# S32G Linux PFE Driver User's Manual

# Contents

# Chapter 1

# Revision History

| Revision | Change Description |
|---|---|
| preEAR 0.4.0 | Initial version for preEAR (FPGA/x86 platform) [JPet] |
| preEAR 0.4.1 | Added VDK info [JPet] |
| preEAR 0.4.3 | Removed VDK, Added S32G [JPet] |
| EAR 0.8.0 | Added device-tree config [JPet] |
| EAR 0.8.0 P1 | Added libfci [JPet] |
| BETA 0.9.0 | DTS update for kernel 5.4-rt [JPet] |
| BETA 0.9.1 | Added Master/Slave [JPet] |
| BETA 0.9.2 | Added performance info [JPet] |
| BETA 0.9.3 | Added PTP [OSpac], Added cut1.1 info [JPet] |
| BETA 0.9.4 | Added mem internals [CMan], STR, DT updates [JPet] |
| BETA 0.9.5 | Updated STR, DT, Added AUX [JPet]<br>Removed cut 1.1 references. [CMan] |

Table 1.1: Revision History

# Chapter 2

# Introduction

The Linux PFE driver is, in general, an OS-specific SW component responsible for the complex PFE management. It consists of three main functional blocks:

- The platform driver
  This part covers the initial, low-level PFE HW bring-up, configuration, firmware upload, and the SW representation of the PFE HW components. This part can be described as a low-level PFE driver providing the interface to the hardware (including the firmware).

- The data-path driver
  Data-path driver covers tasks related to the Ethernet data traffic in terms of passing the packets between the PFE and the networking stack. This includes the implementation of the Host Interface (HIF) driver and connection with the host OS-provided the networking stack.

- Control path driver
  The block in charge of the functionality related to runtime PFE engine configuration and monitoring, implementing the Fast Control Interface (FCI) endpoint which is accessible from the user-space through related FCI library.

The driver runs within the target host OS environment and connects the PFE with networking stack. It provides access to physical Ethernet interfaces via exposing logical interfaces to the OS, and the OS together with user's applications can use the Ethernet connectivity via standard OS-provided interfaces (i.e. network sockets).

# Chapter 3

# Building Procedure

## 3.1 Building the driver

The PFE driver consists of number of smaller software modules. The main module producing the final driver is called *linux-pfeng* and depends on all the others. Successful build gives the final driver library *pfeng.ko* which is an Ethernet driver for particular Linux kernel. Its location within the project tree in `sw/linux-pfeng/`. There are two ways to build the driver:

1. Integrated build by NXP Automotive Linux BSP, powered by Yocto
2. Standalone

### 3.1.1 Variant 1: Building within NXP Auto Linux BSP source tree by Yocto

1. As prerequisite, save the PFE firmware files to any location on the local disk. Ask NXP representative for PFE firmware package.
2. Build standard NXP Auto Linux BSP to check that all necessary componens are ready. This step is not only for verification but it precompiles most of necessary part of BSP which will be used later for creating sdcard image with included pfeng Linux driver.
3. Configure additional component by adding the following two lines to the conf/local.conf

   ```
   DISTRO_FEATURES_append = " pfe "
   PFE_LOCAL_FIRMWARE_DIR = "<directory path where fw file was saved>"
   or (on ALB BSP 28.0 or higher)
   NXP_FIRMWARE_LOCAL_DIR = <directory path where fw file was saved>"
   ```

4. Rebuild BSP to get PFE drivers + firmware included in the sdcard image

### 3.1.2 Variant 2: Building standalone

1. As prerequisite check all necessary development requirements:
   (a) Host development GNU toolchain, including GNU-cc, GNU-make
   (b) Linux kernel development files
2. Go to `sw/linux-pfeng/`.

3. Make sure that the following environment variables are set and points to the right directories:

   (a) KERNELDIR

   The direcory where the Linux kernel development files are located.

   (b) Optionally also PLATFORM

   The name of the GNU toolchain platform. In case of NXP Auto Linux BSP, it is "aarch64-linux-gnu"

4. Clean working directories:

```
make KERNELDIR=<path to kernel> PLATFORM=aarch64-linux-gnu drv-clean
```

5. Build the driver:

```
make KERNELDIR=<path to kernel> PLATFORM=aarch64-linux-gnu all
```

# Chapter 4

# Usage

## 4.1 Prerequisites

### 4.1.1 Hardware: NXP S32G274A silicon platform EVB or RDB2

For usage the software on S32G EVB or RDB2 boards, the following shall be fullfiled:

- Compatible PFE firmware files.

  ☞ *The compatibility requirements can be find in the driver public repository: https://source.codeaurora.org/external/autobsp32/extra/pfeng*

- Auto Linux BSP version 30.0 or higher for creation of sdcard bootable image

  ☞ *Note that PFE firmware is being delivered as a standalone product package.*

### 4.1.2 Software: Kernel and U-boot

The Linux driver has the following software dependencies:

1. PFE Controller reset

   The Auto Linux BSP provides kernel with S32G Reset driver.

   In case no driver is used (no '*resets*' property configured in DT), the PFE reset must be done in bootloader.

2. SGMII support

   To have SGMII connected devices fully configurable, the kernel SerDes driver must be involved ('*phys*' property configured in DT).

   Otherwise S32G SerDes must be preconfigured from bootloader.

3. Clocking

   Native Linux kernel clock driver is required ('*clocks*' property configured in DT).

## 4.2 Supported development boards

NXP offers two development boards for S32G274A:

1. S32G-VNP-EVB
2. S32G-VNP-RDB2

Please refer to board specific user guide to check possible ethernet connectivity. Also check '*NXP Automotive Linux BSP User Manual*' which provides comprehensive information for ethernet controllers, including PFE, with supported configurations for U-boot and Linux.

## 4.3 Running the driver

1. The *pfeng.ko* driver is embedded in the Auto Linux BSP sdcard image and as such is automatically loaded on Linux startup.
2. To check available network interfaces:

   ```
   root@s32g274aevb:~# ifconfig -a
   ```

   there should be some, based on config, *pfeX* interfaces.
3. All network interfaces (pfe0 [,pfe1], pfe2) are configured as DHCP clients. If DHCP is not desired, remove apropriate lines in '/etc/network/interfaces' and configure the IP addresses and bring the interfaces up by executing:

   ```
   root@s32g274aevb:~# ifconfig pfe<0-2> <interface_ip_address>/<mask>
   ```
4. The *ping* utility can be used to test the connection:

   ```
   root@s32g274aevb:~# ping <remote_ip_address>
   ```

## 4.4 The driver configuration - device tree

The driver is configurable by changes in the device-tree pfeng node. Usually, the device tree files are using multi-level organization, for example the *fsl-s32g274a-evb.dtb* is built by integration:

1. Family DT file *fsl-s32-gen1.dtsi*
2. SoC DT file *fsl-s32g274a.dtsi*
3. Board specific DT file *fsl-s32g274a-evb.dts*

Device tree bindings for fsl-pfeng:

```
*
* Copyright 2020-2021 NXP
*
* NXP S32G274A PFE networking accelerator (pfeng)
*

Required properties:
```

```
- compatible : Should be "fsl,s32g274a-pfeng"
- reg : Address and length of the register set for the device
- resets: Specify PFE partition reset
- interrupts : Should contain all pfeng interrupts but hifs: bmu,upegpt,
   safety
- clocks : Should contain at least: pfe_sys, pfe_pe
- memory-region : Physical address space for PFE buffers
                  The following memory regions are required in this exact
                     order:
                  1) reserved region for the BMU2 buffer pool, must be in
                     the range 0x00020000 - 0xbfffffff;
                  "fsl,pfe-bmu2-pool" compatible;
                  2) reserved region for non-cacheable DMA buffers,
                  "shared-dma-pool" compatible;
                  3) reserved region for buffer descriptor rings,
                  "fsl,pfe-bdr-pool" compatible;

Optional properties:
- dma-coherent : Declare driver DMA cohereenty (S32G2xx cut 2.0+ only)
- fsl,fw-class-name : PFE CLASS firmware filename
- fsl,fw-util-name : PFE UTIL firmware filename
- fsl,pfeng-master-hif-channel : [slave only] The number of master's HIF
   channel (0-3)
- phys : Serdes phys for sgmii interfaces

Required subnode:
- hif : specifies the PFE HIF channel
- emac : [master only] specifies the PFE EMAC interface
- ethernet : specifies the logical network interface

Required properties for 'hif' subnode:
- compatible : Should be "fsl,pfeng-hif"
- reg : Small number, indexing the HIF channel
- interrupts : Should contain HIF channel interrupt
- fsl,pfeng-hif-mode : channel mode PFENG_HIF_MODE_EXCLUSIVE/
   PFENG_HIF_MODE_SHARED
- fsl,pfeng-ihc : [master/slave only] Annotate for IHC traffic

Required properties for 'emac' subnode:
- compatible : Should be "fsl,pfeng-emac"
- reg : Small number, indexing the EMAC
- phy-mode : See ethernet.txt file in the same directory

Optional properties for 'emac' subnode:
- clocks : array of TX clocks

Optional subnode for 'emac':
- mdio : specifies the mdio bus, used as a container for phy nodes
  according to phy.txt in the same directory

Requires properties for 'mdio' subnode:
- compatible = Should be "fsl,pfeng-mdio"

Requires properties for 'ethernet' subnode:
- compatible : Should be "fsl,pfeng-logif"
- reg : Small number, indexing the network interfaces
- fsl,pfeng-if-name : Logical interface name visible in the Linux
- fsl,pfeng-logif-mode : PFENG_LOGIF_MODE_TX_INJECT/
```

```
    PFENG_LOGIF_MODE_TX_CLASS/PFENG_LOGIF_MODE_AUX
- fsl,pfeng-hif-channels : array of phandles of 'hif' nodes
- fsl,pfeng-emac-link : [master/standalone only] phandle to coresponding '
    emac' node
- fsl,pfeng-emac-id : [slave only] PFE EMAC id (0..2)

Optional properties for 'ethernet' subnode:
- fsl,pfeng-ihc : [master/slave only] Declares the interface for IHC
    traffic
- fsl,pfeng-emac-router : [slave only] switch linked EMAC to
    IF_OP_FLEX_ROUTER mode
- local-mac-address : MAC address
- phy-handle : phandle to the PHY device connected to this device.
- fixed-link : Assume a fixed link. See fixed-link.txt in the same
    directory.
  Use instead of phy-handle.

Example:

reserved-memory {
        #address-cells = <2>;
        #size-cells = <2>;
        ranges;

        pfe_reserved_bmu2: pfebufs@83000000 {
                compatible = "fsl,pfe-bmu2-pool";
                reg = <0 0x83000000 0 0x200000>;
                no-map;
                status = "okay";
        };

        pfe_reserved: pfebufs@83200000 {
                compatible = "shared-dma-pool";
                reg = <0 0x83200000 0 0x9e0000>;
                no-map;
                status = "okay";
        };

        pfe_reserved_bdr: pfebufs@83bf0000 {
                compatible = "fsl,pfe-bdr-pool";
                reg = <0 0x83be0000 0 0x20000>;
                status = "okay";
        };
};

pfe@46000000 {
        compatible = "fsl,s32g274a-pfeng";
        pfe: pfe@46000000 {
                compatible = "fsl,s32g274a-pfeng";
                reg = <0x0 0x46000000 0x0 0x1000000>,     /* PFE controller
                    */
                        <0x0 0x4007ca00 0x0 0x4>;       /* S32G274a syscon
                             */
                reg-names = "cbus", "gpr";
                #address-cells = <1>;
                #size-cells = <0>;
                interrupt-parent = <&gic>;
                interrupts = <GIC_SPI 194 IRQ_TYPE_EDGE_RISING>,
                              <GIC_SPI 195 IRQ_TYPE_EDGE_RISING>,
                              <GIC_SPI 196 IRQ_TYPE_EDGE_RISING>,
                              <GIC_SPI 197 IRQ_TYPE_EDGE_RISING>;
                interrupt-names = "bmu", "nocpy", "upegpt", "safety";
                dma-coherent;
                resets = <&reset S32GEN1_SCMI_RST_PART2>;
                reset-names = "pfe_part";
                clocks = <&clks S32G_SCMI_CLK_PFE_AXI>,
                         <&clks S32G_SCMI_CLK_PFE_PE>,
                         <&clks S32G_SCMI_CLK_PFE_TS>;
                clock-names = "pfe_sys", "pfe_pe", "pfe_ts";
                memory-region = <&pfe_reserved_bmu2>, <&pfe_reserved>, <&
                    pfe_reserved_bdr>;
                fsl,fw-class-name = "s32g_pfe_class.fw";
                fsl,fw-util-name = "s32g_pfe_util.fw";
                phys = <&serdes1 PHY_TYPE_XPCS 0 0>,
                         <&serdes1 PHY_TYPE_XPCS 1 1>,
                         <&serdes0 PHY_TYPE_XPCS 1 1>;
                phy-names = "emac0_xpcs", "emac1_xpcs", "emac2_xpcs";
                nvmem-cells = <&soc_revision>;
                nvmem-cell-names = "soc_revision";

                /* PFE_HIF_0 */
                pfe_hif0: hif@0 {
                        compatible = "fsl,pfeng-hif";
                        #address-cells = <1>;
                        #size-cells = <0>;
                        status = "disabled";
                        interrupts =  <GIC_SPI 190 IRQ_TYPE_EDGE_RISING>;
                        reg = <0>;
                        fsl,pfeng-hif-mode = <PFENG_HIF_MODE_EXCLUSIVE>;
                        fsl,pfeng-ihc;
                };

                /* PFE_HIF_1 */
                pfe_hif1: hif@1 {
                        compatible = "fsl,pfeng-hif";
                        #address-cells = <1>;
                        #size-cells = <0>;
                        status = "disabled";
                        interrupts =  <GIC_SPI 191 IRQ_TYPE_EDGE_RISING>;
                        reg = <1>;
                        fsl,pfeng-hif-mode = <PFENG_HIF_MODE_SHARED>;
                };

                /* PFE_HIF_2 */
                pfe_hif2: hif@2 {
                        compatible = "fsl,pfeng-hif";
                        #address-cells = <1>;
                        #size-cells = <0>;
                        status = "disabled";
                        interrupts =  <GIC_SPI 192 IRQ_TYPE_EDGE_RISING>;
                        reg = <2>;
                        fsl,pfeng-hif-mode = <PFENG_HIF_MODE_EXCLUSIVE>;
                };

                /* PFE_EMAC_0 */
                pfe_emac0: emac@0 {
```

```
                compatible = "fsl,pfeng-emac";
                #address-cells = <1>;
                #size-cells = <0>;
                status = "okay";
                reg = <0>;
                clocks = <&clks S32G_SCMI_CLK_PFE0_TX_SGMII>,
                         <&clks S32G_SCMI_CLK_PFE0_TX_RGMII>,
                         <&clks S32G_SCMI_CLK_PFE0_TX_RMII>,
                         <&clks S32G_SCMI_CLK_PFE0_TX_MII>,
                         <&clks S32G_SCMI_CLK_PFE0_RX_SGMII>,
                         <&clks S32G_SCMI_CLK_PFE0_RX_RGMII>,
                         <&clks S32G_SCMI_CLK_PFE0_RX_RMII>,
                         <&clks S32G_SCMI_CLK_PFE0_RX_MII>;
                clock-names = "tx_sgmii", "tx_rgmii",
                              "tx_rmii", "tx_mii",
                              "rx_sgmii", "rx_rgmii",
                              "rx_rmii", "rx_mii";
                phy-mode = "sgmii";

                /* mdio */
                pfe_mdio0: mdio@0 {
                        compatible = "fsl,pfeng-mdio";
                        #address-cells = <1>;
                        #size-cells = <0>;
                        reg = <0x0>;
                };
        };

        /* PFE_EMAC_1 */
        pfe_emac1: emac@1 {
                compatible = "fsl,pfeng-emac";
                #address-cells = <1>;
                #size-cells = <0>;
                status = "okay";
                reg = <1>;
                clocks = <&clks S32G_SCMI_CLK_PFE1_TX_SGMII>,
                         <&clks S32G_SCMI_CLK_PFE1_TX_RGMII>,
                         <&clks S32G_SCMI_CLK_PFE1_TX_RMII>,
                         <&clks S32G_SCMI_CLK_PFE1_TX_MII>,
                         <&clks S32G_SCMI_CLK_PFE1_RX_SGMII>,
                         <&clks S32G_SCMI_CLK_PFE1_RX_RGMII>,
                         <&clks S32G_SCMI_CLK_PFE1_RX_RMII>,
                         <&clks S32G_SCMI_CLK_PFE1_RX_MII>;
                clock-names = "tx_sgmii", "tx_rgmii",
                              "tx_rmii", "tx_mii",
                              "rx_sgmii", "rx_rgmii",
                              "rx_rmii", "rx_mii";
                phy-mode = "rgmii-id";

                /* mdio */
                pfe_mdio1: mdio@1 {
                        compatible = "fsl,pfeng-mdio";
                        #address-cells = <1>;
                        #size-cells = <0>;
                        reg = <0x1>;
                };
        };
```

```
/* PFE_EMAC_2 */
pfe_emac2: emac@2 {
        compatible = "fsl,pfeng-emac";
        #address-cells = <1>;
        #size-cells = <0>;
        status = "okay";
        reg = <2>;
        clocks = <&clks S32G_SCMI_CLK_PFE2_TX_SGMII>,
                 <&clks S32G_SCMI_CLK_PFE2_TX_RGMII>,
                 <&clks S32G_SCMI_CLK_PFE2_TX_RMII>,
                 <&clks S32G_SCMI_CLK_PFE2_TX_MII>,
                 <&clks S32G_SCMI_CLK_PFE2_RX_SGMII>,
                 <&clks S32G_SCMI_CLK_PFE2_RX_RGMII>,
                 <&clks S32G_SCMI_CLK_PFE2_RX_RMII>,
                 <&clks S32G_SCMI_CLK_PFE2_RX_MII>;
        clock-names = "tx_sgmii", "tx_rgmii",
                      "tx_rmii", "tx_mii",
                      "rx_sgmii", "rx_rgmii",
                      "rx_rmii", "rx_mii";
        phy-mode = "rgmii-id";

        /* mdio */
        pfe_mdio2: mdio@2 {
                compatible = "fsl,pfeng-mdio";
                #address-cells = <1>;
                #size-cells = <0>;
                reg = <0x2>;
        };
};

/* Logical network interface 'pfe0' */
pfe_logif0: ethernet@0 {
        compatible = "fsl,pfeng-logif";
        #address-cells = <1>;
        #size-cells = <0>;
        status = "disabled";
        local-mac-address = [ 00 04 9F BE EF 00 ];
        fsl,pfeng-if-name = "pfe0";
        fsl,pfeng-logif-mode = <PFENG_LOGIF_MODE_TX_INJECT
            >;
        fsl,pfeng-emac-link = <&pfe_emac0>;
        fsl,pfeng-hif-channels = <&pfe_hif0>;
};

/* Logical network interface 'pfe1' */
pfe_logif1: ethernet@1 {
        compatible = "fsl,pfeng-logif";
        #address-cells = <1>;
        #size-cells = <0>;
        status = "disabled";
        local-mac-address = [ 00 04 9F BE EF 01 ];
        fsl,pfeng-if-name = "pfe1";
        fsl,pfeng-logif-mode = <PFENG_LOGIF_MODE_TX_INJECT
            >;
        fsl,pfeng-emac-link = <&pfe_emac1>;
        fsl,pfeng-hif-channels = <&pfe_hif1>;
};
```

```
                /* Logical network interface 'pfe2' */
                pfe_logif2: ethernet@2 {
                        compatible = "fsl,pfeng-logif";
                        #address-cells = <1>;
                        #size-cells = <0>;
                        status = "disabled";
                        local-mac-address = [ 00 04 9F BE EF 02 ];
                        fsl,pfeng-if-name = "pfe2";
                        fsl,pfeng-logif-mode = <PFENG_LOGIF_MODE_TX_INJECT
                           >;
                        fsl,pfeng-emac-link = <&pfe_emac2>;
                        fsl,pfeng-hif-channels = <&pfe_hif2>;
                };

                /* Logical network interface 'aux0' */
                pfe_aux0: ethernet@3 {
                        compatible = "fsl,pfeng-logif";
                        #address-cells = <1>;
                        #size-cells = <0>;
                        status = "okay";
                        local-mac-address = [ 00 04 9F BE EF 80 ];
                        fsl,pfeng-if-name = "aux0";
                        fsl,pfeng-logif-mode = <PFENG_LOGIF_MODE_AUX>;
                        fsl,pfeng-hif-channels = <&pfe_hif1>;
                };

        };
};

/* Example of PFE slave driver entry. Requires SLAVE mode driver */
pfe_slave: pfe_slave@46000000 {
        compatible = "fsl,s32g274a-pfeng-slave";
        reg = <0x0 0x46000000 0x0 0x1000000>;    /* PFE controller */
        #address-cells = <0x01>;
        #size-cells = <0x00>;
        interrupt-parent = <&gic>;
        dma-coherent;
        memory-region = <&pfe_reserved_slave>,
                        <&pfe_reserved_bdr_slave>;
        fsl,pfeng-master-hif-channel = <0>;
        status = "disabled";

        /* PFE_HIF_3 */
        pfesl_hif3: hif@3 {
                compatible = "fsl,pfeng-hif";
                #address-cells = <1>;
                #size-cells = <0>;
                status = "okay";
                interrupts = <GIC_SPI 193 IRQ_TYPE_EDGE_RISING>;
                reg = <3>;
                fsl,pfeng-hif-mode = <PFENG_HIF_MODE_SHARED>;
                fsl,pfeng-ihc;
        };

        /* Logical network interface 'pfe0sl' */
        pfesl_logif0: ethernet@0 {
                compatible = "fsl,pfeng-logif";
                #address-cells = <1>;
                #size-cells = <0>;
                status = "okay";
                local-mac-address = [ 00 04 9F BE FF 10 ];
                fsl,pfeng-if-name = "pfe0sl";
                fsl,pfeng-logif-mode = <PFENG_LOGIF_MODE_TX_INJECT>;
                fsl,pfeng-emac-id = <0>; /* PFE_EMAC_0 */
                fsl,pfeng-emac-router;
                fsl,pfeng-hif-channels = <&pfesl_hif3>;
        };

        /* Logical network interface 'pfe1sl' */
        pfesl_logif1: ethernet@1 {
                compatible = "fsl,pfeng-logif";
                #address-cells = <1>;
                #size-cells = <0>;
                status = "okay";
                local-mac-address = [ 00 04 9F BE FF 11 ];
                fsl,pfeng-if-name = "pfe1sl";
                fsl,pfeng-logif-mode = <PFENG_LOGIF_MODE_TX_INJECT>;
                fsl,pfeng-emac-router;
                fsl,pfeng-emac-id = <1>; /* PFE_EMAC_1 */
                fsl,pfeng-hif-channels = <&pfesl_hif3>;
        };

        /* Logical network interface 'aux0sl' */
        pfesl_aux0: ethernet@11 {
                compatible = "fsl,pfeng-logif";
                #address-cells = <1>;
                #size-cells = <0>;
                status = "okay";
                local-mac-address = [ 00 04 9F BE FF 80 ];
                fsl,pfeng-if-name = "aux0sl";
                fsl,pfeng-logif-mode = <PFENG_LOGIF_MODE_AUX>;
                fsl,pfeng-hif-channels = <&pfesl_hif3>;
        };

};
```

## 4.5 Performance consideration

To gain highest possible throughput on host interface, several options exist to enhance performance, loosing other features or resources:

- Double key ring size
  - Better throughput can be achieved by doubling HIF channel ring size. By default there is 256 entries in the ring. It's compile time option which must be changed in source code header file 'pfe_platform/public/pfe_platform_cfg.h', in named constant PFE_- HIF_RING_CFG_LENGTH.
  - Restriction applies: not all interfaces can be enabled or reserved memory region has to be enlarged in device tree.
- Restrict RAM to 1GB (mem=1g)
  - Restrict Linux kernel memory to 1G limit. It minimizes bounce buffering on TX.

  - Disadvantage: waste memory.

# Chapter 5

# Power management

## 5.1 Prerequisite

- Linux kernel with PM support (it's default)
- Arm Trusted Firmware

### 5.1.1 Compile time

- Linux kernel source configured with CONFIG_PM and CONFIG_SUSPEND

### 5.1.2 Run time

- Linux kernel with support for Suspend to RAM (STR) together with kernel reset controller driver for S32G

  Please read more about STR in *'Linux BSP User Manual'*, chapter '*Power Management*'.

- Hardware, capable of doing suspend

☞ *For hardware reasons, the power management features are only supported on S32G-VNP-RDB2 boards*

## 5.2 Suspend to RAM

The driver implements *.suspend()* and *.resume()* callbacks, which are used by system to manage power states.

### 5.2.1 Suspend

During suspend, the driver does ordered stopping of all managed components:

- Network interfaces (pfe0, pfe1 ...)
- PFE EMACs and its clocks (RX/TX)
- PFE HIF channels

- PFE Platform
- PFE clocks

### 5.2.2 Resume

On resume, the steps are way around, but preceding by S32G PFE partition reset.

☞ *The driver don't preserve any configuration set by FCI API.*

In case of additional, post-startup changes, made by FCI API, it is user's reponsibility to restore them after return from suspend.

# Chapter 6

# Master-Slave feature

## 6.1 Prerequisite

The driver can run in multiple instances within the same system to allow sharing the PFE-provided connectivity using dedicated host interfaces. To enable Master-Slave, the driver gets extended to two binaries, ***pfeng.ko*** which additionaly contains Inter-HIF-communication (aka 'IHC') support and which acts as Master and ***pfeng-slave.ko***, which also contains IHC, but has lack of direct PFE controller manageability.

☞ *Master-Slave feature requires additional configuration in device tree*

## 6.2 Master

To build PFE driver with Master functionality, additional compile-time option is required:

```
make <options> PFE_CFG_MULTI_INSTANCE_SUPPORT=1 PFE_CFG_PFE_MASTER=1 ...
```

The Master requires marking one of the configured HIF interface for IHC transport in device tree by setting *fsl,pfeng-ihc* property, for example:

```
        pfe_hif0: hif@0 {
                compatible = "fsl,pfeng-hif";
                #address-cells = <1>;
                #size-cells = <0>;
                status = "okay";
                interrupts = <GIC_SPI 190 IRQ_TYPE_EDGE_RISING>;
                reg = <0>;
                fsl,pfeng-hif-mode = <PFENG_HIF_MODE_EXCLUSIVE>;
                fsl,pfeng-ihc;
        };
```

☞ *Only one network interface can carry 'fsl,pfeng-ihc property'*

## 6.3 Slave

To build PFE driver with Slave functionality, additional comiple-time option is required:

```
make <options> PFE_CFG_MULTI_INSTANCE_SUPPORT=1 PFE_CFG_PFE_MASTER=0 ...
```

The Slave requires two configuration options in device tree:

- setting global configuration option 'fsl,pfeng-master-hif-channel' with Master's HIF channel number (see example in Master section). The option can be overwritten by command line parameter *master_ihc_chnl*.
- marking one of the configured network interface for IHC transport in device tree by setting *fsl,pfeng-ihc* property, similarly like in Master section 6.2.

☞ *Only one network interface can carry 'fsl,pfeng-ihc property'*

## 6.4 Running

When running a master-slave scenario, following points apply:

- Master driver must always be executed first to allow slave drivers to connect to it during their start-up phase. Also, the network interface for IHC communication has to be up.
- Slave driver(s) configure the PFE to ensure that packets can reach the driver. The PFE then distributes traffic to particular driver instances using destination MAC address of received packets following given rules:
  - Packets received via PFE EMAC interfaces with destination MAC address matching some running slave driver instance are delivered to that instance.
  - All remaining traffic is delivered to the master driver instance.
  - Default MAC address-based traffic distribution can be changed using FCI.
  - When master driver is reset, all slaves must be subsequently reset too

# Chapter 7

# Network interface modes: INJECT, CLASS, AUX

The kernel network interface is described in DTS in ethernet sub-entry, '*fsl-pfeng-logif*' compatible.

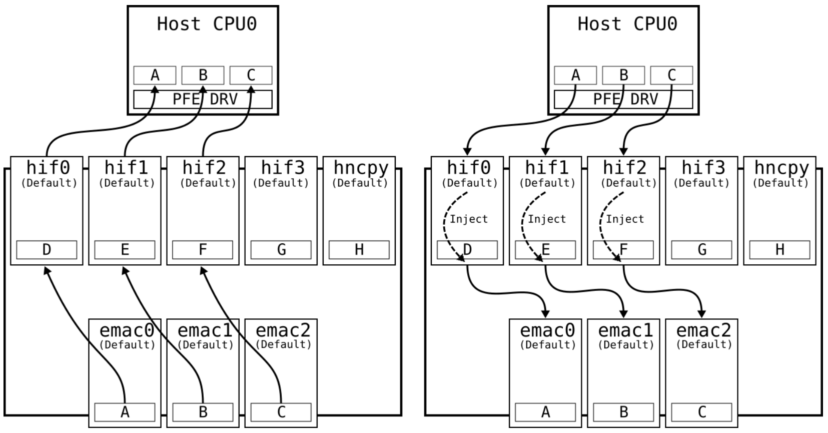
Interface mode is set in property '*fsl,pfeng-logif-mode*' and can be:

- PFENG_LOGIF_MODE_TX_INJECT
- PFENG_LOGIF_MODE_TX_CLASS
- PFENG_LOGIF_MODE_AUX

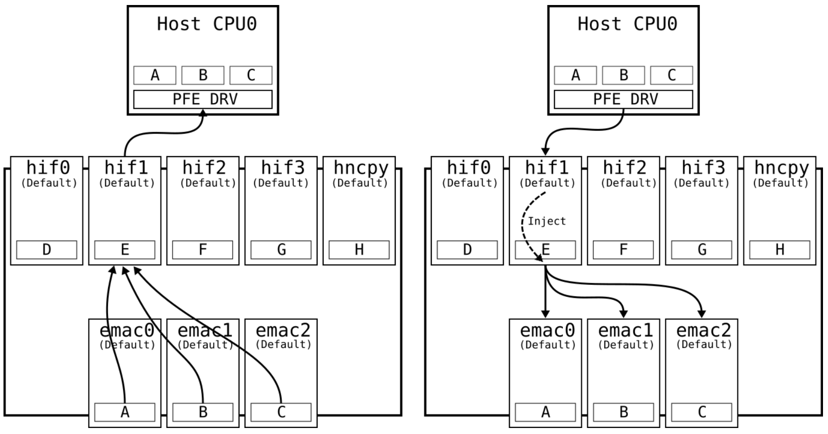
## 7.1 INJECT mode: PFENG_LOGIF_MODE_TX_INJECT

In default Linux network interface configuration, every interface is receiving packets only from EMAC physical interface with which it is configured (property '*f*sl,pfeng-emac-link').

First picture represents multiHIF configuration, when each HIF channel serves exactly one network interface:



Second picture represents sharedHIF configuration, when all network interfaces are using one HIF channel for data passing.



Both pictures are using INJECT mode for TX path. Injecting means possibility to skip PFE classifier and pass traffic directly to external EMAC physical interface.

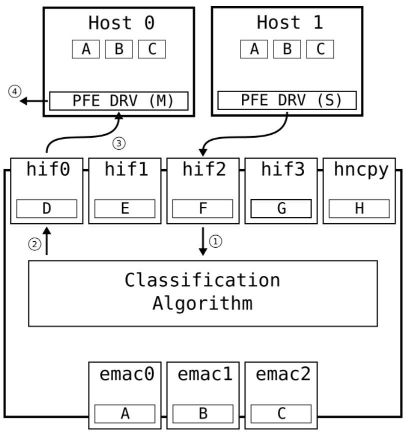
## 7.2 CLASS mode: PFENG_LOGIF_MODE_TX_CLASS

Second mode for TX datapath is passing traffic to the PFE classifier and allow it to process packet according rules using classifier's Flexible Parser and Flexible Filter features. Read more about them in document '*FCI API Reference*'.

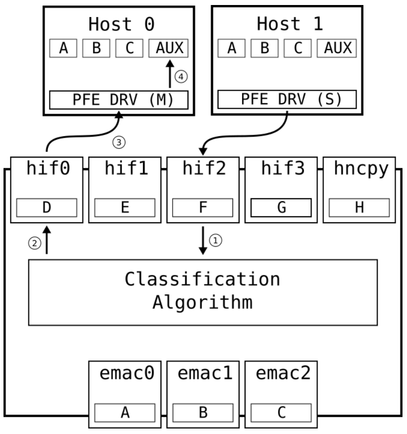
## 7.3 AUX mode: PFENG_LOGIF_MODE_AUX

The AUX interface mode is somehow special. It allows to receive traffic from any physical interfaces, not only EMACs but also HIFs, allowing to build more enhanced configuration, including PFE accelerated L2 bridge and/or L3 router.

When no AUX exists, the traffic from unknown source is discarded:



In case of existing AUX interface, packet is processed there:


Host 0
A
B
C
AUX
④
PFE DRV (M)
Host 1
A
B
C
AUX
PFE DRV (S)
③
hif0
D
hif1
E
hif2
F
hif3
G
hncpy
H
②
①
Classification Algorithm
emac0
A
emac1
B
emac2
C

# Chapter 8

# IEEE1588 Support

The PFE is capable of providing precise, adjustable time reference with ability to timestamp ingress and egress PTP frames. Once enabled the driver ensures that ingress and egress PTP frames are timestamped at MAC level and provides timestamp to the stack.
Supported synchronization modes:

- End to End - SYNC, Follow_Up, Delay_Req, Delay_Resp
  - For UDP supported only on IP: 224.0.1.129, 224.0.1.130, 224.0.1.131, 224.0.1.132
- Peer to Peer - Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up
  - For UDP supported only on IP: 224.0.0.107

Transport layers:

- Ethernet - Supported
- UDP over IPv4 - Supporetd
- UDP over IPV6 - Currently not supported. Support will be added in future.

## 8.1 Clock configuration

S32G platform allows multiple clock configurations depicted in figure 8.1.

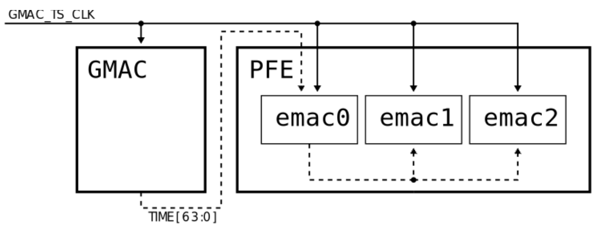


Figure 8.1: S32G2 IEEE1588 clock distribution

### 8.1.1 Internal Timestamp Mode(full line figure 8.1)

In this mode the PFE_EMAC is maintaining system time using internal timer clocked by reference signal and uses the timer value to timestamp frames or report current system time. GMAC_TS_CLK is used as reference. This is the only supported configuration. For this to work you have to add "pfe_ts" clock to the device tree.
Example (from *fsl-s32g274a.dtsi*):

```
clocks = <&clks S32G_SCMI_CLK_PFE_AXI>,
     <&clks S32G_SCMI_CLK_PFE_PE>,
     <&clks S32G_SCMI_CLK_PFE_TS>;
clock-names = "pfe_sys", "pfe_pe", "pfe_ts";
```

☞ *In case the "pfe_ts" clock is not provided, the timestamping feature is disabled*

In Internal Timestamp Mode the ‘GMAC_TS_CLK’ (RM clock name) resp. ’S32G_SCMI_-CLK_PFE_TS’ (SCMI clock name) should be configured with respect to recommended frequency:

| **Mode** | **Minimum Frequency** |
|---|---|
| 10 Mbps Full Duplex | 5 MHz |
| 10 Mbps Half Duplex | 5 MHz |
| 100 Mbps Full Duplex | 5 MHz |
| 100 Mbps Half Duplex | 5 MHz |
| 1 Gbps Full Duplex | 5 MHz |
| 1 Gbps Half Duplex | 18.75 MHz |
| 2.5 Gbps Full Duplex | 11.72 MHz |
| 2.5 Gbps Half Duplex | 46.875 MHz |

### 8.1.2 External Timestamp Mode(dashed line figure 8.1)

The internal PFE_EMAC timer is disabled and PFE_EMAC is using time information delivered from external source. This source can be PFE_EMAC0 or the GMAC instance as shown in figure with the dashed line. NOTE: This configuration is currently not supported.

## 8.2 Driver interface

Driver implements standard Linux API for HW timestamping configuration.

### 8.2.1 IOCTL

The driver supports standart ioctl for timestamp configuration. This API is typically called by PTP stack.

- SIOCSHWTSTAMP - Set configuration (struct hwtstamp_config)
  - TX configuration: Timestamping on/off
  - RX configuration: Message specific configuration currently not supported, all P2P and E2E packets are always timestamped.

- SIOCGHWTSTAMP - Get configuration (struct hwtstamp_config)

### 8.2.2 PTP hardware clock device

After driver initialization there should be one device /dev/ptpX for each PFE_EMAC. Association of clock and interface is available in ethtool as "PTP Hardware Clock" number.

### 8.2.3 ethtool

Driver implements standard ethtool API for getting timestamp options via ETHTOOL_GET_-TS_INFO.
Example output:

```
root@s32g274aevb:~# ethtool -T pfe0
Time stamping parameters for pfe0:
Capabilities:
        hardware-transmit     (SOF_TIMESTAMPING_TX_HARDWARE)
        software-transmit     (SOF_TIMESTAMPING_TX_SOFTWARE)
        hardware-receive      (SOF_TIMESTAMPING_RX_HARDWARE)
        software-receive      (SOF_TIMESTAMPING_RX_SOFTWARE)
        software-system-clock (SOF_TIMESTAMPING_SOFTWARE)
        hardware-raw-clock    (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 0
Hardware Transmit Timestamp Modes:
        off                   (HWTSTAMP_TX_OFF)
        on                    (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
        none                  (HWTSTAMP_FILTER_NONE)
        some                  (HWTSTAMP_FILTER_SOME)
        ptpv1-l4-event        (HWTSTAMP_FILTER_PTP_V1_L4_EVENT)
        ptpv1-l4-sync         (HWTSTAMP_FILTER_PTP_V1_L4_SYNC)
        ptpv1-l4-delay-req    (HWTSTAMP_FILTER_PTP_V1_L4_DELAY_REQ)
        ptpv2-l4-event        (HWTSTAMP_FILTER_PTP_V2_L4_EVENT)
        ptpv2-l4-sync         (HWTSTAMP_FILTER_PTP_V2_L4_SYNC)
        ptpv2-l4-delay-req    (HWTSTAMP_FILTER_PTP_V2_L4_DELAY_REQ)
        ptpv2-l2-event        (HWTSTAMP_FILTER_PTP_V2_L2_EVENT)
        ptpv2-l2-sync         (HWTSTAMP_FILTER_PTP_V2_L2_SYNC)
        ptpv2-l2-delay-req    (HWTSTAMP_FILTER_PTP_V2_L2_DELAY_REQ)
        ptpv2-event           (HWTSTAMP_FILTER_PTP_V2_EVENT)
        ptpv2-sync            (HWTSTAMP_FILTER_PTP_V2_SYNC)
        ptpv2-delay-req       (HWTSTAMP_FILTER_PTP_V2_DELAY_REQ)
```

## 8.3 BSP yocto support

Hint: add package **'linuxptp'** to your Yocto configuration file. With this configuration you get acces to Linux PTP stack **ptp4l** and clock synchronization demon "phc2sys" for advanced configurations.

```
IMAGE_INSTALL_append = " linuxptp"
```

# Chapter 9

# LibFCI

## 9.1 The libfci

Additional features provided by the PFE Ethernet accelerator can be managed using FCI API. The driver release package contains library sources which can be used to control PFE from custom user's application. Details about usage of the library can be found in *'FCI API Reference'* document.

## 9.2 Building libfci library

The simplest way, how to get in touch with libfci framework, is using the Auto Linux BSP facitilies. The BSP contains libfci recipe in **`meta-alb/recipes-extended`** subdirectory, which is intendent for quick jump-in to the FCI development. By default, the libfci is not enabled, so the first step is to create libfci.a library:

```
user@dev:~/fsl-auto-yocto-bsp/build_s32g274aevb$ bitbake -c compile libfci
```

## 9.3 Entering Yocto development environment

Yocto offers extra task called devshell. The task will deposit all the libfci source code into a directory, apply all patches included in the recipe, and then open a terminal in that directory:

```
user@dev:~/fsl-auto-yocto-bsp/build_s32g274aevb$ bitbake -c devshell libfci
```

When invoked, all environmental variables are set up exactly like for compilation. Use predefined ***$CC***, ***$CXX***, ***$PATH***. The libfci.a library, which was compiled in previous step, is located in `sw/xfci/libfci/build/release` directory, what can checked by *find* command, inside devshell terminal:

```
root@dev:~/fsl-auto-.../libfci/0.9.4-r0/git# find . -name libfci.a
./sw/xfci/libfci/build/release/libfci.a
```

The same way it can be checked for required header files:

```
root@dev:~/fsl-auto-.../libfci/0.9.4-r0/git# find . -name libfci.h
./sw/xfci/libfci/public/libfci.h
```

The above information is enough to have all necessary bits for libfci-enabled application development.

## 9.4 Example tool lifci_cli

The driver release package also provides example command line tool libfci_cli, to demonstrate ability of libFCI to manage accelerated features in PFE controller.

Users of Auto Linux BSP root filesystem can find the libfci_cli prebuilt there.

Because the tool is served as example, it can be updated/refactor or whatever any time, so consult current features in sources and/or by command line help.

# Chapter 10

# Reserved Memory Regions

The PFE accesses various data structures stored in the host memory. The location of some of these data structures is configurable to allow the integrator to place them in such a way as to either optimize performance (place some data structures in faster memory) and/or tune the application from security/safety perspective (place some data structures into dedicated container with restricted access permissions). The following reserved memory regions are currently defined for the Linux PFE driver:

## 10.1 PFE System Buffers region

This is the memory region where Ethernet frames should be stored while they are being processed by the PFE. The region is defined in the device tree by a dedicated ‘reserved-memory’ sub-node having the ‘**fsl,pfe-bmu2-pool**’ compatibility string. The memory region is exclusively reserved for PFE’s BMU2 hardware block managing the buffers for in-transit Ethernet frames. The physical start address, alignment and size of this region meet the restrictions imposed by the BMU2 hardware block. The allocation of the BMU2 buffer pool is also controlled by the PFE_CFG_SYS_MEM compile time configuration option. To allocate the BMU2 buffer pool inside this exclusive memory region PFE_CFG_SYS_MEM should be set to ‘pfe-bmu2-pool’ (current default). The BMU2 buffer pool can also be allocated inside the DMA shared pool region by setting PFE_CFG_SYS_MEM to ‘pfe_ddr’ but this is not recommended (sub-optimal).

## 10.2 Buffer Descriptor Rings region

This is a cache enabled memory region reserved for performance critical data structures. It is intended mainly for the Rx and Tx buffer descriptor rings, but it is also used for storing pre-allocated Tx packet metadata headers. The memory region is defined in the device tree by a dedicated ‘reserved-memory’ sub-node having the ‘**fsl,pfe-bdr-pool**’ compatibility string. This is a cache coherent memory region that meets the DMA address range restrictions of the PFE hardware. Allocation in this region is also controlled by the PFE_CFG_BDR_MEM compile time configuration option, which should be set to ’pfe-bdr-pool’ (current default). By setting PFE_CFG_BDR_MEM to ’pfe_ddr’, all allocations targeting the PFE_CFG_BDR_MEM memory pool (buffer descriptors and Tx headers) will be made in the non-cacheable DMA shared pool region resulting in performance degradation.

## 10.3 DMA shared pool region

This is the common memory region reserved for the remaining DMA structures used by PFE, most notable the routing table. It is a '**shared-dma-pool**' compatible memory region, managed by the Linux kernel system code, and all DMA-API allocations are performed from here. Linux maps this zone as non-cacheable. To allocate memory pools inside this shared memory region set the corresponding PFE_CFG_*_MEM compile time configuration options to 'pfe_ddr'.