picocom

PCOMware_NPU_Driver_User_Guides

Release 1

Picocom

Mar 29, 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 Picocoms 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 that based on PC802_KDriver is shown in Fig. 1.2.

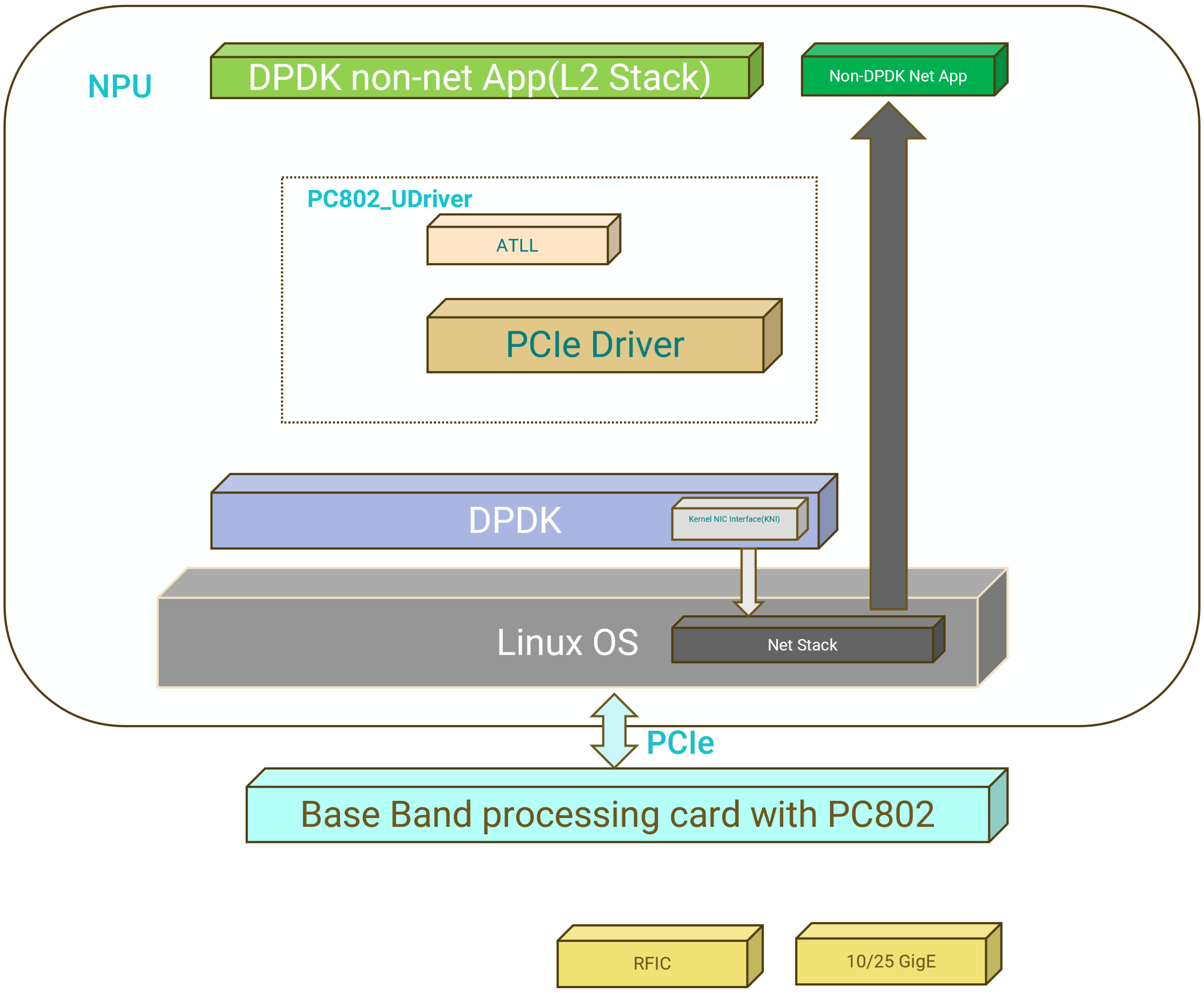
NPU
DPDK non-net App(L2 Stack)
Non-DPDK Net App
PC802_UDriver
ATLL
PCIe Driver
DPDK
Kernel NIC Interface(KNI)
Linux OS
Net Stack
PCIe
Base Band processing card with PC802
RFIC
10/25 GigE

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

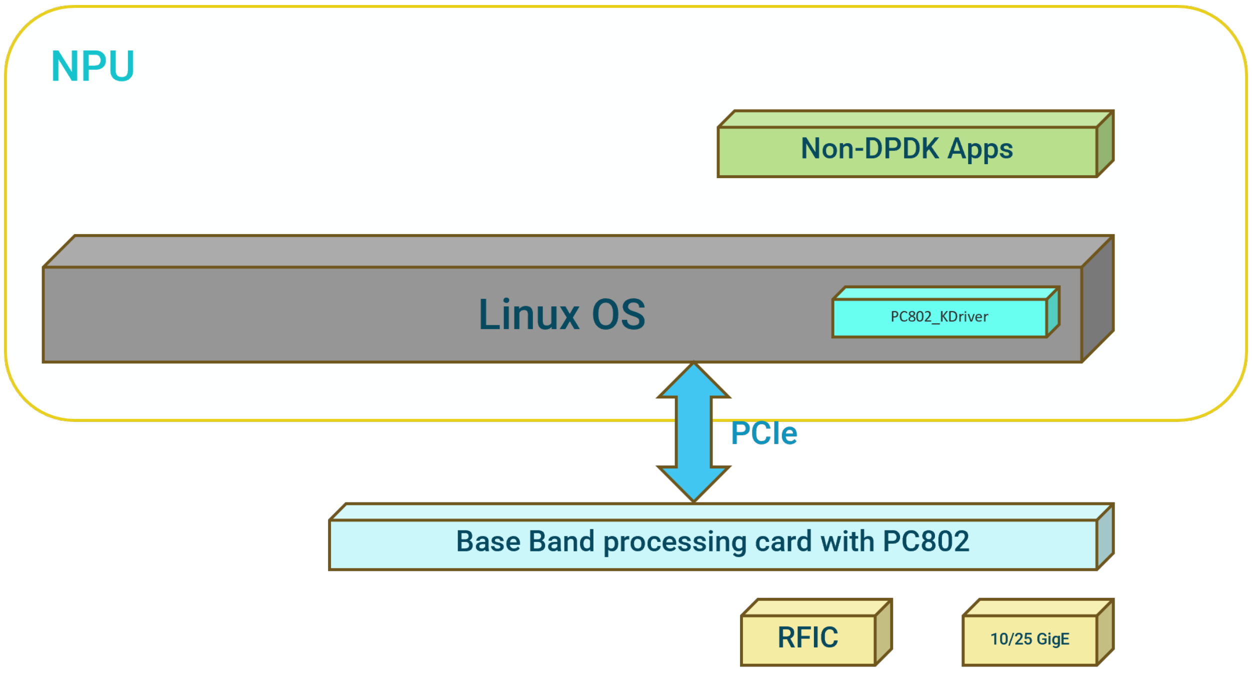

Fig. 1.2: More details shown in *Kernel space driver*.

CHAPTER

TWO

INTRODUCTION

2.1 Feature overview

The PC802 driver has the following features:

- Support 5G EMBB data exchange between PHY and MAC.
- Support 4G LTE data exchange between PHY and MAC.
- Support 5G URLLC data exchange between PHY and MAC.
- Support NIC (Network interface card) function.
- Support OAM (operation and maintenance) channel.

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
	5G EMBB Data	256k	EMBB FAPI PDSCH/PUSCH data
	5G EMBB Control	256k	EMBB FAPI Control Message
	5G URLLC	16k	URLLC FAPI Control/Data Message
	4G LTE Data	64k	LTE FAPI PDSCH/PUSCH data
	4G LTE Control	64k	LTE FAPI Control Message
	OAM	16K	Operation and maintenance Message
PC802R	Ethernet	2k	None-ecpri Ethernet package
	OAM	16K	Operation and maintenance Message

CHAPTER

THREE

USER SPACE DRIVER

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

You need insmod igb_uio.ko to bind PC802 to linux kernel:

Download kmod code from DPDK Kmode repo.

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
ninja -C build
cd build
#default path is /usr/local
ninja install
```

2. 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
ninja -C aarch64-build-gcc
```

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

Note: If you are binary customer, please contact Picocom to get PC-002897-DC-A-PC802_UDriver_libs

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.

CMakeLists.txt example:

```
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
 -l:librte_net_cnxk.a
 -l:librte_net_cxgbe.a
 -l:librte_net_dpaa.a
 -l:librte_net_dpaa2.a
```

(continues on next page)

(continued from previous page)

```
-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
-l:librte_crypto_dpaa_sec.a
-l:librte_crypto_dpaa2_sec.a
-l:librte_crypto_nitrox.a
-l:librte_crypto_null.a
```

(continues on next page)

(continued from previous page)

```
-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
-l:librte_gro.a
-l:librte_eventdev.a
-l:librte_efd.a
-l:librte_distributor.a
```

(continues on next page)

(continued from previous page)

```
-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
-lrte_eventdev
-lrte_efd
-lrte_distributor
-lrte_cryptodev
```

(continues on next page)

(continued from previous page)

```
-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,XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
)

#message(STATUS PCIE_DRIVER_LIBS =${PCIE_DRIVER_LIBS})

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

```
./usertools/dpdk-devbind.py -s
```

```
Network devices using kernel driver
===================================
...
Other Network devices
=====================
0000:af:00.0 'DWC_usb31 abcf' unused=windrvr1440
```

(continues on next page)

(continued from previous page)

```
No 'Crypto' devices detected
============================
No 'Eventdev' devices detected
============================
No 'Mempool' devices detected
============================
No 'Compress' devices detected
============================
```

Optional driver `af: 00.0` appears:

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

```
Network devices using DPDK-compatible driver
============================================
0000:af:00.0 'DWC_usb31 abcf' drv=vfio-pci unused=windrvr1440
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

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, start with 0
- **`queue_id`** – **[in]** Queue Number for non-ethernet traffic, start with 1
- **`block_size`** – **[in]** memory block size in byte (buffer header + message body)
- **`block_num`** – **[in]** number of memory blocks in the pool of the queue
- **`nb_desc`** – **[in]** number of message descriptors, should be less than block_num

Returns returns 0 if open success, or else return 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, start with 0
- **queue_id** – **[in]** Queue Number for non-ethernet traffic, start with 1
- **block_size** – **[in]** memory block size in byte (buffer header + message body)
- **block_num** – **[in]** number of memory blocks in the pool of the queue
- **nb_desc** – **[in]** number of message descriptors, should be less than block_num

Returns returns 0 if open success, or else return error

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

Allocated one message memory from current block in used for tx.

Parameters

- **port_id** – **[in]** PC802 chip number,start with 0
- **queue_id** – **[in]** Queue Number for non-ethernet traffic, start with 1

Returns return pointer to message body, 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)

Create control queues for Tx and Rx.

Parameters **info** – **[in]** Register Tx and Rx callback functions

Returns returns 0 if open success, or else return error

void **pcxxCtrlClose**(void)

Close and free the control Shared memory.

Parameters **none** –

Returns none

int **pcxxCtrlAlloc**(char **buf, uint32_t *availableSize)

Allocated one control message memory from current block in used.

Parameters

- **buf** – **[out]** the allocated memory address
- **availableSize** – **[out]** the current available size in this block

Returns returns 0 if open success, or else return error

int **pcxxCtrlSend**(const char *buf, uint32_t bufLen)

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

Parameters

- `buf` – **[in]** write memory data
- `bufLen` – **[in]** length of data written

Returns returns 0 if open success, or else return error

int **pcxxCtrlRecv**(void)

Checks the number of received control messages. Application thread may poll at function till it detects there is message from Tx side.

Parameters `none` –

Returns returns 0 if handle the received messages success, or else return error

Data channel

int **pcxxDataOpen**(const pcxxInfo_s *info)

Create data queues for Rx and Tx.

Parameters `info` – **[in]** Register Tx and Rx callback functions

Returns returns 0 if open success, or else return error

void **pcxxDataClose**(void)

Close and free the data queue.

Parameters `none` –

Returns none

int **pcxxDataAlloc**(uint32_t bufSize, char **buf, uint32_t *offset)

Allocated one data message memory from current block in used.

Parameters

- `bufSize` – **[in]** the alloc memory size
- `buf` – **[out]** the available memory address
- `offset` – **[out]** the data memory offset from first address

Returns returns 0 if alloc success, or else return error

int **pcxxDataSend**(uint32_t offset, uint32_t bufLen)

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

Parameters

- `offset` – **[in]** the offset value from first address of data queue
- `bufLen` – **[in]** the write data length

Returns returns 0 if send success, or else return error

void ***pcxxDataRecv**(uint32_t offset, uint32_t len)

Recv data from queue by offset.

Parameters

- `offset` – **[in]** offset of read data queue, data memory offset from first address
- `len` – **[in]** read data queue length

Returns return pointer to the data if success, else return NULL

OAM channel

int **`pcxxOamOpen`**(const pcxxInfo_s *info)

Create OAM queues for Tx and Rx.

Parameters **`info`** – **[in]** Register Tx and Rx callback functions

Returns returns 0 if open success, or else return error

void **`pcxxOamClose`**(void)

Close and free the OAM Shared memory.

Parameters **`none`** –

Returns none

int **`pcxxOamAlloc`**(char **buf, uint32_t *availableSize)

Allocated one OAM message memory from current block in used.

Parameters

- **`buf`** – **[out]** the allocated memory address
- **`availableSize`** – **[out]** the current available size in this block

Returns returns 0 if open success, or else return error

int **`pcxxOamSend`**(const char *buf, uint32_t bufLen)

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

Parameters

- **`buf`** – **[in]** write memory data
- **`bufLen`** – **[in]** length of data written

Returns returns 0 if open success, or else return error

int **`pcxxOamRecv`**(void)

Checks the number of received OAM messages. Application thread may poll at function till it detects there is message from Tx side.

Parameters **`none`** –

Returns returns 0 if handle the received messages success, or else return error

3.3.3 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.

```
static int port_init(uint16_t port){
    struct rte_mempool* mbuf_pool;
    struct rte_eth_dev_info dev_info;
    struct rte_eth_txconf tx_conf;
    int socket_id;

    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);

    pcxxDataOpen(&data_cb_info);

    pcxxCtrlOpen(&ctrl_cb_info);

    pcxxOamOpen(&log_cb_info);

    rte_eth_dev_start(port);

}
```

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();
}
```

Create an infinite loop of task to poll the oam channels

```
while (1) {
    pcxxOamRecv();
}
```

Data exchange process

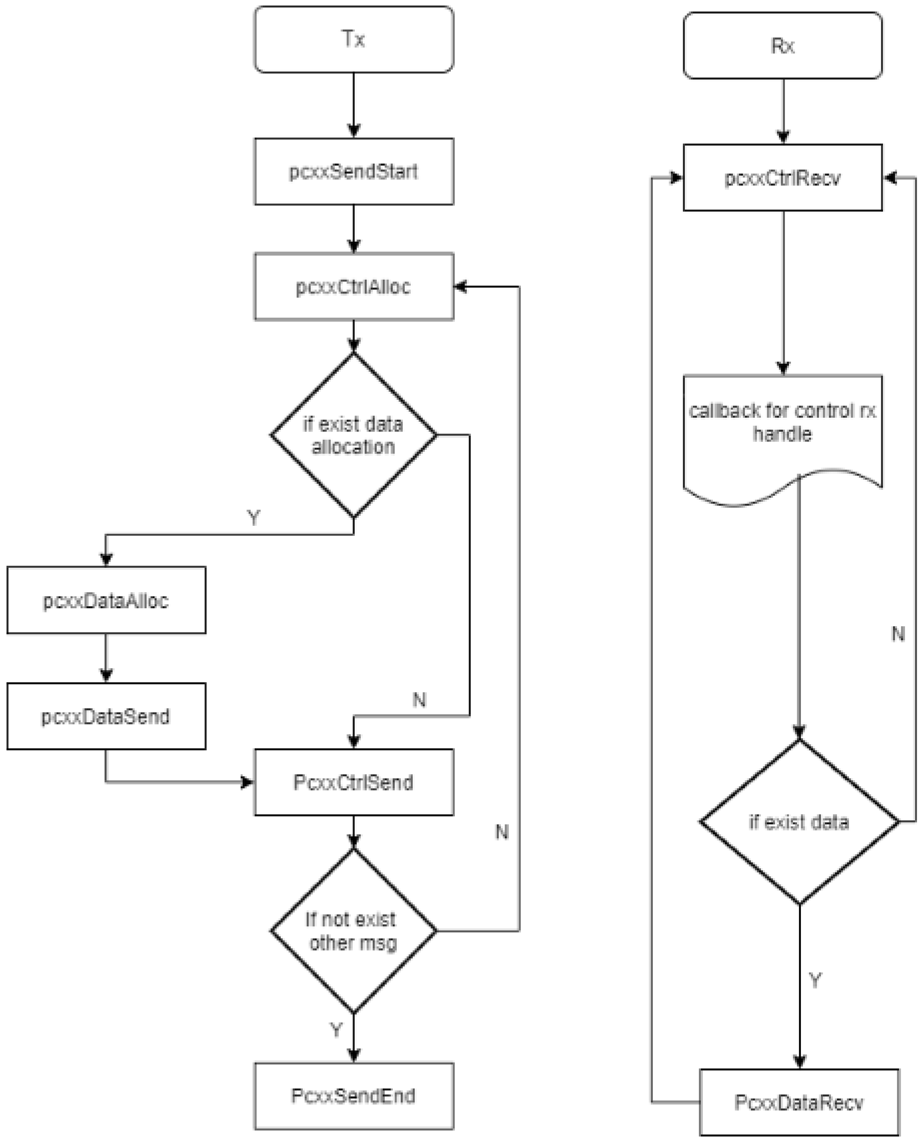

Fig. 3.1: Data exchange process between PHY and MAC

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 Quick Start

The PC802 Small Cell Development Board (PC802SCB) is a flexible 5G NR/LTE development platform for evaluating the PC802 for different small cell use cases. The PC802SCB includes an on-board NXP Network Processing Unit (NPU). For more hardware information on PC802SCB, contact Picocom to get "PC802 Small Cell Development Board Product Brief".

4.1.1 Building PC802SCB image

Download LSDK from NXP Semiconductor website

Extract and cd it to the root folder of the SDK:

```
tar zxvf flexbuild_lsdk2108.tgz
cd flexbuild_lsdk2108
```

Setup the environment variables and import the build script:

```
source setup.env
```

Get the component source code from the remote git repository:

```
flex-builder -i repo-update
flex-builder -i repo-fetch
```

Patch the component source to support Pico SCB:

```
# patch the linux kernel
cd components/linux/linux
git checkout -b LSDK-21.08-LS1046A-PSCB-PR1
patch -p1 < {path_to_patch}/linux.patch
cd -

# patch the atf
cd components/firmware/atf
```

(continues on next page)

(continued from previous page)

```
patch -p1 < {path_to_patch}/atf.patch
cd -

# patch the rcw
cd components/firmware/rcw
patch -p1 < {path_to_patch}/rcw.patch
cd -

# patch the uboot
cd components/firmware/uboot
patch -p1 < {path_to_patch}/uboot.patch
cd -

# patch the lsdk
patch -p1 < {path_to_patch}/flexbuild_lsdk2108.patch
```

If the development environment is not Ubuntu-20.04, you need to start docker ubuntu-20.04:

```
flex-builder docker
```

Build all the images:

```
flex-builder -i mkrfs
flex-builder -i packrfs
flex-builder -i mkboot -m ls1046apscb
flex-builder -i mkfw -m ls1046apscb -b sd
flex-builder -i mkfw -m ls1046apscb -b qspi
```

Build and run PC802_KDriver:

1. Run pcitest to test the PCIe basic function:

   ```
   $ 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 pcieptest.ko
   $ export PATH=$PATH:.
   $ ./pcitest.sh
   ```

2. Run the 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
   ```

4.2 SD card image

4.2.1 Formatting SD card

To partition and format the target SD/eMMC/USB disk and install your custom distro image:

```
$ flex-installer -i pf -d /dev/sdx                  (default 4 partitions as 4P=128M:2G:5G:-1)
or
$ flex-installer -i pf -d /dev/sdx -p 4P=128M:3G:6G:-1   (specify your custom partitions as␣
↪4P=128M:3G:6G:-1)
```

4.2.2 Flashing firmware to SD card

Assume your SD card has a device node at /dev/sdb, then you can flash firmware image to SD card by using the following command line:

```
The device nodes are not exactly same on different machines, it could be sdc, sdd, sde ..., or /
↪dev/mmcblk0::

    - Install composite firmware image:
    $ sudo dd if=firmware_ls1046apscb_sdboot.img of=/dev/sdX bs=512 seek=8 conv=sync
```

4.2.3 Extracting the boot and rootfs images to SD card

Assume your SD card partitions are already mounted to the system. For example, on Ubuntu 18.04, the boot partition /dev/sdb2 and rootfs partition /dev/sdb4 will be mounted into the filesystem folder at /mnt/sdb2, /mnt/sdb4. Then you can extract boot and rootfs images to your SD card by using the following command line:

```
- Install boot partition image:
$ sudo tar -xvf boot_LS_arm64_lts_5.10.tgz -C /media/user/boot/

- Install root partition image:
$ sudo tar -xvf rootfs_lsdk2108_ubuntu_main_arm64.tgz -C /media/user/data4/
```

4.2.4 Downloading and deploying LSDK composite firmware in Windows environment

- Download the DD for Windows tool and install it.
- Run the DD tool for Windows as an administrator.
- Click Choose disk, Choose file, and Restore to program the newly generated composite firmware (with _4k suffix in file name) into the target SD card.
- Unplug your SD card from the Windows host machine and plug it on the target board.
- Set the DIP switch for SD boot or run sd_bootcmd at U-Boot prompt.

4.2.5 Programming QSPI flash

Copy the qspi flash boot image to the mmc card boot partition, and then run the following command in the uboot shell:

```
load mmc 0:2 $load_addr firmware_ls1046apscb_qspiboot.img
sf probe 0:0
sf erase 0 +$filesize && sf write $load_addr 0 $filesize
```

4.3 Configuring the Board

4.3.1 Configuring MAC address for Ethernet in uboot

Press any key in u-boot stage to enter the u-boot shell, and then run the following command line:

```
setenv ethaddr 00:04:9F:03:05:E4
saveenv
```

4.3.2 Configuring static IP address for the board

In the ubuntu shell after the board is booted to Ubuntu, run the following command:

```
ifconfig fm1-mac5 192.168.1.90/24
```

In the u-boot shell, run the following commands:

```
setenv ipaddr 192.168.1.90
setenv netmask 255.255.255.0
saveenv
```

4.3.3 Configuring the board to get IP address from DHCP server

In the ubuntu shell after the board is booted to Ubuntu, run the following command:

```
dhclient -v fm1-mac5
```

In the u-boot shell, run the following command:

```
dhcp
```

INDEX