PCOMware_NPU_Driver_User_Guides

Release 2

Picocom

Dec 13, 2022

CONTENTS

CHAPTER

ONE

USER GUIDE FOR PCIE DRIVER IN NPU

This document provides information for developers on how to develop, and port applications on NPU to integrate with Picocom` s PC802 Chip (SoC). Picocom provides two modes of NPU driver:

- user space driver(PC802_UDriver)
- kernel space driver(PC802_KDriver)

The software architecture based on PC802_UDriver is shown in the Fig. 1.1, and the one based on PC802_KDriver is shown in Fig. 1.2.

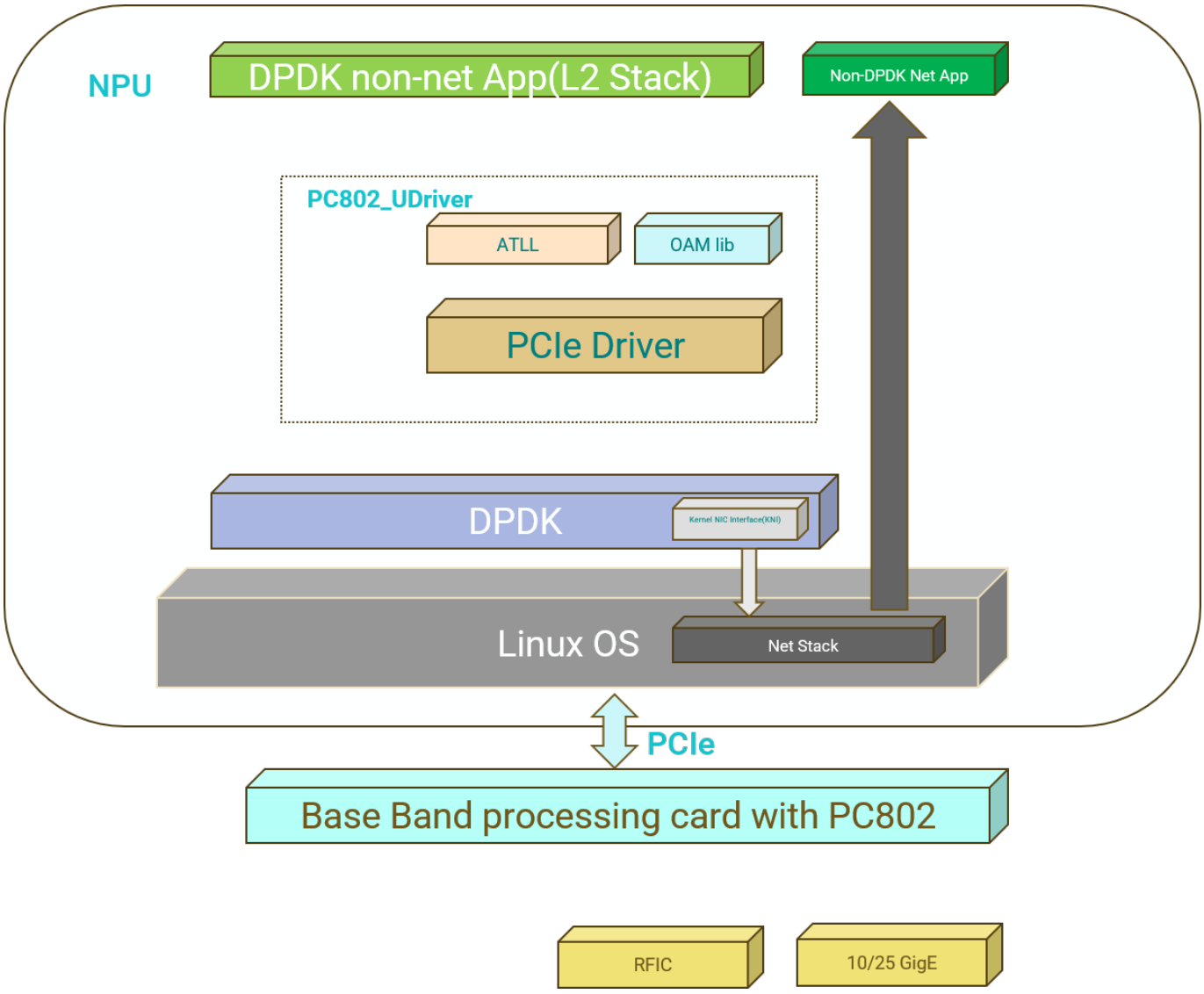

Fig. 1.1: More details shown in *User space Driver*

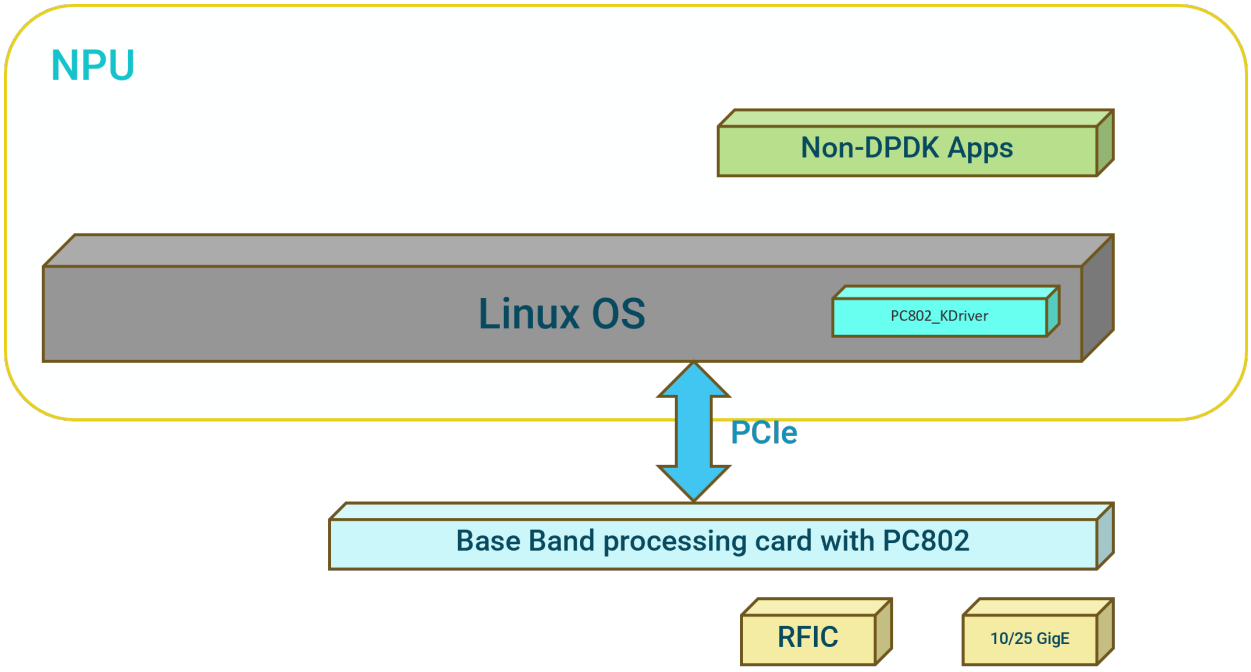

Fig. 1.2: More details shown in *Kernel space driver*

CHAPTER

TWO

INTRODUCTION

2.1 Feature overview

The PC802 driver has the following features:

- 2 * C/U split traffic channels
- 1 * legacy(none C/U split) traffic channel
- Ethernet channel
- OAM channel
- PC802 debugging function

To support the above features, a different queue is designed for each feature. In this way, the application can use different tasks to operate queues, which makes parallel processing possible and convenient:

- There is one pair of UL/DL direction circular ring per queue.
- Different queues have different maximum buffer sizes. Table 2.1 lists the maximum buffer size for single-time data transaction.

Table 2.1: Maximum buffer size definition

SOC Type	Queue Name	Block Size	Description
PC802	Ethernet	2k	None-ecpri Ethernet package
	TRAFFIC DATA 1	256k	CELL 0 FAPI PDSCH/PUSCH data
	TRAFFIC CTRL 1	256k	CELL 0 FAPI Control Message
	LEGACY	16k	Legacy(none C/U split) cell Message
	TRAFFIC DATA 2	256k	CELL 1 FAPI PDSCH/PUSCH data
	TRAFFIC CTRL 2	256k	CELL 1 FAPI Control Message
	OAM	8K	Operation and maintenance Message
PC802R	Ethernet	2k	None-ecpri Ethernet package
	OAM	8K	Operation and maintenance Message

CHAPTER

THREE

USER SPACE DRIVER

PC802_UDriver includes two parts, Abstract Transport Layer Lib(*ATLL interfaces*). and *PC802 PCIe driver*.

- PC802 PCIe driver: A high-performance packet processing driver, leverages the Data Plane Development Kit (DPDK) to take advantage of fast I/O.
- ATLL: Designed as fast communication interface between PHY and MAC layers. Abstract from platform-specific transport mechanisms (SHM, PCIe) and to offer a homogeneous interface to the application layer.

L2/L3 can run as a DPDK application to do the FAPI/OAM data transaction with PC802 in a Linux environment via integrating with PC802_UDriver. Meanwhile, through the KNI (Kernel NIC Interface) kernel network interface card interface, PC802 PCIe driver simulates a virtual network port to provide communication between dpdk applications and linux kernels. The KNI interface allows packets to be received from user mode and forwarded to the linux protocol stack. The chapter describes how to compile and run a application based on PC802_UDriver.

3.1 System Requirements

This section describes the packages required to compile the PC802_UDriver.

3.1.1 BIOS setting prerequisite on x86

For the majority of platforms, no special BIOS settings are needed to use PC802 PCIe driver. However, for power management functionality and high performance of small packets, BIOS setting changes may be needed.

Note: If UEFI secure boot is enabled, the Linux kernel may disallow the use of UIO on the system. Therefore, devices for use by DPDK should be bound to the `igb_uio` or `uio_pci_generic`.

3.1.2 Linux OS

Required:

- Kernel version >= 4.4

  The kernel version required is based on the oldest long-term stable kernel available at kernel.org. Compatibility for recent distribution kernels will be kept, notably RHEL/CentOS 7.

  The kernel version in use can be checked by using the command:

```
uname -r
```

- glibc >= 2.7 (for features related to cpuset)

  The version can be checked using the `ldd --version` command.

3.1.3 DPDK

Required:

X86 platform:

DPDK Version DPDK 21.08.0

LXP LS1046A platform:

DPDK Version in LSDK 21.08

3.2 Compiling the PC802 PCIe driver from source code

If you use igb_uio.ko to bind PC802 to linux kernel:

The source code can be cloned as follows:

```
git clone https://dpdk.org/git/dpdk-kmods
```

build igb_uio

```
cd path/to/git/repo/dpdk-kmods/linux/igb_uio
make
insmod igb_uio.ko
```

3.2.1 Build PC802 PCIe driver libs on NPU side from source code

Contact Picocom to get <PC-002911-DC - Picocom PC802_UDriver code> and enter the root directory of DPDK.

1. X86 Platform:

```
cd ${your_DPDK_PATH}
patch -p1 < ../Picocom-PC802-PCIe-UDriver-based-on-DPDK-21.08.patch
meson build -Denable_multi_pc802=true
#default path is /usr/local
ninja -C build install
```

1. ARM Platform:

```
cd ${your_flexbuild_lsdk2108_PATH}
cd components/apps/networking/dpdk
patch  -p1 < ../Picocom-PC802-PCIe-UDriver-based-on-flexbuild-lsdk2108.patch
meson aarch64-build-gcc --cross-file config/arm/arm64_armv8_linux_gcc
meson configure -Dprefix=~/dpdk_arm_libs -Denable_multi_pc802=true aarch64-build-gcc
#cross compile libs output to "~/dpdk_arm_libs"
ninja -C aarch64-build-gcc install
```

More information on how to compile the DPDK, see DPDK Documentation .

Note: If no source code, please contact Picocom to get PC-002897-DC-A-PC802_UDriver_libs

Note: The new version supports multiple PC802. If you don't need this function, you can compile without the "-Denable_multi_pc802=true" option, and the interface is the same as release 1.

3.2.2 Compiling DPDK application using cmake with link static libraries

Please refer to the CMakeLists.txt to compile the DPDK application through CMake.

Example of CMakeLists.txt:

```
set (PCIE_DRIVER_LIBS
 -L/usr/local/lib/x86_64-linux-gnu
 -Wl,--as-needed
 -Wl,--no-undefined
 -Wl,-O1
 -Wl,--whole-archive
 -Wl,--start-group
 -l:librte_common_cpt.a
 -l:librte_common_dpaax.a
 -l:librte_common_iavf.a
 -l:librte_common_octeontx.a
 -l:librte_common_octeontx2.a
 -l:librte_bus_auxiliary.a
 -l:librte_bus_dpaa.a
 -l:librte_bus_fslmc.a
 -l:librte_bus_ifpga.a
 -l:librte_bus_pci.a
 -l:librte_bus_vdev.a
 -l:librte_bus_vmbus.a
 -l:librte_common_cnxk.a
 -l:librte_common_qat.a
 -l:librte_common_sfc_efx.a
 -l:librte_mempool_bucket.a
 -l:librte_mempool_cnxk.a
 -l:librte_mempool_dpaa.a
 -l:librte_mempool_dpaa2.a
 -l:librte_mempool_octeontx.a
 -l:librte_mempool_octeontx2.a
 -l:librte_mempool_ring.a
 -l:librte_mempool_stack.a
 -l:librte_net_af_packet.a
 -l:librte_net_ark.a
 -l:librte_net_atlantic.a
 -l:librte_net_avp.a
 -l:librte_net_axgbe.a
 -l:librte_net_bnxt.a
 -l:librte_net_bond.a
```

(continues on next page)

(continued from previous page)

```
-l:librte_net_cnxk.a
-l:librte_net_cxgbe.a
-l:librte_net_dpaa.a
-l:librte_net_dpaa2.a
-l:librte_net_e1000.a
-l:librte_net_ena.a
-l:librte_net_enetc.a
-l:librte_net_enic.a
-l:librte_net_failsafe.a
-l:librte_net_fm10k.a
-l:librte_net_hinic.a
-l:librte_net_hns3.a
-l:librte_net_i40e.a
-l:librte_net_iavf.a
-l:librte_net_ice.a
-l:librte_net_igc.a
-l:librte_net_ionic.a
-l:librte_net_ixgbe.a
-l:librte_net_kni.a
-l:librte_net_liquidio.a
-l:librte_net_memif.a
-l:librte_net_netvsc.a
-l:librte_net_nfp.a
-l:librte_net_ngbe.a
-l:librte_net_null.a
-l:librte_net_octeontx.a
-l:librte_net_octeontx2.a
-l:librte_net_octeontx_ep.a
-l:librte_net_pc802.a
-l:librte_net_pfe.a
-l:librte_net_qede.a
-l:librte_net_ring.a
-l:librte_net_sfc.a
-l:librte_net_softnic.a
-l:librte_net_tap.a
-l:librte_net_thunderx.a
-l:librte_net_txgbe.a
-l:librte_net_vdev_netvsc.a
-l:librte_net_vhost.a
-l:librte_net_virtio.a
-l:librte_net_vmxnet3.a
-l:librte_raw_cnxk_bphy.a
-l:librte_raw_dpaa2_cmdif.a
-l:librte_raw_dpaa2_qdma.a
-l:librte_raw_ioat.a
-l:librte_raw_ntb.a
-l:librte_raw_octeontx2_dma.a
-l:librte_raw_octeontx2_ep.a
-l:librte_raw_skeleton.a
-l:librte_crypto_bcmfs.a
-l:librte_crypto_caam_jr.a
-l:librte_crypto_cnxk.a
```

(continues on next page)

(continued from previous page)

```
-l:librte_crypto_dpaa_sec.a
-l:librte_crypto_dpaa2_sec.a
-l:librte_crypto_nitrox.a
-l:librte_crypto_null.a
-l:librte_crypto_octeontx.a
-l:librte_crypto_octeontx2.a
-l:librte_crypto_scheduler.a
-l:librte_crypto_virtio.a
-l:librte_compress_octeontx.a
-l:librte_regex_octeontx2.a
-l:librte_vdpa_ifc.a
-l:librte_event_cnxk.a
-l:librte_event_dlb2.a
-l:librte_event_dpaa.a
-l:librte_event_dpaa2.a
-l:librte_event_dsw.a
-l:librte_event_octeontx2.a
-l:librte_event_opdl.a
-l:librte_event_skeleton.a
-l:librte_event_sw.a
-l:librte_event_octeontx.a
-l:librte_baseband_acc100.a
-l:librte_baseband_fpga_5gnr_fec.a
-l:librte_baseband_fpga_lte_fec.a
-l:librte_baseband_null.a
-l:librte_baseband_turbo_sw.a
-l:librte_node.a
-l:librte_graph.a
-l:librte_bpf.a
-l:librte_flow_classify.a
-l:librte_pipeline.a
-l:librte_table.a
-l:librte_port.a
-l:librte_fib.a
-l:librte_ipsec.a
-l:librte_vhost.a
-l:librte_stack.a
-l:librte_security.a
-l:librte_sched.a
-l:librte_reorder.a
-l:librte_rib.a
-l:librte_regexdev.a
-l:librte_rawdev.a
-l:librte_pdump.a
-l:librte_power.a
-l:librte_member.a
-l:librte_lpm.a
-l:librte_latencystats.a
-l:librte_kni.a
-l:librte_jobstats.a
-l:librte_ip_frag.a
-l:librte_gso.a
```

(continues on next page)

(continued from previous page)

```
-l:librte_gro.a
-l:librte_eventdev.a
-l:librte_efd.a
-l:librte_distributor.a
-l:librte_cryptodev.a
-l:librte_compressdev.a
-l:librte_cfgfile.a
-l:librte_bitratestats.a
-l:librte_bbdev.a
-l:librte_acl.a
-l:librte_timer.a
-l:librte_hash.a
-l:librte_metrics.a
-l:librte_cmdline.a
-l:librte_pci.a
-l:librte_ethdev.a
-l:librte_meter.a
-l:librte_net.a
-l:librte_mbuf.a
-l:librte_mempool.a
-l:librte_rcu.a
-l:librte_ring.a
-l:librte_eal.a
-l:librte_telemetry.a
-l:librte_kvargs.a
-lrte_node
-lrte_graph
-lrte_bpf
-lrte_flow_classify
-lrte_pipeline
-lrte_table
-lrte_port
-lrte_fib
-lrte_ipsec
-lrte_vhost
-lrte_stack
-lrte_security
-lrte_sched
-lrte_reorder
-lrte_rib
-lrte_regexdev
-lrte_rawdev
-lrte_pdump
-lrte_power
-lrte_member
-lrte_lpm
-lrte_latencystats
-lrte_kni
-lrte_jobstats
-lrte_ip_frag
-lrte_gso
-lrte_gro
```

(continues on next page)

(continued from previous page)

```
-lrte_eventdev
-lrte_efd
-lrte_distributor
-lrte_cryptodev
-lrte_compressdev
-lrte_cfgfile
-lrte_bitratestats
-lrte_bbdev
-lrte_acl
-lrte_timer
-lrte_hash
-lrte_metrics
-lrte_cmdline
-lrte_pci
-lrte_ethdev
-lrte_meter
-lrte_net
-lrte_mbuf
-lrte_mempool
-lrte_rcu
-lrte_ring
-lrte_eal
-lrte_telemetry
-lrte_kvargs
-Wl,--no-whole-archive
-Wl,--no-as-needed
-pthread
-lm
-ldl
-lnuma
-Wl,--export-dynamic
-latomic
-Wl,--end-group
-Wl,-rpath,XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
)

execute_process(COMMAND pkg-config --cflags libdpdk
    OUTPUT_VARIABLE PCIE_DRIVER_C_FLAGS
    OUTPUT_STRIP_TRAILING_WHITESPACE)

set(PCIE_DRIVER_C_FLAGS "${PCIE_DRIVER_C_FLAGS} -m64 -pthread -D_GNU_SOURCE")
```

3.2.3 Check if PC802 is active

```
cd ${your_DPDK_PATH}
./usertools/dpdk-devbind.py -s
```

```
Network devices using kernel driver
===================================
...
Other Network devices
=====================
0000:01:00.0 'Device 0802' unused=vfio-pci
No 'Crypto' devices detected
============================
No 'Eventdev' devices detected
============================
No 'Mempool' devices detected
============================
No 'Compress' devices detected
============================
```

Optional driver `01:  00.0` appears:

```
usertools/dpdk-devbind.py -b igb_uio 01:00.0
```

```
Network devices using DPDK-compatible driver
============================================
0000:01:00.0 'Device 0802' drv=igb_uio unused=vfio-pci
Network devices using kernel driver
===================================
...
No 'Crypto' devices detected
============================
No 'Eventdev' devices detected
============================
No 'Mempool' devices detected
============================
No 'Compress' devices detected
============================
```

Output shown in the above figure means that the binding is successful.

3.3 Programmer's Guide

3.3.1 PC802 PCIe driver

Interfaces

uint16_t **pc802_get_count**(void)

Get the total number of pc802 devices that have been successfully initialised.

Returns

The total number of usable pc802 devices.

int **pc802_get_port_id**(uint16_t pc802_index)

get valid pc802 port id.

Parameters

pc802_index – **[in]** PC802 index, value is 0-PC802_INDEX_MAX

Returns

return >=0 is the PC802 device id, or else return error

int **pc802_create_rx_queue**(uint16_t port_id, uint16_t queue_id, uint32_t block_size, uint32_t block_num, uint16_t nb_desc)

Create Rx queue for queue_id >= 1.

Parameters

- **port_id** – **[in]** PC802 chip number, starting from 0
- **queue_id** – **[in]** Queue ID for non-Ethernet traffic, starting from 1
- **block_size** – **[in]** memory block size in bytes (buffer header + message body)
- **block_num** – **[in]** number of memory blocks in the pool of queues
- **nb_desc** – **[in]** number of message descriptors, which should be less than block_num

Returns

returns 0 if opened successfully, otherwise returns error

int **pc802_create_tx_queue**(uint16_t port_id, uint16_t queue_id, uint32_t block_size, uint32_t block_num, uint16_t nb_desc)

Create Tx queue for queue_id >= 1.

Parameters

- **port_id** – **[in]** PC802 chip number, starting from 0
- **queue_id** – **[in]** Queue ID for non-Ethernet traffic, starting from 1
- **block_size** – **[in]** memory block size in bytes (buffer header + message body)
- **block_num** – **[in]** number of memory blocks in the pool of queues
- **nb_desc** – **[in]** number of message descriptors, which should be less than block_num

Returns

returns 0 if opened successfully, otherwise returns error

PC802_Mem_Block_t ***pc802_alloc_tx_mem_block**(uint16_t port_id, uint16_t queue_id)

Allocate one message memory from the current block in use for tx.

Parameters

- **port_id** – **[in]** PC802 chip number, starting from 0
- **queue_id** – **[in]** Queue ID for non-Ethernet traffic, starting from 1

Returns

returns pointer to message body if allocated successfully; returns Null when failure

uint16_t **pc802_rx_mblk_burst**(uint16_t port_id, uint16_t queue_id, PC802_Mem_Block_t **rx_blks, uint16_t nb_blks)

3.3.2 ATLL interfaces

Ctrl channel

int **pcxxCtrlOpen**(const pcxxInfo_s *info, uint16_t dev_index, uint16_t cell_index)

Create control queues for Tx and Rx.

Parameters

- **info** – **[in]** Register Tx and Rx callback functions
- **dev_index** – **[in]** baseband device index
- **cell_index** – **[in]** baseband device cell index

Returns

returns 0 if opened successfully, otherwise returns error

void **pcxxCtrlClose**(uint16_t dev_index, uint16_t cell_index)

Close and free the control shared memory.

Parameters

- **dev_index** – **[in]** baseband device index
- **cell_index** – **[in]** baseband device cell index

Returns

none

int **pcxxCtrlAlloc**(char **buf, uint32_t *availableSize, uint16_t dev_index, uint16_t cell_index)

Allocate one control message memory from the current block in use.

Parameters

- **buf** – **[out]** the allocated memory address
- **availableSize** – **[out]** the current available size in this block
- **dev_index** – **[in]** baseband device index
- **cell_index** – **[in]** baseband device cell index

Returns

returns 0 if opened successfully, otherwise returns error

int **pcxxCtrlSend**(const char *buf, uint32_t bufLen, uint16_t dev_index, uint16_t cell_index)

Update block header when the content of one control message is completed.

Parameters

- **buf** – **[in]** write memory data
- **bufLen** – **[in]** length of data written
- **dev_index** – **[in]** baseband device index
- **cell_index** – **[in]** baseband device cell index

Returns

returns 0 if opened successfully, otherwise returns error

int **pcxxCtrlRecv**(uint16_t dev_index, uint16_t cell_index)

Check the number of received control messages. Application thread may poll this function till it detects messages from Tx side.

Parameters

- **dev_index** – **[in]** baseband device index
- **cell_index** – **[in]** baseband device cell index

Returns

returns 0 if the received messaged are handled successfully, otherwise returns error

Data channel

int **pcxxDataOpen**(const pcxxInfo_s *info, uint16_t dev_index, uint16_t cell_index)

Create data queues for Rx and Tx.

Parameters

- **info** – **[in]** Register Tx and Rx callback functions
- **dev_index** – **[in]** baseband device index
- **cell_index** – **[in]** baseband device cell index

Returns

returns 0 if opened successfully, otherwise returns error

void **pcxxDataClose**(uint16_t dev_index, uint16_t cell_index)

Close and free the data queue.

Parameters

- **dev_index** – **[in]** baseband device index
- **cell_index** – **[in]** baseband device cell index

Returns

none

int **pcxxDataAlloc**(uint32_t bufSize, char **buf, uint32_t *offset, uint16_t dev_index, uint16_t cell_index)

Allocate one data message memory from the current block in use.

Parameters

- **bufSize** – **[in]** the alloc memory size
- **buf** – **[out]** the available memory address
- **offset** – **[out]** the data memory offset from the first address
- **dev_index** – **[in]** baseband device index
- **cell_index** – **[in]** baseband device cell index

Returns

returns 0 if allocated successfully, otherwise returns error

int **pcxxDataSend**(uint32_t offset, uint32_t bufLen, uint16_t dev_index, uint16_t cell_index)

Update data queue context when the content of one data message is completed.

Parameters

- **offset** – **[in]** the offset value from the first address of data queue

- **bufLen** – **[in]** the length of the written data
- **dev_index** – **[in]** baseband device index
- **cell_index** – **[in]** baseband device cell index

Returns
returns 0 if sent successfully, otherwise returns error

void ***pcxxDataRecv**(uint32_t offset, uint32_t len, uint16_t dev_index, uint16_t cell_index)

Receive data from queue by offset.

Parameters

- **offset** – **[in]** offset of the read data queue, data memory offset from the first address
- **len** – **[in]** the length of the read data queue
- **dev_index** – **[in]** baseband device index
- **cell_index** – **[in]** baseband device cell index

Returns
returns pointer to the data if received successfully, otherwise returns NULL

3.3.3 OAM lib interfaces

Interfaces

int **pcxx_oam_init**(void)

Initialize the oam function.

Returns
returns 0 if success, or else return error

int **pcxx_oam_register**(uint32_t msg_type, pcxx_oam_cb_fn cb_fun, void *arg)

Register the callback function received by the oam message.

Parameters

- **msg_type** – **[in]** Oam messages type.
- **cb_fun** – **[in]** User supplied callback function to be called.
- **arg** – **[in]** Pointer to the parameters for the registered callback.

Returns
returns 0 if success, or else return error.

int **pcxx_oam_unregister**(uint32_t msg_type)

Unregister the callback function received by the oam message.

Parameters
msg_type – **[in]** Oam messages type

Returns
returns 0

int **pcxx_oam_sub_msg_register**(uint32_t msg_type, uint16_t sub_msg_id, pcxx_oam_cb_fn cb_fun, void *arg)

Register the callback function received by the oam submessage.

Parameters

- **msg_type** – **[in]** Oam messages type.
- **sub_msg_id** – **[in]** Oam submessage id.
- **cb_fun** – **[in]** User supplied callback function to be called.
- **arg** – **[in]** Pointer to the parameters for the registered callback.

Returns
returns 0 if success, or else return error.

int **pcxx_oam_sub_msg_unregister**(uint32_t msg_type, uint16_t sub_msg_id)
Unregister the callback function received by the oam submessage.

Parameters

- **msg_type** – **[in]** Oam messages type.
- **sub_msg_id** – **[in]** Oam submessage id.

Returns
returns 0

int **pcxx_oam_send_msg**(uint16_t dev_index, uint32_t msg_type, const pcxx_oam_sub_msg_t **sub_msg, uint32_t msg_num)
Send oam messages.

Parameters

- **dev_index** – **[in]** Baseband device index.
- **msg_type** – **[in]** Bam messages type.
- **sub_msg** – **[in]** Send submessage memory pointer.
- **msg_num** – **[in]** Number of submessage.

Returns
returns 0 if send success, or else return error.

3.3.4 How to use the API

This section describes how to use the API provided by the PC802_UDriver.

Initialization process

Before running DPDK app, all the required PCIe devices (including PC802) should be bound to DPDK low layer driver (for example, igb_uio) by using a tool like dpdk-devbind.py. Then the ret_eal_init( ) will probe these devices and allocate their port IDs.

The port IDs are allocated by DPDK EAL. All PCIe devices are bound to DPDK driver and a port ID is allocated for each DPDK virtual device. For all DPDK bound PCIe Ethernet devices (including PC802), their port IDs are in the increasing order of their specific value computed by their PCIe address. The value = (domain << 24) | (bus << 16) | (device << 8) | function. The larger this value is, the larger the port ID is. You can enter the index of the PC802 through the pc802_get_port_id interface to get the port ID.

```
int main(int argc, char** argv)
{
    int diag;
    int pc802_index = 0;
```

(continues on next page)

(continued from previous page)

```
    ...

    diag = rte_eal_init(argc, argv);
    if (diag < 0)
        rte_panic("Cannot init EAL\n");

    pcxx_oam_init();
    for ( pc802_index=0; pc802_index<pcxxGetDevCount(); pc802_index++ )
    {
        port_init(pc802_index);
    }

    ...

    return 0;
}

static int port_init(uint16_t pc802_index){
    struct rte_mempool* mbuf_pool;
    struct rte_eth_dev_info dev_info;
    struct rte_eth_txconf tx_conf;
    int socket_id;
    uint16_t cell;
    int port = pc802_get_port_id(pc802_index);

    rte_eth_dev_info_get(port, &dev_info);
    socket_id = dev_info.device->numa_node;

    mbuf_pool = rte_pktmbuf_pool_create("MBUF_POOL_ETH_TX", 2048,
        128, 0, RTE_MBUF_DEFAULT_BUF_SIZE, socket_id);
    if (mbuf_pool == NULL)
        rte_exit(EXIT_FAILURE, "Cannot create mbuf pool on Line %d\n", __LINE__);
    mpool_pc802_tx = mbuf_pool;

    mbuf_pool = rte_pktmbuf_pool_create("MBUF_POOL_ETH_RX", 2048,
        128, 0, RTE_MBUF_DEFAULT_BUF_SIZE, socket_id);
    if (mbuf_pool == NULL)
        rte_exit(EXIT_FAILURE, "Cannot create mbuf pool on Line %d\n", __LINE__);

    rte_eth_dev_configure(port, 1, 1, &dev_conf);
    tx_conf = dev_info.default_txconf;
    rte_eth_tx_queue_setup(port, 0, 128, socket_id, &tx_conf);
    rte_eth_rx_queue_setup(port, 0, 128, socket_id, NULL, mbuf_pool);

    for (cell = 0; cell < CELL_NUM_PRE_DEV; cell++)
    {
        pcxxDataOpen(&data_cb_info, pc802_index, cell);
        pcxxCtrlOpen(&ctrl_cb_info, pc802_index, cell);
    }

    rte_eth_dev_start(port);
```

(continues on next page)

(continued from previous page)

```
}
```

Create an infinite loop of task to poll the ctrl channels. Because data is an accompanying channel to a ctrl channel, there is no need to create a separate receiving task.

```
while (1) {
    pcxxCtrlRecv(pc802_index, cell_index);
}
```

To enable the OAM function, you need to call pcxx_oam_init for initialization. Then call OAM lib interfaces, Register the callback function to receive OAM messages and send OAM messages.

Note: OAM messages are defined in document PC-003615-DC-1-NR_PHY_OAM_API_Specification, please contact Picocom.

Data exchange process

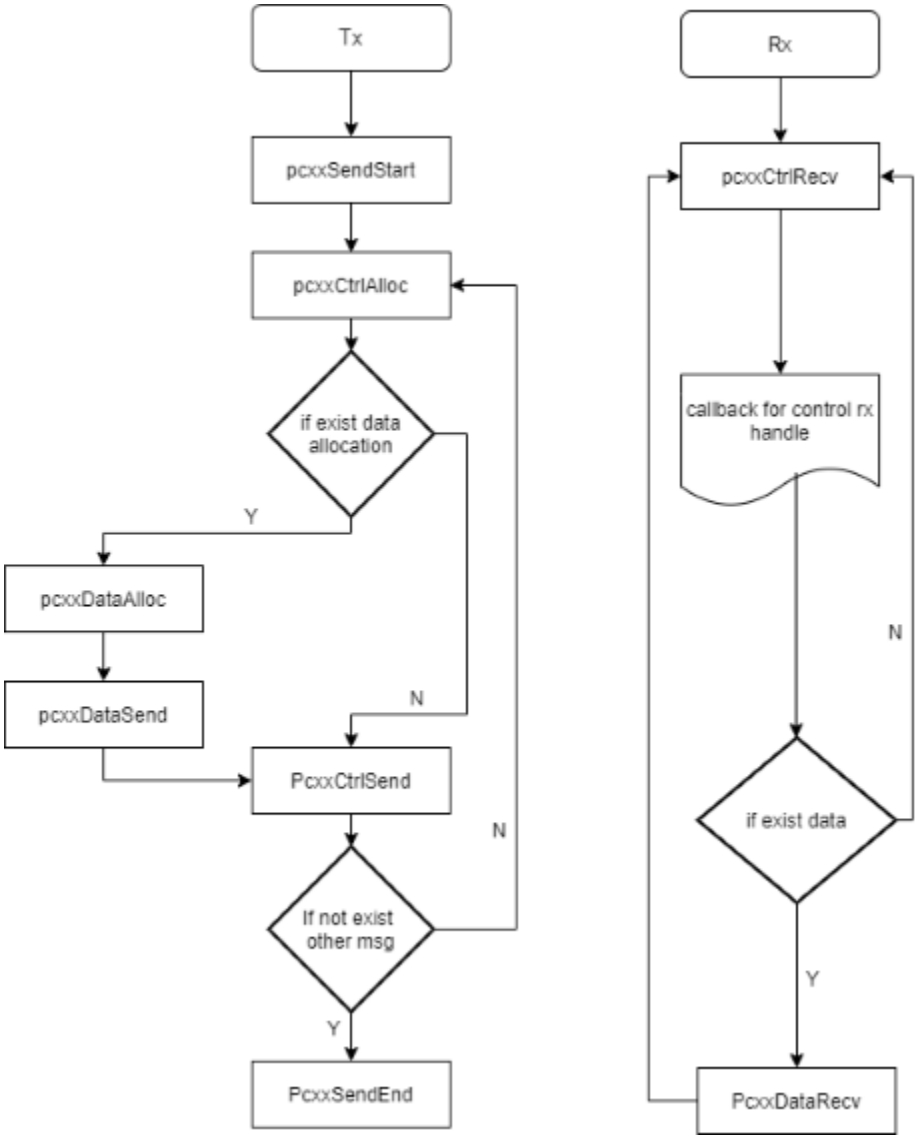

Fig. 3.1: Data exchange process between PHY and MAC

3.4 HowTo Guides

This section describes how to use some features of PC802_UDriver.

The features is integrated in the PC802_UDriver, and is supported by all NPU applications that are compiled with PC802_UDriver, such as dpdk-testpc802 in ${DPDK_PATH}app/test-PC802/.

3.4.1 PC802 log function

PC802_UDriver supports PC802 log output for debugging purposes.

This function default enable when PC802 is initialized, and save in syslog.

example:

```
#Boot up dpdk-testpc802 enable log:
dpdk-testpc802
```

Observe syslog:

```
tail -f /var/log/syslog
Apr 24 04:43:56 localhost dpdk-testpc802[904]: PFI 0 event[00000]: 0x4B3C2D1E(0x12CF0, 11550)
Apr 24 04:43:56 localhost dpdk-testpc802[904]: PFI 0 event[00001]: 0x881F8C97(0x2207E, 3223)
Apr 24 04:43:56 localhost dpdk-testpc802[904]: PFI 0 event[00002]: 0x0000018C(0x00000, 0396)
Apr 24 04:43:56 localhost dpdk-testpc802[904]: PFI 0 event[00003]: 0x00000190(0x00000, 0400)
Apr 24 04:43:56 localhost dpdk-testpc802[904]: PFI 0 event[00004]: 0x1E2D3C4B(0x078B4, 15435)
Apr 24 06:14:59 localhost dpdk-testpc802[857]: PFI 0 event[00000]: 0x4B3C2D1E(0x12CF0, 11550)
Apr 24 06:14:59 localhost dpdk-testpc802[857]: PFI 0 event[00001]: 0x881F8C97(0x2207E, 3223)
Apr 24 06:14:59 localhost dpdk-testpc802[857]: PFI 0 event[00002]: 0x0000018C(0x00000, 0396)
Apr 24 06:14:59 localhost dpdk-testpc802[857]: PFI 0 event[00003]: 0x00000190(0x00000, 0400)
Apr 24 06:14:59 localhost dpdk-testpc802[857]: PFI 0 event[00004]: 0x1E2D3C4B(0x078B4, 15435)
Apr 24 06:15:00 localhost dpdk-testpc802[857]: PFI 0 PRINTF: run_from_ilm : 58 : cid =  0
```

You can disable by adding “–log-level=pc802.printf:1” to the EAL argument:

```
#Boot up dpdk-testpc802 disable log:
dpdk-testpc802 --log-level=pc802.printf:1
```

3.4.2 Eanble KNI function

PC802_UDriver supports PC802 ethernet traffic forward to virtual KNI port.

When this feature is enabled, will create a KNI Linux virtual network interface for PC802.

Packets sent to the KNI Linux interface will be received by the PC802_UDriver, and PC802_UDriver may forward packets to PC802 eCPRI interface, and forward between two.

Using this function requires KNI kernel module be inserted.

You can enable by adding “–vdev=net_kni0” to the EAL argument.

example:

```
#build with kmods
meson -Denable_kmods=true build
ninja -C build install
#insert rte_kni.ko
insmod /lib/modules/$(uname -r)/extra/dpdk/rte_kni.ko carrier=on
#Boot up dpdk-testpc802 forward PC802 ethernet traffic to kni0 virtual port:
dpdk-testpc802 --vdev=net_kni0
```

Observe Linux interfaces:

```
ifconfig kni0
kni0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 2034
        inet 192.168.1.1  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::14d9:d3ff:fe2b:d796  prefixlen 64  scopeid 0x20<link>
        ether 16:d9:d3:2b:d7:96  txqueuelen 1000  (Ethernet)
        RX packets 119  bytes 9938 (9.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 119  bytes 9938 (9.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

3.4.3 Capturing PC802 PCIe queue data

PC802_UDriver supports capture data on PC802 PCIe queues using the dpdk-pdump tool.

dpdk-pdump usage instructions refer to the DPDK documentation.

Some parameter descriptions:

- queue: queue mask, 2-7 bits are valid (queue 0 is ethernet, which can be captured directly through the network port), input * means all queues

example:

```
#Boot up dpdk-testpc802 enable log:
dpdk-testpc802

#in another terminal
#Capturing PC802 PCIe queue data to file:
dpdk-pdump -l 0 -- --pdump 'port=0,queue=*,rx-dev=/tmp/pc802.pcap,tx-dev=/tmp/pc802.pcap,mbuf-
↪size=32768'
#Capturing PC802 PCIe queue data to interface:
dpdk-pdump -l 0 -- --pdump 'port=0,queue=*,rx-dev=lo,tx-dev=lo,mbuf-size=32768'
```

capture data description:

- The capture data is saved as a pcap file;
- The captured queue data is stored in the UDP payload;
- Different queues are distinguished by destination ports, and queues 1-6 correspond to ports 6881-6886 respectively;
- The uplink source port is 8021, and the downlink source port is 8022;
- The option of the ip header contains the ip.opt.time_stamp field, which records the original sending and receiving time stamp of the message, and the unit is us.

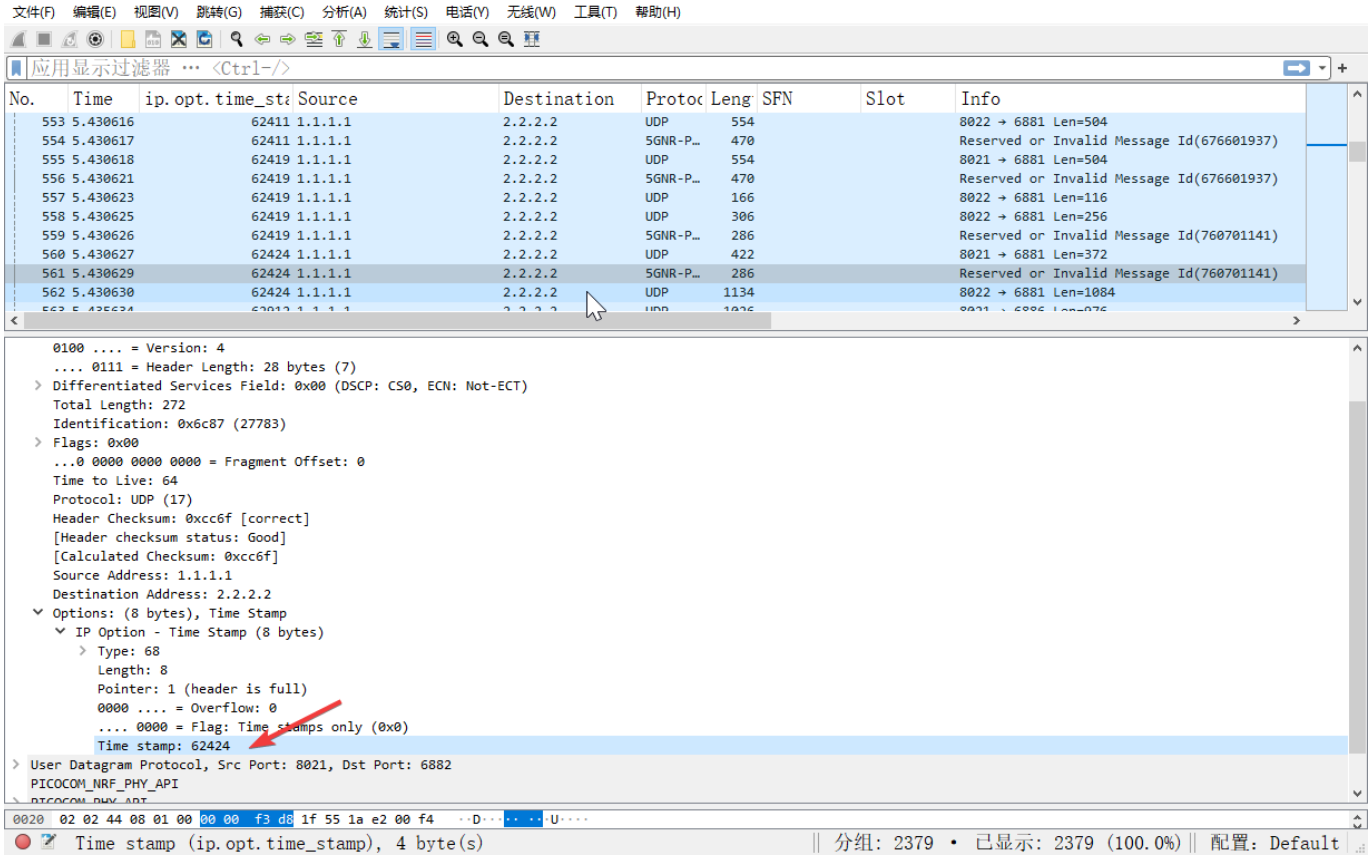

Fig. 3.2: Captured PC802 PCIe queue data

CHAPTER

FOUR

KERNEL SPACE DRIVER

In Kenel driver mode, customers can use the I/O interface provided by the Linux kernel to operate on the PC802. However, due to the addition of data copy from user mode to kernel mode, the performance of this mode is lower than that of userspace mode.

4.1 Build and run kernel driver

1. Run pcitest to test PCI-E basic function:

```
$ make KERNEL_DIR=<your kernel source folder> all
$ sudo insmod pcieptest.ko
$ export PATH=$PATH:.
$ ./pcitest.sh
```

2. Run PC802 traffic test:

```
$ sudo mkdir /lib/firmware/pico
$ cp <path of pc802.img> to /lib/firmware/pico/pc802.img
$ make KERNEL_DIR=<your kernel source folder> all
$ sudo insmod pcsc.ko
$ sudo ./pcsc_test
```

3. Build local loopback mode driver (PCI-E independent):

```
$ sudo mkdir /lib/firmware/pico
$ cp <path of pc802.img> to /lib/firmware/pico/pc802.img
$ make KERNEL_DIR=<your kernel source folder> CONFIG_PCSC_LOOPBACK=y all
$ sudo insmod pcsc_lb.ko
$ sudo ./pcsc_test
```

CHAPTER

FIVE

RELEASE NOTES

5.1 PC802 driver 2.0

5.1.1 New features

- SI-1013: Added support for capturing traffic from PC802 PCIe
- SI-1011: Optimized the cpu and memory consumption of the PC802 log function
- SI-939: Added support for multi-process mode with PC802
- SI-909: Added OAM lib to support DU OM features
- SI-855: Added support for PCIe register to transfer slot indication
- SI-998: Added support for multi-threads mode with PC802
- SI-801: Added support for ORANIC board
- SI-808: Added support for rerun app without rebooting NPU
- SI-806: Added support for enable KNI interface for PC802

5.1.2 Bug fixes

5.1.3 Known issues

5.1.4 API Changes

- The `dev_inde` and `cell_index` input parameters are added to the ATLL interface below to support multiple PC802s, which can be turned on by the MULTI_PC802 macro, and turned off by default:
  - `pcxxCtrlOpen`
  - `pcxxCtrlClose`
  - `pcxxDataOpen`
  - `pcxxDataClose`
  - `pcxxSendStart`
  - `pcxxSendEnd`
  - `pcxxCtrlAlloc`
  - `pcxxCtrlSend`

  - pcxxCtrlRecv
  - pcxxDataAlloc
  - pcxxDataSend

5.2 PC802 driver 2.0.1

5.2.1 New features

- SI-1017: Added support for Different firmware&vector file for different chip in ORANIC
- SI-1337: Optimized PCIe msg jitter

5.2.2 Bug fixes

- SI-1369: Fix PC802 driver(v2.0) PCIe channel failure on FSBL V0

INDEX

P