DPDK

DATA PLANE DEVELOPMENT KIT

Platform Specific Guides

Release 19.08.2

Nov 15, 2019

The following are platform specific guides and setup information.

CHAPTER

ONE

MELLANOX BLUEFIELD BOARD SUPPORT PACKAGE

This document has information about steps to setup Mellanox BlueField platform and common offload HW drivers of **Mellanox BlueField** family SoC.

1.1 Supported BlueField family SoCs

- BlueField

1.2 Supported BlueField Platforms

- BlueField SmartNIC
- BlueField Reference Platforms
- BlueField Controller Card

1.3 Common Offload HW Drivers

1. **NIC Driver**

   See ../nics/mlx5 for Mellanox mlx5 NIC driver information.

2. **Cryptodev Driver**

   This is based on the crypto extension support of armv8. See ../cryptodevs/armv8 for armv8 crypto driver information.

Note: BlueField has a variant having no armv8 crypto extension support.

1.4 Steps To Setup Platform

Toolchains, OS and drivers can be downloaded and installed individually from the Web. But it is recommended to follow instructions at Mellanox BlueField Software Website.

1.5 Compile DPDK

DPDK can be compiled either natively on BlueField platforms or cross-compiled on an x86 based platform.

1.5.1 Native Compilation

Refer to ../nics/mlx5 for prerequisites. Either Mellanox OFED/EN or rdma-core library with corresponding kernel drivers is required.

make build

```
make config T=arm64-bluefield-linux-gcc
make -j
```

meson build

```
meson build
ninja -C build
```

1.5.2 Cross Compilation

Refer to ../linux_gsg/cross_build_dpdk_for_arm64 to install the cross toolchain for ARM64. Base on that, additional header files and libraries are required:

- libibverbs
- libmlx5
- libnl-3
- libnl-route-3

Such header files and libraries can be cross-compiled and installed on to the cross toolchain directory like depicted in arm_cross_build_getting_the_prerequisite_library, but those can also be simply copied from the filesystem of a working BlueField platform. The following script can be run on a BlueField platform in order to create a supplementary tarball for the cross toolchain.

```
mkdir -p aarch64-linux-gnu/libc
pushd $PWD
cd aarch64-linux-gnu/libc

# Copy libraries
mkdir -p lib64
cp -a /lib64/libibverbs* lib64/
cp -a /lib64/libmlx5* lib64/
cp -a /lib64/libnl-3* lib64/
cp -a /lib64/libnl-route-3* lib64/

# Copy header files
mkdir -p usr/include/infiniband
cp -a /usr/include/infiniband/ib_user_ioctl_verbs.h usr/include/infiniband/
cp -a /usr/include/infiniband/mlx5*.h usr/include/infiniband/
cp -a /usr/include/infiniband/tm_types.h usr/include/infiniband/
cp -a /usr/include/infiniband/verbs*.h usr/include/infiniband/
```

```
# Create supplementary tarball
popd
tar cf aarch64-linux-gnu-mlx.tar aarch64-linux-gnu/
```

Then, untar the tarball at the cross toolchain directory on the x86 host.

```
cd $(dirname $(which aarch64-linux-gnu-gcc))/..
tar xf aarch64-linux-gnu-mlx.tar
```

make build

```
make config T=arm64-bluefield-linux-gcc
make -j CROSS=aarch64-linux-gnu- CONFIG_RTE_KNI_KMOD=n CONFIG_RTE_EAL_IGB_UIO=n
```

meson build

```
meson build --cross-file config/arm/arm64_bluefield_linux_gcc
ninja -C build
```

CHAPTER

TWO

NXP QORIQ DPAA BOARD SUPPORT PACKAGE

This doc has information about steps to setup QorIQ dpaa based layerscape platform and information about common offload hw block drivers of **NXP QorIQ DPAA** SoC family.

2.1 Supported DPAA SoCs

- LS1046A/LS1026A
- LS1043A/LS1023A

More information about SoC can be found at NXP Official Website.

2.2 Common Offload HW Block Drivers

1. **Nics Driver**

   See ../nics/dpaa for NXP dpaa nic driver information.

2. **Cryptodev Driver**

   See ../cryptodevs/dpaa_sec for NXP dpaa cryptodev driver information.

3. **Eventdev Driver**

   See ../eventdevs/dpaa for NXP dpaa eventdev driver information.

2.3 Steps To Setup Platform

There are four main pre-requisites for executing DPAA PMD on a DPAA 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 16.04 LTS (Xenial) or 18.04 (Bionic) userland which can be obtained from here.

4. **FMC Tool**

   Before any DPDK application can be executed, the Frame Manager Configuration Tool (FMC) need to be executed to set the configurations of the queues. This includes the queue state, RSS and other policies. This tool can be obtained from NXP (Freescale) Public Git Repository.

   This tool needs configuration files which are available in the *DPDK Extra Scripts*, described below for DPDK usages.

As an alternative method, DPAA PMD 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 scripts as provided in the DPDK Extra repository.

  DPDK Extras Scripts.

Currently supported by DPDK:

- NXP SDK **2.0+** (preferred: LSDK 18.09).
- Supported architectures: **arm64 LE**.
- Follow the DPDK Getting Started Guide for Linux to setup the basic DPDK environment.

CHAPTER

THREE

NXP QORIQ DPAA2 BOARD SUPPORT PACKAGE

This doc has information about steps to setup NXP QorIQ DPAA2 platform and information about common offload hw block drivers of **NXP QorIQ DPAA2** SoC family.

3.1 Supported DPAA2 SoCs

- LX2160A
- LS2084A/LS2044A
- LS2088A/LS2048A
- LS1088A/LS1048A

More information about SoC can be found at NXP Official Website.

3.2 Common Offload HW Block Drivers

1. **Nics Driver**

   See ../nics/dpaa2 for NXP dpaa2 nic driver information.

2. **Cryptodev Driver**

   See ../cryptodevs/dpaa2_sec for NXP dpaa2 cryptodev driver information.

3. **Eventdev Driver**

   See ../eventdevs/dpaa2 for NXP dpaa2 eventdev driver information.

4. **Rawdev AIOP CMDIF Driver**

   See ../rawdevs/dpaa2_cmdif for NXP dpaa2 AIOP command interface driver information.

5. **Rawdev QDMA Driver**

   See ../rawdevs/dpaa2_qdma for NXP dpaa2 QDMA driver information.

3.3 Steps To Setup Platform

There are four main pre-requisites for executing DPAA2 PMD 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 16.04 LTS (Xenial) or 18.04 (Bionic) userland which can be obtained from here.

4. **Resource Scripts**

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

As an alternative method, DPAA2 PMD 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+** (preferred: LSDK 19.03).
- MC Firmware version **10.14.0** and higher.
- Supported architectures: **arm64 LE**.
- Follow the DPDK Getting Started Guide for Linux to setup the basic DPDK environment.

CHAPTER

FOUR

OCTEON TX BOARD SUPPORT PACKAGE

This doc has information about steps to setup OCTEON TX platform and information about common offload hw block drivers of **Cavium OCTEON TX** SoC family.

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

4.1 Common Offload HW Block Drivers

1. **Crypto Driver** See ../cryptodevs/octeontx for octeontx crypto driver information.
2. **Eventdev Driver** See ../eventdevs/octeontx for octeontx ssovf eventdev driver information.
3. **Mempool Driver** See ../mempool/octeontx for octeontx fpavf mempool driver information.

4.2 Steps To Setup Platform

There are three main pre-prerequisites for setting up Platform drivers on OCTEON TX compatible board:

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

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

Note: The PF driver and the required microcode for the crypto offload block will be available with OCTEON TX SDK only. So for using crypto offload, follow the steps mentioned in *Setup Platform Using OCTEON TX SDK*.

---

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, Platform drivers 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 OCTEON TX board. Please refer *Setup Platform Using OCTEON TX SDK*.

- Follow the DPDK ../linux_gsg/index to setup the basic DPDK environment.

4.3 Setup Platform Using OCTEON TX SDK

The OCTEON TX platform drivers can be compiled either natively on **OCTEON TX** ® board or cross-compiled on an x86 based platform.

The **OCTEON TX** ® board must be running the linux kernel based on OCTEON TX SDK 6.2.0 patch 3. In this, the PF drivers for all hardware offload blocks are already built in.

4.3.1 Native Compilation

If the kernel and modules are cross-compiled and copied to the target board, some intermediate binaries required for native build would be missing on the target board. To make sure all the required binaries are available in the native architecture, the linux sources need to be compiled once natively.

```
cd /lib/modules/$(uname -r)/source
make menuconfig
make
```

The above steps would rebuild the modules and the required intermediate binaries. Once the target is ready for native compilation, the OCTEON TX platform drivers can be compiled with the following steps,

```
cd <dpdk directory>
make config T=arm64-thunderx-linux-gcc
make
```

The example applications can be compiled using the following:

```
cd <dpdk directory>
export RTE_SDK=$PWD
export RTE_TARGET=build
cd examples/<application>
make
```

4.3.2 Cross Compilation

The DPDK applications can be cross-compiled on any x86 based platform. The OCTEON TX SDK need to be installed on the build system. The SDK package will provide the required toolchain etc.

Refer to ../linux_gsg/cross_build_dpdk_for_arm64 for further steps on compilation. The 'host' & 'CC' to be used in the commands would change, in addition to the paths to which libnuma related files have to be copied.

The following steps can be used to perform cross-compilation with OCTEON TX SDK 6.2.0 patch 3:

```
cd <sdk_install_dir>
source env-setup

git clone https://github.com/numactl/numactl.git
cd numactl
git checkout v2.0.11 -b v2.0.11
./autogen.sh
autoconf -i
./configure --host=aarch64-thunderx-linux CC=aarch64-thunderx-linux-gnu-gcc --prefix=<numa inst
make install
```

The above steps will prepare build system with numa additions. Now this build system can be used to build applications for **OCTEON TX** ® platforms.

```
cd <dpdk directory>
export RTE_SDK=$PWD
export RTE_KERNELDIR=$THUNDER_ROOT/linux/kernel/linux
make config T=arm64-thunderx-linux-gcc
make -j CROSS=aarch64-thunderx-linux-gnu- CONFIG_RTE_KNI_KMOD=n CONFIG_RTE_EAL_IGB_UIO=n EXTRA_
```

If NUMA support is not required, it can be disabled as explained in ../linux_gsg/cross_build_dpdk_for_arm64.

Following steps could be used in that case.

```
make config T=arm64-thunderx-linux-gcc
make CROSS=aarch64-thunderx-linux-gnu-
```

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

CHAPTER

FIVE

MARVELL OCTEON TX2 PLATFORM GUIDE

This document gives an overview of **Marvell OCTEON TX2** RVU H/W block, packet flow and procedure to build DPDK on OCTEON TX2 platform.

More information about OCTEON TX2 SoC can be found at Marvell Official Website.

5.1 Supported OCTEON TX2 SoCs

- CN96xx
- CN93xx
- CNF95xx

5.2 OCTEON TX2 Resource Virtualization Unit architecture

The Fig. 5.1 diagram depicts the RVU architecture and a resource provisioning example.

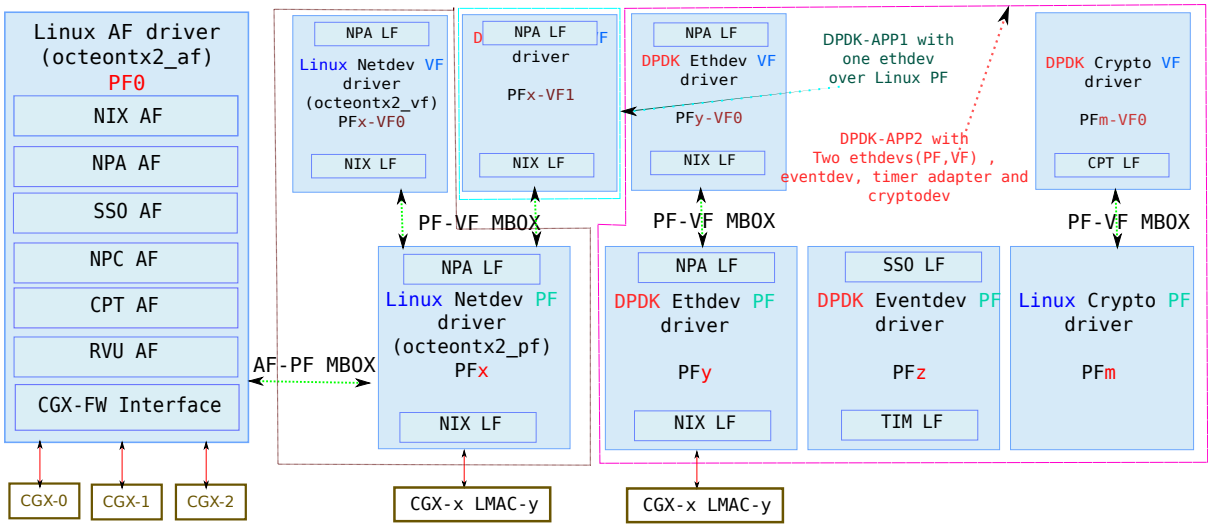

Fig. 5.1: OCTEON TX2 Resource virtualization architecture and provisioning example

Resource Virtualization Unit (RVU) on Marvell's OCTEON TX2 SoC maps HW resources belonging to the network, crypto and other functional blocks onto PCI-compatible physical and virtual functions.

Each functional block has multiple local functions (LFs) for provisioning to different PCIe devices. RVU supports multiple PCIe SRIOV physical functions (PFs) and virtual functions (VFs).

The Table 5.1 shows the various local functions (LFs) provided by the RVU and its functional mapping to DPDK subsystem.

Table 5.1: RVU managed functional blocks and its mapping to DPDK subsystem

#	LF	DPDK subsystem mapping
1	NIX	rte_ethdev, rte_tm, rte_event_eth_[rt]x_adapter, rte_security
2	NPA	rte_mempool
3	NPC	rte_flow
4	CPT	rte_cryptodev, rte_event_crypto_adapter
5	SSO	rte_eventdev
6	TIM	rte_event_timer_adapter
7	LBK	rte_ethdev
8	DPI	rte_rawdev

PF0 is called the administrative / admin function (AF) and has exclusive privileges to provision RVU functional block's LFs to each of the PF/VF.

PF/VFs communicates with AF via a shared memory region (mailbox).Upon receiving requests from PF/VF, AF does resource provisioning and other HW configuration.

AF is always attached to host, but PF/VFs may be used by host kernel itself, or attached to VMs or to userspace applications like DPDK, etc. So, AF has to handle provisioning/configuration requests sent by any device from any domain.

The AF driver does not receive or process any data. It is only a configuration driver used in control path.

The Fig. 5.1 diagram also shows a resource provisioning example where,

1. PFx and PFx-VF0 bound to Linux netdev driver.
2. PFx-VF1 ethdev driver bound to the first DPDK application.
3. PFy ethdev driver, PFy-VF0 ethdev driver, PFz eventdev driver, PFm-VF0 cryptodev driver bound to the second DPDK application.

5.3 LBK HW Access

Loopback HW Unit (LBK) receives packets from NIX-RX and sends packets back to NIX-TX. The loopback block has N channels and contains data buffering that is shared across all channels. The LBK HW Unit is abstracted using ethdev subsystem, Where PF0's VFs are exposed as ethdev device and odd-even pairs of VFs are tied together, that is, packets sent on odd VF end up received on even VF and vice versa. This would enable HW accelerated means of communication between two domains where even VF bound to the first domain and odd VF bound to the second domain.

Typical application usage models are,

1. Communication between the Linux kernel and DPDK application.
2. Exception path to Linux kernel from DPDK application as SW `KNI` replacement.
3. Communication between two different DPDK applications.

5.4 OCTEON TX2 packet flow

The Fig. 5.2 diagram depicts the packet flow on OCTEON TX2 SoC in conjunction with use of various HW accelerators.

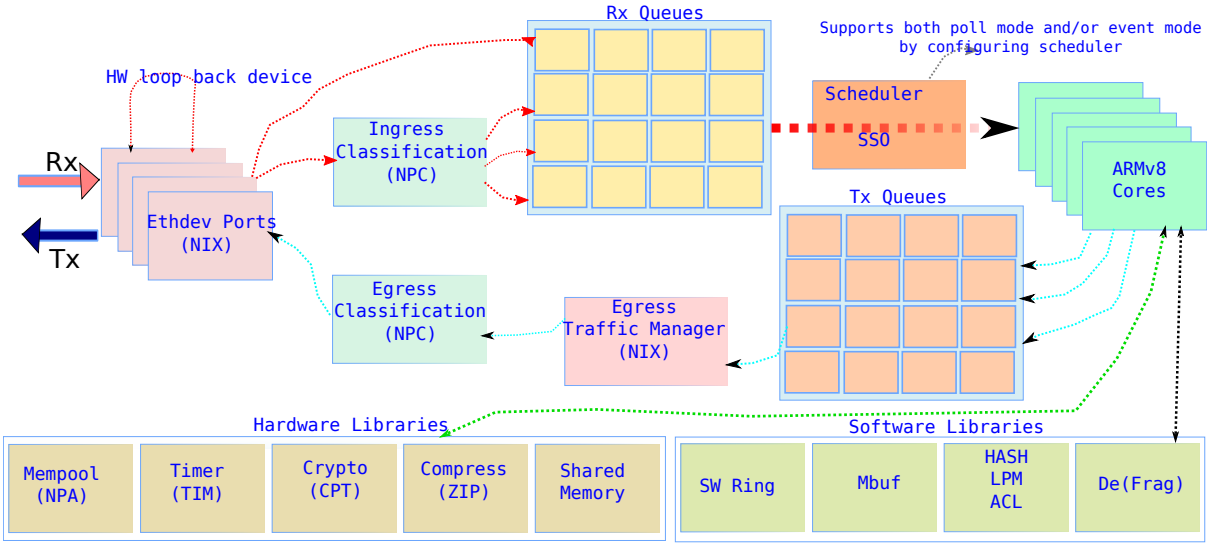

Fig. 5.2: OCTEON TX2 packet flow in conjunction with use of HW accelerators

5.5 HW Offload Drivers

This section lists dataplane H/W block(s) available in OCTEON TX2 SoC.

1. **Ethdev Driver** See ../nics/octeontx2 for NIX Ethdev driver information.
2. **Mempool Driver** See ../mempool/octeontx2 for NPA mempool driver information.
3. **Event Device Driver** See ../eventdevs/octeontx2 for SSO event device driver information.
4. **DMA Rawdev Driver** See ../rawdevs/octeontx2_dma for DMA driver information.

5.6 Procedure to Setup Platform

There are three main prerequisites for setting up DPDK on OCTEON TX2 compatible board:

1. **OCTEON TX2 Linux kernel driver**

   The dependent kernel drivers can be obtained from the kernel.org.

   Alternatively, the Marvell SDK also provides the required kernel drivers.

   Linux kernel should be configured with the following features enabled:

```
# 64K pages enabled for better performance
CONFIG_ARM64_64K_PAGES=y
CONFIG_ARM64_VA_BITS_48=y
# huge pages support enabled
CONFIG_HUGETLBFS=y
CONFIG_HUGETLB_PAGE=y
# VFIO enabled with TYPE1 IOMMU at minimum
```

```
CONFIG_VFIO_IOMMU_TYPE1=y
CONFIG_VFIO_VIRQFD=y
CONFIG_VFIO=y
CONFIG_VFIO_NOIOMMU=y
CONFIG_VFIO_PCI=y
CONFIG_VFIO_PCI_MMAP=y
# SMMUv3 driver
CONFIG_ARM_SMMU_V3=y
# ARMv8.1 LSE atomics
CONFIG_ARM64_LSE_ATOMICS=y
# OCTEONTX2 drivers
CONFIG_OCTEONTX2_MBOX=y
CONFIG_OCTEONTX2_AF=y
# Enable if netdev PF driver required
CONFIG_OCTEONTX2_PF=y
# Enable if netdev VF driver required
CONFIG_OCTEONTX2_VF=y
CONFIG_CRYPTO_DEV_OCTEONTX2_CPT=y
# Enable if OCTEONTX2 DMA PF driver required
CONFIG_OCTEONTX2_DPI_PF=n
```

2. **ARM64 Linux Tool Chain**

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

   Alternatively, the Marvell SDK also provides GNU GCC toolchain, which is optimized for OCTEON TX2 CPU.

3. **Rootfile system**

   Any *aarch64* supporting filesystem may 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.

   Alternatively, the Marvell SDK provides the buildroot based root filesystem. The SDK includes all the above prerequisites necessary to bring up the OCTEON TX2 board.

- Follow the DPDK ../linux_gsg/index to setup the basic DPDK environment.

5.7 Debugging Options

Table 5.2: OCTEON TX2 common debug options

#	Component	EAL log command
1	Common	–log-level='pmd.octeontx2.base,8'
2	Mailbox	–log-level='pmd.octeontx2.mbox,8'

5.7.1 Debugfs support

The **OCTEON TX2 Linux kernel driver** provides support to dump RVU blocks context or stats using debugfs.

Enable `debugfs` by:

1. Compile kernel with debugfs enabled, i.e `CONFIG_DEBUGFS=y`.
2. Boot OCTEON TX2 with debugfs supported kernel.

3. Verify `debugfs` mounted by default "mount | grep -i debugfs" or mount it manually by using.

```
# mount -t debugfs none /sys/kernel/debug
```

Currently `debugfs` supports the following RVU blocks NIX, NPA, NPC, NDC, SSO & CGX.

The file structure under `/sys/kernel/debug` is as follows

```
octeontx2/
|-- cgx
|   |-- cgx0
|   |   '-- lmac0
|   |       '-- stats
|   |-- cgx1
|   |   |-- lmac0
|   |   |   '-- stats
|   |   '-- lmac1
|   |       '-- stats
|   '-- cgx2
|       '-- lmac0
|           '-- stats
|-- cpt
|   |-- cpt_engines_info
|   |-- cpt_engines_sts
|   |-- cpt_err_info
|   |-- cpt_lfs_info
|   '-- cpt_pc
|---- nix
|   |-- cq_ctx
|   |-- ndc_rx_cache
|   |-- ndc_rx_hits_miss
|   |-- ndc_tx_cache
|   |-- ndc_tx_hits_miss
|   |-- qsize
|   |-- rq_ctx
|   |-- sq_ctx
|   '-- tx_stall_hwissue
|-- npa
|   |-- aura_ctx
|   |-- ndc_cache
|   |-- ndc_hits_miss
|   |-- pool_ctx
|   '-- qsize
|-- npc
|    |-- mcam_info
|    '-- rx_miss_act_stats
|-- rsrc_alloc
'-- sso
     |-- hws
     |   '-- sso_hws_info
     '-- hwgrp
         |-- sso_hwgrp_aq_thresh
         |-- sso_hwgrp_iaq_walk
         |-- sso_hwgrp_pc
         |-- sso_hwgrp_free_list_walk
         |-- sso_hwgrp_ient_walk
         '-- sso_hwgrp_taq_walk
```

RVU block LF allocation:

```
cat /sys/kernel/debug/octeontx2/rsrc_alloc

pcifunc    NPA    NIX    SSO GROUP    SSOWS    TIM    CPT
PF1         0       0
PF4                 1
```

```
PF13                          0, 1     0, 1      0
```

CGX example usage:

```
cat /sys/kernel/debug/octeontx2/cgx/cgx2/lmac0/stats

=======Link Status======
Link is UP 40000 Mbps
=======RX_STATS======
Received packets: 0
Octets of received packets: 0
Received PAUSE packets: 0
Received PAUSE and control packets: 0
Filtered DMAC0 (NIX-bound) packets: 0
Filtered DMAC0 (NIX-bound) octets: 0
Packets dropped due to RX FIFO full: 0
Octets dropped due to RX FIFO full: 0
Error packets: 0
Filtered DMAC1 (NCSI-bound) packets: 0
Filtered DMAC1 (NCSI-bound) octets: 0
NCSI-bound packets dropped: 0
NCSI-bound octets dropped: 0
=======TX_STATS======
Packets dropped due to excessive collisions: 0
Packets dropped due to excessive deferral: 0
Multiple collisions before successful transmission: 0
Single collisions before successful transmission: 0
Total octets sent on the interface: 0
Total frames sent on the interface: 0
Packets sent with an octet count < 64: 0
Packets sent with an octet count == 64: 0
Packets sent with an octet count of 65127: 0
Packets sent with an octet count of 128-255: 0
Packets sent with an octet count of 256-511: 0
Packets sent with an octet count of 512-1023: 0
Packets sent with an octet count of 1024-1518: 0
Packets sent with an octet count of > 1518: 0
Packets sent to a broadcast DMAC: 0
Packets sent to the multicast DMAC: 0
Transmit underflow and were truncated: 0
Control/PAUSE packets sent: 0
```

CPT example usage:

```
cat /sys/kernel/debug/octeontx2/cpt/cpt_pc

CPT instruction requests   0
CPT instruction latency    0
CPT NCB read requests      0
CPT NCB read latency       0
CPT read requests caused by UC fills   0
CPT active cycles pc       1395642
CPT clock count pc         5579867595493
```

NIX example usage:

```
Usage: echo <nixlf> [cq number/all] > /sys/kernel/debug/octeontx2/nix/cq_ctx
       cat /sys/kernel/debug/octeontx2/nix/cq_ctx
echo 0 0 > /sys/kernel/debug/octeontx2/nix/cq_ctx
cat /sys/kernel/debug/octeontx2/nix/cq_ctx

=====cq_ctx for nixlf:0 and qidx:0 is=====
W0: base                        158ef1a00

W1: wrptr                       0
```

```
W1: avg_con                     0
W1: cint_idx                    0
W1: cq_err                      0
W1: qint_idx                    0
W1: bpid                        0
W1: bp_ena                      0

W2: update_time                 31043
W2:avg_level                    255
W2: head                        0
W2:tail                         0

W3: cq_err_int_ena              5
W3:cq_err_int                   0
W3: qsize                       4
W3:caching                      1
W3: substream                   0x000
W3: ena                                 1
W3: drop_ena                    1
W3: drop                        64
W3: bp                          0
```

NPA example usage:

```
Usage: echo <npalf> [pool number/all] > /sys/kernel/debug/octeontx2/npa/pool_ctx
       cat /sys/kernel/debug/octeontx2/npa/pool_ctx
echo 0 0 > /sys/kernel/debug/octeontx2/npa/pool_ctx
cat /sys/kernel/debug/octeontx2/npa/pool_ctx

======POOL : 0=======
W0: Stack base          1375bff00
W1: ena                 1
W1: nat_align           1
W1: stack_caching       1
W1: stack_way_mask      0
W1: buf_offset          1
W1: buf_size            19
W2: stack_max_pages     24315
W2: stack_pages         24314
W3: op_pc               267456
W4: stack_offset        2
W4: shift               5
W4: avg_level           255
W4: avg_con             0
W4: fc_ena              0
W4: fc_stype            0
W4: fc_hyst_bits        0
W4: fc_up_crossing      0
W4: update_time         62993
W5: fc_addr             0
W6: ptr_start           1593adf00
W7: ptr_end             180000000
W8: err_int             0
W8: err_int_ena         7
W8: thresh_int          0
W8: thresh_int_ena      0
W8: thresh_up           0
W8: thresh_qint_idx     0
W8: err_qint_idx        0
```

NPC example usage:

```
cat /sys/kernel/debug/octeontx2/npc/mcam_info
```

```
NPC MCAM info:
RX keywidth    : 224bits
TX keywidth    : 224bits

MCAM entries   : 2048
Reserved       : 158
Available      : 1890

MCAM counters  : 512
Reserved       : 1
Available      : 511
```

SSO example usage:

```
Usage: echo [<hws>/all] > /sys/kernel/debug/octeontx2/sso/hws/sso_hws_info
echo 0 > /sys/kernel/debug/octeontx2/sso/hws/sso_hws_info

================================================
SSOW HWS[0] Arbitration State      0x0
SSOW HWS[0] Guest Machine Control  0x0
SSOW HWS[0] SET[0] Group Mask[0] 0xffffffffffffffff
SSOW HWS[0] SET[0] Group Mask[1] 0xffffffffffffffff
SSOW HWS[0] SET[0] Group Mask[2] 0xffffffffffffffff
SSOW HWS[0] SET[0] Group Mask[3] 0xffffffffffffffff
SSOW HWS[0] SET[1] Group Mask[0] 0xffffffffffffffff
SSOW HWS[0] SET[1] Group Mask[1] 0xffffffffffffffff
SSOW HWS[0] SET[1] Group Mask[2] 0xffffffffffffffff
SSOW HWS[0] SET[1] Group Mask[3] 0xffffffffffffffff
================================================
```

5.8 Compile DPDK

DPDK may be compiled either natively on OCTEON TX2 platform or cross-compiled on an x86 based platform.

5.8.1 Native Compilation

make build

```
make config T=arm64-octeontx2-linux-gcc
make -j
```

The example applications can be compiled using the following:

```
cd <dpdk directory>
export RTE_SDK=$PWD
export RTE_TARGET=build
cd examples/<application>
make -j
```

meson build

```
meson build
ninja -C build
```

5.8.2 Cross Compilation

Refer to ../linux_gsg/cross_build_dpdk_for_arm64 for generic arm64 details.

make build

```
make config T=arm64-octeontx2-linux-gcc
make -j CROSS=aarch64-marvell-linux-gnu- CONFIG_RTE_KNI_KMOD=n
```

meson build

```
meson build --cross-file config/arm/arm64_octeontx2_linux_gcc
ninja -C build
```

Note: By default, meson cross compilation uses `aarch64-linux-gnu-gcc` toolchain, if Marvell toolchain is available then it can be used by overriding the c, cpp, ar, strip `binaries` attributes to respective Marvell toolchain binaries in `config/arm/arm64_octeontx2_linux_gcc` file.
