DPDK

DATA PLANE DEVELOPMENT KIT

Rawdev Drivers

Release 18.08.1

April 02, 2019

The following are a list of raw device PMDs, which can be used from an application through rawdev API.

CHAPTER

ONE

NXP DPAA2 CMDIF DRIVER

The DPAA2 CMDIF is an implementation of the rawdev API, that provides communication between the GPP and AIOP (Firmware). This is achieved via using the DPCI devices exposed by MC for GPP <–> AIOP interaction.

More information can be found at NXP Official Website.

1.1 Features

The DPAA2 CMDIF implements following features in the rawdev API;

- Getting the object ID of the device (DPCI) using attributes
- I/O to and from the AIOP device using DPCI

1.2 Supported DPAA2 SoCs

- LS2084A/LS2044A
- LS2088A/LS2048A
- LS1088A/LS1048A

1.3 Prerequisites

There are three main pre-requisities for executing DPAA2 CMDIF on a DPAA2 compatible board:

1. **ARM 64 Tool Chain**

   For example, the *aarch64* Linaro Toolchain.

2. **Linux Kernel**

   It can be obtained from NXP's Github hosting.

3. **Rootfile system**

   Any *aarch64* supporting filesystem can be used. For example, Ubuntu 15.10 (Wily) or 16.04 LTS (Xenial) userland which can be obtained from here.

As an alternative method, DPAA2 CMDIF can also be executed using images provided as part of SDK from NXP. The SDK includes all the above prerequisites necessary to bring up a DPAA2 board.

The following dependencies are not part of DPDK and must be installed separately:

- **NXP Linux SDK**

  NXP Linux software development kit (SDK) includes support for family of QorIQ® ARM-Architecture-based system on chip (SoC) processors and corresponding boards.

  It includes the Linux board support packages (BSPs) for NXP SoCs, a fully operational tool chain, kernel and board specific modules.

  SDK and related information can be obtained from: NXP QorIQ SDK.

- **DPDK Extra Scripts**

  DPAA2 based resources can be configured easily with the help of ready scripts as provided in the DPDK Extra repository.

  DPDK Extras Scripts.

Currently supported by DPDK:

- NXP SDK **2.0+**.
- MC Firmware version **10.0.0** and higher.
- Supported architectures: **arm64 LE**.
- Follow the DPDK Getting Started Guide for Linux to setup the basic DPDK environment.

Note: Some part of fslmc bus code (mc flib - object library) routines are dual licensed (BSD & GPLv2).

1.4 Pre-Installation Configuration

1.4.1 Config File Options

The following options can be modified in the `config` file.

- `CONFIG_RTE_LIBRTE_PMD_DPAA2_CMDIF_RAWDEV` (default `y`)

  Toggle compilation of the `lrte_pmd_dpaa2_cmdif` driver.

1.5 Enabling logs

For enabling logs, use the following EAL parameter:

```
./your_cmdif_application <EAL args> --log-level=pmd.raw.dpaa2.cmdif,<level>
```

Using `pmd.raw.dpaa2.cmdif` as log matching criteria, all Event PMD logs can be enabled which are lower than logging `level`.

1.5.1 Driver Compilation

To compile the DPAA2 CMDIF PMD for Linux arm64 gcc target, run the following `make` command:

```
cd <DPDK-source-directory>
make config T=arm64-dpaa2-linuxapp-gcc install
```

1.6 Initialization

The DPAA2 CMDIF is exposed as a vdev device which consists of dpci devices. On EAL initialization, dpci devices will be probed and then vdev device can be created from the application code by

- Invoking `rte_vdev_init("dpaa2_dpci")` from the application
- Using `--vdev="dpaa2_dpci"` in the EAL options, which will call rte_vdev_init() internally

Example:

```
./your_cmdif_application <EAL args> --vdev="dpaa2_dpci"
```

1.6.1 Platform Requirement

DPAA2 drivers for DPDK can only work on NXP SoCs as listed in the `Supported DPAA2 SoCs`.

CHAPTER

TWO

NXP DPAA2 QDMA DRIVER

The DPAA2 QDMA is an implementation of the rawdev API, that provide means to initiate a DMA transaction from CPU. The initiated DMA is performed without CPU being involved in the actual DMA transaction. This is achieved via using the DPDMAI device exposed by MC.

More information can be found at NXP Official Website.

2.1 Features

The DPAA2 QDMA implements following features in the rawdev API;

- Supports issuing DMA of data within memory without hogging CPU while performing DMA operation.
- Supports configuring to optionally get status of the DMA translation on per DMA operation basis.

2.2 Supported DPAA2 SoCs

- LS2084A/LS2044A
- LS2088A/LS2048A
- LS1088A/LS1048A

2.3 Prerequisites

There are three main pre-requisities for executing DPAA2 QDMA on a DPAA2 compatible board:

1. **ARM 64 Tool Chain**

   For example, the *aarch64* Linaro Toolchain.

2. **Linux Kernel**

   It can be obtained from NXP's Github hosting.

3. **Rootfile system**

   Any *aarch64* supporting filesystem can be used. For example, Ubuntu 15.10 (Wily) or 16.04 LTS (Xenial) userland which can be obtained from here.

As an alternative method, DPAA2 QDMA can also be executed using images provided as part of SDK from NXP. The SDK includes all the above prerequisites necessary to bring up a DPAA2 board.

The following dependencies are not part of DPDK and must be installed separately:

- **NXP Linux SDK**

  NXP Linux software development kit (SDK) includes support for family of QorIQ® ARM-Architecture-based system on chip (SoC) processors and corresponding boards.

  It includes the Linux board support packages (BSPs) for NXP SoCs, a fully operational tool chain, kernel and board specific modules.

  SDK and related information can be obtained from: NXP QorIQ SDK.

- **DPDK Extra Scripts**

  DPAA2 based resources can be configured easily with the help of ready scripts as provided in the DPDK Extra repository.

  DPDK Extras Scripts.

Currently supported by DPDK:

- NXP LSDK **17.12+**.
- MC Firmware version **10.3.0** and higher.
- Supported architectures: **arm64 LE**.
- Follow the DPDK Getting Started Guide for Linux to setup the basic DPDK environment.

Note: Some part of fslmc bus code (mc flib - object library) routines are dual licensed (BSD & GPLv2).

2.4 Pre-Installation Configuration

2.4.1 Config File Options

The following options can be modified in the `config` file.

- `CONFIG_RTE_LIBRTE_PMD_DPAA2_QDMA_RAWDEV` (default `y`)

  Toggle compilation of the `lrte_pmd_dpaa2_qdma` driver.

2.5 Enabling logs

For enabling logs, use the following EAL parameter:

```
./your_qdma_application <EAL args> --log-level=pmd.raw.dpaa2.qdma,<level>
```

Using `pmd.raw.dpaa2.qdma` as log matching criteria, all Event PMD logs can be enabled which are lower than logging `level`.

2.5.1 Driver Compilation

To compile the DPAA2 QDMA PMD for Linux arm64 gcc target, run the following `make` command:

```
cd <DPDK-source-directory>
make config T=arm64-dpaa2-linuxapp-gcc install
```

2.6 Initialization

The DPAA2 QDMA is exposed as a vdev device which consists of dpdmai devices. On EAL initialization, dpdmai devices will be probed and populated into the rawdevices. The rawdev ID of the device can be obtained using

- Invoking `rte_rawdev_get_dev_id("dpdmai.x")` from the application where x is the object ID of the DPDMAI object created by MC. Use can use this index for further rawdev function calls.

2.6.1 Platform Requirement

DPAA2 drivers for DPDK can only work on NXP SoCs as listed in the `Supported DPAA2 SoCs`.

CHAPTER

THREE

IFPGA RAWDEV DRIVER

FPGA is used more and more widely in Cloud and NFV, one primary reason is that FPGA not only provides ASIC performance but also it's more flexible than ASIC.

FPGA uses Partial Reconfigure (PR) Parts of Bit Stream to achieve its flexibility. That means one FPGA Device Bit Stream is divided into many Parts of Bit Stream(each Part of Bit Stream is defined as AFU-Accelerated Function Unit), and each AFU is a hardware acceleration unit which can be dynamically reloaded respectively.

By PR (Partial Reconfiguration) AFUs, one FPGA resources can be time-shared by different users. FPGA hot upgrade and fault tolerance can be provided easily.

The SW IFPGA Rawdev Driver (**ifpga_rawdev**) provides a Rawdev driver that utilizes Intel FPGA Software Stack OPAE(Open Programmable Acceleration Engine) for FPGA management.

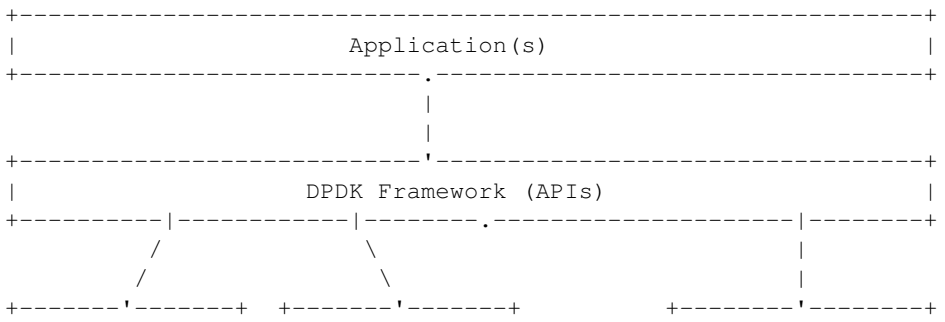
3.1 Implementation details

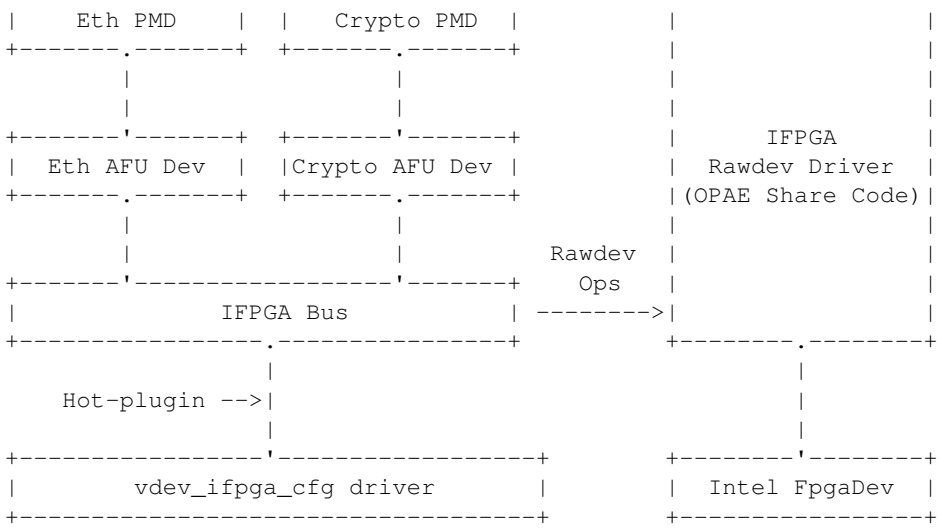
Each instance of IFPGA Rawdev Driver is probed by Intel FpgaDev. In coordination with OPAE share code IFPGA Rawdev Driver provides common FPGA management ops for FPGA operation, OPAE provides all following operations: - FPGA PR (Partial Reconfiguration) management - FPGA AFUs Identifying - FPGA Thermal Management - FPGA Power Management - FPGA Performance reporting - FPGA Remote Debug

All configuration parameters are taken by vdev_ifpga_cfg driver. Besides configuration, vdev_ifpga_cfg driver also hot plugs in IFPGA Bus.

All of the AFUs of one FPGA may share same PCI BDF and AFUs scan depend on IFPGA Rawdev Driver so IFPGA Bus takes AFU device scan and AFU drivers probe. All AFU device driver bind to AFU device by its UUID (Universally Unique Identifier).

To avoid unnecessary code duplication and ensure maximum performance, handling of AFU devices is left to different PMDs; all the design as summarized by the following block diagram:





3.2 Build options

- `CONFIG_RTE_LIBRTE_IFPGA_BUS` (default `y`)

  Toggle compilation of IFPGA Bus library.

- `CONFIG_RTE_LIBRTE_IFPGA_RAWDEV` (default `y`)

  Toggle compilation of the `ifpga_rawdev` driver.

3.3 Run-time parameters

This driver is invoked automatically in systems added with Intel FPGA, but PR and IFPGA Bus scan is trigged by command line using `--vdev 'ifpga_rawdev_cfg` EAL option.

The following device parameters are supported:

- `ifpga` [string]

  Provide a specific Intel FPGA device PCI BDF. Can be provided multiple times for additional instances.

- `port` [int]

  Each FPGA can provide many channels to PR AFU by software, each channels is identified by this parameter.

- `afu_bts` [string]

  If null, the AFU Bit Stream has been PR in FPGA, if not forces PR and identifies AFU Bit Stream file.