

FRS SoC Evaluation Design Specification

This document could contain technical inaccuracies or typographical errors. TTTech Flexibilis Oy may make changes in the product described in this document at any time.

Please, email comments about this document to support@flexibilis.com.

© Copyright TTTech Flexibilis Oy 2021. All rights reserved.

Trademarks

All trademarks are the property of their respective owners.

Contents

1 About This Document	7
1.1 Conventions Used in This Document	7
2 General	9
3 FPGA Evaluation Design	10
3.1 Top Level	10
3.2 QSYS Design	10
3.2.1 FES System	12
3.2.2 HPS	14
3.3 FES System Configuration	15
3.3.1 Generic Configuration	15
3.3.2 Interface Options	17
3.3.3 Interface Configuration	18
3.3.3.1 Port Interface Type	19
3.3.3.2 Port Address Configuration	20
3.3.4 Adapter Address Configuration	21
3.4 Interface Adapters	21
3.5 Avalon/AXI Address Map	21
3.6 Compilation	23
3.6.1 Folder Structure	23
3.6.2 QSYS Generation	23
3.6.3 Quartus Project	24
4 SW Evaluation Design	25
4.1 Boot	26
4.1.1 ROM Boot Code	27
4.1.2 Preloader	27
4.1.3 U-Boot	27
4.1.4 Linux	29
4.2 Kernel and Drivers	29
4.2.1 Linux	29
4.2.2 Device Tree	29
4.2.3 flx_frs (FES)	30
4.2.3.1 Device Tree Bindings	30
4.2.3.2 Principle of Operation	34
4.2.3.3 Accessing Switch Features	35
4.2.3.4 Port Link Mode Management	35
4.2.3.5 Managing Port Forwarding Mode	35
4.2.3.6 Accessing IPO Entries	35
4.2.3.7 Accessing Port Statistics Counters	36
4.2.3.8 Accessing MAC Address Table	36
4.2.3.9 Accessing Static MAC Address Table	36
4.2.3.10 Traffic Shaping	36
4.2.3.11 Traffic Policing	37
4.2.3.12 Configuring MACsec	37
4.2.3.13 Auxiliary Network Interfaces	37
4.2.3.14 Independent Interfaces	38
4.2.4 flx_frtc (FRTC)	38
4.2.5 flx_time	39
4.2.6 flx_pio (Altera PIO)	39
4.2.7 flx_eth_mdio (Altera MDIO Core)	39
4.2.8 i2c_gpio	40
4.2.9 stmmac (EMAC)	40
4.2.10 Marvell (PHY)	41
4.3 User Space	41

4.3.1 XR7 PTP.....	41
4.3.2 XR7 Redundancy Supervision	42
4.3.3 flx_fes_lib.....	42
4.3.4 XR7 Management Software	42
4.3.4.1 XR7 FCM	42
4.3.4.2 XR7 IFM.....	43
4.3.4.3 XR7 GUI.....	43
4.3.5 SSH Server.....	43
4.3.6 Debian	43
4.4 Compilation	43
4.4.1 Toolchains	43
4.4.2 Preloader and U-Boot.....	44
4.4.3 Linux Kernel	45
4.4.4 Linux Drivers for Flexibilis IPs	45
4.4.5 Device Tree	46
4.4.6 Other Flexibilis Software	46
4.4.7 Third Party Software.....	46
4.5 SD-Card	46
5 Customization	49
5.1 Changing FES CPU Port Speed.....	49
5.2 Changing PHY address	49
5.3 FES Port without a PHY	49
5.4 Adding an FES Port	49
5.5 Change AXI Bus Type	50
6 Troubleshooting	51
6.1 Driver Loading.....	51
6.1.1 Letting Drivers Load Automatically	51
6.1.2 Loading Drivers Explicitly	51
6.1.3 Driver Load Verification	51
6.2 FES	52
6.2.1 Missing Functionality	52
6.2.2 FES Switch Register Access.....	52
6.2.3 FES Port Register Access.....	53
6.2.4 FES Port Adapter Register Access	53
6.2.5 FES Port Link Status and Speed for Copper Interfaces	53
6.2.6 Use of Correct PHY Driver	54
6.2.7 FES Port Link Status and Speed for Fiber Interfaces	55
6.2.8 SFP Module Change Detection.....	55
6.2.9 Traffic Problems	55
6.2.9.1 RGMII.....	56
6.3 FRTC.....	56
6.3.1 Checking FRTC Is Running.....	56
6.3.2 Rough FRTC Frequency Check.....	57
7 Abbreviations	58
8 References	59

Figures

Figure 1. Bit and Byte Order.....	7
Figure 2. Evaluation Design Block Diagram	10
Figure 3. SOC System Design Block Diagram	11
Figure 4. FES System with RGMII Component Block Diagram	13
Figure 5. HPS interfaces	14
Figure 6. Generics	15

Figure 7. Interface Options	17
Figure 8. FRTC/FPTS Options	18
Figure 9. Interface Configurations	19
Figure 10. Port Address Configurations	20
Figure 11. Adapter Address Configurations	21
Figure 12. Folder Structure.....	23
Figure 13. SW Evaluation Design.....	25
Figure 14. SW Evaluation Design boot process.....	27

Tables

Table 1. Avalon Address Map	23
Table 2. Use of drivers with supported boards.....	29
Table 3. FES driver API header files	35
Table 4. flx_fes_lib files	42
Table 5. SD-card structure	46

Revision History

Rev	Date	Comments
0.1	15.5.2014	Draft
0.2	2.10.2014	Typo fixes
0.3	12.2.2015	SoC EDS 14.0.2 update More information about FRS driver
1.0	12.2.2015	Approved in review
1.1	26.10.2015	Added troubleshooting chapter Driver information update
1.2	23.3.2016	Updated SW module names and descriptions. Replaced ALT_TSE adapter with SGMII/1000Base-X
1.3	3.8.2016	Updated QSYS design description. FRS Redbox was replaced by FES system. Updated flx_frs driver descriptions for new features and related flx_frs_tool command examples. Added LED control descriptions.
1.4	30.8.2016	Added support for Novtech NOV SOM CVLite and NetLeap board combination.
1.5	26.4.2021	Changed the support for FRS SoC Eval board.

1 About This Document

This document describes the evaluation design for Flexibilis Redundant Switch (FRS) [7] for Cyclone V SoC FPGA. FRS is an Ethernet switch Intellectual Property (IP) core targeted at programmable hardware platforms. FRS is a variation of Flexibilis Ethernet Switch (FES) [7].

The purpose of the Evaluation Design is to provide an FRS evaluation platform and it may be used as is or modified as required. The Evaluation Design implements a HSR/PRP RedBox with support for 1588 PTP ordinary and transparent clocks. The Evaluation Design is provided for FRS SOC Evaluation board, which is available via TTTech Flexibilis. The reference design is downloadable from www.flexibilis.com or <https://www.tttech-industrial.com/products/flexibilis/>. There is also a Reference Design for Cyclone IV GX and Cyclone V GX/GT evaluation boards, but it is not covered by this document.

Chapter 2 describes the Evaluation Design in general. Chapter 3 describes the FPGA part of the Evaluation Design, i.e. VHDL, IP and QSYS issues. Chapter 3 describes the software included in the Evaluation Design. Chapter 5 contains information for customizing the design. Chapter 6 includes troubleshooting information and debugging tips. Chapter 7 contains abbreviations and chapter 8 references.

1.1 Conventions Used in This Document

Register descriptions in this document follow these rules: Unless otherwise stated, all the bits that activate or enable something are active when their value is 1 and inactive when their value is 0. The explanation of the bit types is the following:

RO = Read Capable Only. The bits marked with RO can be read. Writing to these bits is allowed if not otherwise stated. If writing is allowed, it does not affect the value of the bit.

R/W = Read and Write capable. The bits can be read and written. Writing 1 to the bit makes its value 1. Writing 0 to the bit makes its value 0.

R/C = Read and Clear capable. The bits can be read and cleared. Writing 0 to the bit makes its value 0. Writing 1 does nothing.

R/SC = Read and Self Clear. The bits can be read. After reading bits, the value automatically returns back to 0.

R/W/SC = Read, Write and Self Clear. The bits can be read and written. Writing 0 to the bit does nothing. Writing 1 to the bit makes its value 1 for a while, but after that the value automatically returns back to 0.

The bits marked as *Reserved* should not be written anything but 0, even if they are marked as read capable only, because their function may change in future versions.

Bit and byte order used for 16, 32 and 64 registers is depicted in Figure 1.

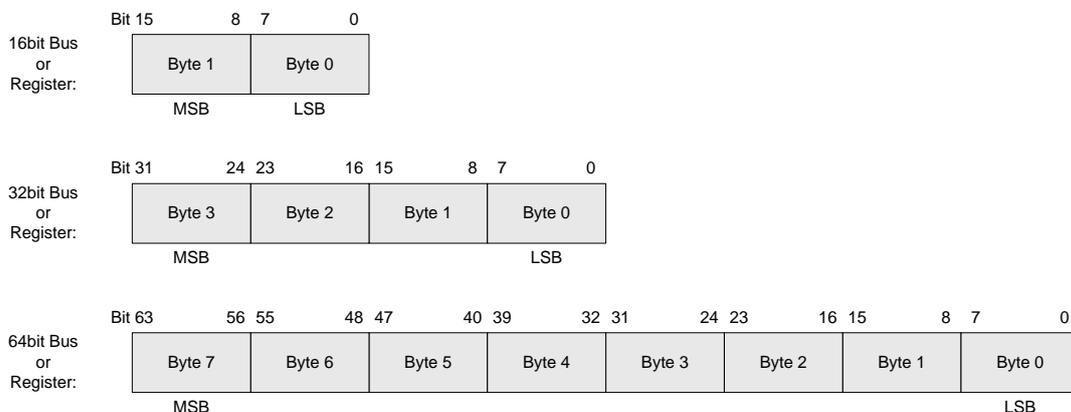


Figure 1. Bit and Byte Order

Signal names are written in document with `SignalName` style. Block names are written with Capital first letter. Pseudo code is written with `PseudoCode` style and command line commands are written with `CommandLine` style.

2 General

The FRS Evaluation Design consists of FPGA and SW design. The SW is run on ARM Cortex-A9 located in the Cyclone V Hard Processor System (HPS). The FPGA design is implemented using Altera QSYS tool that provides a graphical design tool for FPGA systems. FPGA design compilations are made using Altera Quartus II tool.

Software is built using cross-compiler toolchains. A bare-metal cross-compiler toolchain is used to build bootloaders (Preloader and U-Boot). Another cross-compiler toolchain is used to build Linux kernel, drivers and user space software. Additional GNU/Linux utilities are also needed to generate bootable SD-card image. Section 4.4.1 contains more information about the needed tools.

Version information about the tools used, the IP blocks and the SW components are listed in the Evaluation Design release notes included in the release package.

3 FPGA Evaluation Design

The VHDL and QSYS part of the design is described in this chapter.

3.1 Top Level

The Evaluation Design block diagram with external interfaces is described below.

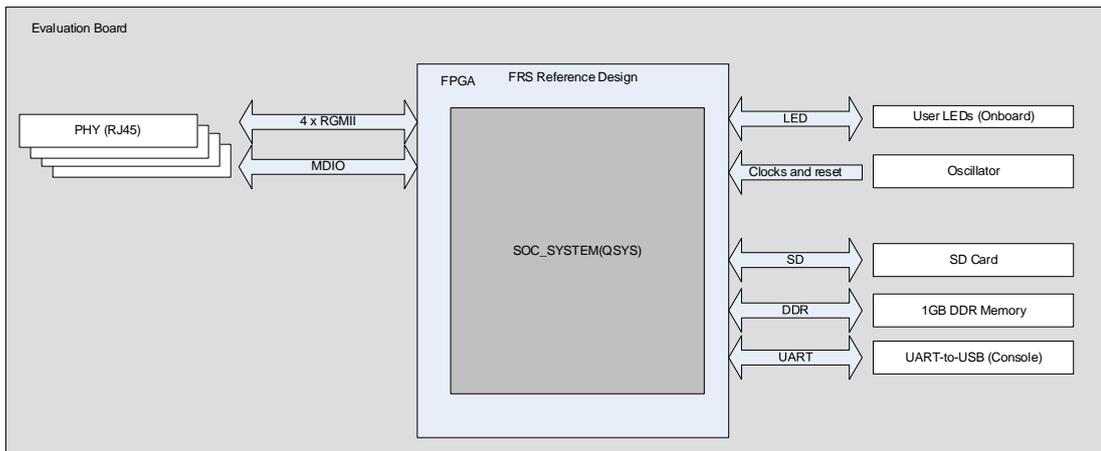


Figure 2. Evaluation Design Block Diagram

The Evaluation Design provides the following interfaces

- 4 external interfaces
 - o There are 4 local PHY's (RGMII) connected to FRS
 - o Redundant Port A
 - o Redundant Port B
 - o 2 Interlinks
- STA MDIO(MDIO for PHY management)
- Link LEDs
- DDR for 1 GB DDR memory
- SDMMC/SDIO for SD card
- UART for Console connection via UART-to-USB bridge
- Clocks and reset
 - o 50MHz or 25 MHz
 - Routed through PLL to 125MHz and 25 MHz clocks
 - Routed through PLL to SerDes clock
 - Routed directly for Avalon clock

The QSYS Design, SOC_SYSTEM is instantiated in soc_system_top.vhd top level design file.

3.2 QSYS Design

The QSYS design, soc_system, is a combination of many QSYS components and their sub components. The block diagram below describes the QSYS system in the highest level.

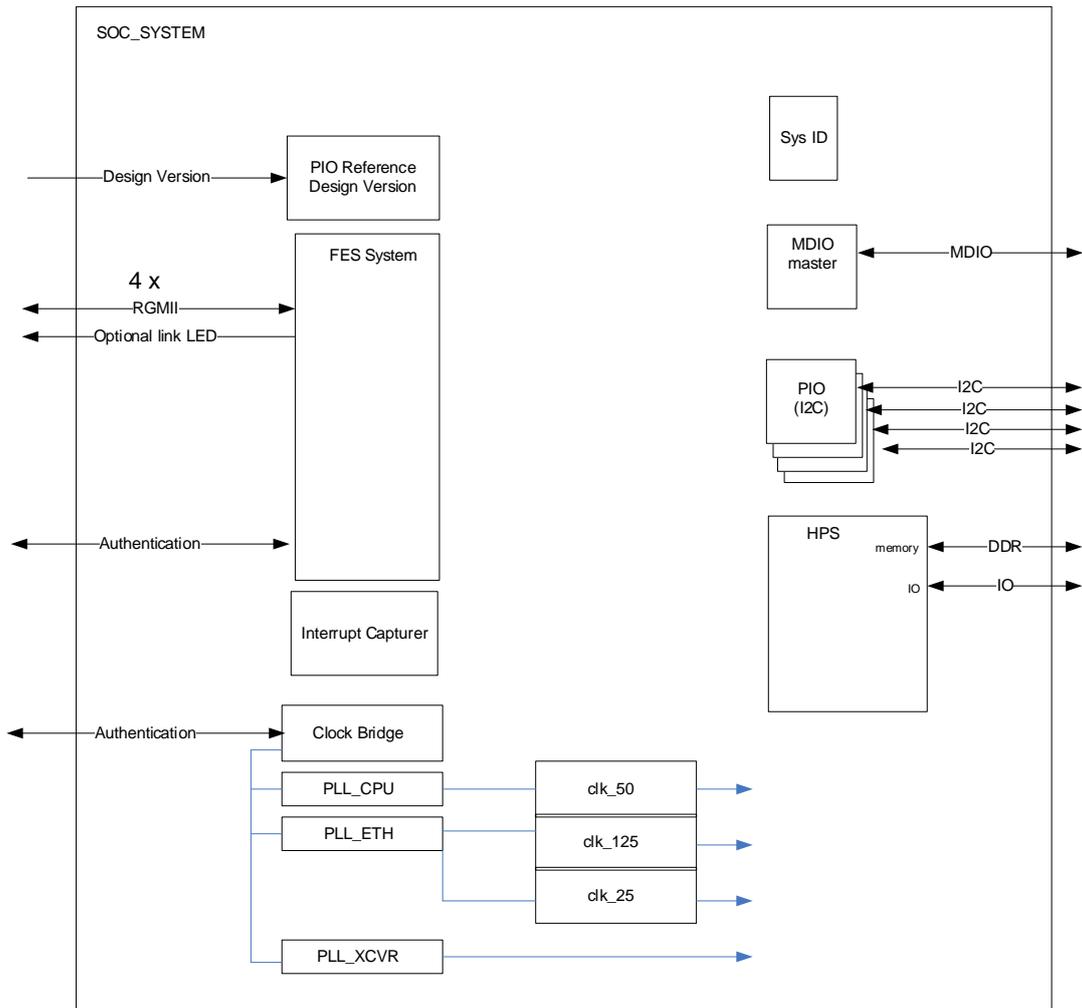


Figure 3. SOC System Design Block Diagram

Soc_system includes the following components:

1. Clock bridge
 - a. Clock bridge is used to introduce the input clock for the QSYS system
2. PLLs
 - a. PLL_CPU, 50 MHz clock for HPS and Avalon
 - b. PLL_ETH, 25 MHz for Internal MII and 125MHz for FES system clock and reconfiguration clock
 - c. PLL_XCVR, 125 MHz for transceivers
3. Clock and reset for 25, 50 and 125MHz.
 - a. Clock source components are used in QSYS designs to feed clock and resets into the QSYS system/components
4. FES System
 - a. More detailed description in 3.2.1
5. Hard Processing System (HPS)
 - a. More detailed Description in 3.2.2
6. Evaluation Design Version
 - a. Includes Evaluation Design version number
7. SYS ID
 - a. Includes System ID
8. I2C
 - a. Actually a Parallel IO block (PIO), used to generate I²C interface
9. MDIO master
 - a. Used to access RGMII PHY devices

10. Interrupt Capturer
 - a. Captures Interrupt inputs and presents them as registers readable via Avalon Slave port

3.2.1 FES System

The FES System QSYS component is actually a subsystem i.e. it includes other QSYS components. It is generated based on configurations with tcl scripts, which can be found in the FES System component folder. The FES System component block diagram with RGMII adapters is presented in Figure 4.

In previous Reference design versions, up to 2.9.4, similar block was called FRS Redbox. However, as the Flexibilis Ethernet Switch (FES) and Flexibilis Redundant Switch (FRS) were combined into same IP block, the FRS Redbox component was also replaced with a new QSYS component.

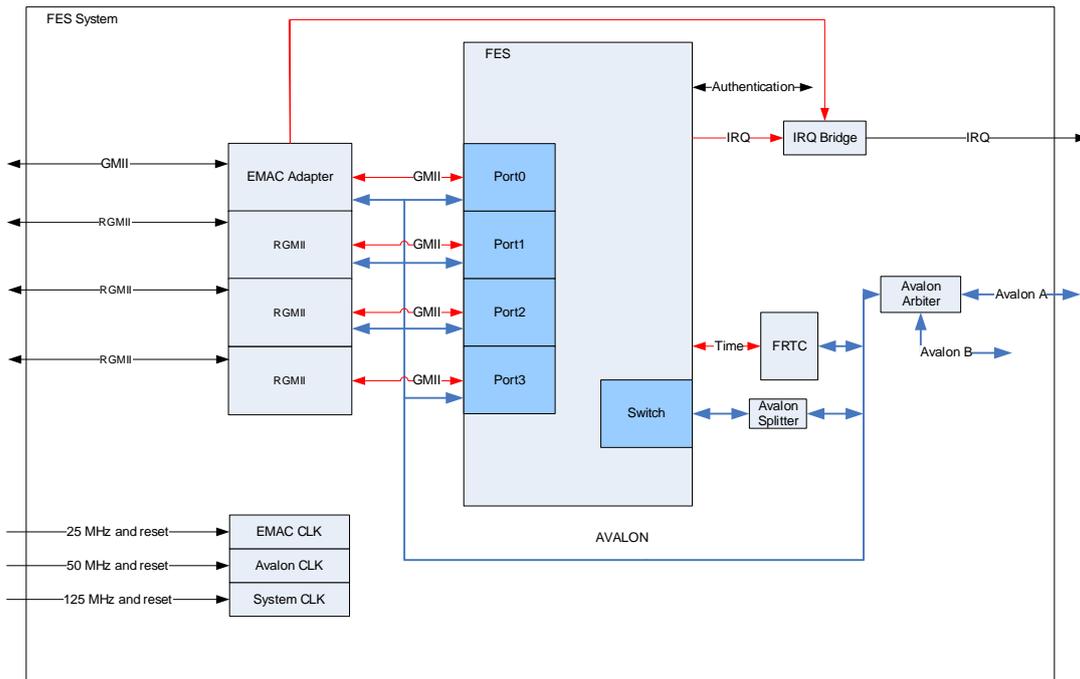


Figure 4. FES System with RGMII Component Block Diagram

FES

FES QSYS component is the Ethernet switch core which instantiated Flexibilis Ethernet Switch [7]. It can be instantiated also in QSYS as a separate component “Flexibilis Ethernet Switch” (FES_core) or in the VHDL code.

RGMII

RGMII adapters provide interface conversion between GMII (FRS) and RGMII. PHY configuration is done with separate MDIO masters (instantiated in QSYS top level).

EMAC ADAPTER

EMAC adapter provides GMII interface conversion so that it is compatible with HPS EMAC.

RECONFIG

Since the design implements Transceivers in the SGMII/1000BASE-X adapter, this block is required to be instantiated. Currently it does not include any functionality.

FRTC

Flexibilis Real Time Clock provides time information for FRS. FRTC can be controlled via Avalon.

IRQ BRIDGE

IRQ Bridge component provides a QSYS supported method for interrupt mapping.

CLOCK SOURCES

FES System component includes four clock source components (mii_clk, avalon_clk, system_clk, reconfig_clk), that are used to map clock signals into QSYS components.

AVALON ARBITER

The Avalon Arbiter is able to provide arbitration functionalities for two different Avalon interfaces. However, in this case only one Avalon interface is used.

AVALON SPLITTER

The Avalon Splitter component is generated automatically by the QSYS to support older FRS control mechanisms, even though it would not be required in this case.

3.2.2 HPS

The HPS QSYS component provides the user the possibility to modify HPS interfaces and certain configurations. If these setting or configurations are changed, it is important to always regenerate the Preloader (SPL).

The HPS interfaces are shown in Figure 5.

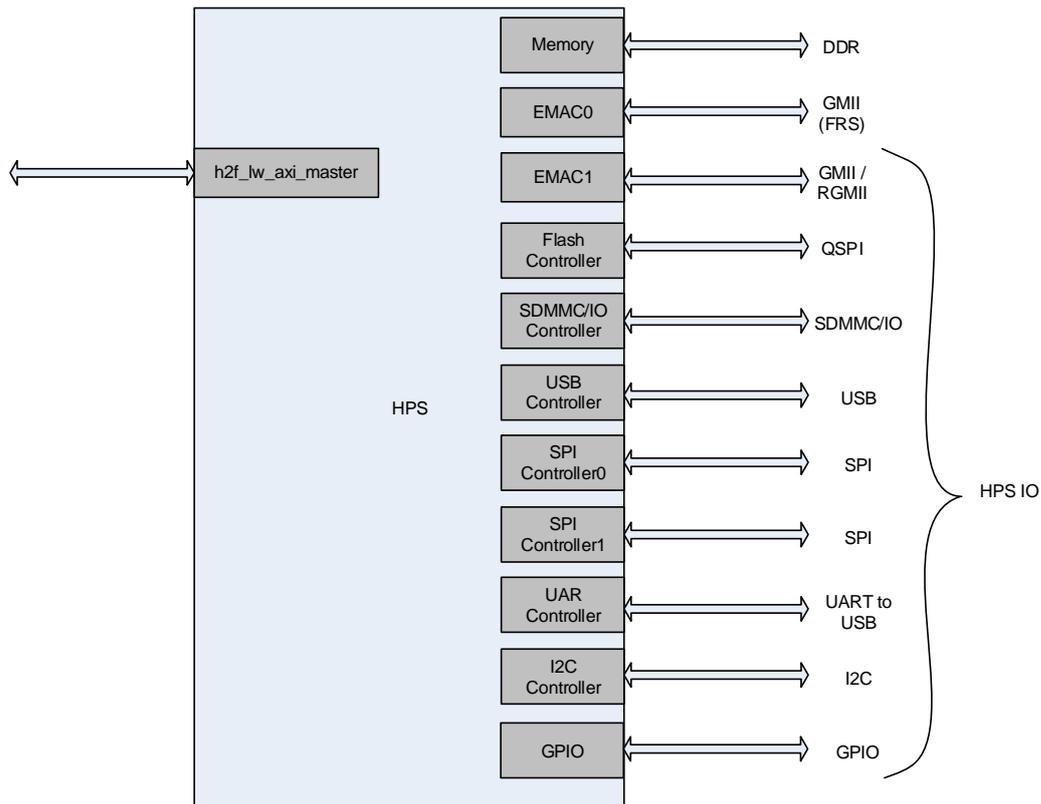


Figure 5. HPS interfaces

The HSP configuration includes only one AXI interface (h2f_lw_axi_master), which is used to communicate with the FPGA side of the SOC chip. The HPS acts as the bus master in this bus.

The most important HPS interfaces, in addition to the AXI interfaces, are two Ethernet MAC (EMAC) interfaces, DDR interface and the SDMMC/IO interface. EMAC0 is connected to the

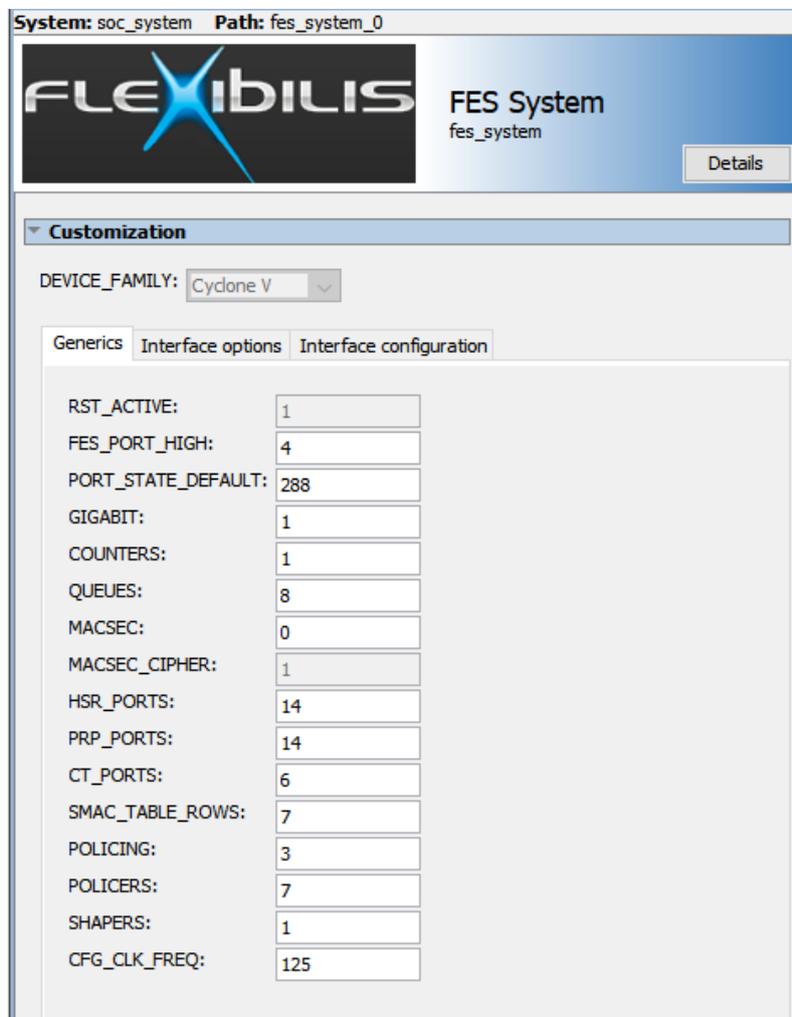
FRS Port0 and it is used to send and receive traffic over the FRS. The EMAC1 provides a change of external Ethernet port for the HPS that is not related to the FRS. EMAC0 interface is always GMII. The SDMMC/IO interface is connected to an external SD card slot and the SD card is used as the boot source of the whole SOC system, while the DDR is used as the runtime memory. Most of the other interfaces are not used in this Evaluation Design or are just used for development purposes. On the QSYS level most interfaces are part of the HPS IO interface, but Memory and EMAC interfaces are shown separately.

3.3 FES System Configuration

This chapter describes how FES system component is configured and how configurations affect the actual design.

To ease the configuration, instantiation and mapping of VHDL designs, the QSYS component provides automatic component and signal mapping and instantiation based on the configuration made in QSYS GUI.

3.3.1 Generic Configuration



The screenshot shows the 'FES System' configuration window for 'fes_system'. The 'Customization' section is expanded, and the 'Generics' tab is selected. The 'DEVICE_FAMILY' is set to 'Cyclone V'. The following table lists the generic parameters and their values:

Parameter	Value
RST_ACTIVE:	1
FES_PORT_HIGH:	4
PORT_STATE_DEFAULT:	288
GIGABIT:	1
COUNTERS:	1
QUEUES:	8
MACSEC:	0
MACSEC_CIPHER:	1
HSR_PORTS:	14
PRP_PORTS:	14
CT_PORTS:	6
SMAC_TABLE_ROWS:	7
POLICING:	3
POLICERS:	7
SHAPERS:	1
CFG_CLK_FREQ:	125

Figure 6. Generics

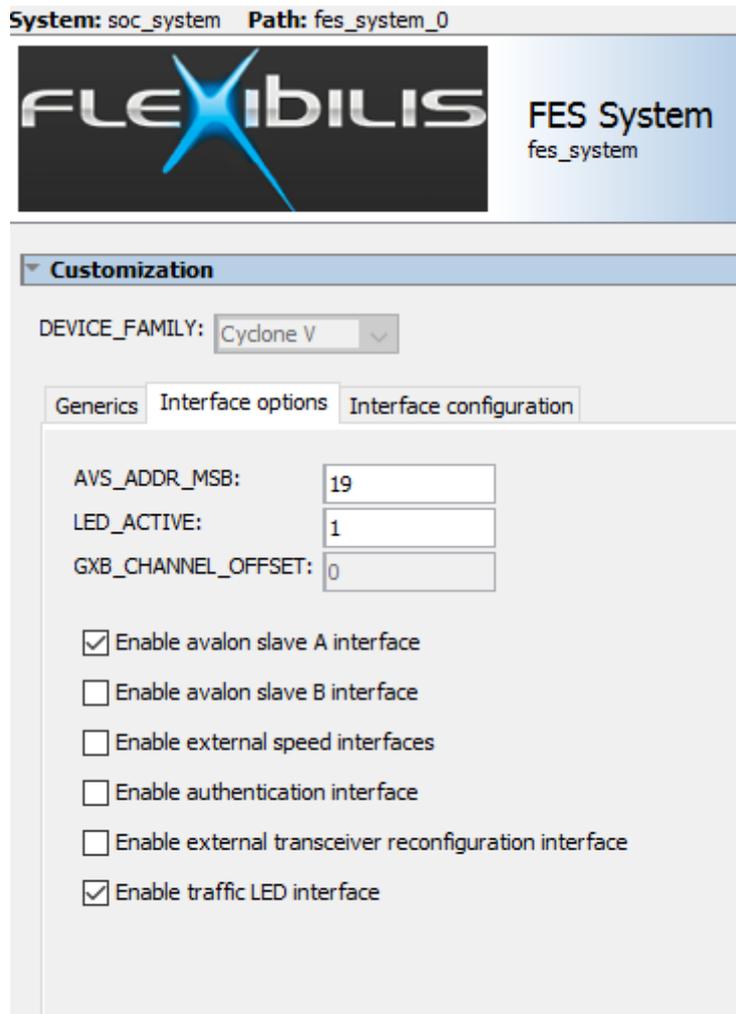
FES System Generics configurations page is shown in Figure 1, with settings used in this case. The configurable generics are:

- FES_PORT_HIGH
 - o Sets the port count. Value 4 means that there are five ports.

- PORT_STATE_DEFAULT
 - o Sets the default value for all PORT_STATE registers. Value 288 is 0x0120, which sets ports to forwarding mode, GMII and 1000Mbps
- GIGABIT
 - o Enables Gigabit operation (optional feature of FES). For more information see FES user manual [7]
 - o Gigabit operation is enabled in the reference design
- COUNTERS
 - o If COUNTERS = '1' then the COUNTERS block is synthesized (optional feature of FES). Use value '0' when no COUNTERS are needed to save logic.
 - o Counters are enabled in the reference design.
- QUEUES
 - o Number of priority queues (optional feature of FES). Possible values are 4 and 8. For more information see FES user manual [7].
 - o Reference design uses 8 priority queues. **Note that the basic FRS variation supports 4 queues and 8 queues is an extension.**
- MACSEC
 - o MACSEC port mask. Enables MACSEC functionality on corresponding port (optional feature of FES). For more information see FES user manual [7].
 - o MACSEC is supported in port 1 and 2. **Note that the basic FRS variation does not support MACSEC but is available as an extension.**
- HSR_PORTS
 - o Defines which ports support HSR. HSR support is optional features of FES.
 - o Decimal value 14 means that ports 1,2 and 3 include HSR capabilities
- PRP_PORTS
 - o Defines which ports support PRP. PRP support is optional features of FES.
 - o Decimal value 14 means that ports 1,2 and 3 include PRP capabilities
- CT_PORTS
 - o Enables Cut-through operation between two HSR ports. Cut-through is an optional feature of FES. For more information see FES user manual [7].
 - o Value 6, means ports 1 and 2.
- SMAC_TABLE_ROWS
 - o Defines how many rows are supported by the SMAC. SMAC is an optional feature of FES [7].
 - o In this reference design 128 SMAC table rows are supported. **Note that the basic FRS variation does not support SMAC, but it is available as an extension.**
- POLICING
 - o Enables Policer functionalities (optional feature of FES). For more information see FES user manual [7].
 - Value 0: no policing functionalities (bandwidth limiting) supported
 - Value 1: Policing supported via IPO
 - Value 2: Policing supported via SMAC
 - Value 3: Policing supported via SMAC/IPO, selectable via registers configurations
 - o In the reference design policing is enabled via SMAC and IPO. **Note that basic FRS variation does not support policing, but it is available as an extension.**
- POLICERS
 - o Defines how many policers are supported per port. Value 7 means 128 policers, value 8 means 256 policers etc.
 - o Requires that POLICING generic has a value 1,2 or 3.
- SHAPERS
 - o Enables Shaping functionalities (optional feature of FES). For more information see FES user manual [7].
 - o Shaping is supported by the reference design. **Note that the basic FRS variation does not support shaping, but it is available as an extension.**

- CFG_CLK_FREQ
 - o This defines the system clock frequency for the FRS and it must be set to match the actual clock frequency. In this case it is 125 MHz

3.3.2 Interface Options



System: soc_system Path: fes_system_0

FLEXIBILIS FES System
fes_system

Customization

DEVICE_FAMILY: Cyclone V

Generics Interface options Interface configuration

AVS_ADDR_MSB: 19

LED_ACTIVE: 1

GXB_CHANNEL_OFFSET: 0

Enable avalon slave A interface

Enable avalon slave B interface

Enable external speed interfaces

Enable authentication interface

Enable external transceiver reconfiguration interface

Enable traffic LED interface

Figure 7. Interface Options

In the Interface Options page, Figure 7, it is possible to select what interfaces the FES System provides.

1. Avalon slave A interface
 - a. Enabled in the reference design and connected to the NIOS in the QSYS
2. Avalon slave B interface
 - a. Disabled
3. External Speed interface
 - a. Not enabled in the reference design. Could be used to set speed/interface mode for the FRS ports.
4. Authentication Interface
 - a. Not enabled in the reference design. Must be exported and used, if an external security chip is used for IP license validation
5. Transceiver reconfiguration interface

- a. Not enabled in the reference design. Could be used for providing reconfiguration interface from external controller. Usually used in designs with multiple FES cores using transceiver interfaces.
- 6. Traffic LED interface
 - a. Enabled in the reference design. Provides signals for driving traffic LEDs (if provided by the interface adapters)

In addition this page defines the following interface related generics:

- AVR_ADDR_MSB
 - o Sets the Avalon address bus width. As a MSB definition it is actual width minus one.
- LED_ACTIVE
 - o Selects logic level when Link LEDs should lit.
- GXB_CHANNEL_OFFSET
 - o Not applicable since only one FRS is instantiated inside one FPGA chip.
 - o In case there would be multiple FRS inside one FPGA, the GXB_CHANNEL_OFFSET needs to be individual for each.

3.3.3 Interface Configuration

In the Interface configurations page there are three sub pages.

FRTC configurations are common for Interface configuration pages.

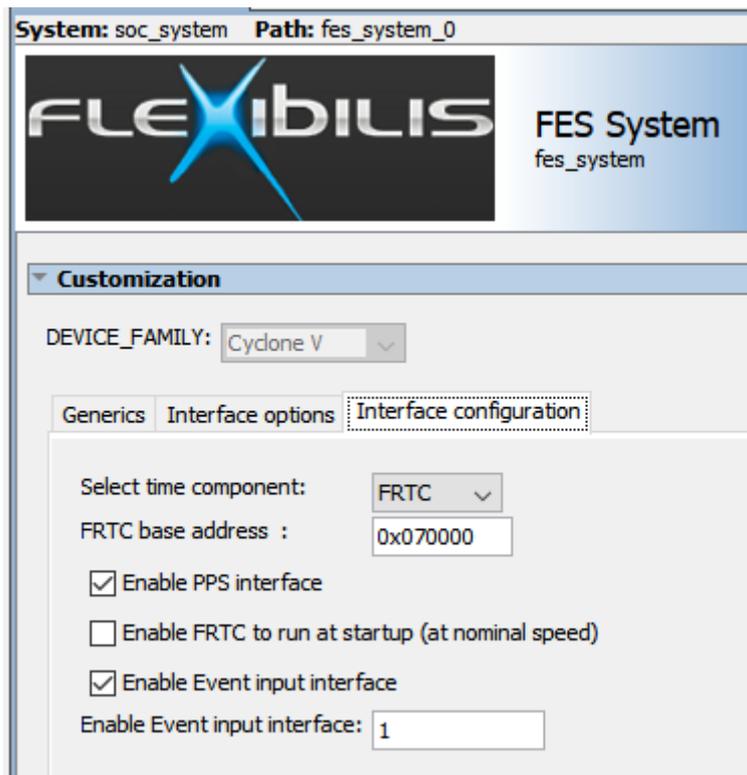


Figure 8. FRTC/FPTS Options

with these setting it is possible to enable the PPS signal output and start FRTC clock without register configurations. The Event input requires a separately licensable block called Flexibilis PPX Timestamper (FPTS).

3.3.3.1 Port Interface Type

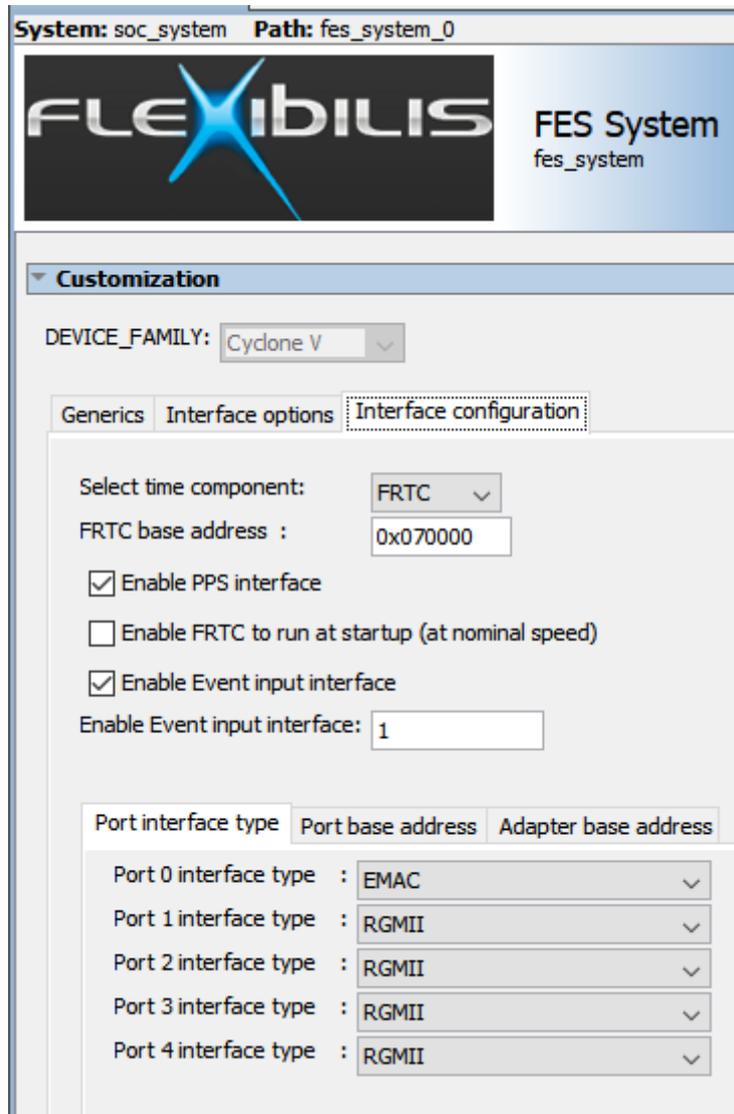


Figure 9. Interface Configurations

The Port Interface type, Figure 9, is used to select interface type for each port. In this case:

- Port 0 is used for the HPS EMAC, therefore it includes an EMAC adapter.
- Ports 1, 2, 3 and 4 are RGMII

Other possible interface types include:

- MII PHY mode
 - o Provides interface, which is compatible with most PHY interfaces
- GMII/MII Native (None)
 - o Provides FRS GMII interface without modifications
 - o Mainly for internal use i.e. QuadBox
- 1000BASE-X only
- 100BASE-FX
- RMII
- SGMII/1000base-X v2
- SGMII/1000base-X/100base-FX Triple mode adapter
- AFEC (Advanced Flexibilis Ethernet controller)

3.3.3.2 Port Address Configuration

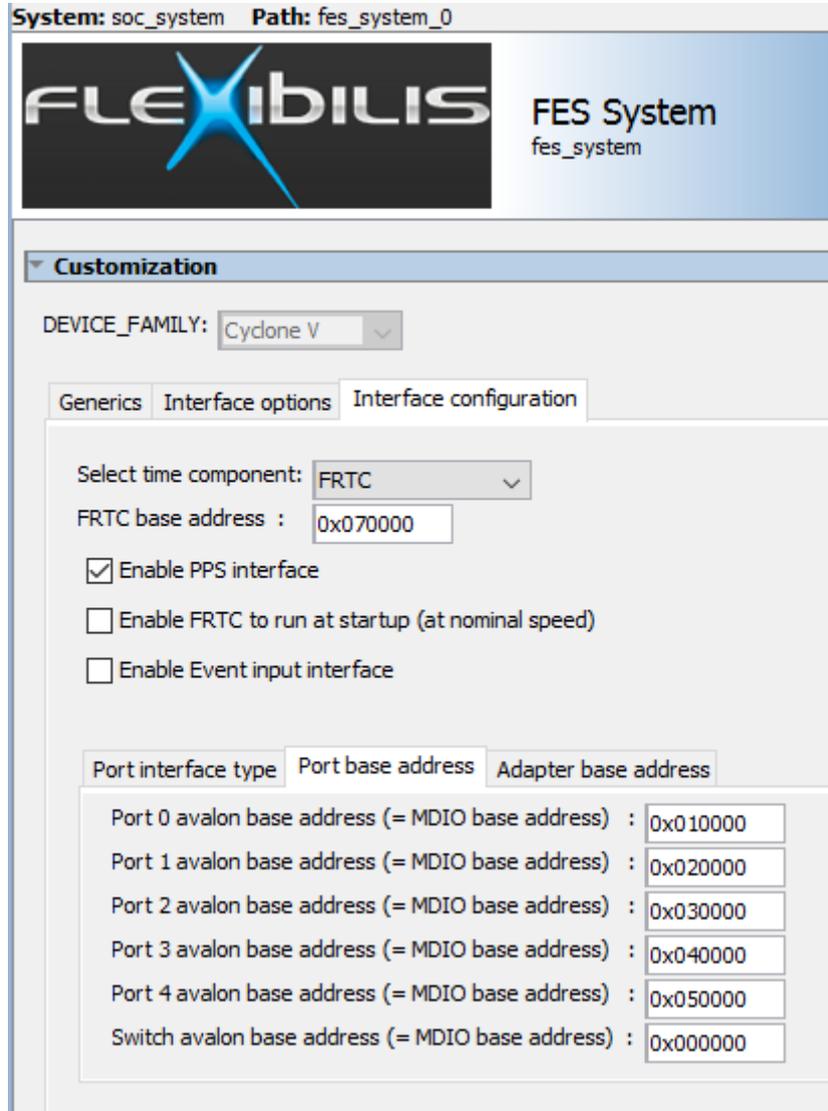


Figure 10. Port Address Configurations

Port base address configuration, Figure 10, defines the Avalon address offset from FES System base address for FRS Port and Switch configuration registers. Addresses are defined as word addresses. Avalon address map is defined in Chapter 3.5.

3.3.4 Adapter Address Configuration

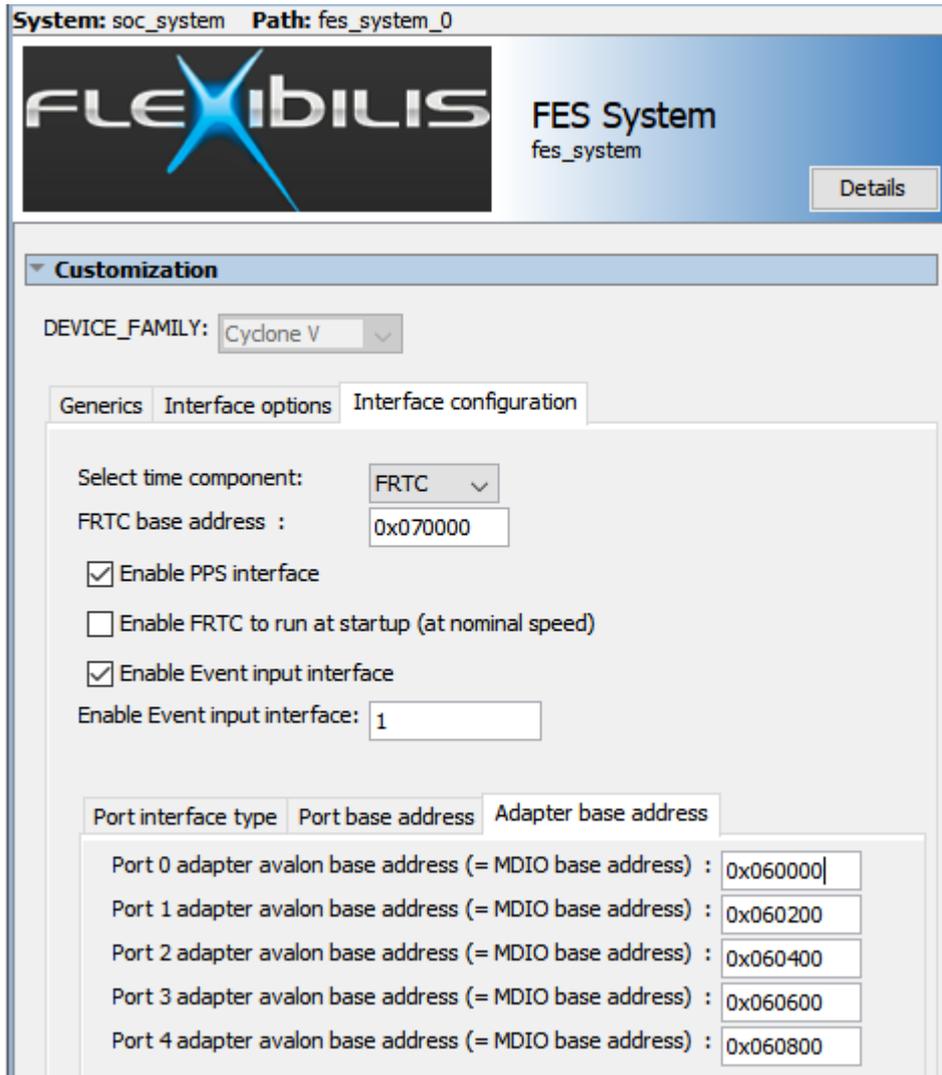


Figure 11. Adapter Address Configurations

Figure 11 presents adapter base address configuration, which defines the Avalon address offset from the FES System base address for adapter registers. Addresses are defined as word addresses. Avalon address map is defined in Chapter 3.5.

3.4 Interface Adapters

Interface adapters are used with FRS to provide other Ethernet interface types than the FRS native MII/GMII. Figure 9 illustrates the Adapters used in this Evaluation Design. However, the FES System provides also other interface adapters. Interface adapters are defined in separate specifications

3.5 Avalon/AXI Address Map

The Avalon Address Map is based on the settings in QSYS and its components. The Table 1 below defines the Address map for this Evaluation Design.

Component	BUS and Base Address
MDIO master	0x0001_2000
Sys ID	0x0001_0000
Rev ID	0x0004_0500
Interupt capturer	0x0000_0000
FES System	0x0010_0000
Switch core	
<i>Switch registers</i>	0x0010_0000
<i>TS</i>	0x0010_1000
<i>VLAN</i>	0x0010_2000
FRS Port 0	
<i>GEN</i>	0x0011_0000
<i>HSR</i>	0x0011_1000
<i>PTP</i>	0x0011_2000
<i>CNT</i>	0x0011_3000
<i>IPO</i>	0x0011_4000
FRS Port 1	
<i>GEN</i>	0x0012_0000
<i>HSR</i>	0x0012_1000
<i>PTP</i>	0x0012_2000
<i>CNT</i>	0x0012_3000
<i>IPO</i>	0x0012_4000
FRS Port 2	
<i>GEN</i>	0x0013_0000
<i>HSR</i>	0x0013_1000
<i>PTP</i>	0x0013_2000
<i>CNT</i>	0x0013_3000
<i>IPO</i>	0x0013_4000
FRS Port 3	
<i>GEN</i>	0x0014_0000
<i>HSR</i>	0x0014_1000
<i>PTP</i>	0x0014_2000
<i>CNT</i>	0x0014_3000
<i>IPO</i>	0x0014_4000
FRS Port 4	
<i>GEN</i>	0x0015_0000
<i>HSR</i>	0x0015_1000
<i>PTP</i>	0x0015_2000
<i>CNT</i>	0x0015_3000
<i>IPO</i>	0x0015_4000
Adapter P0	
<i>EMAC (No Content)</i>	0x0016_0000

Adapter P1	
RGMII	0x0016_0200
Adapter P2	
RGMII	0x0016_0400
Adapter P3	
RGMII	0x0016_0600
Adapter P4	
RGMII	0x0016_0800
FRTC	0x0017_0000

Table 1. Avalon Address Map

3.6 Compilation

This chapter describes shortly the Evaluation Design compilation steps. For more detailed information about Quartus, please refer to the documentation available in Altera’s website.

The Evaluation Design package includes ready-to-use program files, so compilation is not a mandatory step.

3.6.1 Folder Structure

The Evaluation Design folder structure is illustrated in Figure 12. The *fpga* folder includes *bin_qsys* folder, which includes most of the quartus project files. In addition there is an *ip* folder, which includes QSYS component files. The FPGA project expects that *external* and *encrypted_ip* folders are in place and have the correct content. Therefore the user should place separate FRS and FRTC cores into the correct folders or change the file path settings in Quartus project accordingly.

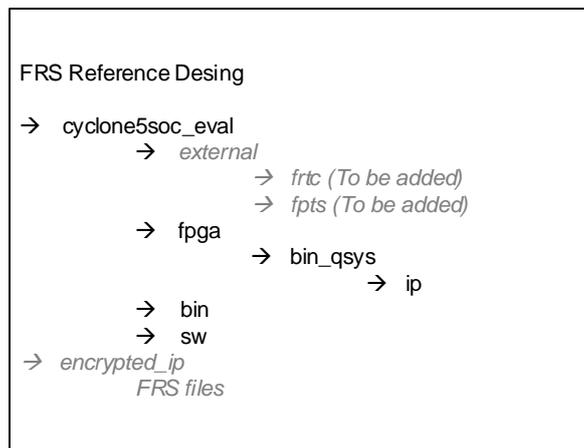


Figure 12. Folder Structure

3.6.2 QSYS Generation

The Evaluation Design includes *soc_system.qsys* file, which includes information about the QSYS system used in this design. The QSYS system needs to be generated before the first compilation is done, since the QSYS result design files are not included in the package. Normally Quartus tool generates the files automatically during normal compilation flow.

In order to modify the QSYS system it must be opened from Quartus. First, open the soc_system.qpf (Quartus Project File) with quartus. Then on the toolbar open the QSYS . Once QSYS opens, it will query the qsys system file to be opened. This is the soc_system.qsys file that is located in the bin_qsys folder. Once correct system has opened the user can either make changes or then directly generate a new qsys design. Generation is activated from the Generation page, pressing the Generate button. There should not be any errors or warnings during the generation. Once generation has finished, all necessary files have been generated and user can return to the Quartus project.

3.6.3 Quartus Project

The quartus project is configured with the correct assignments (pinout, IO Voltage, clock constrains etc.) and file references. The user should also make sure that the links to the files are correct, especially for the external IP cores.

To be able to compile the design, the user should make sure that the required licenses are available for the Quartus tool. The license setup can be found in the Tools menu. This Evaluation Design requires:

- FRS (Flexibilis)
- FRTC (Flexibilis) and FPTS
 - o OCP license provided for evaluation
- GMII to RGMII adapter (TTTech Flexibilis)

Once the QSYS system is generated and the licenses are set, the design can be compiled in normal way. The design will generate warnings, and they should be looked thought. However, they all should be generated by Altera blocks and should not affect the operations. In case of suspicious warnings please contact TTTech Flexibilis.

4 SW Evaluation Design

SW Evaluation Design is implemented as a modular design. The modules are depicted in Figure 13.

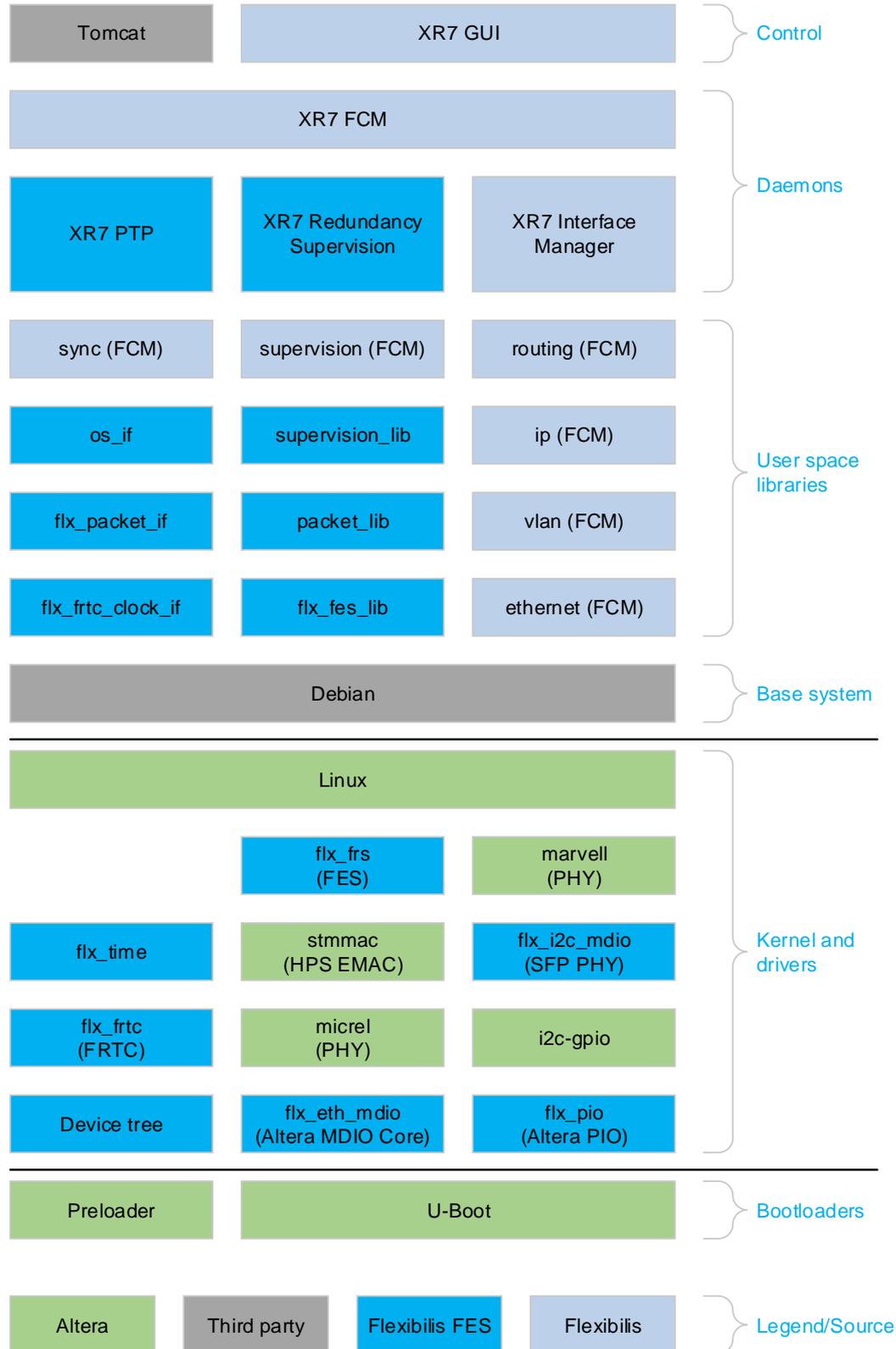


Figure 13. SW Evaluation Design

The modules depicted in Figure 13 as Flexibilis FRS with blue color are important core SW components of the SW Evaluation Design. They are needed to make use of FRS features like HSR, PRP and PTP. The other Flexibilis modules provide NETCONF [11] interface to daemons and GUI, which uses the NETCONF interface.

Green modules are needed for low level SoC FPGA configuration and booting OS. Altera SoC EDS includes source code for HPS Preloader and U-Boot and tools to build them. Handoff files from FPGA design are needed during the build process. SoC EDS contains also information for downloading matching Linux kernel source tree.

SW Evaluation Design is based on Debian [12] GNU/Linux distribution. The core SW modules for FES require only supported kernel (Linux in this case) and a C library. Although SW Evaluation Design is based on Debian, it is not a requirement for using these Flexibilis SW modules.

Source codes for protocol stacks (XR7 PTP Time daemon and XR7 Redundancy supervision) and for other software from Flexibilis Oy except Linux drivers, are delivered separately and they require a license with Flexibilis Oy. All protocol stacks and other Flexibilis software are included in the Evaluation Design release in a binary format and the functionality can be verified using the evaluation boards.

In the following sections SW Evaluation Design boot process and modules is described first, followed by descriptions of each major section of the SW stack and its components.

4.1 Boot

FRS SoC SW Evaluation Design uses SD/MMC boot and is thus booted from an SD-card. The boot process is depicted in Figure 14. Structure of the SD-card is described in section 4.5, see Table 5.

The following sections explain the role of each boot related module and how it is used in the SW Evaluation Design. For more details about the general SoC boot process and other boot alternatives please refer to Altera SoC documentation.

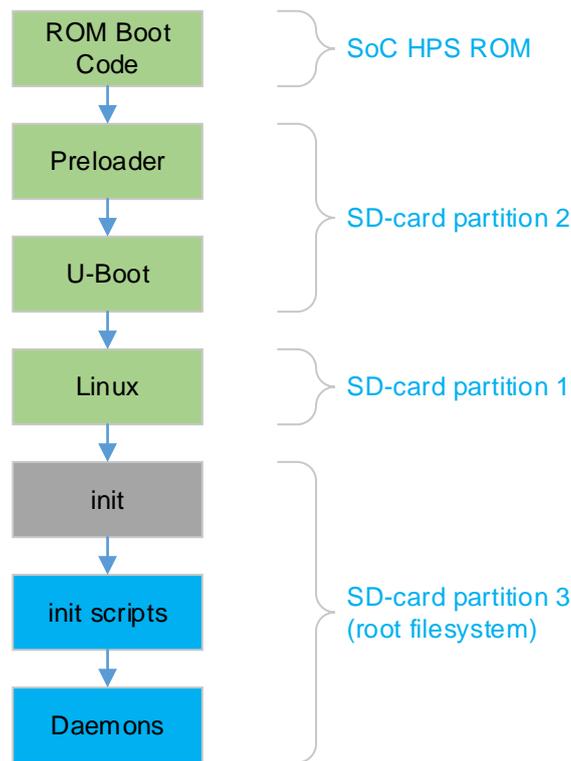


Figure 14. SW Evaluation Design boot process

4.1.1 ROM Boot Code

SoC HPS boot mode is selected using external signals, typically using DIP switches or jumpers on evaluation boards. When powered on, HPS ROM Boot Code checks the selected boot mode and operates accordingly. With SD/MMC boot it checks the SD-card for an MBR. If it finds one, it looks for a partition with type 0xA2. When found, it reads bootloader from start of the partition to on-chip RAM (OCRAM) at address 0xFFFF0000 and executes the bootloader from there.

4.1.2 Preloader

HPS Software Preloader, sometimes referred to as SPL, is the first piece of software run by HPS after Boot ROM. Preloader generation tools are part of Altera SoC EDS.

Some settings in Altera Qsys or Quartus II affect the Preloader, which means that the Preloader binary needs to be rebuilt from sources whenever such changes are made to FPGA design.

See Altera SoC Embedded Design Suite User Guide [13] for more information about the Preloader.

Because the size of OCRAM is limited (64 KiB), the whole U-Boot bootloader does not fit in there and Altera SoC Linux boot uses so called Software Preloader. It is a minimalistic bootloader which initializes various pieces of hardware, including SDRAM, and then reads the actual U-Boot bootloader from the 0xA2 partition offset 0x40000 (256 KiB) into SDRAM and executes the U-Boot bootloader from there.

Preloader behavior can be changed at compile time, for example to boot from QSPI flash instead.

4.1.3 U-Boot

U-Boot is a Linux bootloader for embedded systems and it is used to boot Linux on Altera SoC. It is the second piece of software run by HPS after Boot ROM. See Altera SoC

Embedded Design Suite User Guide [13] for more information about the U-Boot bootloader for Altera SoC.

U-Boot in turn reads its environment right after the MBR. If it finds a valid environment, it loads it, otherwise it uses hard-coded environment which is defined at U-Boot compile time. U-Boot environment contains U-Boot variables which include also the U-Boot commands used to boot Linux. Environment used in the SW Evaluation Design is shown in Listing 1, with important settings in **bold**.

```

ECC_SDRAM=0
ECC_SDRAM_DBE=0
ECC_SDRAM_SBE=0
baudrate=115200
bootargs=console=ttyS0,115200 root=/dev/mmcblk0p2 rw rootwait
bootcmd=run fpgaload ; run mmcload ; run mmcboot
bootdelay=5
bootimage=uImage
bootimagesize=0x500000
ethact=mii0
fdtaddr=0x00000100
fdtimage=socfpga.dtb
fdtimagesize=0x2000
filesize=37C3B4
fpgabin=soc_system.rbf
fpgadata=0x2000000
fpgadatasize=0x700000
fpgaload=mmc rescan ;
    ${mmcloadcmd} mmc 0:1 ${fpgadata} ${fpgabin} ;
    fpga load 0 ${fpgadata} ${filesize}
loadaddr=0x00007fc0
mmcboot=setenv bootargs console=ttyS0,115200 root=${mmcroot} rw
    rootwait ; bootm ${loadaddr} - ${fdtaddr}
mmcload=mmc rescan ;
    ${mmcloadcmd} mmc 0:${mmcloadpart} ${loadaddr} ${bootimage}
    ; ${mmcloadcmd} mmc 0:${mmcloadpart} ${fdtaddr} ${fdtimage}
mmcloadcmd=ext2load
mmcloadpart=1
mmcroot=/dev/mmcblk0p3
qspiboot=setenv bootargs console=ttyS0,115200 root=${qspiroot} rw
    rootfstype=${qspirootfstype} ;
    bootm ${loadaddr} - ${fdtaddr}
qspibootimageaddr=0xa0000
qspifdtaddr=0x50000
qspiload=sf probe ${qspiloadcs} ;
    sf read ${loadaddr} ${qspibootimageaddr} ${bootimagesize} ;
    sf read ${fdtaddr} ${qspifdtaddr} ${fdtimagesize}
qspiloadcs=0
qspiroot=/dev/mtdblock1
qspirootfstype=jffs2
ramboot=setenv bootargs console=ttyS0,115200 ;
    bootm ${loadaddr} - ${fdtaddr}
stderr=serial
stdin=serial
stdout=serial
verify=n

```

Listing 1. SW Evaluation Design U-Boot environment

U-Boot runs the commands from `bootcmd` variable. SW Evaluation Design loads FPGA image file to SDRAM and configures the FPGA with it. Then it loads Linux kernel image and device tree to SDRAM, to different addresses. Finally it sets Linux kernel command line and passes control to Linux kernel passing it also address of the device tree.

4.1.4 Linux

Linux kernel does its own boot time initializations, mounts root filesystem and starts the init process. From then on system uses Debian boot scheme. SW Evaluation Design includes init scripts to load the FES and FRTC drivers, which are built as kernel modules, and start the daemons.

The i2c devices defined for Linux i2c-gpio driver in the device tree do not get automatically bound to the i2c-gpio driver. For that reason the SW Evaluation Design contains the following code in one of the boot scripts after loading the `flx_pio` drivers

```
for i in /sys/devices/soc.0/i2c.* ; do
    echo ${i##*/} > /sys/bus/platform/drivers/i2c-gpio/bind
done
```

4.2 Kernel and Drivers

Summary of main drivers used with each supported board are listed in Table 2.

Driver	Use
<code>flx_frs</code>	FES
<code>flx_time</code>	Time interface
<code>flx_frtc</code>	FRTC
<code>Stmmac</code>	HPS EMAC
<code>Marvell</code>	Copper SFP PHYs
<code>flx_eth_mdio</code>	MDIO busses to FPGA Ethernet PHYs
<code>flx_pio</code>	FPGA PHY reset and LEDs

Table 2. Use of drivers with supported boards

4.2.1 Linux

Linux kernel for Altera SoC is used in the FRS SoC SW Evaluation Design. Some small patches had to be applied to make it work in a desired way, mostly to support HPS EMAC operation also without PHY using the fixed-link Linux PHY driver (`CONFIG_FIXED_PHY`). The default kernel configuration is also changed slightly to enable needed features.

See Altera SoC Embedded Design Suite User Guide [13] for more information about the Linux kernel for Altera SoC.

SW Evaluation Design Linux drivers for Flexibilis IPs are modular. All needed Linux drivers from Flexibilis are licensed under GPL v2. Descriptions of Flexibilis Linux drivers used in the SW Evaluation Design follow.

4.2.2 Device Tree

Linux kernel for Altera SoC HPS needs device tree to function correctly or at all. Device tree contains information about the system and its components or devices and is used by device drivers to handle the hardware correctly. Device tree is thus a way to tell Linux which devices are present in the system, and various device specific configuration information. Linux drivers can then be bound to the devices automatically and drivers have a way to get configuration information of its devices.

Device trees are used in binary form (`.dtb` files), but manipulated as text files in source form (`.dts` files). Conversion between source and binary is possible in both directions without loss of functionality using device tree compiler `dtc`, but some source form constructs may get replaced by something else in the binary form.

There are a couple of ways to generate device tree for a system. It can be written from scratch, device tree from Linux kernel can be used as basis, or Altera SoC EDS device tree generator utility `sopc2dts` can be used, at least as a starting point. Device tree from Linux kernel was used as basis with the FRS SoC SW Evaluation Design. It was modified slightly to match the QSYS settings, then FES and FRTC IP related parts were added.

Device tree bindings of Flexibilis drivers are described together with each driver in the following sections. Complete device trees for supported boards are available in the download package.

4.2.3 flx_frs (FES)

`Flx_frs` is a driver for FES and XRS RS. Flexibilis Ethernet Switch (FES) and Flexibilis Redundant Switch (FRS) were combined into same FES IP block. Available IP features must be described for the driver in device tree.

Driver creates a Linux net device for each switch port. Net devices of the external ports are attached to PHY devices, if so configured. This allows existing Linux PHY drivers to be used for link mode monitoring in order to keep FES registers synchronized with current link mode. Link mode can be managed through `ETHTOOL` `ioctl`. Driver provides also its own `ioctl` interface to user space for accessing FES features. Many examples in this chapter use `flx_frs_tool` which is built on `flx_fes_lib` APIs.

4.2.3.1 Device Tree Bindings

Driver needs information about each switch and switch port in device tree to function correctly. Network interface names are defined in device tree. By convention CPU ports are named `SE01`, `SE02`, and so on while external ports are named `CE01`, `CE02`, `CE03` and so on, although any valid names can be used.

FES definitions for the FRS SoC evaluation board are shown in Listing 2.

```

fes@ff300000 {
    #address-cells = <1>;
    #size-cells = <1>;
    compatible = "flx,fes";
    /* Switch registers */
    reg = <0xff300000 0x8000>;
    /* FPGA interrupts start from 72 (-32 for .dts) */
    interrupts = <0 43 0x4>;
    mac_name = "eth0";
    ptp-clock = <&frtc0>;
    if_name = "SE01";
    features {
        clock-frequency = <125000000>;
        device-id;
        gigabit;
        statistics-counters;
        mac-address-table;
        vlan;
        traffic-shaper;
        priority-queues = <8>;
        traffic-policers = <128>;
        static-mac-address-table-rows = <128>;
        hsr-ports = <0x0e>;
        prp-ports = <0x0e>;
        cut-through-ports = <0x06>;
    };
    port0 {
        if_name = "SE01";
        medium_type = <5>;
        cpu-port;
        /* port and port adapter registers */
        reg = <0xff310000 0x10000>;
    };
    port1 {
        if_name = "CE01";
        medium_type = <1>;
        reg = <0xff320000 0x10000 0xff360200 0x200>;
        phy-handle = <&phy1>;
        phy-mode = "rgmii-id";
    };
    port2 {
        if_name = "CE02";
        medium_type = <1>;
        reg = <0xff330000 0x10000 0xff360400 0x200>;
        phy-handle = <&phy2>;
        phy-mode = "rgmii-id";
    };
    port3 {
        if_name = "CE03";
        medium_type = <1>;
        reg = <0xff340000 0x10000 0xff360600 0x200>;
        phy-handle = <&phy3>;
        phy-mode = "rgmii-id";
    };
    port4 {
        if_name = "CE04";
        medium_type = <1>;
        reg = <0xff350000 0x10000 0xff360800 0x200>;
        phy-handle = <&phy4>;
        phy-mode = "rgmii-id";
    };
};

```

Listing 2. FES device tree definition

Switch device tree bindings are listed in the following.

compatible

Value should be "flx,fes" for FES IP.

reg

Address of FES switch registers and length in octets.

interrupts

Interrupt definition in format specific to the parent interrupt controller.

mac_name

Name of the Linux net device whose Ethernet MAC is connected to CPU port.

features

Enabled FES IP features are described in this node. Many of them can be taken directly from FES Generics, see section 3.3.1, although often the setting is in device tree in a slightly different form. Some of these features affect driver operation, some are currently only informational but may have some effect in the future. Active features can be checked from procfs file

`/proc/driver/flx_frs/device<NN>_features` or through `ioctl` interface (4.2.3.3).

clock-frequency (integer)

Switch clock frequency in Hz. Software may use this for example when calculating traffic control (policing and shaping) configurations for desired bit rates.

device-id (boolean) [FES]

Switch IP contains device ID registers. Always valid for standard FES IP.

gigabit (boolean) [FES Generics]

Gigabit Ethernet is supported. Without this all ports are limited to 10/100 Mb/s speeds.

statistics-counters (boolean)

Port statistics counters are available. Always valid for standard FES IP.

mac-address-table (boolean)

Switch MAC address table is accessible. Always valid for standard FES IP.

vlan (boolean)

Switch supports VLANs. Always valid for standard FES IP.

traffic-shaper (boolean) [FES Generics]

Traffic shapers are enabled for all ports.

priority-queues (integer) [FES Generics]

Defines number of output queues per port.

traffic-policers (integer) [FES Generics]

Defines number of traffic policers per port. Missing or zero value disables policer support.

static-mac-address-table-rows (integer) [FES Generics]

Defines number of rows in static MAC address table. Missing or zero value disables static MAC address table support.

hsr-ports (integer) [FES Generics]

Defines switch ports that support HSR. Value is a bit mask of port numbers.

prp-ports (integer) [FES Generics]

Defines switch ports that support PRP. Value is a bit mask of port numbers.

macsec-ports (integer) [FES Generics]
 Defines switch ports that support MACsec. Value is a bit mask of port numbers. Missing or zero value disables MACsec support.

port<N>
 FES port definitions for port <N>.

Port device tree bindings are:

if_name
 Linux net device name for the port. Driver creates new net device for each port using this name.

medium_type
 Type of the medium used with this port. This affects how the driver deals with the port. Possible values are:

- 0 NONE, port is not used
- 1 SFP, port connected to SFP (fiber or copper) and may have a PHY
- 2 PHY, port hard-wired to a PHY
- 5 NO PHY, there is no PHY, speed can be changed at runtime

Parameters phy-handle and phy-mode must be used with SFP and PHY medium types for the FES driver to be able to attach the Linux PHY device to the FES port net device. Parameter sfp-phy-handle can be used with SFP medium type to define access to PHY within copper SFP module. Parameter sfp-EEPROM can be used with SFP medium type to detect SFP type from SFP EEPROM contents.

cpu-port
 Indicates a port connected to CPU.

interlink-port
 Indicates a port connected to another FES. This is used for example when building a QuadBox using a design with one CPU port and two interconnected switches.

auto-speed-select
 Configure FES port to select speed from external signals or configure XRS RS port to select speed automatically.

reg
 Address of port registers and length in octets, optionally followed by address of port adapter registers and length in octets. Note that adapter registers should be defined if port is connected to any of the adapters specified in section 3.4 for the link to work correctly.

phy-handle
 Link to PHY device which is defined somewhere else in the device tree. This is required for ports with medium type PHY and optional for ports with medium type SFP. If this is left out, FES driver assumes there is no PHY.

Copper SFP modules may have a PHY, too. Parameter sfp-phy-handle should be used for them instead of phy-handle.

For medium type SFP both phy-handle and sfp-phy-handle can be specified, when there is a separate PHY in addition to SFP PHY for a port. This may be necessary for example to put both PHYs in correct mode.

phy-mode
 PHY interface mode to use with the Linux PHY driver framework. This is required when phy-handle is set. See Linux source file `drivers/of/of_net.c` for possible values.

sfp-eeprom

Link to I²C slave device of SFP EEPROM defined somewhere else in the device tree. If this is specified for ports which have medium_type value 1 (SFP), SFP module type is detected from SFP EEPROM contents. This is needed with some designs for the port to function correctly with different SFP modules.

sfp-phy-handle

Link to PHY device within copper SFP module. Copper SFP modules typically contain a PHY device which is accessed via I²C.

For medium type SFP both phy-handle and sfp-phy-handle can be specified, when there is a separate PHY in addition to SFP PHY for a port. This may be necessary for example to put both PHYs in correct mode.

sgmii-phy-mode

If present, FES driver uses SGMII/1000Base-X port adapter in SGMII PHY mode when its SGMII mode is selected. Otherwise the adapter is used in SGMII MAC mode.

4.2.3.2 Principle of Operation

FES driver needs an Ethernet MAC device to work with. It catches all frames received by the MAC using Linux net device API and handles them itself. Because of this the original MAC network interface cannot be used for networking. So IP addresses and routes at OS level, for example, are configured to use the FES CPU network interface (typically SE01) instead of the original network interface.

All frames sent by OS to FES port network interfaces will be forwarded to the original MAC driver for actual sending. Before that the driver adds management trailer to all frames. Frames sent to FES CPU port network interface will get management trailer value zero, which causes FES to choose where to forward the frame. Frames to other FES port network interfaces (CExx) will get a management trailer with only the bit of that port set, causing the frame to be sent only through that FES port. Thus the driver relies on FES management trailer feature to work correctly.

All frames received from the MAC have the management trailer, too, which indicates the receiving external FES port. Received frame handling in FES driver depends on interface mode: normal or independent.

For normal external ports FES driver passes all frames to OS as coming from FES CPU port network interface, with the exception of HSR/PRP supervision frames and PTP frames. They are passed to OS as coming from the external FES port network interface so that the software can detect the original port. Because of this the external FES port network interfaces cannot be used for normal networking at OS level. But they can be used to link mode monitoring and enforcing a certain link mode, to retrieve FES port statistics counters and to access FES port registers.

When port is configured as an independent interface, FES driver passes all frames to OS as coming from the external FES port network interface. Independent interfaces feature is described in section 4.2.3.14.

When FES receives a PTP frame from CPU to the CPU port, it timestamps the frame and captures the frame into FES registers and generates an interrupt. The driver detects this and retrieves the original frame sent by software and its timestamp and passes the frame back with the timestamp to OS as coming from the first PTP enabled port. The local PTP software can detect that the frame was actually sent by itself and retrieve the PTP header information and actual send time and do corrections based on the information available. This is used for peer link delay measurement.

4.2.3.3 Accessing Switch Features

FES driver provides an ioctl interface for accessing switch features from application code. See the driver header files for more information, files are listed in Table 3.

File	Description
Flx_frs_iflib.h	FES specific ioctl API definitions
Flx_frs_if.h	FES register definitions

Table 3. FES driver API header files

The preferred method is to use the provided `flx_fes_lib` API rather than the ioctl directly. See section 4.3.3. Note that software can also ask available switch features from the driver.

There is also a command line utility `flx_frs_tool` which supports most of the API features. Use the following command to see its usage.

```
flx_frs_tool -h
```

Example commands to see available switch features:

```
cat /proc/driver/flx_frs/device00_features
flx_frs_tool -F SE01
```

4.2.3.4 Port Link Mode Management

Normal Linux ETHTOOL ioctl can be used to monitor and manage external FES port link status and mode. There is also `ethtool` command available. Example command to get CE01 port link status:

```
ethtool CE01
```

Example command to force CE01 to 1000 Mbit/s full-duplex mode:

```
ethtool -s CE01 autoneg off speed 1000 duplex full
```

Example command to enable autonegotiation on CE01:

```
ethtool -s CE01 autoneg on
```

4.2.3.5 Managing Port Forwarding Mode

Normally port is in disabled mode when the corresponding network interface is down or there is no link, and in forwarding mode when link is also up. FES ports have also a third mode: learning.

Driver ioctl can be used to control FES port forwarding mode. When set in non-automatic mode, driver still keeps the port in disabled mode when network interface is down or there is no link, and in specified mode when link is up. Normal behavior can be returned by setting port back to automatic mode. This is useful for example in implementing rapid spanning tree protocol (RSTP).

Command `flx_frs_tool` can be used to control the forwarding modes. Examples:

```
flx_frs_tool -f CE03 learning
flx_frs_tool -f CE03 auto
```

4.2.3.6 Accessing IPO Entries

`Flx_fes_lib` provides functions for accessing FES IPO rules, which can also be seen from `procfs` file. Example command:

```
cat /proc/driver/flx_frs/device00_ipo_registers
```

4.2.3.7 Accessing Port Statistics Counters

All the statistics counters provided by FES ports are available through ETHTOOL ioctl as Linux network interface specific statistics. FES statistics counters are often much more useful than ordinary Linux net device statistics counters for diagnosing problems.

Example `ethtool` command to retrieve statistics counters from port CE01:

```
ethtool -S CE01
```

4.2.3.8 Accessing MAC Address Table

Driver `ioctl` interface can be used to read FES MAC address table and to clear MAC address table entries of select ports.

Command `flx_frs_tool` can be used to read the switch MAC table. Example:

```
flx_frs_tool -m SE01
```

MAC address can also be read from a proc file, too. Example:

```
cat /proc/driver/flx_frs/device00_mac_table
```

Command `flx_frs_tool` can be used to clear switch MAC table entries of select ports.

Note that it is normally not recommended for redundant ports. Example:

```
flx_frs_tool -c CE03
```

4.2.3.9 Accessing Static MAC Address Table

Driver `ioctl` can be used for reading and writing FES static MAC address table (SMAC table) entries. SMAC table can also be seen from `procf`s file and accessed using `flx_fes_lib` API.

Command `flx_frs_tool` can be used for testing.

Example command to list all enabled SMAC table entries:

```
flx_frs_tool -M SE01
```

Example command to list the whole SMAC table:

```
flx_frs_tool -M SE01 '*'
```

Example command to add an SMAC table entry for MAC address 72:c7:c6:13:51:be to forward VLAN ID 5 frames to ports 1 and 2 using next unused column number:

```
flx_frs_tool -E SE01 72:c7:c6:13:51:be + 0x9000 0x6 0 0 5
```

Example command to remove all SMAC table entries for MAC address 96:0d:59:62:11:8a:

```
flx_frs_tool -R SE01 96:0d:59:62:11:8a
```

Example command to remove all SMAC table entries:

```
flx_frs_tool -R SE01 '*'
```

4.2.3.10 Traffic Shaping

`Flx_fes_lib` provides an API for managing FES traffic shaping feature. Command

`flx_frs_tool -S` can be used for testing the shaping feature. Example command to limit interface CE01 output queue 0 to 50 Mb/s when switch clock frequency has been defined in device tree:

```
flx_frs_tool -S CE01 0 50000000 -C 0
```

Example command to show traffic shaping of output queue0:

```
flx_frs_tool -S CE01 0 -C 125000000
```

Example command to stop shaping traffic in output queue 0:

```
flx_frs_tool -S CE01 0 2000000000 -C 0
```

4.2.3.11 Traffic Policing

Flx_fes_lib provides an API for managing FES traffic policing feature. Command `flx_frs_tool -P` can be used for testing the policing feature. Example command to limit interface CE01 policer 0 (default) to 120 Mb/s with limit 15360 B when switch clock frequency has been defined in device tree:

```
flx_frs_tool -P CE01 0 120000000 15360 -C 0
```

Example command to show traffic policing of policer 0:

```
flx_frs_tool -P CE01 0 -C 0
```

Example command to stop policing traffic in policer 0:

```
flx_frs_tool -P CE01 0 2000000000 65535 -C 0
```

4.2.3.12 Configuring MACsec

Note: MACsec is not enabled in the reference design.

Flx_fes_lib provides an API for managing FES MACsec feature. Flx_frs_tool commands `-I`, `-K` and `-T` can be used for testing MACsec feature. Note that driver requires CAP_NET_ADMIN Linux capability for all accesses to port MACsec registers.

Example commands to write and read back Secure Channel Identifiers (SCI) of CE01. LSB of the RX SCI is 0x00 and MSB is 0x77.

```
flx_frs_tool -I CE01 rx 7766554433221100
flx_frs_tool -I CE01 rx
flx_frs_tool -I CE01 tx fedcba9876543210
flx_frs_tool -I CE01 tx
```

Example commands to write and read back TX key 0 and RX key 0 of CE01. 128 bit keys are used in this example, most significant bits of the full 256-bit keys will be set to zeros. LSB of the RX key is 0x00 and MSB (of the 128-bit key) is 0xFF.

```
flx_frs_tool -K CE01 rx 0 ffeeddccbbaa99887766554433221100
flx_frs_tool -K CE01 rx 0
flx_frs_tool -K CE01 tx 0 fedcba9876543210fedcba9876543210
flx_frs_tool -K CE01 tx 0
```

Example command to enable MACsec on CE01 using TX key 0 so that SC bit is set, i.e. SCI will be included in SecTAG. Both RX and TX security association (SA) 0 packet number (PN) counters are also reset.

```
flx_frs_tool -T CE01 enable 0 0x1 0x1
```

Example commands to write TX key 1 and to switch to using it. Also SC bit is set and SA 1 TX packet number counters are reset.

```
flx_frs_tool -K CE01 tx 1 fffeedddccbbbaaa99977766655544
flx_frs_tool -T CE01 enable 1 0x1 0
```

Example command to disable MACsec on CE01 retaining packet number counter values:

```
flx_frs_tool -T CE01 disable 0 0 0
```

4.2.3.13 Auxiliary Network Interfaces

Management trailers can be used to send frames from only select switch ports. Driver automatically creates a network interface for each port, as described in section 4.2.3.2. Additional, so called auxiliary network interfaces can be created for other uses. Multiple ports can be added to each auxiliary network interface, which allows sending frames to all those ports at the same time.

Auxiliary network interfaces are used with XR7 PTP to support PTP boundary clock feature, and some other PTP usage scenarios.

Auxiliary network interfaces can be managed using the `flx_frs_tool` command. Example commands to create a new net device OC01 and add ports CE01 and CE02 to it:

```
flx_frs_tool -A SE01 OC01
flx_frs_tool -a OC01 CE01
flx_frs_tool -a OC01 CE02
```

Example command to list FES ports of auxiliary network interface OC01:

```
flx_frs_tool -l OC01
```

Example command to remove auxiliary network interface OC01:

```
flx_frs_tool -D OC01
```

4.2.3.14 Independent Interfaces

As FES is basically an Ethernet switch, it forwards traffic between its ports. However it is possible to configure switch so that some of its ports appear as independent network interfaces, just like an interface of a regular Ethernet network interface card. Driver supports this use case by allowing ports to be defined as being independent interfaces.

Module parameter `ifacemodes` can be set to a bitmask, where each bit corresponds to an FES port. Ports whose bit is set are treated as independent interfaces. Net devices of those external ports can then be used as if they were ordinary network interfaces. Normally a different MAC address should be set for such net devices. Command `ip` can be used for that when net device is down:

```
ip link set dev CExx address XX:XX:XX:XX:XX:XX
```

In case of multiple FES instances `ifacemodes` parameter can be set to a comma separated list of bitmask values, one for each FES instance.

Note that the implementation uses switch features like port forward mask, IPO rules and management trailers.

4.2.4 flx_frtc (FRTC)

`Flx_frtc` is a driver for both FRTC (Flexibilis Real-Time Clock) and XRS RTC. When access to FRTC is needed from software, the device must be defined in device tree. Device tree definition is shown in Listing 3. FRTC driver is used from user space through the interface driver `flx_time`.

```
frtc@c8100000 {
    compatible = "flx,frtc";
    reg = <0xff370000 0x10000>;
    /* Step size in nanoseconds and subnanoseconds */
    step-size = <8 0>;
};
```

Listing 3. FRTC device tree definition

Device tree bindings for FRTC are listed below.

`reg`

Address of FRTC registers and length in octets.

`step-size`

NCO step size as two numbers: nanoseconds and subnanoseconds.

Subnanoseconds is a 32-bit number, each second is divided to 2^{32} subnanoseconds.

See FRTC user manual [8] for details.

4.2.5 flx_time

Flx_time driver provides a common user space interface for all Flexibilis time related IPs and blocks, each of which has its own driver. The drivers provides character device `/dev/flx_time0`.

Driver does not have device tree bindings.

4.2.6 flx_pio (Altera PIO)

Flx_pio is a small driver providing Linux GPIO interface to Altera PIO components.

PIO device tree configuration for one PIO block is shown in Listing 4. If GPIO direction is fixed, that should be told to the driver by setting `direction` to correct bitmask, zero for input and one for output.

```
/* PIO for FPGA PHY reset */
pio0: gpio@ff240400 {
    #gpio-cells = <2>;
    compatible = "flx,pio";
    reg = <0xff240400 0x20>;
    gpio-controller;
    width = <1>;
    direction = <0x1>;
};
```

Listing 4. PIO to GPIO definition in device tree

Custom designs may choose a different way to access PHYs, or may lack PHYs entirely. In such cases this may not be needed at all. On the other hand this could be used to turn PIO into Linux GPIO for other purposes.

4.2.7 flx_eth_mdio (Altera MDIO Core)

Flx_eth_mdio is a small MDIO bus driver for Altera MDIO Core components. It is used with FRS SoC Evaluation board to access Ethernet PHYs connected to FPGA. Device tree definition is shown in Listing 5.

```

/* MDIO bus for FES port PHY */
eth_mdio@ff212000 {
    #address-cells = <1>;
    #size-cells = <0>;
    compatible = "flx,eth-mdio";
    reg = <0xff212000 0x100>;

    phy1: phy@0 {
        compatible = "ethernet-phy-ieee802.3-c22";
        reg = <0x0>;
    };

    phy2: phy@1 {
        compatible = "ethernet-phy-ieee802.3-c22";
        reg = <0x1>;
    };

    phy3: phy@2 {
        compatible = "ethernet-phy-ieee802.3-c22";
        reg = <0x2>;
    };

    phy4: phy@3 {
        compatible = "ethernet-phy-ieee802.3-c22";
        reg = <0x3>;
    };
};

```

Listing 5. MDIO bus definition in device tree

4.2.8 i2c_gpio

This is the standard Linux i2c-gpio driver to turn two Linux GPIO signals to an I²C bus. on FRS SoC Evaluation Design SFP EEPROM and SFP PHY devices are accessed using this driver.

4.2.9 stmmac (EMAC)

The SW Evaluation Design makes use of also other Linux drivers. Driver stmmac (CONFIG_STMMAC_ETH) is one of them. It creates a Linux net device for each Ethernet MAC. In this Evaluation Design EMAC is connected to FES CPU port.

EMAC settings in Qsys affect how the Ethernet interfaces need to be represented in device tree. Listing 6 shows definitions for FRS SoC evaluation board.

The phy-mode definition is used to tell EMAC driver how PHY is connected to the EMAC.

The fixed-link definition is used to tell EMAC driver that there is no real PHY, and to set the EMAC speed. It is used when EMAC is connected to an FES port. It causes a virtual PHY device with given PHY address to appear in virtual MDIO bus called "fixed-0" and to associate the EMAC net device with that PHY device. This requires the fixed-link driver (CONFIG_FIXED_PHY).

```

/* EMAC0, eth0 in Linux, connected to FES port 0 on FPGA */
&gmac0 {
    phy-mode = "gmii";
    status = "okay";

    fixed-link {
        speed = <100>;
        full-duplex;
    };

    /*
     * MDIO bus is not really used, but keep it
     * to avoid driver scanning the whole bus.
     */
};

```

Listing 6. Ethernet interface definitions

In custom designs the EMACs could be used differently and the device tree configuration would have to be adapted accordingly. For example, one might decide to use the other EMAC, so the other Ethernet interface definition would be added to device tree.

4.2.10 Marvell (PHY)

The SW Evaluation Design makes use of also other Linux drivers. Driver marvell (CONFIG_MARVELL_PHY) is one of them. It is a Linux PHY driver for Marvell 88E1111 PHYs present in most copper SFP modules.

4.3 User Space

The core user space environment consists of system programs, utilities and libraries from Debian GNU/Linux distribution [12]. Additional software from Flexibilis provides support for the features of FRS SoC Evaluation Design and its FES device. Those are described in the following.

4.3.1 XR7 PTP

XR7 PTP implements Precision Time Protocol. It is described in detail in XR7 PTP Design Specification [4]. XR7 PTP is modular software and uses dynamically linked shared object libraries on GNU/Linux systems. The following PTP modules are used in XRS Reference Software.

xr7ptp
This is the main program (daemon) which includes the XR7 PTP library, i.e. the main functionality of the PTP stack.

os_if
This library implements the OS interface on GNU/Linux for XR7 PTP library.

flx_frtc_clock_if
This library implements the Clock interface for the XR7 PTP library and uses FRTC as local clock. Thus when local device is a PTP slave, this library keeps local FRTC time synchronized with the PTP master clock. It also enables certain features of FES. Flx_time driver user space API (ioctl) is used to access the FRTC and library flx_fes_lib is used to access the FES.

flx_packet_if
This library implements the Packet interface for the XR7 PTP library and uses FES timestamping features. Library flx_fes_lib is used to access FES.

netconf (XR7 FCM “sync” module)

This library provides NETCONF interface to the PTP stack. It uses the Control and Configuration interfaces of the XR7 PTP library and implements XR7 FCM module API. FCM support for XR7 PTP is an optional component and requires XR7 FCM, see chapter 4.3.4.1 for more information.

host_clock_adj

This is a separate daemon which keeps the OS (Linux) clock synchronized to FRTC time.

4.3.2 XR7 Redundancy Supervision

XR7 Redundancy Supervision implements HSR/PRP Supervision protocol. It is described in detail in XR7 Redundancy Supervision Design Specification [5]. It is modular software and uses dynamically linked shared object libraries on GNU/Linux systems. Here is summary of the supervision modules used in FRS SoC SW Evaluation Design.

xr7_redundancy_supervision

This is the main program (control daemon).

supervision_lib

Supervision library implements the HSR/PRP Supervision protocol.

packet_lib

Packet library provides access to the Linux network stack and interfaces. It uses library lfx_fes_lib to access FES.

netconf (XR7 FCM “redundancy_supervision” module)

This library provides NETCONF interface to redundancy supervision. NETCONF is an optional feature and requires the XR7 FCM module, see chapter 4.3.4.1 for more information.

4.3.3 flx_fes_lib

The library contains helper functions for managing FES. The source code files are listed in Table 4.

File	Description
flx_fes.h flx_fes.c	Helper functions for configuring FES IP [7]. Includes for example reading and writing of FES registers and IPO settings.
flx_fes_rstp.h flx_fes_rstp.c	Helper functions for implementing RSTP.
flx_fes_aux.h flx_fes_aux.c	Functions for managing FES auxiliary network interfaces.
flx_fes_tc.h flx_fes_tc.c	Functions for managing FES traffic control (policing and shaping) features.
flx_fes_macsec.h flx_fes_macsec.c	Functions for managing FES MACsec features.

Table 4. flx_fes_lib files

4.3.4 XR7 Management Software

XR7 Management Software provides a web interface and an XML based protocol for configuring devices and examining their status. It consists of three parts which are briefly described in the following.

More information is available from Flexibilis, please send E-mail to contact@flexibilis.com.

4.3.4.1 XR7 FCM

XR7 FCM stands for Flexibilis Configuration Manager. It is an implementation of IETF NETCONF [11] network management protocol. FCM design is modular. The daemon itself implements the protocol while FCM modules, implemented as dynamically linked shared

object libraries, provide NETCONF support for specific system components like XR7 PTP, XR7 Redundancy Supervision, network interfaces and so on.

FCM modules communicate with FCM using local sockets and can thus be integrated into other daemons that actually handle the tasks related to the FCM module. For example FCM module named sync implements time synchronization using XR7 PTP and runs in `xr7ptp` daemon process.

4.3.4.2 XR7 IFM

XR7 Interface Manager (IFM) is a daemon which provides NETCONF support for various network interfaces. Its design is modular, each module is implemented as a dynamically linked shared object library with XR7 FCM module interface. The following modules are used in the XRS Reference Software.

ethernet

This module provides Ethernet interface status and configuration for external Ethernet interfaces CE01, CE02, CE03 and CE04. For example link speed and mode can be changed or current auto-negotiation status can be retrieved. It also provides FES port statistic counters.

vlan

This module provides VLAN configuration support for FES.

ip

This module provides IP address configuration for the system. Note that the FES CPU port net interface is used for normal networking, so the IP address is set to that Linux network interface.

routing

This module provides static routing configuration for the system.

4.3.4.3 XR7 GUI

XR7 GUI provides web interface to the device for presenting status information and for configuring the system as desired by user. It is implemented as a Java servlet and uses NETCONF to access device resources.

In FRS SoC SW Evaluation Design Apache Tomcat is used as the web server and servlet engine. User can access the GUI from address <https://192.168.7.1/> (when default IP address is configured).

4.3.5 SSH Server

OpenSSH server is used. SSH subsystem name `netconf` is configured to use `fcm_manager` so that NETCONF requests over SSH are forwarded to FCM.

4.3.6 Debian

FRS SoC SW Evaluation Design is built on Debian [12] GNU/Linux distribution.

Most of the software from Flexibilis do not require any specific GNU/Linux distribution, and many can be used in other operating systems than Linux.

4.4 Compilation

Each software module in FRS SoC Evaluation Design is built from sources separately.

4.4.1 Toolchains

A bare-metal cross-compiler toolchain is needed for building the bootloaders. One is included with Altera SoC EDS and it is used in the SW Evaluation Design from Embedded Shell.

Another cross-compiler toolchain is needed to build the Linux kernel, drivers and user space software. There are a couple of alternatives available, including but not limited to:

- One from ARM Development Studio 5 provided with Altera SoC EDS
- Linaro GCC
- Mentor Graphics Sourcery CodeBench
- GCC from Emdebian [13]
- Building own toolchain

GCC for ARM GNU/Linux from Emdebian is used with the SW Evaluation Design.

Device tree compiler is needed to turn device tree from source form to binary form or vice versa. It is available in the Linux kernel source tree at `scripts/dtc/`. Many GNU/Linux distributions also package the `dtc` utility, on systems based on Debian it is available from package `device-tree-compiler`.

Preparing an SD-card or SD-card image requires various GNU/Linux tools. Following is a non-complete list of some of the non-POSIX.1 tools that are used with the SW Evaluation Design:

- GNU parted
- kpartx
- mkfs.ext4
- tune2fs
- mount
- umount

Additionally following tools are used with the SW Evaluation Design to collect files needed on SD-card to a separate directory before putting them in SD-card image:

- debootstrap
- qemu-arm-static
- apt-get and other related Debian package management tools

Custom designs may naturally choose a different way and different tools for boot image generation related phases.

4.4.2 Preloader and U-Boot

Preloader and U-Boot are built according to Altera SoC EDS instructions within its embedded shell. They can be built at the same time. Note that Qsys and Quartus II settings can affect the resulting Preloader binary. Sample Preloader and U-Boot build commands are shown in Listing 7. Names of the resulting files are `preloader-mkpimage.bin` and `u-boot.img`. See Figure 12 for directory structure.

```
# Prepare to rebuild.
mkdir -p cyclone5soc_eval/fpga/bin_qsys/bsp
cd cyclone5soc_eval/fpga/bin_qsys/bsp
rm -rf *

bsp-create-settings \
    --settings settings.bsp \
    --type spl \
    --preloader-settings-dir \
    ../hps_isw_handoff/soc_system_hps_0/ \
    --bsp-dir .

# Target all builds Preloader and target uboot builds U-Boot.
make all uboot

# Take final U-Boot binary from intermediate directory.
cp uboot-socfpga/u-boot.img .
```

Listing 7. Preloader and U-Boot generation

See Table 5 for placement of Preloader and U-Boot binaries on the SD-card. Example command to install custom built Preloader from a GNU/Linux system to the SD-card:

```
sudo dd if=preloader-mkimage.bin of=/dev/sdX2 bs=64k
```

Example command to install custom built U-Boot to the SD-card:

```
sudo dd if=u-boot.img of=/dev/sdX2 bs=64k seek=4
```

Above commands assume that SD-card is available as block device `/dev/sdX`. The same commands can be used to install the Preloader and U-Boot to an SD-card image by replacing the `of` parameter value with corresponding device from device mapper.

4.4.3 Linux Kernel

Linux kernel source code is available from a Git repository. Altera SoC EDS provides a script to download and set it up. See also the README file provided with the FRS SoC Evaluation Design.

The patch for EMAC to FPGA connection provided with FRS SoC Evaluation Design must be applied. Provided Linux kernel configuration must also be used. Detailed build instructions are provided with the FRS SoC Evaluation Design release.

In order to install custom built Linux kernel to the SD-card just copy the resulting `uImage` file to the first SD-card partition using a GNU/Linux system. See Table 5 for details.

4.4.4 Linux Drivers for Flexibilis IPs

Linux driver source code is available with the FRS SoC Evaluation Design release. Drivers are built against configured Linux kernel source tree. Example build commands:

```
OPTS="ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- KDIR=<KERNEL_DIR>"
make $OPTS -C flx_time
make $OPTS -C flx_frtc
make $OPTS -C flx_frs
make $OPTS -C flx_pio
```

Replace `<KERNEL_DIR>` with a path to the configured Linux kernel source tree.

4.4.5 Device Tree

Device tree is compiled from source form (.dts) to binary form (.dtb) using device tree compiler `dtc`. The following command turns source file `socfpga.dts` to binary file `socfpga.dtb`:

```
dtc -I dts -O dtb -o socfpga.dtb socfpga.dts
```

The following command turns a binary file `socfpga.dtb` to source file `socfpga.dts`, but note that some constructs may appear differently than in the original source file:

```
dtc -I dtb -O dts -o socfpga.dts socfpga.dtb
```

4.4.6 Other Flexibilis Software

Each Flexibilis product is delivered as a separate release package with detailed build instructions. Please follow them to build the software.

Note that a license is required from Flexibilis for the source code of Flexibilis products.

4.4.7 Third Party Software

FRS SoC SW Evaluation Design SD-card image contains software from Debian GNU/Linux distribution. The SW Evaluation Design is provided for FES evaluation purposes and for that there is no need to rebuild all the software.

The following core Flexibilis software in the FRS SoC SW Evaluation Design do not require any specific third party libraries or other third party software, other than a toolchain for building and a C-library for user space parts:

- Linux drivers for FES and FRTC
- XR7 PTP without FCMD support
- XR7 Redundancy Supervision without FCMD support
- `flx_fes_lib`

Information about other Flexibilis software is available from Flexibilis, please send E-mail to contact@flexibilis.com.

4.5 SD-Card

FRS SoC SW Evaluation Design is booted from SD-card. Structure of the SD-card is shown in Table 5. Ext4 filesystem is chosen for partition 1 because U-Boot supports it and ext4 filesystem supports POSIX.1 file access permissions, unlike FAT filesystems. It works better with Debian package management. Additionally, ext4 filesystem is more immune to situations like abrupt power loss, than for example ext2. However not all ext4 features can be used because they are not supported or do not work reliably in U-Boot.

Partition	Type	Filesystem	Contents
-	-	-	MBR U-Boot environment
1	0x83	ext4	soc_system.rbf socfpga.dtb ulmage
2	0xA2	-	Preloader U-Boot
3	0x83	ext4	Root filesystem

Table 5. SD-card structure

Sample GNU/Linux commands to create an SD-card image file with partitions are shown in Listing 8. GNU parted is used to create the partition table. Unfortunately it does not support custom partition types, so partition 2 type is changed separately. The script uses sectors (of size 512 bytes) as units so that partition start addresses and sizes can be guaranteed for desired alignment (here 4 MiB). Proper alignment is good for flash memory endurance and

performance.

```
# Create sparse SD-card image file socfpga.raw of size 512 MiB.
image="socfpga.raw"
dd of="$image" bs=$((1*1024*1024)) count=0 seek=512

# Create MBR and partitions with 4 MiB alignment.
parted -s "$image" \
    mklabel msdos \
    mkpart primary ext4 8192s 139263s \
    mkpart primary ext4 139264s 147455s \
    mkpart primary ext4 147456s 786431s \
    quit

# Change partition 2 ID to 0xA2.
partition=2
type=0xA2
printf '\$(echo "obase=8 ; $((($type)))" | bc) |
    dd of="$image" conv=notrunc \
    bs=1 seek=$((0x1BE + 16*($partition - 1) + 4)) count=1
```

Listing 8. Script for creating partitions on SD-card image

Filesystem creation and mounting for populating is shown in Listing 9.

```
# Use device mapper to create block devices for partitions
# on image file.
devs=`kpartx -a -v "$image" |
    awk '/^add map .*p[0-9]+/ { print $3 }'`
part1=/dev/mapper/`echo "$devs" | sed -n 1p`
part2=/dev/mapper/`echo "$devs" | sed -n 2p`
part3=/dev/mapper/`echo "$devs" | sed -n 3p`

# Create filesystem on partition 1.
mkfs.ext4 -q -L boot -O ^huge_file,^extent "$part1"

# Make it more tolerant for power losses.
tune2fs -o journal_data "$part1"

# Create filesystem on partition 3.
mkfs.ext4 -q -L rootfs -O ^huge_file "$part3"

# Make it more tolerant for power losses.
tune2fs -o journal_data "$part3"

# Mount root and boot filesystems on $MNT.
MNT="$PWD/tmp"
mount -o noatime,data=writeback,journal_async_commit \
    "$part3" "$MNT"
mkdir "$MNT/boot"
mount -o noatime "$part1" "$MNT/boot"

# Populate filesystems here.
# ...

# Unmount filesystems.
umount "$MNT/boot"
umount "$MNT"

# Remove partitions from device mapper.
kpartx -d "$image"
```

Listing 9. Script for creating filesystems on SD-card image

5 Customization

Evaluation design can be modified in various ways to either evaluation or test different designs, or to create own designs. This chapter explains how to make some changes.

5.1 Changing FES CPU Port Speed

The following must be set correctly to connect SoC EMAC to FES CPU port successfully:

- EMAC PHY interface
- EMAC link mode (speed and duplex)
- FES port mode (GMII vs. MII) and speed

EMAC PHY interface in SoC HPS system manager EMAC group control register is determined from EMAC device tree phy-mode parameter. The register has three choices:

- GMII/MII
- RGMII
- RMII

The EMAC connected to FES CPU port normally must be set to GMII/MII mode.

EMAC link mode (speed and duplex) is set by the EMAC driver according to what PHY reports as active link mode. Because there is no PHY, the fixed-link PHY driver is used for the purpose. To set the speed, a fixed-link device tree parameter must be set correctly. See Listing 6.

FES port link mode must also be set in FES CPU port PORT_STATE register. The (virtual) PHY device reports the link mode to the EMAC driver, but not to the FES CPU port driver. FES CPU port link mode can be set after FES driver is loaded by ETHTOOL ioctl, for example using ethtool command from an init script. Example:

```
ethtool -s SE01 autoneg off speed 100 duplex full
```

5.2 Changing PHY address

EMAC and SFP module PHY addresses are configured in device tree. EMAC PHY address is set in the EMAC device node. FES port PHY address is configured in the PHY device node, which is referenced to from FES port node. For example if FES port PHY is connected to a different MDIO bus, a PHY device node needs to be added to the host device of the MDIO bus. Then that PHY device is referenced from the FES port node phy-handle parameter and also phy-mode is set to correct PHY interface mode.

SFP modules seem to have a fixed I²C slave address to PHY address mapping.

5.3 FES Port without a PHY

Use correct FES port adapter in the FPGA design. If FPGA rebuild is necessary, update the following:

- .rbf file to SD-card
- Rebuild Preloader and U-Boot and write to SD-card 0xA2 partition
- FES port adapter register address in device tree
- FES port medium type value in device tree
- If necessary, use ethtool command to set the speed correctly, example:

```
ethtool -s CE01 autoneg off speed 1000 duplex full
```

5.4 Adding an FES Port

Do the following:

- Update FPGA design, use correct adapter

- Set and note new FES port and adapter register addresses in Qsys
- Regenerate Qsys
- Recompile the design
- Rebuild Preloader and U-Boot
- Update BSP device tree
 - o Add new node for the new port
 - o Add pio and i2c nodes with correct content for the new port if necessary
- Convert device tree from source form (.dts) to binary form (.dtb)
- Update the following to already created SD-card
 - o New FPGA .rbf file to partition 1
 - o Device tree binary file to partition 1
 - o Preloader and U-Boot to SD-card 0xA2 partition
 - o Add new FES port to boot time configuration, for example to FCM Ethernet module and FCM sync module factory and startup configurations

5.5 Change AXI Bus Type

FPGA blocks on Altera SoC FPGA can be connected either to the lightweight AXI bus or normal AXI bus. HPS sees the buses at different addresses. Base address of the normal AXI bus is 0xC0000000 and base address of the lightweight AXI bus is 0xFF200000. Also note that the size of the lightweight AXI bus address range is much smaller.

Do the following to change the AXI bus type:

- Update FPGA design
- Set and note new register addresses in Qsys
- Regenerate Qsys
- Recompile the design
- Rebuild Preloader and U-Boot
- Update all FES switch, FES port, FES port adapter, FRTC and PIO registers in device tree
- Convert device tree from source form (.dts) to binary form (.dtb)
- Update the following to already created SD-card
 - o New FPGA .rbf file to partition 1
 - o Device tree binary file to partition 1
 - o Preloader and U-Boot to SD-card 0xA2 partition

6 Troubleshooting

This chapter describes various methods to verify and debug FES designs. These instructions and tips are generally valid also for custom FES designs, however the following is assumed:

- There is some HW which includes an FES design
- Linux is used
- Drivers from Flexibilis are used for FES and FRTC
- Device tree for design contains adapted FES and FRTC information
- System is bootable to Linux
- Console is available, either serial console or over network, for example using SSH
- proc filesystem is mounted in /proc
- At least the following common GNU/Linux utilities are available:
 - o cat, ls, find, grep
 - o depmod, insmod, modprobe, lsmod
 - o dmesg
 - o ip
 - o ethtool

If all the ports of the FRS SoC Evaluation board stop working, it's likely that the evaluation time limit has expired. The only possible fix for this is power recycle.

6.1 Driver Loading

Typically Flexibilis drivers are built as kernel modules, although it is possible to build them also directly into the Linux kernel.

Use `lsmod` command to see list of loaded kernel modules.

6.1.1 Letting Drivers Load Automatically

Often kernel modules are placed under `/lib/modules/<VERSION>` where `<VERSION>` is the Linux kernel version number, for example `3.10.31-ltsi`. Then drivers can be loaded automatically, which requires that `depmod` command has been used to update module dependencies. If necessary, run `depmod` and `reboot`.

In some cases it may be necessary to use `modprobe` command to load the drivers.

With some HW and FPGA designs the drivers may have to be loaded in specific order. In that case it is recommended to load drivers explicitly as described in the following section.

6.1.2 Loading Drivers Explicitly

If drivers are placed elsewhere than under `/lib/modules/<VERSION>`, then `insmod` command must be used to load the drivers. Typically this is done from an init script at system boot time, when drivers have to be loaded in specific order or at specific time. This may also be useful when the set of drivers that needs to be loaded may change between boots.

6.1.3 Driver Load Verification

Flexibilis drivers typically output something to kernel ring buffer (dmesg log), when drivers are bound to devices. Use `dmesg` command after loading the drivers to see them. Remember that usually not everything output to kernel ring buffer is output to console, so it is better to use the `dmesg` command than to rely on boot time prints on console.

Examples of such prints are:

```
flx-pio ff240000.gpio: Added PIO device ff240000.gpio GPIOs 254 ..
255
flx-pio ff240100.gpio: Added PIO device ff240100.gpio GPIOs 252 ..
253
```

```
flx-pio ff240200.gpio: Added PIO device ff240200.gpio GPIOs 250 ..
251
flx-pio ff240300.gpio: Added PIO device ff240300.gpio GPIOs 248 ..
249
```

```
FLX_TIME: Register NCO component(s).
FLX_TIME: probe for device ff370000.frtc.
```

```
flx_frs: Setup device 0 for memory mapped access
flx_frs ff300000.frs0: Device uses memory mapped access:
0xff300000/0x8000 -> c0990000
flx_frs ff300000.frs0: Port 0 uses memory mapped access:
0xff310000/0x10000 -> c0ae0000
flx_frs ff300000.frs0: Port 0 adapter uses memory mapped access:
0xff360000/0x200 -> c098e000
flx_frs ff300000.frs0: Port 1 uses memory mapped access:
0xff320000/0x10000 -> c0b00000
flx_frs ff300000.frs0: Port 1 adapter uses memory mapped access:
0xff360200/0x200 -> c099a200
flx_frs ff300000.frs0: Port 2 uses memory mapped access:
0xff330000/0x10000 -> c0b20000
flx_frs ff300000.frs0: Port 2 adapter uses memory mapped access:
0xff360400/0x200 -> c099c400
flx_frs ff300000.frs0: Port 3 uses memory mapped access:
0xff340000/0x10000 -> c0b40000
flx_frs ff300000.frs0: Port 3 adapter uses memory mapped access:
0xff360600/0x200 -> c099e600
flx_frs ff300000.frs0: Port 4 uses memory mapped access:
0xff350000/0x10000 -> c0b60000
flx_frs ff300000.frs0: Port 4 adapter uses memory mapped access:
0xff3f0600/0x200 -> c0ac6600
flx_frs ff300000.frs0: FRS IRQ 75 allocated
flx_frs ff300000.frs0: FRS SW reset done
```

FES and FRTC drivers also create files in `/proc/driver` for each device. Use `find` or `ls` command to see the file names and `cat` command to see their contents. Example:

```
find /proc/driver
```

If there are no files even though driver is loaded, the driver is not bound to a device. This may mean for example a problem with device tree contents or a problem with device initialization.

6.2 FES

FES specific troubleshooting and debug tips follow.

6.2.1 Missing Functionality

In case something seems to be missing verify that device tree contains correct set of features and that feature parameters are correct inside the `features` node. See sections 4.2.3.1 and 4.2.3.3.

6.2.2 FES Switch Register Access

Use FES driver `/proc` files to verify that FES switch registers can be accessed correctly. Compare for example ID and configuration ID values to FES manual or to a working system. This could reveal for example a typo in device tree switch register address (FES instance `reg` parameter value). This could also reveal for example bus access timing problems or bus byte order problems in case of custom HW design. Example:

```
# cat /proc/driver/flx_frs/device00_common_registers
```

```
Common Registers of device 0:
FRS ID0                (0x0000): 0x0000
FRS ID1                (0x0001): 0x4000
FRS configuration ID   (0x0002):      6
...
```

6.2.3 FES Port Register Access

Use FES driver `/proc` files to verify that FES port registers can be accessed correctly. This could reveal for example typos in device tree (FES port instance `reg` parameter value).

Example:

```
# cat /proc/driver/flx_frs/device00_port_registers

Port registers of device 0      (REG):  PORT0  PORT1  PORT2  PORT3  PORT4

State                          (0x0000):  0x0204 0x0120 0x0002 0x0002 0x0120
VLAN                           (0x0008):  0x8fff 0x8fff 0x8fff 0x8fff 0x8fff
...
```

6.2.4 FES Port Adapter Register Access

Some designs use FES port adapters to connect FES ports to other interface types. Use FES driver `/proc` files to verify that FES port adapter registers can be accessed correctly and that design contains correct adapters for each port. This could also reveal for example typos in device tree (FES port instance `reg` parameter value). Example:

```
# cat /proc/driver/flx_frs/device00_adapter_registers

Adapter registers of device 0

          (REG):  PORT0      PORT1      PORT2      PORT3      PORT4

ID          (0x0000):  0x0000      0x01b2      0x01b2      0x01b2      0x01b2
Type        :          -          SGMII/1000  SGMII/1000  SGMII/1000  SGMII/1000
Link status (0x0001):  0x4000      0x0001      0x0000      0x0000      0x0001
...
```

Note that some adapters provide link status information in adapter registers, while others expect current link status to be written to adapter registers. FES driver handles it automatically for known and recognized adapter types when adapter register addresses have been configured correctly.

Unrecognized or custom adapters may require use of separate drivers. In that case adapter register addresses should not be defined for FES, unless FES driver is modified to handle also those adapters.

6.2.5 FES Port Link Status and Speed for Copper Interfaces

Copper interfaces use an Ethernet PHY. Thus the Linux net device for such external ports, for example CE01, should be attached to a Linux PHY device. This is configured in device tree and indicated in the kernel ring buffer (`dmesg log`) with a line like this:

```
CE01: Attached PHY driver [Marvell 88E1111] (mii_bus:phy_addr=flx-
i2c-mdio-0:16)
```

If there is no such line, then there is likely a problem with accessing the PHY and the interface may not work correctly in all situations. For example it may or may not work with autonegotiation, but forcing a specific link speed would not be possible. At least not the normal way.

Use `ethtool` command to check the link mode:

```
# ethtool CE01
```

Settings for CE01:

```
Supported ports: [ TP MII ]
Supported link modes:  10baseT/Full
                      100baseT/Full
                      1000baseT/Full

Supported pause frame use: No
Supports auto-negotiation: Yes
Advertised link modes: 10baseT/Full
                      100baseT/Full
                      1000baseT/Full

Advertised pause frame use: No
Advertised auto-negotiation: Yes
Speed: 1000Mb/s
Duplex: Full
Port: MII
PHYAD: 22
Transceiver: external
Auto-negotiation: on
Supports Wake-on: d
Wake-on: d
Current message level: 0x00000007 (7)
                      drv probe link

Link detected: yes
```

Use FES driver `/proc/driver/flx_frs/device<NN>_port_registers` file to verify that FES port is in correct state (GMII vs. MII and correct speed).

Use FES driver `/proc/driver/flx_frs/device<NN>_adapter_registers` file to verify that FES port adapter, if such is used, is in correct state. See the adapter documentation for meaning of register values. In some cases this may reveal problems with the FPGA or HW design.

6.2.6 Use of Correct PHY Driver

Linux PHY framework detects the type of PHY from PHY ID registers (register numbers two and three). It uses that information to decide which of the loaded PHY drivers to use to handle the PHY device.

Many PHY devices can work with the generic PHY driver. But many PHY devices require a specific driver to work correctly. This can also depend on the HW. It is possible that on some HW design PHY can work with the generic PHY driver while on a different HW design a specific PHY driver is needed, for example if HW design is such that certain PHY specific register values must be changed from their default values.

FES driver shows the name of PHY driver used for an FES port in kernel ring buffer (dmesg log). Example:

```
CE01: Attached PHY driver [Generic PHY] (mii_bus:phy_addr=flx-i2c-
mdio-0:16)
```

If the link does not work with the generic PHY driver, check if the Linux kernel contains a specific PHY driver for your PHY. The PHY drivers are in `drivers/net/phy` directory in kernel source tree. Enable the relevant PHY driver in kernel configuration and ensure the module is loaded at boot time before FES driver, or build the PHY driver directly into Linux kernel. Use `dmesg` command to see that the PHY driver really accepts the PHY.

In some cases a newer Linux kernel may contain a suitable PHY driver. In other cases it may be necessary to write a new driver for the PHY.

6.2.7 FES Port Link Status and Speed for Fiber Interfaces

Fiber interfaces do not use an Ethernet PHY. For some adapter types FES driver can configure the FES port speed and adapter correctly from available information, for example using the adapter type whether configured PHY device was found or not. But in some cases FES driver may have to be told which mode to use by ETHTOOL ioctl, see section 4.2.3.4.

6.2.8 SFP Module Change Detection

When FES port has been configured for medium type SFP, FES driver tries to detect when SFP module is changed. This may not work in all cases, because Linux PHY framework is not really designed to handle dynamically disappearing and appearing of PHY devices on an MDIO bus.

Together with flx_i2c_mdio driver SFP module change detection works. If flx_i2c_mdio is used with other drivers it may be necessary to disable this feature by this directive in device tree for flx_i2c_mdio node:

```
disable-change-detection;
```

6.2.9 Traffic Problems

First check that link is up and link status is correct in all places: from PHY driver, in FES port state register, in FES port adapter registers, and also on the link partner side.

Then check FES port statistics counters. They are very useful in some cases to narrow down the problem, because different counters for good and bad octets and for different types of frames are available for each FES port and for both directions. This makes it possible to track the flow of frames from source to destination through FES and back.

Use `ethtool` command to see the statistics counters. Example:

```
ethtool -S SE01
ethtool -S CE01
```

Example scenario: Communication is attempted from CPU with another device through FES. Frames are sent from CPU through an EMAC connected to FES CPU port. Frames are expected to be forwarded by FES from CPU port SE01 to external interface CE01 which is directly connected to the destination device. The destination device is expected to send responses back and it is expected that the responses travel the same path in the reverse order.

First ensure that there are no daemons running on either side which could generate any extra traffic and that FES and FES ports are in correct state. Then start to send frames from CPU. Check that the OS actually tries to send packets to the EMAC which is connected to FES CPU port. The transmit counters of both FES CPU port (SE01) and EMAC net device (for example eth0) should increase without error counters increasing. Example:

```
cat /proc/net/dev ; sleep 1 ; cat /proc/net/dev
```

Then verify that FES CPU port RX counters increase (`rx_good_octets`):

```
ethtool -S SE01
```

Then verify that FES forwards the frames to CE01: `tx_octets` should increase.

```
ethtool -S CE01
```

Then verify that the destination device receives the frames.

If everything looks good, verify that the problem is not in the other direction. Verify that destination device actually sends the responses back.

Then verify that the FES CE01 port receives the frames: `rx_good_octets` should increase.

```
ethtool -S CE01
```

Then verify that FES forwards the response frames to CPU port: tx_octets should increase:

```
ethtool -S SE01
```

Then verify that the frames are pass through the EMAC by checking FES CPU port net device receive counters (this time SE01 only, as FES driver passes received frames always from its own net devices to OS).

```
cat /proc/net/dev
```

If frames are sent by OS towards EMAC, but FES CPU port RX counter never increases, the problem may be related to the EMAC.

If frames received to FES CPU port are not forwarded to other FES ports, or responses are not forwarded from external ports to FES CPU port, the problem may be in FES IPO entries or in FES VLAN configuration. Check also FES port forward mask register value.

If frames sent out from external FES port do not appear at the destination, there may be a problem with the adapter connection or PHY or with the external device. It may also be useful to test with other types of destination devices to rule out some HW interoperability problems. PHY devices may also provide useful counters or status information. Also make sure the PHY device is put in correct interface mode.

Check also FPGA design clocks and try with different link speeds.

Here's a simple checklist:

- link status and mode (net device, adapter, link partner)
- statistics counters
- port modes (normal, HSR, PRP, interlink)
- IPO configuration
- VLAN configuration
- SMAC table
- traffic policing configuration
- traffic shaper configuration
- MACsec configuration
- FPGA design
- device tree correctness
- HW

6.2.9.1 RGMII

Remember that RGMII requires clock signal delay. Many PHYs support different internal delay modes, which can be enabled using `phy-mode` parameter in device tree if the PHY driver supports it. The relevant modes are: "rgmii-id", "rgmii", "rgmii-rxid" and "rgmii-txid". In some cases some other method may have to be used to ensure clock compatibility.

6.3 FRTC

A few checks for FRTC follow.

6.3.1 Checking FRTC Is Running

Use `/proc/driver/flx_time/component_<NN>_registers` file to see FRTC register values. <NN> is flx_time index number of the clock, starting from 00. Example:

```
# cat /proc/driver/flx_time/component_00_registers
```

```
Component index: 0
name           : Local NCO
device id      : 0x0090
revision id    : 0x02
```

```

properties      :    0x1f

Time read:
seconds        : 137409
nanoseconds    : 434575818
subnsecs       : 0x0000
clk cycle cnt: 0x00000f9f29d35bf1

Register content:
nco subnsec reg      :    0x00000000
nco nsec reg         :    0x19e719ca
nco sec reg          : 0x0000000218c1
nco ccnt reg         : 0x0f9f29d35bf1
nco step subnsec reg :    0x00000000
nco step nsec reg    :         0x08
nco adj nsec reg     :    0x1614ecf2
nco adj sec reg      : 0x000000000012
nco cmd reg          :         0x00

```

By default FRTC step size register values can be zero. When FRTC driver is loaded, it writes configured nominal step size value to the step size registers and FRTC starts running. Check that seconds and nanoseconds values increase.

6.3.2 Rough FRTC Frequency Check

Get FRTC register values at for example ten seconds apart and compare elapsed time to wall clock time. Example:

```

# (cat /proc/driver/flx_time/component_00_registers ; sleep 10 ; cat
/proc/driver/flx_time/component_00_registers) | grep seconds
seconds      : 137739
nanoseconds  : 771582498
seconds      : 137749
nanoseconds  : 778224842

```

Note that if OS clock does not run at correct frequency, actual time between above prints may deviate a lot from ten seconds wall clock time. Calculate elapsed FRTC time and divide it by elapsed wall clock time. The result should be close to one.

If FRTC does not run at roughly the correct frequency, check that the nominal step size value is configured correctly. You may also want to check that FPGA design clock part is correct and that the board in question actually has a correct clock chip.

7 Abbreviations

Term	Description
AFEC	Advanced Flexibilis Ethernet Controller
BSP	Board Support Package
CPU	Central Processing Unit
EMAC	Ethernet Media Access Control
FCM	Flexibilis Configuration Manager
FES	Flexibilis Ethernet Switch
FPGA	Field Programmable Gate Array
FRS	Flexibilis Redundant Switch
FRTC	Flexibilis Real Time Clock
HPS	Hard Processor System
HSR	High-availability Seamless Redundancy
MAC	Media Access Control
MDIO	Management Data Input/Output
MII	Media-Independent Interface
MMC	MultiMediaCard
NCO	Numerically Controlled Oscillator
NETCONF	Network Configuration Protocol
OCRAM	On-Chip Random-Access Memory
OS	Operating System
PIO	Parallel Input/Output
PLL	Phase Locked Loop
PPS	Pulse Per Second
PRP	Parallel Redundancy Protocol
PTP	Precision Time Protocol
RGMII	Reduced Gigabit Media-Independent Interface
ROM	Read-Only Memory
RSTP	Rapid Spanning Tree Protocol
SD	Secure Digital
SDRAM	Synchronous Dynamic Random-Access Memory
SFP	Small Form-factor Pluggable transceiver
SoC	System-on-a-Chip
SPL	Software Preloader

8 References

- [1] SGMII/1000Base-X Adapter Specification, FLXD816, version 1.0
- [2] Standard IEC 62439-3:2011
- [3] IEEE standard 1588-2008
- [4] XR7 PTP design specification, xr5_ptp_design.pdf
- [5] XR7 Redundancy Supervision design specification, xr7_redundancy_supervision_design.pdf
- [6] Advanced Flexibilis Ethernet Controller (AFEC) User Manual, AFEC_user_manual.pdf
- [7] Flexibilis Ethernet Switch (FES) Manual, FES_Manual.pdf
- [8] FRTC User Manual, FRTC_user_manual.pdf
- [9] IP License Authentication, Security Chip, Manual, version 1.2.
http://www.flexibilis.com/downloads/Security_Chip.pdf
- [10] Interface Adapter Specification, FLXD817, version 1.0
- [11] NETCONF, RFC 4741, <http://tools.ietf.org/html/rfc4741>
- [12] Debian, <https://www.debian.org/>
- [13] Emdebian, <http://emdebian.org/>
- [14] Altera SoC Embedded Design Suite User Guide,
http://www.altera.com/literature/ug/ug_soc_eds.pdf
- [15] TTTech Flexibilis download portal
<https://www.ttech-industrial.com/products/flexibilis/> or
https://www.flexibilis.com/downloads/refdesigncheck_soc.php
- [16]