DPAA2 (Data Path Acceleration Architecture Gen2) Overview

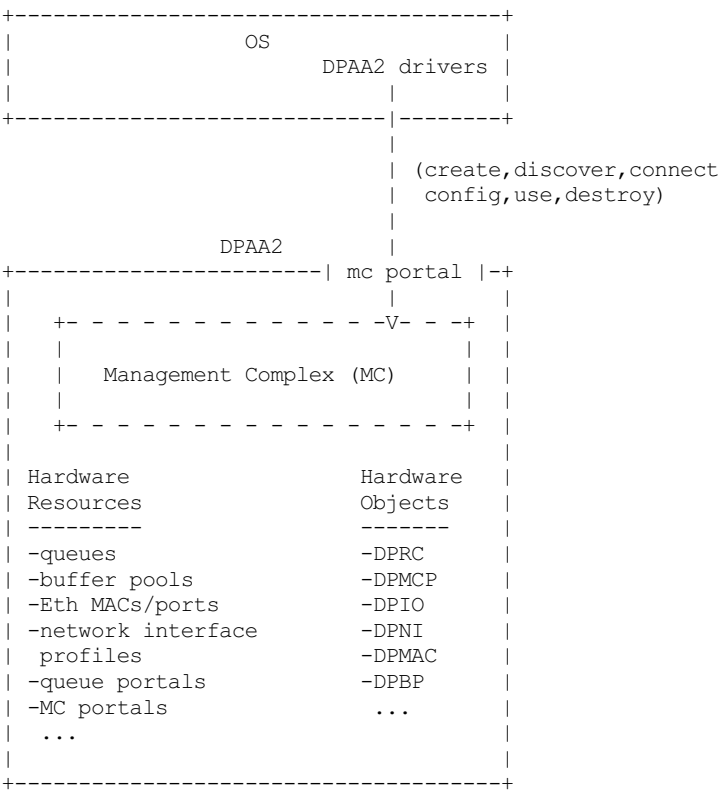
This document provides an overview of the Freescale DPAA2 architecture and how it is integrated into the Linux kernel.

Introduction

DPAA2 is a hardware architecture designed for high-speeed network packet processing. DPAA2 consists of sophisticated mechanisms for processing Ethernet packets, queue management, buffer management, autonomous L2 switching, virtual Ethernet bridging, and accelerator (e.g. crypto) sharing.

A DPAA2 hardware component called the Management Complex (or MC) manages the DPAA2 hardware resources. The MC provides an object-based abstraction for software drivers to use the DPAA2 hardware. The MC uses DPAA2 hardware resources such as queues, buffer pools, and network ports to create functional objects/devices such as network interfaces, an L2 switch, or accelerator instances. The MC provides memory-mapped I/O command interfaces (MC portals) which DPAA2 software drivers use to operate on DPAA2 objects.

The diagram below shows an overview of the DPAA2 resource management architecture:



The MC mediates operations such as create, discover, connect, configuration, and destroy. Fast-path operations on data, such as packet transmit/receive, are not mediated by the MC and are done directly using memory mapped regions in DPIO objects.

Overview of DPAA2 Objects

The section provides a brief overview of some key DPAA2 objects. A simple scenario is described illustrating the objects involved in creating a network interfaces.

DPRC (Datapath Resource Container)

A DPRC is a container object that holds all the other types of DPAA2 objects. In the example diagram below there are 8 objects of 5 types (DPMCP, DPIO, DPBP, DPNI, and DPMAC) in the container.

```
|  +-------+  +-------+  +-------+  +---+---+  +---+---+  |
|  | DPMCP |  | DPIO  |                                    |
|  +-------+  +-------+                                    |
|  | DPMCP |                                              |
|  +-------+                                              |
|                                                         |
+---------------------------------------------------------+
```

From the point of view of an OS, a DPRC behaves similar to a plug and play bus, like PCI. DPRC commands can be used to enumerate the contents of the DPRC, discover the hardware objects present (including mappable regions and interrupts).

```
DPRC.1 (bus)
   |
   +--+--------+-------+-------+-------+
      |        |       |       |       |
    DPMCP.1  DPIO.1  DPBP.1  DPNI.1  DPMAC.1
    DPMCP.2  DPIO.2
    DPMCP.3
```

Hardware objects can be created and destroyed dynamically, providing the ability to hot plug/unplug objects in and out of the DPRC.

A DPRC has a mappable MMIO region (an MC portal) that can be used to send MC commands. It has an interrupt for status events (like hotplug). All objects in a container share the same hardware "isolation context". This means that with respect to an IOMMU the isolation granularity is at the DPRC (container) level, not at the individual object level.

DPRCs can be defined statically and populated with objects via a config file passed to the MC when firmware starts it.

DPAA2 Objects for an Ethernet Network Interface

A typical Ethernet NIC is monolithic-- the NIC device contains TX/RX queuing mechanisms, configuration mechanisms, buffer management, physical ports, and interrupts. DPAA2 uses a more granular approach utilizing multiple hardware objects. Each object provides specialized functions. Groups of these objects are used by software to provide Ethernet network interface functionality. This approach provides efficient use of finite hardware resources, flexibility, and performance advantages.

The diagram below shows the objects needed for a simple network interface configuration on a system with 2 CPUs.

```
+---+---+ +---+---+
   CPU0     CPU1
+---+---+ +---+---+
    |         |
+---+---+ +---+---+
   DPIO     DPIO
+---+---+ +---+---+
    \     /
     \   /
      \ /
   +---+---+
      DPNI  --- DPBP,DPMCP
   +---+---+
       |
       |
   +---+---+
     DPMAC
   +---+---+
       |
   port/PHY
```

Below the objects are described. For each object a brief description is provided along with a summary of the kinds of operations the object supports and a summary of key resources of the object (MMIO regions and IRQs).

DPMAC (Datapath Ethernet MAC)

Represents an Ethernet MAC, a hardware device that connects to an Ethernet PHY and allows physical transmission and reception of Ethernet frames.

- MMIO regions: none
- IRQs: DPNI link change
- commands: set link up/down, link config, get stats, IRQ config, enable, reset

DPNI (Datapath Network Interface)

Contains TX/RX queues, network interface configuration, and RX buffer pool configuration mechanisms. The TX/RX queues are in memory and are identified by queue number.

- MMIO regions: none
- IRQs: link state
- commands: port config, offload config, queue config, parse/classify config, IRQ config, enable, reset

DPIO (Datapath I/O)

Provides interfaces to enqueue and dequeue packets and do hardware buffer pool management operations. The DPAA2 architecture separates the mechanism to access queues (the DPIO object) from the queues themselves. The DPIO provides an MMIO interface to enqueue/dequeue packets. To enqueue something a descriptor is written to the DPIO MMIO region, which includes the target queue number. There will typically be one DPIO assigned to each CPU. This allows all CPUs to simultaneously perform enqueue/dequeued operations. DPIOs are expected to be shared by different DPAA2 drivers.

- MMIO regions: queue operations, buffer management
- IRQs: data availability, congestion notification, buffer pool depletion
- commands: IRQ config, enable, reset

DPBP (Datapath Buffer Pool)

Represents a hardware buffer pool.

- MMIO regions: none
- IRQs: none
- commands: enable, reset

DPMCP (Datapath MC Portal)

Provides an MC command portal. Used by drivers to send commands to the MC to manage objects.

- MMIO regions: MC command portal
- IRQs: command completion
- commands: IRQ config, enable, reset

Object Connections

Some objects have explicit relationships that must be configured:

- DPNI <--> DPMAC
- DPNI <--> DPNI
- DPNI <--> L2-switch-port

  A DPNI must be connected to something such as a DPMAC, another DPNI, or L2 switch port. The DPNI connection is made via a DPRC command.

```
+-------+  +-------+
| DPNI  |  | DPMAC |
+---+---+  +---+---+
    |          |
    +==========+
```

- DPNI <--> DPBP

  A network interface requires a 'buffer pool' (DPBP object) which provides a list of pointers to memory where received Ethernet data is to be copied. The Ethernet driver configures the DPBPs associated with the network interface.

Interrupts

All interrupts generated by DPAA2 objects are message interrupts. At the hardware level message interrupts generated by devices will normally have 3 components-- 1) a non-spoofable 'device-id' expressed on the hardware bus, 2) an address, 3) a data value.

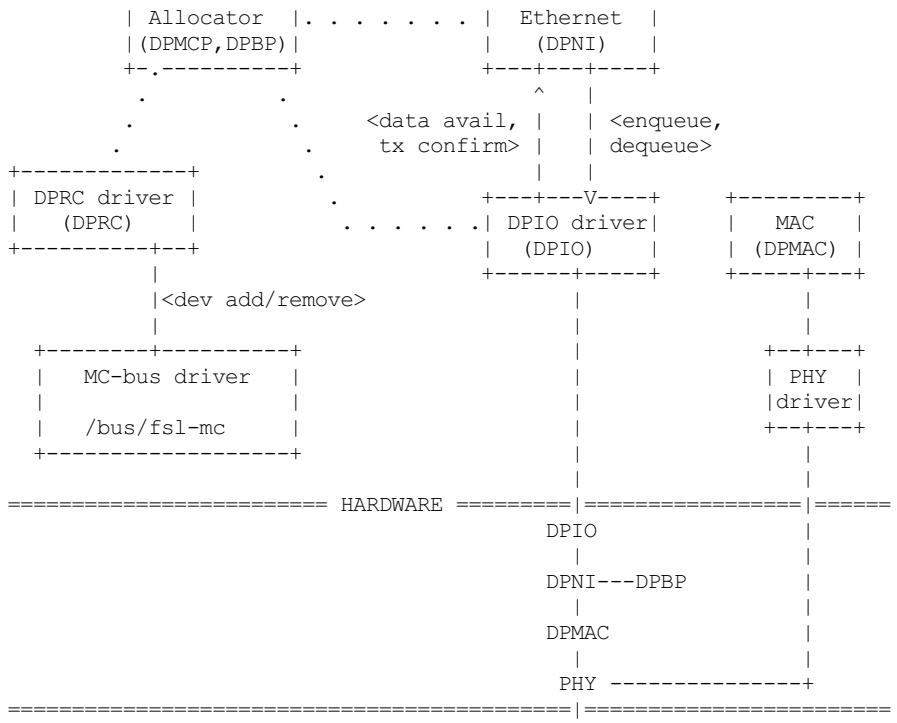
In the case of DPAA2 devices/objects, all objects in the same container/DPRC share the same 'device-id'. For ARM-based SoC this is the same as the stream ID.

DPAA2 Linux Drivers Overview

This section provides an overview of the Linux kernel drivers for DPAA2-- 1) the bus driver and associated "DPAA2 infrastructure" drivers and 2) functional object drivers (such as Ethernet).

As described previously, a DPRC is a container that holds the other types of DPAA2 objects. It is functionally similar to a plug-and-play bus controller. Each object in the DPRC is a Linux "device" and is bound to a driver. The diagram below shows the Linux drivers involved in a networking scenario and the objects bound to each driver. A brief description of each driver follows.

```
                                  +------------+
                                  | OS Network |
                                  |   Stack    |
        +------------+            +------------+
```



A brief description of each driver is provided below.

MC-bus driver

The MC-bus driver is a platform driver and is probed from a node in the device tree (compatible "fsl,qoriq-mc") passed in by boot firmware. It is responsible for bootstrapping the DPAA2 kernel infrastructure. Key functions include:

- registering a new bus type named "fsl-mc" with the kernel, and implementing bus call-backs (e.g. match/uevent/dev_groups)
- implementing APIs for DPAA2 driver registration and for device add/remove
- creates an MSI IRQ domain
- doing a 'device add' to expose the 'root' DPRC, in turn triggering a bind of the root DPRC to the DPRC driver

The binding for the MC-bus device-tree node can be consulted at *Documentation/devicetree/bindings/misc/fsl,qoriq-mc.txt*. The sysfs bind/unbind interfaces for the MC-bus can be consulted at *Documentation/ABI/testing/sysfs-bus-fsl-mc*.

DPRC driver

The DPRC driver is bound to DPRC objects and does runtime management of a bus instance. It performs the initial bus scan of the DPRC and handles interrupts for container events such as hot plug by re-scanning the DPRC.

Allocator

Certain objects such as DPMCP and DPBP are generic and fungible, and are intended to be used by other drivers. For example, the DPAA2 Ethernet driver needs:

- DPMCPs to send MC commands, to configure network interfaces
- DPBPs for network buffer pools

The allocator driver registers for these allocatable object types and those objects are bound to the allocator when the bus is probed. The allocator maintains a pool of objects that are available for allocation by other DPAA2 drivers.

DPIO driver

The DPIO driver is bound to DPIO objects and provides services that allow other drivers such as the Ethernet driver to enqueue and dequeue data for their respective objects. Key services include:

- data availability notifications
- hardware queuing operations (enqueue and dequeue of data)
- hardware buffer pool management

To transmit a packet the Ethernet driver puts data on a queue and invokes a DPIO API. For receive, the Ethernet driver registers a data availability notification callback. To dequeue a packet a DPIO API is used. There is typically one DPIO object per physical CPU for optimum performance, allowing different CPUs to simultaneously enqueue and dequeue data.

The DPIO driver operates on behalf of all DPAA2 drivers active in the kernel-- Ethernet, crypto, compression, etc.

Ethernet driver

The Ethernet driver is bound to a DPNI and implements the kernel interfaces needed to connect the DPAA2 network interface to the network stack. Each DPNI corresponds to a Linux network interface.

MAC driver

An Ethernet PHY is an off-chip, board specific component and is managed by the appropriate PHY driver via an mdio bus. The MAC driver plays a role of being a proxy between the PHY driver and the MC. It does this proxy via the MC commands to a DPMAC object. If the PHY driver signals a link change, the MAC driver notifies the MC via a DPMAC command. If a network interface is brought up or down, the MC notifies the DPMAC driver via an interrupt and the driver can take appropriate action.