



S32G Linux PFE Driver User's Manual

Contents

1	Revision History	2
2	Introduction	3
3	Building Procedure	4
3.1	Building the driver	4
3.1.1	Building standalone	4
3.1.2	Building within Linux kernel source tree	5
4	Usage	6
4.1	Prerequisites	6
4.1.1	Pre-silicon platform FPGA/x86	6
4.1.2	S32G silicon platform NXP S32G-EVB	6
4.2	Running the driver	6

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]

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 is in *.../linux-pfeng/*. There are two ways to build the driver: standalone and embedded within the Linux kernel sources.

3.1.1 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, usually installed by host package manager as linux-kernel-devel or linux-dev
2. Go to *.../linux-pfeng/*.
3. Make sure that the following environment variables are set and points to the right directories:
 - (a) KERNELDIR
The directory where Linux kernel header files are located. Usually it is somewhere under */usr/src/...*
 - (b) optionally also PLATFORM
The name of the GNU toolchain platform. For example it is "x86_64-linux-gnu" for Ubuntu or "x86_64-redhat-linux" for Redhat/Fedora. In Linux driver Makefile, in the directory *sw/linux-pfeng* there are already included some default values for different hardware targets. One for FPGA/x86 and second for arm64.
4. Run 'make drv-clean' to clean working directories
5. Build the driver:
 - (a) On FPGA/x86 run:


```
make KERNELDIR=<path to kernel> GLOBAL_CFG_IP_VERSION=IP_VERSION_FPGA_5_0_4 all
```
 - (b) on arm64: run:


```
make KERNELDIR=<path to kernel> all
```

 *Note that much straightforward option is to use Auto Linux BSP integration support. Read next section.*

3.1.2 Building within Linux kernel source tree

1. As prerequisite, save the PFE firmware file 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 components are ready. This step is not only for verification but it precompile 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 = "<some directory path where the fw file was saved>"
```
4. Rebuild BSP to get PFE driver + firmware included in the sdcard image

Chapter 4

Usage

4.1 Prerequisites

4.1.1 Pre-silicon platform FPGA/x86

For the usage of PFE in FPGA on PCIe card, following shall be fulfilled:

- An x86 platform with PCIe support and one PCI-e x8 slot
- VIRTEX UltraScale FPGA connected with the x86 via PCIe.
- FPGA up and running with the bitfile loaded.
- The Linux kernel 4.14 or newer running on the target x86 platform.
- The pfeng.ko available on the target (see [Building the driver](#)).
- The firmware binary called *pfe-s32g-class.fw* available on the target and served by the kernel-firmware subsystem

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

4.1.2 S32G silicon platform NXP S32G-EVB

For usage the software on S32G EVB, the following shall be fulfilled:

- Compatible PFE firmware file
- Auto Linux BSP version 22.6 or higher for creation of sdcard bootable image

4.2 Running the driver


1. Once all prerequisites are met, one can start the driver by executing:

```
insmod pfeng.ko
```

Command

```
ifconfig -a
```

can then be used to obtain a list of available interfaces. The three new interfaces should be available.

 The exact names of the interfaces vary. Usually, it depends on the OS underlayer subsystems like udev/systemd

1. Configure the IP addresses and bring the interfaces up by executing:


```
ifconfig pfe<0-2> <interface_ip_address>/<mask>
```

2. The network interfaces usage

- (a) Silicon S32G: The limitation of usage of the ports depends of board configuration. For example NXP S32G-EVB has only pfe0 and pfe1 ports linked with PHY interfaces.
- (b) FPGA/x86: Connect the Ethernet cable or cables to the connector(s) on the HW platform board. The assignment of interfaces to the connectors on the FPGA/x86 platform board is as follows:

Interface	"ETHERNETFMC" Connector
pfe0	PORT0
pfe1	PORT2
pfe2	N/A

Table 4.1: FPGA/x86 Interface Assignment

 In case of FPGA/x86 platform it shall be ensured that the physical connection will **only** be established using 100Mbps/full duplex.

3. The "ping" utility can be used to test the connection:

```
ping <remote_ip_address>
```