* Remote loopback         */
#define SLBK_LOCAL          (1 << 1)  /* Local loopback          */
#define SLBK_GENERATION     (1 << 2)  /* Loopcode generation     */
#define SLBK_DETECTION      (1 << 3)  /* Loopcode detection      */

/*--------------------------------------*/
/* DEVICE STATUS:                       */
/*  Ds3 interface status                */
/*  Ds2 interface status                */
/*  Ds1 interface status                */
/*  Test Unit status                    */
/*  Tributary Mapper status             */
/*--------------------------------------*/
typedef struct
{
  Y_E_clock clock;              /* loop-timing / normal              */
  Y_E_ecm   error_counters;     /* Polling / 1 second interrupt      */
  Y_E_m13   framing;            /* M13 / C-bit parity                */
  Y_E_m23   multiplexer;        /* channelized / Full payload        */
  Y_E_state feac;               /* FEAC channel ON / OFF             */
  Y_E_state mdl;                /* MDL channel ON / OFF              */
  WORD8     aic;                /* AIC-bit detect / gen              */
  WORD8     alarms;             /* DS3 Alarms detect / gen           */
  WORD8     loopbacks;          /* DS3 Loopback Remote / Local       */
}Y_S_ds3_status;

typedef struct
{
  WORD8     mode;               /* E1/T1 mode                        */
  WORD8     alarms;             /* DS2 Alarms detect / gen           */
  WORD8     loopbacks;          /* DS2 Loopback Remote/detect/gen    */
}Y_S_ds2_status;
```

```
typedef struct
{
  WORD8     alarms;            /* T1 AIS generation, toward DS3/DS1    */
  WORD8 loopbacks;             /* DS1 Loopback Remote/Local/detect/gen */
}Y_S_ds1_status;

typedef struct
{
  Y_E_stu     generator;        /* Fixed-Pattern/PRBS/OFF    */
  Y_E_stu     framer;           /* E1/T1/OFF                 */
}Y_S_tu_status;

typedef struct
{
  Y_E_smap     rx;              /* Straight / mapped        */
  Y_E_smap     tx;              /* Straight / mapped        */
}Y_S_map_status;

/*--------------------*/
/* Device STATUS:     */
/*--------------------*/
typedef struct
{
  WORD32          port;   /* device number   */
  void*           base;   /* base address    */
  Y_S_ds3_status  ds3;    /* ds3 interface   */
  Y_S_ds2_status  ds2;    /* ds2 interface   */
  Y_S_ds1_status  ds1;    /* ds1 interface   */
  Y_S_tu_status   tu;     /* test unit       */
  Y_S_map_status  map;    /* mapper          */
}Y_S_device_status;

/*----------------------*/
/* MDL channel STATUS: */
/*----------------------*/
typedef struct
{
  Y_R_ref          Ref;               /* channel reference         */
  Y_R_size         BufferSize;        /* Rx and Tx max buffer size */
  WORD8            Threshold;         /* Reception FIFO threshold  */
  BOOL             CheckCrc;          /* Rx Check CRC              */
  BOOL             TransferCrc;       /* Rx transfer CRC           */
  BOOL             GenCrc;            /* Tx CRC generation         */
  BOOL             ShareFlag;         /* Tx Share Flag             */
  BOOL             Riftf;             /* Rx Inter Frame report     */

  Y_R_Wcount       RxValid_ct;        /* Rx Valid Frames           */
```

```
  Y_R_Wcount       RxTooLong_ct;       /* Rx Too long Frames         */
  Y_R_Wcount       RxOverflow_ct;      /* Rx Data Overflow           */
  Y_R_Wcount       RxAbort_ct;         /* Rx Abort                   */
  Y_R_Wcount       RxNotOctet_ct;      /* Rx Not Octet               */
  Y_R_Wcount       RxCrcError_ct;      /* Rx CRC Error               */
  Y_R_Wcount       RxChannelOff_ct;    /* Rx Channel Off             */
  Y_R_Wcount       RxSyncFlag_ct;      /* Rx Synchronization Flag    */
  Y_R_Wcount       RxSyncFF_ct;        /* Rx Synchronization FF      */

  Y_R_Wcount       TxValid_ct;         /* Tx Valid Frames            */
  Y_R_Wcount       TxBusy_ct;          /* Tx Txfifo Busy             */
  Y_R_Wcount       TxUnderrun_ct;      /* Tx Data underrun           */
}Y_S_mdl_status;

/*-----------------------*/
/* FEAC channel STATUS: */
/*-----------------------*/
typedef enum
{
  BOM_NORMAL_MODE,            /* receiver in normal mode */
  BOM_FILTER_MODE,            /* receiver in filter mode */
  BOM_CONTINUOUS_MODE,        /* reception continuously  */
  BOM_10PACKETS_MODE          /* reception in 10 packets */
} Y_E_bom;

typedef struct
{
  Y_E_state   cmde;           /* ON/OFF command */
  WORD8       code;           /* BOM to send    */
} Y_S_feac_rsend;

typedef struct
{
  Y_R_ref         Ref;         /* Channel reference         */
  Y_R_size        BufferSize;  /* Rx and Tx max buffer size */
  WORD8           Threshold;   /* Reception FIFO threshold  */
  Y_E_bom         FilterMode;  /* BOM filter mode           */
  Y_E_bom         ReceiveMode; /* BOM receive mode          */
  WORD8           Rsend;       /* BOM transmit repeat state */
}Y_S_feac_status;
```

### 10.11.1.6 TEST UNIT structures

```
#define TU_OFF 0xffff       /* Test Unit turn Off */

              /*---------------------------------------*/
```

```
              /* Test Unit GENERATOR                    */
              /*----------------------------------------*/
typedef enum
{
  FIXED_PATTERN,          /* fixed-pattern generation */
  PRBS                    /* Pseudo-random generation */
} Y_E_tu_mode_gen;

typedef WORD8  Y_R_tu_len;
typedef WORD16 Y_R_tu_pattern;

typedef enum
{
  TU_LOOPBACK,            /* Loopback insertion point      */
  TU_DS1_TRIBUTARY,       /* DS1 tributary insertion point */
  TU_DS2_TRIBUTARY,       /* DS2 tributary insertion point */
  TU_DS2_PAYLOAD,         /* DS2 payload insertion point   */
  TU_DS3_PAYLOAD          /* DS3 payload insertion point   */
}Y_E_tu_tp;

/*-------------------------------------------------------------*/
/* Generator error insertion:                                  */
/* INSERT_NO_ERRORS (0)         Insert NO errors               */
/* INSERT_NO_ERRORS +1          Insert 10^-1 error rate        */
/* INSERT_NO_ERRORS +2          Insert 10^-2 error rate        */
/* INSERT_NO_ERRORS +3          Insert 10^-3 error rate        */
/* INSERT_NO_ERRORS +4          Insert 10^-4 error rate        */
/* INSERT_NO_ERRORS +5          Insert 10^-5 error rate        */
/* INSERT_NO_ERRORS +6          Insert 10^-6 error rate        */
/* INSERT_NO_ERRORS +7          Insert 10^-7 error rate        */
/* INSERT_SINGLE_BIT_ERROR(0xff)Insert SINGLE error            */
/*-------------------------------------------------------------*/
typedef enum
{
  INSERT_NO_ERRORS,
  INSERT_SINGLE_BIT_ERROR =0xff
}Y_E_tu_err_generator;




/*---------------------------*/
/* Generator setup structure */
/*---------------------------*/
typedef struct
{
  Y_E_tu_mode_gen  mode;        /* Generator mode   */
```

```
  Y_R_tu_pattern   patternH;    /* pattern HIGH     */
  Y_R_tu_pattern   patternL;    /* pattern LOW      */
  Y_R_tu_len       len;         /* pattern length   */
    BOOL           zs_enable;   /* zero suppression */
}Y_S_tu_generator;

/*---------------------------------*/
/* test point insertion structure */
/*---------------------------------*/
typedef struct
{
  Y_E_tu_tp            tp;         /* test point       */
  Y_R_tn               tn;         /* tributary number */
}Y_S_tu_tp;

/*---------------------------------*/
/* Fixed-pattern receive structure */
/*---------------------------------*/
typedef struct
{
  WORD16            rx_status;  /* receiver status           */
  Y_R_tu_len        len;        /* pattern length            */
  Y_R_tu_pattern    patternH;   /* pattern HIGH              */
  Y_R_tu_pattern    patternL;   /* pattern LOW               */
  Y_E_tu_tp         tp;         /* test point                */
  Y_R_tn            tn;         /* tributary number (Ds1,Ds2) */
}Y_S_tu_rx_pattern;

/*----------------------------*/
/* receive counters structure */
/*----------------------------*/
typedef struct
{
  Y_E_tu_mode_gen mode;      /* mode                  */
  WORD16          rx_status; /* receiver status       */
  WORD8           ierr;      /* error insertion status */
  Y_R_Wcount      bbitH;     /* bit count HIGH        */
  Y_R_Wcount      bbitL;     /* bit count LOW         */
  Y_R_Wcount      errorH;    /* error count HIGH      */
  Y_R_Wcount      errorL;    /* error count HIGH      */
}Y_S_tu_rcv_counters;
              /*---------------------------------------*/
              /* Test Unit FRAMER                       */
              /*---------------------------------------*/
typedef enum
{
  TU_T1,                  /* T1 mode   */
```

```
  TU_E1                   /* E1 mode   */
} Y_E_tu_mode_framer;

typedef enum
{
  TU_ESF,                 /* T1 ESF format          */
  TU_SF,                  /* T1 SF format           */
  TU_DOUBLE_FRAME,        /* E1 Double Frame format */
  TU_MULTI_FRAME          /* E1 Double Frame format */
} Y_E_tu_format_framer;

typedef enum
{
  RAI_SF_FS,             /* T1 Remote Alarm format : Frame bit */
  RAI_SF_BIT2            /* T1 Remote Alarm format : Bit 2     */
} Y_E_tu_rai_format;

typedef enum
{
  FRAMING_ERROR,         /* Framing error insertion    */
  CRC_ERROR,             /* CRC error insertion        */
  E_BIT_ERROR,           /* E-bit error insertion      */
  RAI                    /* T1 Remote Alarm generation */
}Y_E_tu_err_framer;

/*-------------------------*/
/* Framer setup structure */
/*-------------------------*/
typedef struct
{
  Y_E_tu_mode_framer   mode;     /* framer mode            */
  Y_E_tu_format_framer format;   /* framing format         */
  Y_E_tu_rai_format    rai;      /* T1 Remote Alarm format */
}Y_S_tu_framer;

/*---------------------------*/
/* Error Insertion structure */
/*---------------------------*/
typedef struct
{
  Y_E_action            action;  /* SET, RESET             */
  Y_E_tu_err_framer     error;   /* Error selection        */
}Y_S_tu_err_framer;

/*---------------------------------*/
/* Framer error counters structure */
/*---------------------------------*/
```

```
typedef struct
{
  WORD8                rx_status; /* Framer status                */
  Y_E_tu_mode_framer   mode;      /* T1/E1                        */
  Y_E_tu_format_framer format;    /* ESF-SF/ DOUBLE -MULTI FRAME */
  WORD8                ierr;      /* error insertion status      */
  Y_R_Wcount           fec;       /* framing error counter       */
  Y_R_Wcount           crcc;      /* CRC error counter           */
  Y_R_Wcount           ebitc;     /* E-bit error counter         */
}Y_S_tu_counters_framer;
```

### 10.11.1.7 ERROR COUNTERS structures

```
/*---------------------------*/
/* error counters commands   */
/*---------------------------*/
typedef enum
{
  READ_COUNTERS,               /* Read Err. counters (context)        */
  RESET_COUNTERS,             /* Reset Err. counters (context)       */
  GET_AND_CLEAR_COUNTERS,      /* Get and clear Err. counters (Regs)  */
  GET_AND_NOT_CLEAR_COUNTERS   /* Get and notclear Err.counters(Regs) */
 } Y_E_error_selection;


/*-------------------------------*/
/* DS2 error counters structure  */
/*-------------------------------*/
typedef struct
{
  Y_R_Wcount         rfec2;
  Y_R_Wcount         rpec2;
} Y_S_DS2_err_counters;


/*-------------------------------*/
/* Error counters structure      */
/*-------------------------------*/
typedef struct
{
  Y_E_error_selection   action;   /* Error counter command          */
  Y_R_Wcount            rcve3;    /* DS3, B3ZS line code violations */
  Y_R_Wcount            rexz3;    /* DS3, Excessive Zeroes          */
  Y_R_Wcount            rfec3;    /* DS3, Framing bit errors        */
  Y_R_Wcount            rpec3;    /* DS3, Parity errors             */
  Y_R_Wcount            rcpec3;   /* DS3, Path parity errors        */
```

```
  Y_R_Wcount            rfebec3;  /* DS3, FEBE errors              */
  Y_S_DS2_err_counters  rds2errc[DS2_TN_MAX]; /* DS2 errors         */
} Y_S_error_counters;
```

#### 10.11.1.8 TRIBUTARY MAPPER structure

```
/*-----------------------------------------*/
/* MAPPER configuration parameters set     */
/*-----------------------------------------*/
typedef W ORD8  Y_R_itf;  /* Interface number */

typedef struct
{
  Y_R_itf     one;                /* Configure only one entry  */
  Y_R_itf     rxt [DS1_TN_MAX+4]; /* receive tributary side    */
  Y_R_itf     txt [DS1_TN_MAX+4]; /* transmit tributary side   */
} Y_S_mapper_cfg;
```

### 10.11.2 M13FX OPI INTERFACE HEADER FILE (opiext.h)

Refer to M13FX OPI interface chapter.

# Infineon goes for Business Excellence

“Business excellence means intelligent approaches and clearly defined processes, which are both constantly under review and ultimately lead to good operating results.
Better operating results and business excellence mean less idleness and wastefulness for all of us, more professional success, more accurate information, a better overview and, thereby, less frustration and more satisfaction.”

Dr. Ulrich Schumacher

http://www.infineon.com

Published by Infineon Technologies AG