DATA PLANE DEVELOPMENT KIT

Event Device Drivers

Release 18.05.1

September 07, 2018

CONTENTS

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

CHAPTER

ONE

NXP DPAA EVENTDEV DRIVER

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

More information can be found at NXP Official Website.

1.1 Features

The DPAA EVENTDEV implements many features in the eventdev API;

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

1.2 Supported DPAA SoCs

- LS1046A
- LS1043A

1.3 Prerequisites

There are following pre-requisites for executing EVENTDEV on a DPAA compatible platform:

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, DPAA 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 DPAA 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**

  DPAA based resources can be configured easily with the help of ready to use xml files as provided in the DPDK Extra repository.

  DPDK Extras Scripts.

Currently supported by DPDK:

- NXP SDK **2.0+** or LSDK **17.09+**
- Supported architectures: **arm64 LE**.
- Follow the DPDK Getting Started Guide for Linux to setup the basic DPDK environment.

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_DPAA_EVENTDEV` (default `y`)

  Toggle compilation of the `librte_pmd_dpaa_event` driver.

1.4.2 Driver Compilation

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

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

1.5 Initialization

The dpaa eventdev is exposed as a vdev device which consists of a set of channels and queues. On EAL initialization, dpaa components will be probed and then vdev device can be created from the application code by

- Invoking `rte_vdev_init("event_dpaa1")` from the application
- Using `--vdev="event_dpaa1"` in the EAL options, which will call rte_vdev_init() internally

Example:

```
./your_eventdev_application --vdev="event_dpaa1"
```

1.6 Limitations

1. DPAA eventdev can not work with DPAA PUSH mode queues configured for ethdev. Please configure export DPAA_NUM_PUSH_QUEUES=0

1.6.1 Platform Requirement

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

1.6.2 Port-core Binding

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

CHAPTER

TWO

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.

2.1 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

2.2 Supported DPAA2 SoCs

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

2.3 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).

2.4 Pre-Installation Configuration

2.4.1 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.

2.4.2 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
```

2.5 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"
```

2.6 Enabling logs

For enabling logs, use the following EAL parameter:

```
./your_eventdev_application <EAL args> --log-level=pmd.event.dpaa2,<level>
```

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

2.7 Limitations

2.7.1 Platform Requirement

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

2.7.2 Port-core binding

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

CHAPTER

THREE

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.

3.1 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

3.2 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"
```

3.2.1 Scheduling Quanta

The scheduling quanta sets the number of events that the device attempts to schedule in a single schedule call performed by the service core. 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"
```

3.2.2 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"
```

3.3 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.

3.3.1 "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.

3.3.2 Distributed Scheduler

The software eventdev is a centralized scheduler, requiring a service 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.

3.3.3 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

FOUR

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.

4.1 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
- HW managed event timers support through TIMVF, with high precision and time granularity of 1us.
- Up to 64 event timer adapters.

4.2 Supported OCTEONTX SoCs

- CN83xx

4.3 Prerequisites

See `../platform/octeontx` for setup information.

4.4 Pre-Installation Configuration

4.4.1 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.

4.4.2 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
```

4.5 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"
```

4.6 Selftest

The functionality of octeontx eventdev can be verified using this option, various unit and functional tests are run to verify the sanity. The tests are run once the vdev creation is successfully complete.

```
--vdev="event_octeontx,selftest=1"
```

4.7 Enable TIMvf stats

TIMvf stats can be enabled by using this option, by default the stats are disabled.

```
--vdev="event_octeontx,timvf_stats=1"
```

4.8 Limitations

4.8.1 Burst mode support

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

4.8.2 Rx adapter support

When eth_octeontx is used as Rx adapter event schedule type `RTE_SCHED_TYPE_PARALLEL` is not supported.

4.8.3 Event timer adapter support

When timvf is used as Event timer adapter the clock source mapping is as follows:

```
RTE_EVENT_TIMER_ADAPTER_CPU_CLK  = TIM_CLK_SRC_SCLK
RTE_EVENT_TIMER_ADAPTER_EXT_CLK0 = TIM_CLK_SRC_GPIO
RTE_EVENT_TIMER_ADAPTER_EXT_CLK1 = TIM_CLK_SRC_GTI
RTE_EVENT_TIMER_ADAPTER_EXT_CLK2 = TIM_CLK_SRC_PTP
```

When timvf is used as Event timer adapter event schedule type `RTE_SCHED_TYPE_PARALLEL` is not supported.

CHAPTER

FIVE

OPDL EVENTDEV POLL MODE DRIVER

The OPDL (Ordered Packet Distribution Library) eventdev is a specificimplementation of the eventdev API. It is particularly suited to packetprocessing workloads that have high throughput and low latency requirements.All packets follow the same path through the device. The order in whichpackets follow is determinted by the order in which queues are set up.Events are left on the ring until they are transmitted. As a result packetsdo not go out of order

5.1 Features

The OPDL eventdev implements a subset of features of the eventdev API;

Queues

- Atomic
- Ordered (Parallel is supported as parallel is a subset of Ordered)
- Single-Link

Ports

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

5.2 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_opdl0")` from the application
- Use `--vdev="event_opdl0"` in the EAL options, which will call rte_vdev_init() internally

Example:

```
./your_eventdev_application --vdev="event_opdl0"
```

5.2.1 Single Port Queue

It is possible to create a Single Port Queue `RTE_EVENT_QUEUE_CFG_SINGLE_LINK`. Packets dequeued from this queue do not need to be re-enqueued (as is the case with an ordered queue). The purpose of this queue is to allow for asynchronous handling of packets in the middle of a pipeline. Ordered queues in the middle of a pipeline cannot delete packets.

5.2.2 Queue Dependencies

As stated the order in which packets travel through queues is static in nature. They go through the queues in the order the queues are setup at initialisation `rte_event_queue_setup()`. For example if an application sets up 3 queues, Q0, Q1, Q2 and has 3 assoicated ports P0, P1, P2 and P3 then packets must be

- Enqueued onto Q0 (typically through P0), then
- Dequeued from Q0 (typically through P1), then
- Enqueued onto Q1 (also through P1), then
- Dequeued from Q2 (typically through P2), then
- Enqueued onto Q3 (also through P2), then
- Dequeued from Q3 (typically through P3) and then transmitted on the relevant eth port

5.3 Limitations

The opdl implementation has a number of limitations. These limitations are due to the static nature of the underlying queues. It is because of this that the implementation can achieve such high throughput and low latency

The following list is a comprehensive outline of the what is supported and the limitations / restrictions imposed by the opdl pmd

- The order in which packets moved between queues is static and fixed (dynamic scheduling is not supported).
- NEW, RELEASE are not explicitly supported. RX (first enqueue) implicitly adds NEW event types, and TX (last dequeue) implicitly does RELEASE event types.
- All packets follow the same path through device queues.
- Flows within queues are NOT supported.
- Event priority is NOT supported.
- Once the device is stopped all inflight events are lost. Applications should clear all inflight events before stopping it.
- Each port can only be associated with one queue.
- Each queue can have multiple ports associated with it.
- Each worker core has to dequeue the maximum burst size for that port.
- For performance, the rte_event flow_id should not be updated once packetis enqueued on RX.

5.3.1 Validation & Statistics

Validation can be turned on through a command line parameter

```
--vdev="event_opdl0,do_validation=1,self_test=1"
```

If validation is turned on every packet (as opposed to just the first in each burst), is validated to have come from the right queue. Statistics are also produced in this mode. The statistics are available through the eventdev xstats API. Statistics are per port as follows:

- claim_pkts_requested
- claim_pkts_granted
- claim_non_empty
- claim_empty
- total_cycles