User Guide

For

QDMA Ethernet Platform

(Linux Kernel Driver)

Project Number: 2000-0158

Author: Pankaj Kumar

CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

1.1 Document Overview

This User Guide provides the setup procedure and software usage instructions of the Linux Kernel driver for Network PF of QDMA Ethernet Platform.

1.2 Document Reference

Document References	Version
[1] QDMA Subsystem for PCI Express (PG302)	3.0
[2] Integrated 100G Ethernet Subsystem (PG203)	2.6
[3] QEP control application https://github.com/Xilinx/qep-drivers/, subdirectory qep-ctl	

Table 1-1: Document Reference

1.3 Glossary

Acronym / Term	Description
BAR	Base Address Register
BDF	Bus, Device, Function of a PCIe device
C2H	Card to Host
CMAC	100G Media Access Control (Ethernet Layer 2)
DMA	Direct Memory Access
H2C	Host to Card
IP	Intellectual Property
MAC	Media Access Control
PF	Physical Function
QDMA	Multi Queue Direct Memory Access
QEP	QDMA Ethernet Platform
STM	Streaming Traffic Manager
STM-N	Streaming Traffic Manager for Networking

Table 1-2: Glossary

2 System Overview

This design adds networking support to QDMA based SDAccel platform. The Ethernet Subsystem is added to the static region of the shell. The platform has three physical functions, two physical functions for device management (PF0) and compute acceleration (PF1), and one physical function (PF2) for Network acceleration. The Ethernet subsystem is accessible to the host via PF2.

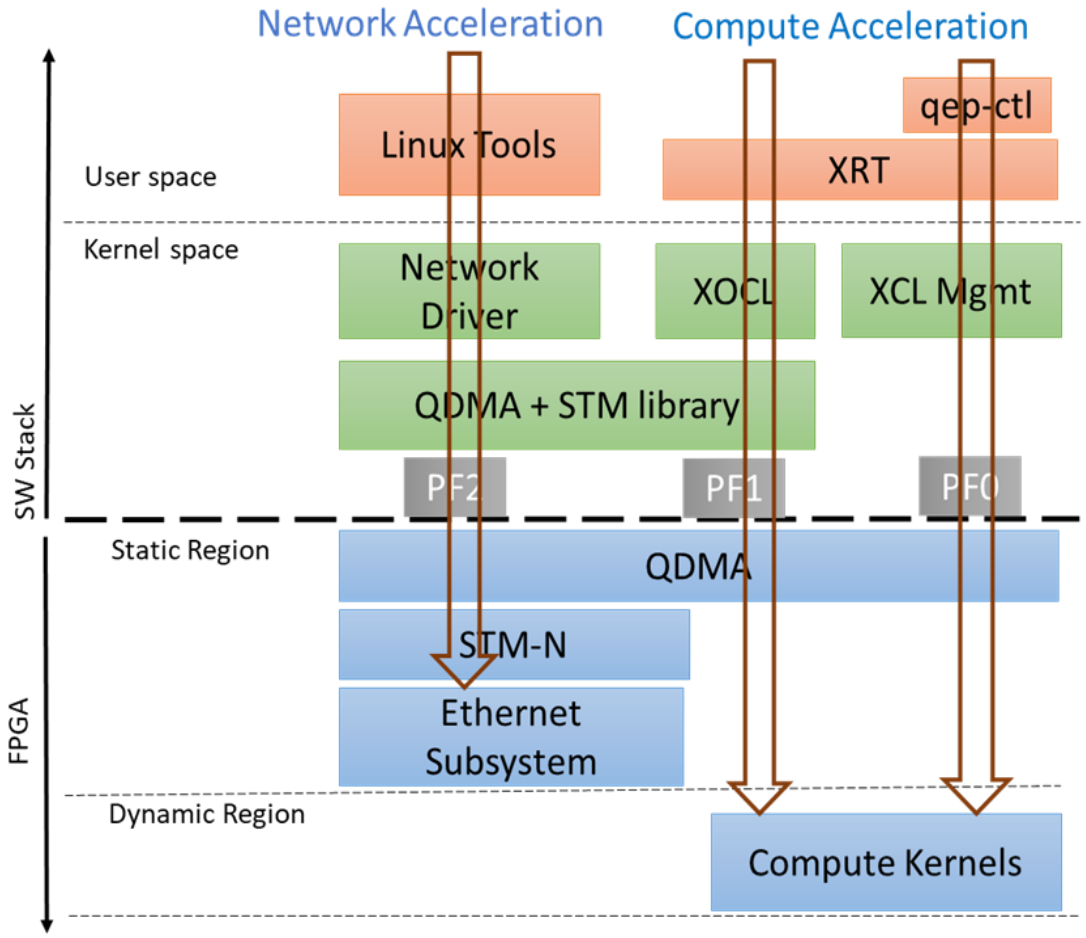

Figure 1: Software Architecture

3 Linux Kernel Driver

QEP Linux Kernel Driver configures the underlying PCI device, initializes the QDMA, STM-N and CMAC, setups the queues, and registers the device as network device to the Linux kernel. The Linux Kernel Driver supports

- Standard Network Device Operations
- Ethtool Operations
- Debugfs

The following sections provide a brief set of instructions to build and deploy the Linux Kernel Driver and run a few tests to validate.

Note: **All the instructions must be executed with root privilege.**

3.1 System Setup

This release was tested with the following system configuration.

Host System Configuration	Description
Operating System	Ubuntu 18.04.1 LTS
Linux Kernel	4.15.0-64-generic
RAM	32GB on local NUMA node

Table 3-1: System Configuration

3.2 Environment

For QEP Linux Kernel driver following system requirements are to be met:

1. Host System with at least one Gen3 x16 PCIe slot and minimum 16GB RAM on same CPU node.
2. U250 Board
3. USB Digilent cables to connect the U250 board to the Host System.
4. Xilinx 2019.1 Vivado tools for programming the FPGA.
5. SDAccel Platform DSA with networking subsystem

Program the U250 board with QEP platform. Please refer to https://www.xilinx.com/products/boards-and-kits/alveo/u250.html#gettingStarted for bringing up U250 boards. Once the device is programmed with QEP platform, execute the following command on the server:

lspci -d 10ee:

```
-Following is sample output

04:00.0 Processing accelerators: Xilinx Corporation Device 7000
04:00.1 Processing accelerators: Xilinx Corporation Device 7001
04:00.2 Ethernet controller: Xilinx Corporation Device 7002
```

Figure 2: Lspci Output

Note: There should be three PFs shown and the PCIe device ID should be shown as 7002 for the device enumerated as Ethernet controller.

3.3 Build Kernel Module

Below Table describes the QEP Linux driver database structure and its contents on the Xilinx Github https://github.com/Xilinx/qep-drivers, subdirectory linux-kernel.

Directory	Description
qep_drivers/linux-kernel/driver	Linux Kernel Driver Source
qep_drivers/linux-kernel/ug02_2000_0158.pdf	This document (User guide)
qep_drivers/linux-kernel/RELEASE.txt	Release Notes

Table 3-2: Linux Kernel Software Database Content

Steps to build the driver

```
# sudo apt-get install linux-headers-$(uname -r) build-essential
# git clone https://github.com/Xilinx/qep-drivers.git
# cd qep_drivers/linux-kernel/driver
# make clean
# make
```

This will generate qep_drv.ko (Kernel Module)

3.4 Device MAC Address Configuration

Configure the Ethernet MAC address of the device on Management PF using qep-ctl application [3].

```
#./qep-ctl config  -d 81:00.0 -m 00:5D:03:00:00:02
In this example 81:00.0 is the BDF of the management PF.
```

This command just sets the MAC address of the network device but doesn't enable MAC and VLAN filtering. Refer to Readme.txt in qep-ctl application for usage details.

3.5 Inserting Kernel Module

Execute the following command to load kernel module:

```
#sudo insmod qep_drv.ko
-Optional module parameters:
      loopback_en: Enable loopback mode at CMAC IP
                   0: Disable (Default)
                   1: Enable
```

Upon successful insertion of the driver, a new network interface is created, that can be used for sending and receiving traffic.

To list all network interfaces, use the following command:

```
# ifconfig -a
```

Below is sample output for 100G network interface

```
enp4s0    Link encap:Ethernet  HWaddr 00:00:00:00:00:00
          inet6 addr: fe80::200:ff:fe00:0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:9600  Metric:1
          RX packets:17625026 errors:0 dropped:9738388 overruns:0 frame:0
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8817175776 (8.8 GB)  TX bytes:2462 (2.4 KB)
```

3.6 Unloading Kernel Module

Execute the following command to unload kernel module:

```
# sudo rmmod qep_drv
```

3.7 Driver Validation

3.7.1 Data Path Test in Loopback

This is a simple test by putting CMAC IP in loopback mode. In loopback mode, CMAC core puts the data coming on TX path onto its RX path. So, the packets sent on TX interface would be received on RX interface. The CMAC core can be put in loopback mode by module parameter loopback_en.

Execute the below commands for this test.

```
# sudo apt install hping3
# ifconfig -s -a
```

sample output:

Iface	MTU	RX-OK	RX-ERR	RX-DRP	RX-OVR	TX-OK	TX-ERR	TX-DRP	TX-OVR	Flg
enp0s25	1500	1012365	0	0	0	140785	0	0	0	BMRU
lo	65536	11492	0	0	0	11492	0	0	0	LRU

Figure 3: Ifconfig Output1

```
# insmod qep_drv.ko loopback_en=1
# ifconfig -s -a
```

sample output:

Iface	MTU	RX-OK	RX-ERR	RX-DRP	RX-OVR	TX-OK	TX-ERR	TX-DRP	TX-OVR	Flg
enp0s25	1500	1013255	0	0	0	140876	0	0	0	BMRU
ens4f2	1500	6	0	0	0	6	0	0	0	BMPRU
lo	65536	11500	0	0	0	11500	0	0	0	LRU

Figure 4: Ifconfig Output2

```
# ifconfig ens4f2 -arp
# ifconfig ens4f2 192.168.1.10
# hping3 192.168.1.20 -I ens4f2 -2 -c 10 -d 64
```

Note: Verify that difference in number of TX-OK packet and RX-OK packet before and after executing the above hping3 command is same.

3.7.2 *Ping test with peer NIC*

Ping test ensures connectivity with a peer NIC.

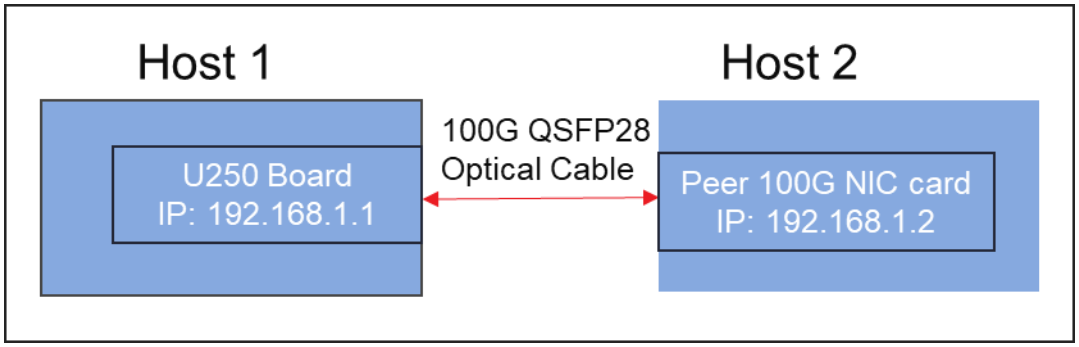

Figure 5: Ping Test System Setup

Following are the steps,

1) Set IP address and netmask on Host1 network interface using following command

```
# ifconfig ens4f2 192.168.1.1 netmask 255.255.255.0
```

2) Set IP address and netmask of Host2 network interface using following command

```
# ifconfig enp175s0 192.168.1.2 netmask 255.255.255.0
```

3) From Host1 execute the following commands to check connectivity

```
# ping -I ens4f2 192.168.1.2
64 bytes from 192.168.1.2: icmp_seq=1 ttl=63 time=0.220 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=63 time=0.274 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=63 time=0.289 ms
```

3.8 *Driver Usage and Configuration*

- To assign an IP address to the network interface

```
ifconfig <interface> <ip address> netmask <subnet mask>
```

- To bring network interface up

```
ifconfig <interface> up
```

- To bring network interface down

```
ifconfig <interface> down
```

- To change MTU (Maximum Transmission Unit) size of network interface

```
ifconfig <interface> mtu <mtu size>
```

- To check driver and device information of network interface

```
ethtool -i <interface>
```

- To print statistics of network interface

```
ethtool -S <interface>
```

- To query ring size values of network interface

```
ethtool -g <interface>
```

- To modify ring size of network interface

```
ethtool -G <interface> rx <Rx Ring Size> tx <Tx Ring Size>
```

- To modify channels (Queues) of network interface

```
ethtool -L <interface> rx <num channel> tx <num channel>
```

- To change the coalescing settings of the specified network device

```
ethtool -C <interface> rx-usecs <N> rx-frames <N>
```

- To dump all Tx/Rx packets of network interface

```
tcpdump -XX -i <interface>
```

- To program the MAC Address and VLAN ID

```
./qep-ctl config -m 01:22:33:44:55:66  -v 4
```

- To enable VLAN and MAC update in TX

```
./qep-ctl config -c TX -e mac,vlan
```

- To enable VLAN and MAC based packet filtering in RX

```
./qep-ctl config -c RX -e mac,vlan
```

- To disable security block

```
./qep-ctl config -e disable
```

- To show security block config

```
./qep-ctl show
```

For more options please refer man page of standard Linux IP utilities (e.g. ifconfig, ethtool, tcpdump, etc.).

4 Debugfs Usage

The QEP Linux Kernel Driver provides rich debugfs support. Along with configuration, status, and statistics, debug related information is also provided through the read-only interface. Debug information is provided for CMAC, STM-N and QDMA subsystems. The debugfs files are located at /sys/kernel/debug/qep_dev/ filesystem. For every device a filesystem tree is populated with directory name as its BDF.

```
>sudo tree /sys/kernel/debug/qep_dev/
```

sample output:

```
/sys/kernel/debug/qep_dev/
└── 04:00:2
    ├── cmac
    ├── qdma_info
    ├── qdma_regs
    ├── qep_config
    ├── queues
    │   └── 0
    │       ├── c2h
    │       │   ├── cntxt
    │       │   ├── desc
    │       │   └── info
    │       ├── cmpt
    │       │   ├── cntxt
    │       │   ├── desc
    │       │   └── info
    │       └── h2c
    │           ├── cntxt
    │           ├── desc
    │           └── info
    └── stmn
```

Figure 6: Debugfs Tree Snapshot

4.1 CMAC

This file contains a snapshot of error, status and statistics registers of CMAC IP. For details refer to CMAC user guide.
Sample Command:

```
# cat /sys/kernel/debug/qep_dev/<BDF>/cmac
```

4.2 QDMA Info

This file provides information about configuration mode of QDMA.

Sample Command:

```
# cat /sys/kernel/debug/qep_dev/<BDF>/qdma_info
```

4.3 QDMA Register Dump

This file provides a snapshot of QDMA registers. For details please refer to QDMA IP user guide

Sample Command:

```
# cat /sys/kernel/debug/qep_dev/<BDF>/qdma_regs
```

4.4 QEP Config

This file provides information on QEP driver configuration modes.

Sample Command:

```
# cat /sys/kernel/debug/qep_dev/<BDF>/qep_config
```

4.5 Queue

This directory tree is dynamic and populates following files corresponding to C2H, H2C and CMPT queue up to max number of configured queues.

- cntxt: Software and Hardware Context of queue
- desc: Raw descriptor queue
- info: Queue info

Sample Command:

```
# cat /sys/kernel/debug/qep_dev/<BDF>/queues/0/c2h/info
```

4.6 STM-N

This file prints a snapshot of STM-N error, drops, status and statistics. After every read, the underlying registers get cleared.

Sample Command:

```
# cat /sys/kernel/debug/qep_dev/<BDF>/stmn
```

5 Document Review History

Version	Date	Review Details	State
1	12-Jul-19	Initial draft of QEP Linux Kernel Driver User Guide	Not Released
2	15-Jul-19	Reviewed and approved for Release	Released
5	30-Sep-19	QEP Linux driver User guide for 1.0.0 release. Reviewed and approved for Release by Pankaj D.	Released

Table 5-1: Document Review History