DPDK

DATA PLANE DEVELOPMENT KIT

Event Device Drivers

Release 17.08.0

August 08, 2017

The following are a list of event device PMDs, which can be used from an application trough the eventdev API.

CHAPTER

ONE

NXP DPAA2 EVENTDEV DRIVER

The dpaa2 eventdev is an implementation of the eventdev API, that provides a wide range of the eventdev features. The eventdev relies on a dpaa2 hw to perform event scheduling.

More information can be found at NXP Official Website.

1.1 Features

The DPAA2 EVENTDEV implements many features in the eventdev API;

- Hardware based event scheduler
- 8 event ports
- 8 event queues
- Parallel flows
- Atomic flows

1.2 Supported DPAA2 SoCs

- LS2080A/LS2040A
- LS2084A/LS2044A
- LS2088A/LS2048A
- LS1088A/LS1048A

1.3 Prerequisites

There are three main pre-requisities for executing DPAA2 EVENTDEV 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 EVENTDEV 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. Please note that enabling debugging options may affect system performance.

- `CONFIG_RTE_LIBRTE_PMD_DPAA2_EVENTDEV` (default `y`)

  Toggle compilation of the `lrte_pmd_dpaa2_event` driver.

- `CONFIG_RTE_LIBRTE_PMD_DPAA2_EVENTDEV_DEBUG` (default `n`)

  Toggle display of generic debugging messages

1.4.2 Driver Compilation

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

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

1.5 Initialization

The dpaa2 eventdev is exposed as a vdev device which consists of a set of dpcon devices and dpci devices. On EAL initialization, dpcon and dpci devices will be probed and then vdev device can be created from the application code by

- Invoking `rte_vdev_init("event_dpaa2")` from the application
- Using `--vdev="event_dpaa2"` in the EAL options, which will call rte_vdev_init() internally

Example:

```
./your_eventdev_application --vdev="event_dpaa2"
```

1.6 Limitations

1.6.1 Platform Requirement

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

1.6.2 Port-core binding

DPAA2 EVENTDEV driver requires event port ‘x’ to be used on core ‘x’.

CHAPTER

TWO

SOFTWARE EVENTDEV POLL MODE DRIVER

The software eventdev is an implementation of the eventdev API, that provides a wide range of the eventdev features. The eventdev relies on a CPU core to perform event scheduling. This PMD can use the service core library to run the scheduling function, allowing an application to utilize the power of service cores to multiplex other work on the same core if required.

2.1 Features

The software eventdev implements many features in the eventdev API;

Queues

- Atomic
- Ordered
- Parallel
- Single-Link

Ports

- Load balanced (for Atomic, Ordered, Parallel queues)
- Single Link (for single-link queues)

Event Priorities

- Each event has a priority, which can be used to provide basic QoS

2.2 Configuration and Options

The software eventdev is a vdev device, and as such can be created from the application code, or from the EAL command line:

- Call `rte_vdev_init("event_sw0")` from the application
- Use `--vdev="event_sw0"` in the EAL options, which will call rte_vdev_init() internally

Example:

```
./your_eventdev_application --vdev="event_sw0"
```

2.2.1 Scheduling Quanta

The scheduling quanta sets the number of events that the device attempts to schedule before returning to the application from the `rte_event_schedule()` function. Note that is a *hint* only, and that fewer or more events may be scheduled in a given iteration.

The scheduling quanta can be set using a string argument to the vdev create call:

```
--vdev="event_sw0,sched_quanta=64"
```

2.2.2 Credit Quanta

The credit quanta is the number of credits that a port will fetch at a time from the instance's credit pool. Higher numbers will cause less overhead in the atomic credit fetch code, however it also reduces the overall number of credits in the system faster. A balanced number (eg 32) ensures that only small numbers of credits are pre-allocated at a time, while also mitigating performance impact of the atomics.

Experimentation with higher values may provide minor performance improvements, at the cost of the whole system having less credits. On the other hand, reducing the quanta may cause measurable performance impact but provide the system with a higher number of credits at all times.

A value of 32 seems a good balance however your specific application may benefit from a higher or reduced quanta size, experimentation is required to verify possible gains.

```
--vdev="event_sw0,credit_quanta=64"
```

2.3 Limitations

The software eventdev implementation has a few limitations. The reason for these limitations is usually that the performance impact of supporting the feature would be significant.

2.3.1 "All Types" Queues

The software eventdev does not support creating queues that handle all types of traffic. An eventdev with this capability allows enqueueing Atomic, Ordered and Parallel traffic to the same queue, but scheduling each of them appropriately.

The reason to not allow Atomic, Ordered and Parallel event types in the same queue is that it causes excessive branching in the code to enqueue packets to the queue, causing a significant performance impact.

The `RTE_EVENT_DEV_CAP_QUEUE_ALL_TYPES` flag is not set in the `event_dev_cap` field of the `rte_event_dev_info` struct for the software eventdev.

2.3.2 Distributed Scheduler

The software eventdev is a centralized scheduler, requiring the `rte_event_schedule()` function to be called by a CPU core to perform the required event distribution. This is not really a limitation but rather a design decision.

The RTE_EVENT_DEV_CAP_DISTRIBUTED_SCHED flag is not set in the event_dev_cap field of the rte_event_dev_info struct for the software eventdev.

2.3.3 Dequeue Timeout

The eventdev API supports a timeout when dequeuing packets using the rte_event_dequeue_burst function. This allows a core to wait for an event to arrive, or until timeout number of ticks have passed. Timeout ticks is not supported by the software eventdev for performance reasons.

CHAPTER

THREE

OCTEONTX SSOVF EVENTDEV DRIVER

The OCTEONTX SSOVF PMD (**librte_pmd_octeontx_ssovf**) provides poll mode eventdev driver support for the inbuilt event device found in the **Cavium OCTEONTX** SoC family as well as their virtual functions (VF) in SR-IOV context.

More information can be found at Cavium, Inc Official Website.

3.1 Features

Features of the OCTEONTX SSOVF PMD are:

- 64 Event queues
- 32 Event ports
- HW event scheduler
- Supports 1M flows per event queue
- Flow based event pipelining
- Flow pinning support in flow based event pipelining
- Queue based event pipelining
- Supports ATOMIC, ORDERED, PARALLEL schedule types per flow
- Event scheduling QoS based on event queue priority
- Open system with configurable amount of outstanding events
- HW accelerated dequeue timeout support to enable power management
- SR-IOV VF

3.2 Supported OCTEONTX SoCs

- CN83xx

3.3 Prerequisites

There are three main pre-perquisites for executing SSOVF PMD on a OCTEONTX compatible board:

1. **OCTEONTX Linux kernel PF driver for Network acceleration HW blocks**

   The OCTEONTX Linux kernel drivers (including the required PF driver for the SSOVF) are available on Github at octeontx-kmod along with build, install and dpdk usage instructions.

2. **ARM64 Tool Chain**

   For example, the *aarch64* Linaro Toolchain, which can be obtained from here.

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 http://cdimage.ubuntu.com/ubuntu-base/releases/16.04/release/ubuntu-base-16.04.1-base-arm64.tar.gz.

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

   SDK and related information can be obtained from: Cavium support site.

- Follow the DPDK Getting Started Guide for Linux to setup the basic DPDK environment.

3.4 Pre-Installation Configuration

3.4.1 Config File Options

The following options can be modified in the `config` file. Please note that enabling debugging options may affect system performance.

- `CONFIG_RTE_LIBRTE_PMD_OCTEONTX_SSOVF` (default `y`)

  Toggle compilation of the `librte_pmd_octeontx_ssovf` driver.

- `CONFIG_RTE_LIBRTE_PMD_OCTEONTX_SSOVF_DEBUG` (default `n`)

  Toggle display of generic debugging messages

3.4.2 Driver Compilation

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

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

3.5 Initialization

The octeontx eventdev is exposed as a vdev device which consists of a set of SSO group and work-slot PCIe VF devices. On EAL initialization, SSO PCIe VF devices will be probed and then the vdev device can be created from the application code, or from the EAL command line based on the number of probed/bound SSO PCIe VF device to DPDK by

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

Example:

```
./your_eventdev_application --vdev="event_octeontx"
```

3.6 Limitations

3.6.1 Burst mode support

Burst mode is not supported. Dequeue and Enqueue functions accepts only single event at a time.