Intel® Ethernet Controller E810
Data Plane Development Kit (DPDK) 20.11

Configuration Guide

Ethernet Products Group (EPG)

December 2020

Revision 1.0
633514-001

Revision History

Revision	Date	Comments
1.0	December 18, 2020	Initial public release.

Contents

NOTE: ***This page intentionally left blank.***

1.0 Known Issues - Read First

None.

2.0 Introduction

This document is designed to provide instructions for configuring and testing Intel® Ethernet 800 Series Network Adapters with Data Plane Development Kit (DPDK).

2.1 DPDK Overview

DPDK is a set of libraries and drivers that perform fast packet processing. This enables a user to create optimized performance with packet processing applications.

DPDK bypasses the OS network stack, avoiding the associated latency, and maps hardware registers to user space. A DPDK-enabled application processes packets faster by allowing NICs to DMA packets directly into an application's address space and having the application poll for packets, thereby avoiding the overhead of interrupts from the NIC.

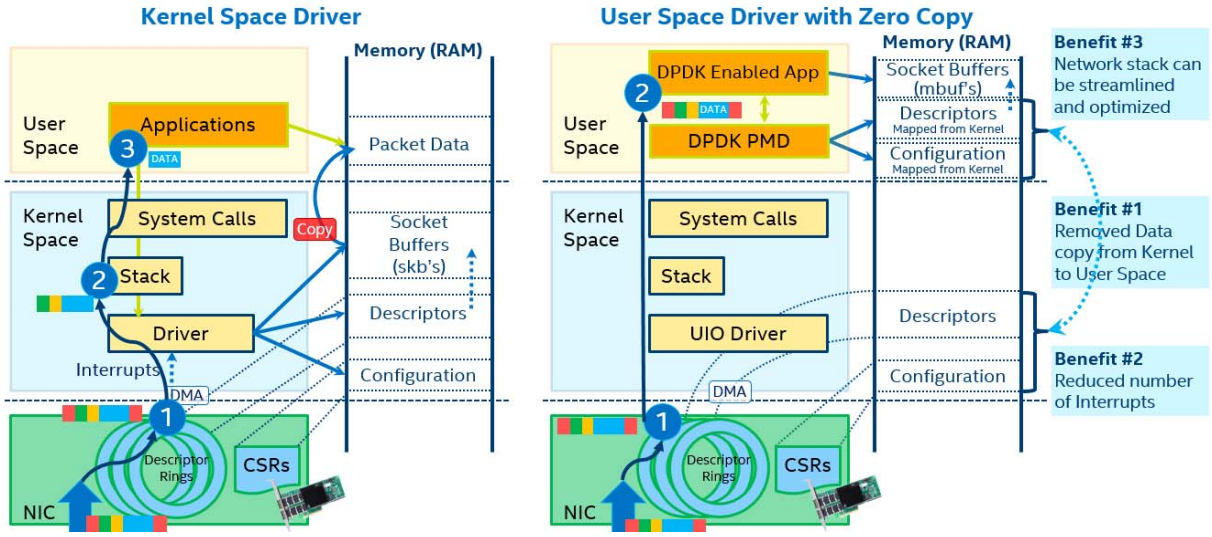

Figure 1. Packet Processing Kernel Space vs. User Space

Key components of DPDK include the following:

- **Environment Abstraction Layer (EAL)** — Provides a generic interface that hides the environment specifics from the application and libraries.
- **Ethernet Poll Mode Driver (PMD)** — Designed to work without asynchronous, interrupt-based signaling mechanisms.
- **Memory management** — Allocates pools of objects in memory created in huge page memory space, uses a ring to store free objects, and spreads objects evenly across DRAM channels to optimize access speed.

- **Buffer management** — Pre-allocates fixed-size buffers that are stored in memory pools, significantly reducing the amount of time spent by the operating system allocating and deallocating buffers.
- **Queue management** — Replaces spinlocks with safe lockless queues, allowing different software components to process packets while avoiding unnecessary wait times.
- **Flow classification** — Improves throughput by implementing Intel® Streaming SIMD Extensions (Intel® SSE) to produce a hash based on tuple information, enabling packets to be placed into flows quickly for processing.
- **Packet Forwarding Algorithm Support** — Includes Hash (librte_hash) and Longest Prefix Match (LPM, librte_lpm) libraries to support the corresponding packet forwarding algorithms.

Refer to dpdk.org for more details.

3.0 DPDK Requirements

3.1 System Requirements

For DPDK system requirements, refer to Section 2 of the following document:

http://doc.dpdk.org/guides/linux_gsg/

3.2 Software Requirements

- Operating System: RHEL 7.4+, Ubuntu 14.04 +, Ubuntu 20.04.1 LTS, SLES 15
- Linux Kernel: 5.0.0-52-generic
- E810 OS Package file: 1.3.20.0
- E810 DDP Comms Package: 1.3.24.0
- E810 *ice* driver: 1.3.2
- E810 *iavf* driver: 4.0.2
- DPDK version: 20.11
- E810 NVM Version: 2.30/2.32
- E810 Firmware Version: 1.5.3.7

3.2.1 Updating the NVM with a DPDK Driver

If all of the following are true:

- You want to update or inventory the device based on the Intel® Ethernet 800 Series.
- You are using the DPDK driver.
- The *ice* device driver is not bound to any port on the device.

Then you must:

1. Bind the kernel driver to the device.
   a. Make sure the *ice* kernel driver is installed.
   b. Use **lspci** to discover the PCI location of the device port you want to update/inventory (in <Bus:Device.Function> format (for example, 04:00.0))

c. Bind the port with the kernel driver:

```
# usertools/dpdk-devbind.py -b <i40e|ice> <B:D.F>
```

2. Run **nvmupdate**.
   - NVM update example:

     ```
     # nvmupdate -u -l -c nvmupdate.cfg
     ```

   - NVM inventory example:

     ```
     # nvmupdate -i -l -c nvmupdate.cfg
     ```

3. Reboot the system.
4. Restore your initial driver configuration by loading the DPDK driver.

   ```
   # usertools/dpdk-devbind.py -b igb_uio <B:D.F>
   ```

4.0 System Under Test (SUT) Installation and Configuration

To run a DPDK application, some customization may be required on the target machine.

For more details, refer Section 2.3 of the following document:

http://doc.dpdk.org/guides/linux_gsg/

Note: The configuration below has been tested on Ubuntu OS with kernel 5.0.0-52-generic.

4.1 Performance BKC

4.1.1 System Configuration

Threads per Core:	2
Cores per Socket:	22
Sockets:	2
NUMA Nodes	2
Vendor ID:	Genuine Intel
CPU Family:	6
Model:	85
Model Name:	Intel® Xeon® Gold 6238 CPU @ 2.10 GHz
Stepping:	7
CPU MHz:	1000.461
CPU Max MHz:	2100.0000
CPU Min MHz:	1000.0000
BogoMIPS:	4200.00
Virtualization:	VT-x
HT:	On
Turbo:	On
Power Management:	Disabled
L1d Cache:	32K
L1i Cache:	32K
L2 Cache:	1024K
L3 Cache:	30976K
NUMA node2 CPUs:	0-21, 44-65
NUMA node1 CPUs:	22-43, 66-87

Note: For best performance, the device must be installed in a PCIe v4.0 x8 or PCIe v3.0 x16 slot.

4.1.2 BIOS Settings

Power Management -> CPU Power and Performance Policy <Performance>

Intel processors have a power management feature where the system goes in power savings mode when it is being underutilized. This feature should be turned off to avoid variance in performance. The system should be configured for maximum performance (BIOS configuration). The downside is that even when the host system is idle, the power consumption is not down.

```
Power and Performance -> CPU Power and Perf Policy -> Performance
Power and Performance -> Workload Configuration -> I/O Sensitive
```

For maximum performance, low-power processor states (C6, C1 enhanced) should be disabled:

```
CPU C-state Disabled
```

P-States, which are designed to optimize power consumption during code execution, should be disabled:

```
CPU P-state Disabled
```

or:

```
Advanced -> Power & Performance -> CPU C State Control -> Package C-State=C0/C1
State
Advanced -> Power & Performance -> CPU C State Control -> C1E=Disabled
```

Turboboost/Speedstep

Speedstep is a CPU feature that dynamically adjusts the frequency of processor to meet processing needs, decreasing the frequency under low CPU-load conditions. Turboboost over-clocks a core when the demand for CPU is high. Turboboost requires that Speedstep is enabled.

These two configurations could introduce a variance in data plane performance when there is a burst of packets. For consistency of behavior, these two features should be disabled.

```
Enhanced Intel® Speedstep® Tech Disabled
Turbo Boost Disabled
```

Virtualization Extensions

Intel virtualization extensions (VT - for VT-x) and VT-d (for direct IO) and DMA remapping (DMAR) must be turned on. VT-d enables IOMMU virtualization capabilities that are required for PCIe pass-through. Also, interrupt remapping should be enabled so that hardware interrupts can be remapped to a VM for PCIe pass-through.

```
Intel VT For directed I/O(VT-d) Enabled
Intel Virtualization Technology (VT-x) Enabled
```

Hyperthreading

Hyperthreading is Intel's simultaneous multi-threading technology For each physical processor core that is present, the operating system addresses two virtual (logical) cores and shares the workload between them when possible. Each logical core shares the resources (L1 and L2 cache, registers) of the physical core. This is controlled by a setting in the BIOS.

In general, data plane performance suffers when hyperthreading is enabled. Therefore, the recommendation is to disable it.

Hyperthreading configuration is a BIOS setting, and changing it requires a reboot.

If hyperthreading is enabled, it is still possible to obtain the same performance as with hyperthreading disabled. To do this, isolate the extra logical cores (see CPU isolation) and do not assign any threads to them.

```
Processor Configuration -> Hyper-threading -> Enabled for Throughput/Disabled for
Latency
```

4.1.3 Hugepages Setup

Hugepage support is required for the large memory pool allocation used for packet buffers. By using hugepage allocations, performance is increased since fewer pages are needed, and therefore less Translation Lookaside Buffers (TLBs, high speed translation caches), which reduce the time it takes to translate a virtual page address to a physical page address. Without hugepages, high TLB miss rates would occur with the standard 4K page size, slowing performance.

For 1 GB pages:

It is not possible to reserve the hugepage memory after the system has booted. The size must be specified explicitly and can also be optionally set as the default hugepage size for the system.

The 1 GB hugepage option can be added in Grub along with IOMMU in kernel command line, as shown in Section 4.1.11.

To reserve 4 GB of hugepage memory in the form of four 1 GB pages, the following options should be passed to the kernel:

```
default_hugepagesz=1G hugepagesz=1G hugepages=4
```

Once the hugepage memory is reserved, to make the memory available for DPDK use, execute the following:

```
mkdir /mnt/huge
mount -t hugetlbfs nodev /mnt/huge
```

For 1 GB pages, the mount point can be made permanent across reboots, by adding the following line to the */etc/fstab* file:

```
nodev /mnt/huge hugetlbfs pagesize=1GB 0 0
```

Note: There are multiple ways to create hugepages under RHEL 7.x, following is one example that shows the steps to create four pages of 1 GB:

1. Create a mounting point for huge pages with auto-mount.

   ```
   cd /mnt
   mkdir huge
   ```

2. Modify */etc/fstab* to include:

   ```
   nodev /mnt/huge hugetlbfs  pagesize=1GB 0 0
   ```

3. Using **grubby** to modify grub directly to setup hugepages.

   ```
   # grubby --default-kernel
   /boot/vmlinuz-3.10.0-957.el7.x86_64
   # grubby --update-kernel=/boot/vmlinuz-3.10.0-957.el7.x86_64 --
   args="intel_iommu=on iommu=pt default_hugepagesz=1G hugepagesz=1G hugepages=4"
   ```

4. Reboot the system and use *dpdk-setup.sh* script to check hugepage allocations. Select option 54 to view the hugepages setup.

```
# $RTE_SDK/usertools/dpdk-setup.sh
Option: 54
AnonHugePages:    153600 kB
HugePages_Total:       4
HugePages_Free:        4
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:    1048576 kB
```

or:

```
# cat /proc/meminfo | grep HugePages_Total
HugePages_Total:       4
# cat /proc/meminfo | grep Hugepagesize
Hugepagesize:    1048576 kB
```

For 2 MB pages:

Hugepages can be allocated after the system has booted. This is done by echoing the number of hugepages required to a *nr_hugepages* file in the */sys/devices/* directory.

For a single-node system, the command to use is as follows (assuming that 1024 pages are required):

```
echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
mount -t hugetlbfs nodev /mnt/huge
```

On a NUMA machine, pages should be allocated explicitly on separate nodes:

```
mkdir -p /mnt/huge
echo 1024 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/
nr_hugepages
echo 1024 > /sys/devices/system/node/node1/hugepages/hugepages-2048kB/
nr_hugepages
mount -t hugetlbfs nodev /mnt/huge
```

4.1.4 IOMMU

In addition, to run the DPDK with Intel® VT-d, the `iommu=pt` kernel parameter must be used when using *igb_uio* driver. This results in pass-through of the DMA Remapping (DMAR) lookup in the host. Also, if *INTEL_IOMMU_DEFAULT_ON* is not set in the kernel, the `intel_iommu=on` kernel parameter must be used as well. This ensures that the Intel IOMMU is being initialized as expected.

Note: While using `iommu=pt` is compulsory for *igb_uio* driver, the vfio-*pci* driver can work with both `iommu=pt` and `iommu=on`.

4.1.5 CPU Isolation

isolcpus is one of the kernel boot parameters that isolates certain CPUs from kernel scheduling, which is especially useful if you want to dedicate some CPUs for special tasks with the least amount or unwanted interruption (but cannot get to 0) in a multi-core system.

4.1.6 RCU Callbacks

To eliminate local timer interrupts, RCU callbacks need to be isolated as well. This is done either in the kernel config, or by the **rcu_nocbs** grub option.

4.1.7 Tickless Kernel

For high-performance applications, using a tickless kernel can result in improved performance. The host kernel must have the cores operating in tickless mode, and the same cores should be dedicated to the application.

The host kernel might have been built with the CONFIG_NO_HZ_FULL_ALL option. If so, tickless operation happens automatically on any core on which the Linux scheduler has only one thread to run. To check for this, look for that string in your Linux kernel config file. This file might be at */boot/<kernel version>* (determine your kernel version with "**uname -a**") or at */proc/config.gz*.

If the kernel was not built with CONFIG_NO_HZ_FULL, it might still be possible to run tickless by configuring it in the grub file (see the Grub File section). Specify the same set of CPUs for both **nohz_full** and **isolcpus**.

4.1.8 vt.handoff

vt.handoff (vt = virtualterminal) is a kernel boot parameter unique to Ubuntu, and is not an upstream kernel boot parameter. Its purpose is to allow the kernel to maintain the current contents of video memory on a virtual terminal. Therefore, when the operating system is booting up, when it moves past the boot loader, **vt.handoff** allows showing of an aubergine background, with Plymouth displaying a logo and progress indicator bar on top of this. Once the display manager comes up, it smoothly replaces this with a login prompt.

4.1.9 NUMA Balancing and MCE

NUMA Balancing inside a kernel automatically optimizes a task scanner for scheduling on the fly, This should be disabled to get a consistent performance during benchmarking.

Also, machine check event exceptions logging is disabled.

4.1.10 Active-State Power Management

Active-State Power Management (ASPM) saves power in the PCIe subsystem by setting a lower power state for PCIe links when the devices to which they connect are not in use. When ASPM is enabled, device latency increases because of the time required to transition the link between different power states. Therefore, ASPM support is disabled here for performance benchmarking.

4.1.11 Kernel Boot Command Line with All the Above System Tuning Parameters

Grub update with iommu, hugepage, isolcpus, nohz_full, rcu_nocbs, vt.handoff, numa_balancing, C-States, P-States, and MCE:

```
vi /etc/default/grub
###
GRUB_CMDLINE_LINUX=" default_hugepagesz=1G hugepagesz=1G hugepages=16 idle=poll
intel_idle.max_cstate=0 pcie_aspm=off intel_iommu=on iommu=pt intel_pstate=disable
processor.max_cstate=1 numa_balancing=disable mce=off isolcpus=$local_cores
nohz_full=$local_cores rcu_nocbs=$local_cores vt.handoff=1"
GRUB_CMDLINE_LINUX_DEFAULT=" default_hugepagesz=1G hugepagesz=1G hugepages=16
idle=poll intel_idle.max_cstate=0 pcie_aspm=off intel_iommu=on iommu=pt
intel_pstate=disable processor.max_cstate=1 numa_balancing=disable mce=off
isolcpus=$local_cores nohz_full=$local_cores rcu_nocbs=$local_cores vt.handoff=1"
###
Update-grub
```

Depending on the system, the **ulimit** command might also be required in Grub:

```
#max locked memory       (kbytes, -l) unlimited
ulimit -l unlimited

#max memory size         (kbytes, -m) unlimited
ulimit -m unlimited
```

Reboot required after grub edit.

Hugepages allocated from the above grub cmd can be verified by:

```
cat /proc/meminfo | grep Huge
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
FileHugePages:         0 kB
HugePages_Total:      16
HugePages_Free:       16
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:    1048576 kB
Hugetlb:        16777216 kB
```

4.1.12 High Performance of Small Packets on 100G NIC- Use 16 Bytes Rx Descriptor Size

ICE PMD supports both 16 and 32 bytes Rx descriptor sizes. The 16 bytes size can provide high performance at small packet sizes. Configuration of CONFIG_RTE_LIBRTE_ICE_16BYTE_RX_DESC in config files can be changed to use 16 bytes size Rx descriptors.

More details to set 16 bytes Rx descriptor size in Section 4.8.1.

4.2 Hardware Packet Generator

For performance benchmarking, Intel recommends a hardware-based packet generator. The traffic parameters for a hardware packet generator like Ixia are as follows:

- Ixia 100 Gigabit Ethernet Traffic Generator
- IxNetwork/IXExlporer: 8.40 EA/ 9.00.1900.15 Patch1
- Acceptable Frame Loss: 0%,
- Traffic Duration: ~20 seconds
- Destination IP: 192.18.0.0 (fixed)
- Source IP: 4096
- Make sure that flow configuration goes to multiple queues to enable RSS.

Note: If you do not have access to a hardware-based packet generator, then a DPDK-based software packet generator, **pktgen**, could be used instead (see Section 5.0). Software-based packet generators like **pktgen** might not be ideal for performance benchmarking (hardware packet generator is recommended), but rather is used for bring-up and basic traffic tests.

4.3 Downloading and Installing Drivers and Modules

Note: This section can be skipped if the DPDK driver is being used instead.

The *ice* driver requires the Dynamic Device Personalization (DDP) package file to enable advanced features, such as dynamic tunneling, Intel® Flow Director, RSS, and ADQ. The driver installation process installs the default DDP package file and creates a soft link *ice.pkg* to the physical package *ice-x.x.x.x.pkg* in the directory. The driver installation process also puts both the driver module and the DDP file in the *initramfs/initrd* image.

1. Download the *ice* driver and uncompress.

```
### download ice-<x.x.x>.tar.gz
tar -xzvf ice-<x.x.x.tar.gz
cd ice-<x.x.x>/src/
```

2. Compile and install the *ice* driver.

```
make -j 8
make install
```

3. Load the *ice* module.

```
 modprobe ice
```

   Note: The Linux driver automatically installs the default DDP package file during driver installation. See the *ice* driver README for general installation and building instructions.

   The DDP package loads during device initialization. The driver looks for *intel/ice/ddp/ice.pkg* in the firmware root (typically */lib/firmware/* or */lib/firmware/updates/*) and checks that it contains a valid DDP package file. The *ice.pkg* file is a symbolic link to the default DDP package file installed by the Linux-firmware software package or the *ice* out-of-tree driver installation.

4. Check to see that the Intel® Ethernet 800 Series Network Adapters all loaded and did not report errors in loading *ice.pkg*.

```
dmesg | grep ice
[1970973.906537] ice: Intel(R) Ethernet Connection E800 Series Linux Driver -
version 1.0.4
[1970973.906539] ice: Copyright (C) 2018-2019, Intel Corporation.
[1970974.253663] ice 0000:3b:00.0: The DDP package was successfully loaded: ICE OS
Default Package version 1.3.13.0
[1970974.302693] ice 0000:3b:00.0 ens1f0: renamed from eth0
[1970974.713958] ice 0000:3b:00.0 ens1f0: device is not ready yet
[1970974.971503] ice 0000:3b:00.0: DCB is enabled in the hardware, max number of
TCs supported on this port are 8
[1970974.971504] ice 0000:3b:00.0: FW LLDP is disabled, DCBx/LLDP in SW mode.
[1970974.971506] ice 0000:3b:00.0: Commit DCB Configuration to the hardware
[1970974.973548] ice 0000:3b:00.0: 126.016 Gb/s available PCIe bandwidth, limited
by 8 GT/s x16 link at 0000:3a:00.0 (capable of 252.048 Gb/s with 16 GT/s x16 link)
[1970975.013680] ice 0000:3b:00.1: DDP package already present on device: ICE OS
Default Package version 1.3.13.0
[1970975.064720] ice 0000:3b:00.1 ens1f1: renamed from eth0
[1970975.098414] ice 0000:3b:00.1: PTP init successful
[1970975.303485] ice 0000:3b:00.1 ens1f1: device is not ready yet
[1970975.489183] ice 0000:3b:00.1: DCB is enabled in the hardware, max number of
TCs supported on this port are 8
[1970975.489184] ice 0000:3b:00.1: FW LLDP is disabled, DCBx/LLDP in SW mode.
[1970975.489186] ice 0000:3b:00.1: Commit DCB Configuration to the hardware
[1970975.491230] ice 0000:3b:00.1: 126.016 Gb/s available PCIe bandwidth, limited
```

```
by 8 GT/s x16 link at 0000:3a:00.0 (capable of 252.048 Gb/s with 16 GT/s x16 link)
```

5. Check to see that the Intel® Ethernet 800 Series Network Adapters all unloaded and did not report errors in loading *package_file_1_0.pkg*.

```
rmmod ice
dmesg | grep ice
#...
#[606237.211534] ice: module unloaded
modprobe ice
```

4.4 Gathering Required Configuration

Gather the required configuration:

```
ifconfig, ethtool -i <ethernet_interface>
cat /sys/class/net/ethernet_interface/device/numa_node,
cat /sys/class/net/ethernet_interface/device/device, and numactl --hardware

##################################
#configuration for pktgen-server
# enp178s0:_device_id=0x1592
# enp178s0:_firmware-version: 1.40 0x80003920 1.2685.0
# enp178s0:_pci_address/bus-info: 0000:b2:00.0
# enp178s0:_mac_address/ether: 68:05:ca:a6:0b:1c
# enp178s0:_numa_node=1
# enp178s0:_ip_address=192.168.103.101
# NUMA node(s):      1
# NUMA node1 CPU(s): 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
43 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87

##################################
#configuration for testpmd-server
# enp24s0:_device_id=0x1592
# enp24s0:_pci_address/bus-info: 0000:18:00.0
# enp24s0:_mac_address/ether: 68:05:ca:a6:0a:b0
# enp24s0:_numa_node=0
# enp24s0:_ip_address=192.168.103.102
# NUMA node(s):      0
# NUMA node0 CPU(s): 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 44 45
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65
```

A sample script in Section 7.1 can also be used to gather required configuration.

Note: "8086:1592" is device ID for the E810.

4.5 Getting the Latest DPDK Code

1. Clone and checkout DPDK 20.11 release from git://dpdk.org/dpdk.

   ```
   mkdir /usr/src/dpdk_latest/
   cd /usr/src/dpdk_latest/
   git clone git://dpdk.org/dpdk
   ```

   or:

   ```
   git clone http://dpdk.org/dpdk
   ##switch to releases branch to checkout version 20.11.0
   git checkout releases
   ```

   or, DPDK can also be downloaded from:

   http://core.dpdk.org/download/ or http://fast.dpdk.org/rel/dpdk-20.11.tar.xz

2. Untar DPDK, if download version is *dpdk-20.11.tar.xz* tarball.

   ```
   tar xf dpdk-20.11.tar.xz
   ```

DPDK is composed of several directories:

- ***lib*** — Source code of DPDK libraries.
- ***drivers*** — Source code of DPDK poll-mode drivers.
- ***app*** — Source code of DPDK applications (automatic tests).
- ***examples*** — Source code of DPDK application examples.
- ***config***, ***buildtools***, ***mk*** — Framework-related makefiles, scripts, and configuration.

4.6 DPDK Target Environment Directory

1. Create a DPDK target environment directory (such *asx86_64-native-linuxapp-gcc*). It contains all libraries and header files required to build an application.

   When compiling an application in the Linux environment on the DPDK, the following variables must be exported:

- **RTE_SDK** — Points to the DPDK installation directory.
- **RTE_TARGET** — Points to the DPDK target environment directory.

  For example:

  ```
  export RTE_SDK=/usr/src/dpdk_latest/dpdk-20.11
  export RTE_TARGET=x86_64-native-linuxapp-gcc
  export DESTDIR=install
  cd $RTE_SDK
  ```

4.7 Prerequisite Library

NUMA:

NUMA is required by most modern machines, but not needed for non-NUMA architectures.

Note: For compiling the NUMA lib, run **libtool -version** to ensure that the **libtool** version is greater than or equal to 2.2. Otherwise, the compilation will fail with errors.

For Ubuntu:

```
sudo apt-get install libnuma-dev
```

For RHEL:

```
yum install numactl-devel
```

or:

```
git clone https://github.com/numactl/numactl.git
cd numactl
./autogen.sh
./configure
make install
```

The NUMA header files and lib file are generated in the *include* and *lib* folders, respectively, under <*numa install dir*>.

PYTHON:

For Ubuntu:

```
sudo apt-get install libnuma-dev
apt-get install python3
```

Create a symlink to it: `sudo ln -s /usr/bin/python3 /usr/bin/python`

MESON:

The MESON tool is required to configure the DPDK build.

For Ubuntu:

```
sudo apt-get install meson
```

If MESON is not available as a suitable package, it can also be installed using the Python 3 pip tool:

```
sudo apt install python3-pip
pip3 install meson
```

Note: pip3 puts the executable under */usr/local/lib/python3.6/dist-packages*, so put that directory in your *PATH*.

For example:

PATH="/usr/local/lib/python3.6/dist-packages:$PATH"

Note: If the following error is seen: "Requires >=0.47.1 but the version of Meson is 0.45.1"

First, remove meson installed by **apt** if installed:

```
sudo apt purge meson -y
```

Then, install via pip3:

```
sudo pip3 install meson
```

Then, make a symlink in bin folder from local bin:

```
sudo ln -s /usr/local/bin/meson /usr/bin/meson
```

Now, check:

```
which meson && meson --version
```

4.8 Installing DPDK

1. Install DPDK.
   a. Do one of the following:
      - Install DPDK Target Environments:

        Install and make targets using the **make install T=**<*target*> command in the top-level DPDK directory.

        ```
        make -j 24 install T=$RTE_TARGET DESTDIR=install
        cp -rf /usr/src/dpdk_latest/dpdk-20.11/lib/lib* /usr/lib/
        ###(if /usr/lib contains older version of DPDK's lib.)
        rsync -a /usr/src/dpdk_latest/dpdk-20.11/lib/ /usr/lib/
        export DPDK_BUILD_DIR=x86_64-native-linuxapp-gcc
        ```

      - Install DPDK using the MESON build system:

        Run the following set of commands from the top-level DPDK directory:

        ```
        meson build
        cd build
        ```

        Run `meson configure` to include kmods:

        ```
        meson configure -Denable_kmods=true
        ```

        ```
        ninja
        sudo ninja install
        ldconfig
        ```

      The DPDK target can also be setup from the *dpdk-setup.sh* script found in the *usertools* subdirectory.

      ```
      cd /usr/src/dpdk_latest/dpdk-20.11/usertools
      ./dpdk-setup.sh
      ```

   b. From Step 1a, select the DPDK environment to build, enter option 36 (x86_64-native-linuxapp-gcc).
   c. Refer to the following link for more details on the *dpdk-setup* script:

      https://doc.dpdk.org/guides/linux_gsg/quick_start.html

2. Create a symbolic link of the ICE DDP package file that was in the */lib/firmware/* directory.

   Notes: The location for the ICE DDP package file for DPDK is:

   /lib/firmware/updates/intel/ice/ddp/ice.pkg

   or

   /lib/firmware/intel/ice/ddp/ice.pkg

   If this file exists, DPDK PMD downloads it into the device. If not, the DPDK PMD goes into Safe Mode where it uses the configuration contained in the device's NVM. This is NOT a supported configuration and many advanced features will not be functional.

   If directory */ice/ddp* does not exist under */lib/firmware/updates/intel/*, create the */ice/ddp* directory before proceeding:

   ```
   cd /lib/firmware/update/intel
   mkdir ice
   cd ice
   mkdir ddp
   ```

3. Copy the renamed DDP package file to */lib/firmware/updates/intel/ice/ddp/*.

   ```
   cp /usr/src/ice-0.x.x/ddp/ice-1.3.20.0.pkg /lib/firmware/updates/intel/ice/ddp/
   ice.pkg
   ```

   You can also install specific DDP package files for different physical devices in the same system:

   For a specific NIC, the DDP package supposed to be loaded can have a filename: *ice-xxxxxx.pkg*, where '*xxxxxx*' is the 64-bit PCIe Device Serial Number of the NIC. For example, if the NIC's device serial number is 00-CC-BB-FF-FF-AA-05-68, the device-specific DDP package filename is i*ce-00ccbbffffaa0568.pkg* (in hex and all low case).

   During initialization, the driver searches in the following paths in order: */lib/firmware/updates/intel/ice/ddp* and */lib/firmware/intel/ice/ddp*. The corresponding device specific DDP package is downloaded first if the file exists. If not, the driver tries to load the default package. The type of loaded package is stored in *ice_adapter->active_pkg_type*. A symbolic link to the DDP package file is also okay. The same package file is used by both the kernel driver and the DPDK PMD.

4.8.1 Setting 16 Bytes Rx Descriptor Size

For better small packet size performance, 16 bytes Rx descriptor can be set from the top-level DPDK directory:

```
cd x86_64-native-linuxapp-gcc/
vi.config
```

Or, From root dpdk directory:

```
vi config/common_base
```

Under the `# Compile burst-oriented ICE PMD driver` section, change `CONFIG_RTE_LIBRTE_ICE_16BYTE_RX_DESC=n` to `CONFIG_RTE_LIBRTE_ICE_16BYTE_RX_DESC=y`, then save and recompile

```
make -j 24 install T=$RTE_TARGET DESTDIR=install
```

4.9 Linux Drivers

UIO is a small kernel module to set up the device, map device memory to user space, and register interrupts.

DPDK includes the kernel **igb_uio** module, which deals with PCI enumeration and handles links status interrupts in user mode, instead of being handled by kernel.

Since DPDK release 1.7 onward provides VFIO support, use of UIO is optional for platforms that support using VFIO.VFIO. This is a more robust and secure driver compared to UIO, relying on IOMMU protection. To make use of **VFIO**, the *vfio-pci* module must be loaded.

4.9.1 Loading *igb_uio* or *vfio-pci* Modules

Load *igb_uio* or *vfio-pci*. Do one of the following:

1. Starting with DPDK 20.11, *igb_uio* module is no longer included with the build. So, *igb_uio* module can be found in the repository *dpdk-kmods*: http://git.dpdk.org/dpdk-kmods/

   Clone:

   ```
   git://dpdk.org/dpdk-kmods
   http://dpdk.org/git/dpdk-kmods
   ```

2. Download and unzip:

   ```
   /usr/src/dpdk_latest/dpdk-20.11-rc3# unzip dpdk-kmods-
   3d231f1bbfa971cdcd723291f709e08fe1041e67.zip
   ```

3. Compile and insert *igb_uio* module:

   ```
   root@paeuser-S2600WFT:/usr/src/dpdk_latest/dpdk-20.11-rc3/dpdk-20.11-rc3/dpdk-
   kmods-3d231f1bbfa971cdcd723291f709e08fe1041e67/linux/igb_uio# make
   sudo modprobe uio
   sudo insmod igb_uio.ko
   ```

   or:

   ```
   modprobe vfio-pci
   ```

   Notes: The **igb_uio** or **VFIO** modules can also be inserted from the *dpdk-setup.sh* script found in the *usertools* subdirectory by selecting options 43 and 44, respectively.

   Refer to the following link for more details on using this script:

   https://doc.dpdk.org/guides/linux_gsg/quick_start.html

4.9.2 Binding and Unbinding Network Ports

1. Bind the ports to the *igb_uio* or *vfio-pci* module for DPDK use.

   ```
   ### Use dpdk-devbind -s to show the available devices and drivers
   install -Dvm 755 /$RTE_SDK/install/sbin/dpdk-devbind /sbin/dpdk-devbind

   ### Check to see if igb_uio is available and that the device is not marked as
   ***Active
   dpdk-devbind -s
   ```

   or:

   ```
   ./usertools/dpdk-devbind.py --status
   ```

2. If the device is in use, bring down the interface and unbind the interface.

```
ifdown interface
dpdk-devbind -u 18:00.0

### Use dpdk-devbind -b to bind the device to igb_uio or back to the kernel ice
driver

### 18:00.0 interface is used with Testpmd
/$RTE_SDK/install/sbin/dpdk-devbind -b igb_uio 18:00.0

### b2:00.0 interface is used with Pktgen
/$RTE_SDK/install/sbin/dpdk-devbind -b igb_uio b2:00.0
```

Note: The ports can also be bind to **igb_uio** or **VFIO** from the *dpdk-setup.sh* script found in the *usertools* subdirectory by selecting options 49 and 50, respectively.

Refer to the following link for more details on using this script:

https://doc.dpdk.org/guides/linux_gsg/quick_start.html

5.0 Test Applications (pktgen and testpmd)

The **pktgen** application is traffic generator powered by DPDK. It is capable of generating traffic at wire rate. For more information on the **pktgen** application, see:

https://pktgen-dpdk.readthedocs.io/en/latest/

testpmd is another reference application distributed with the DPDK package. Its main purpose is to forward packets between Ethernet ports on a network interface. The **testpmd** application provides a number of different throughput tests and access NIC hardware features, such as Intel® Flow Director. For more information on the testpmd application, see:

https://doc.dpdk.org/guides/testpmd_app_ug/index.html

5.1 Getting Source and Destination MAC Addresses

1. Get source and destination MAC Addresses to be used for **testpmd** and **pktgen** configurations.

```
# pktgen server
#src mac port 0 (b2:00.0) 68:05:ca:a6:0b:1c
# testpmd server
#dst mac port 0 (18:00.0) 68:05:ca:a6:0a:b0
```

5.2 Setting Up pktgen Server

1. Get the latest **pktgen** application from dev branch to the DPDK root directory.

```
git clone git://dpdk.org/apps/pktgen-dpdk
```

   Or, download from:

   http://git.dpdk.org/apps/pktgen-dpdk/log/?h=dev

2. Set the required environmental variable, compile target, and build application.

```
export RTE_SDK=/usr/src/dpdk_latest/dpdk-20.11
export RTE_TARGET=x86_64-native-linuxapp-gcc
export DESTDIR=install
cd $RTE_SDK
```

3. Lua is used in **pktgen** to script and configure the application and also to plug into DPDK functions to expose configuration and statistics.

   For Ubuntu:

```
sudo apt-get install liblua5.3-0 liblua5.3-dev libpcap-dev libbsd-dev
```

   For RHEL:

   Download *lua-5.3.5.tar.gz* from https://www.lua.org/download.html, then copy it to a peer system like */usr/local/src/* and extract it:

```
# tar zxf lua-5.3.5.tar.gz
```

   Then follow steps below:

   a. Export the C_INCLUDE_PATH variable.

```
# export C_INCLUDE_PATH=/usr/local/src/lua-5.3.5/src
```

   Note: The directory to which you extracted *lua-5.3.5.tar.gz*.

b. Add the following configuration settings to the */usr/lib64/pkgconfig/lua-5.3.pc* file, where V=5.3 and R=5.3.5.

```
prefix= /usr
exec_prefix=${prefix}
libdir= /usr/lib64
includedir=${prefix}/include
Name: Lua
Description: An Extensible Extension Language
Version: ${R}
Requires:
Libs: -llua -lm -ldl
Cflags: -I${includedir}/lua-${V}
```

c. Add the following configuration settings to the */usr/lib64/pkgconfig/lua5.3.pc* file, where V=5.3 and R=5.3.5

```
prefix=/usr
INSTALL_BIN=${prefix}/bin
INSTALL_INC=${prefix}/include
INSTALL_LIB=${prefix}/lib
INSTALL_MAN=${prefix}/share/man/man1
INSTALL_LMOD=${prefix}/share/lua/${V}
INSTALL_CMOD=${prefix}/lib/lua/${V}
exec_prefix=${prefix}
libdir=${exec_prefix}/lib
includedir=${prefix}/include
Name: Lua
Description: An Extensible Extension Language
Version: ${R}
Requires:
Libs: -L${libdir} -llua -lm -ldl
Cflags: -I${includedir}
```

d. Then run the make command:

```
# cd lua-5.3.5/
# make linux install
```

e. Download *libpcap-devel* from https://centos.pkgs.org/7/centos-x86_64/libpcap-devel-1.5.3-11.el7.x86_64.rpm.html and install it:

```
# rpm -ivh libpcap-devel*
```

4. Install **pktgen**. Do one of the following:

- If git clone is used to download pktgen:

```
cd $RTE_SDK/pktgen-dpdk
git co dev
make rebuild
```

- If pktgen is downloaded from http://git.dpdk.org/apps/pktgen-dpdk/log/?h=dev:

1. Extract Pktgen:

```
root@paeuser-PE-R740xd:/usr/src/dpdk_latest/dpdk-20.11/pktgen# unzip pktgen-dpdk-254cf80fc417192655a2b79d01705df8850a3afd.zip -d pktgen_latest/
```

2. Make:

```
root@paeuser-PE-R740xd:/usr/src/dpdk_latest/dpdk-20.11/pktgen/pktgen_latest/
pktgen-dpdk-254cf80fc417192655a2b79d01705df8850a3afd# make
```

3. Pktgen application is now under pktgen root */usr/local/bin* directory:

```
root@paeuser-PE-R740xd:/usr/src/dpdk_latest/dpdk-20.11/pktgen/pktgen_latest/
pktgen-dpdk-254cf80fc417192655a2b79d01705df8850a3afd/usr/local/bin# ./pktgen
```

5.3 Setting Up testpmd Server

1. Set the required environmental variable, compile target, and build application.

```
export RTE_SDK=/usr/src/dpdk_latest/dpdk-20.11
export RTE_TARGET=x86_64-native-linuxapp-gcc
export DESTDIR=install
cd $RTE_SDK
```

2. Build and run **testpmd**. Do one of the following:

   - If DPDK is installed using meson build system, as in Step 1a in Section 4.8:

     This will compile up some DPDK libs, the PMD and testpmd and install them in */usr/local/*. (On Fedora you will need to add */usr/local/lib64* to your ld path, it's not there by default.)

     Then, you can run testpmd as, for example.

     ```
     sudo /usr/local/bin/dpdk-testpmd -l 2-11 -n 4 -w 18:00.0 --file-prefix
     testpmd18000 --socket-mem=1024,0 --proc-type=auto -- --nb- cores=4 --rxq=4 --
     txq=4 -i --forward-mode=mac --eth-peer=0,68:05:ca:a6:0b:1c
     ```

     Or:

     ```
     sudo /usr/local/bin/dpdk-testpmd -l 2-11 -n 4 -w 18:00.0 --socket-mem=1024,0 -
     - -i --forward-mode=mac -- nb-cores=4 --rxq=4 --txq=4
     ```

   - Otherwise, if make install is being used:

     ```
     cd $RTE_SDK/app/test-pmd
     make

     /usr/src/dpdk_latest/dpdk-next-net-intel/app/test-pmd# ./testpmd -l 2-11 -n 4
     -w 18:00.0 --file-prefix testpmd18000 --socket-mem=1024,0 --proc-type=auto --
     --nb-cores=4 --rxq=4 --txq=4 -i --forward-mode=mac --eth-
     peer=0,68:05:ca:a6:0b:1c
     ```

     or:

     ```
     ./testpmd -l 2-11 -n 4 -w 18:00.0 --socket-mem=1024,0 -- -i --forward-mode=mac
     --nb-cores=4 --rxq=4 --txq=4
     ```

Note: The Ethernet interface 18:00.0 used in **testpmd** is on NUMA Node 0. Refer to Section 4.4 to gather numa_node info to appropriately position cores with the Ethernet interface.

5.4 Example pktgen Configuration

1. Example **pktgen** configuration command:

```
cd $RTE_SDK
root@paeuser-PE-R740xd:/usr/src/dpdk_latest/dpdk-20.11/pktgen/pktgen_latest/
pktgen-dpdk-254cf80fc417192655a2b79d01705df8850a3afd/usr/local/bin# sudo -E ./
pktgen -l 24-32 -n 4 --proc-type auto --log-level 7 --socket- mem=0,1024 --file-
prefix pgb2000 -w b2:00.0 -- -N -P -T -m [25-28:29-32].0
```

Notes:

- A script to configure the above command is available in Section 7.2.
- The Ethernet interface b2:00.0 used in **pktgen** is on NUMA Node 1, Refer to Section 4.4 to gather numa_node info to appropriately position cores with the Ethernet interface.

2. **Pktgen** entropy configuration to enable multiple traffic streams with RSS:

```
Pktgen:/> set 0 proto tcp
Pktgen:/> set 0 size 128
Pktgen:/> set 0 src mac 68:05:ca:a6:0b:1c
Pktgen:/> set 0 dst mac 68:05:ca:a6:0a:b0
Pktgen:/> set 0 src ip 192.168.103.101/24
Pktgen:/> set 0 dst ip 192.168.103.102
Pktgen:/> enable 0 range
Pktgen:/> range 0 size 128 128 128 1
Pktgen:/> range 0 src mac 68:05:ca:a6:0b:1c 68:05:ca:a6:0b:1c 68:05:ca:a6:0b:1c
00:00:00:00:00:01
Pktgen:/> range 0 dst mac 68:05:ca:a6:0a:b0 68:05:ca:a6:0a:b0 68:05:ca:a6:0a:b0
00:00:00:00:00:01
Pktgen:/> range 0 src ip 192.168.103.101 192.168.103.101 192.168.103.101 0.0.0.1
Pktgen:/> range 0 dst ip 192.168.103.102 192.168.103.102 192.168.103.102 0.0.0.1
Pktgen:/> range 0 dst port 2000 2000 2000 1
Pktgen:/> range 0 src port 3000 3000 3016 1
Pktgen:/> start 0
```

3. Example of **pktgen** screen with traffic running to **testpmd** server:

```
Copyright (c) <2010-2019>, Intel Corporation. All rights reserved. Powered by DPDK
EAL: Detected 88 lcore(s)
EAL: Detected 2 NUMA nodes
EAL: Auto-detected process type: PRIMARY
EAL: Multi-process socket /var/run/dpdk/pgb2000/mp_socket
EAL: Selected IOVA mode 'PA'
EAL: Probing VFIO support...
EAL: PCI device 0000:b2:00.0 on NUMA socket 1
EAL:   probe driver: 8086:1592 net_ice

Lua 5.3.3  Copyright (C) 1994-2016 Lua.org, PUC-Rio

*** Copyright (c) <2010-2019>, Intel Corporation. All rights reserved.
*** Pktgen created by: Keith Wiles -- >>> Powered by DPDK <<<

 Port: Name         IfIndex Alias        NUMA  PCI
    0: net_ice         0                   1   8086:1592/b2:00.0

Initialize Port 0 -- TxQ 4, RxQ 4, Src MAC 68:05:ca:a6:0b:1c <Promiscuous mode
Enabled>
```

```
- Ports 0-0 of 1   <Main Page>  Copyright (c) <2010-2019>, Intel Corporation
  Flags:Port        : P------Range       :0
Link State          :        <UP-100000-FD>      ---Total Rate---
Pkts/s Max/Rx       :   4314304803/33489470   4314304803/33489470
       Max/Tx       :     43492105/35677250     43492105/35677250
MBits/s Rx/Tx       :           39651/42241           39651/42241
Broadcast           :                     0
Multicast           :                     0
Sizes 64            :                     0
      65-127        :                     0
      128-255       :          172481371830
      256-511       :                     0
      512-1023      :                     0
      1024-1518     :                     0
Runts/Jumbos        :                   0/0
ARP/ICMP Pkts       :                   0/0
Errors Rx/Tx        :                   0/0
Total Rx Pkts       :          185206642765
      Tx Pkts       :          206415058038
      Rx MBs        :              29705123
      Tx MBs        :              33289196
                    :
Pattern Type        :                 abcd...
Tx Count/% Rate     :         Forever /100%
Pkt Size/Tx Burst   :            128 /   64
TTL/Port Src/Dest   :         4/ 1234/ 5678
Pkt Type:VLAN ID    :       IPv4 / TCP:0001
802.1p CoS/DSCP/IPP :             0/  0/  0
VxLAN Flg/Grp/vid   :      0000/    0/    0
IP  Destination     :       192.168.103.102
    Source          :    192.168.103.101/24
MAC Destination     :     68:05:ca:a6:0a:b0
    Source          :     68:05:ca:a6:0b:1c
PCI Vendor/Addr     :     8086:1592/b2:00.0

-- Pktgen 19.08.0 (DPDK 19.11.0-rc0)  Powered by DPDK  (pid:85897) ------------
Pktgen:/> stop 0
Pktgen:/> set 0 proto tcp
Pktgen:/> set 0 size 128
Pktgen:/> set 0 src mac 68:05:ca:a6:0b:1c
Pktgen:/> set 0 dst mac 68:05:ca:a6:0a:b0
Pktgen:/> set 0 src ip 192.168.103.101/24
Pktgen:/> set 0 dst ip 192.168.103.102
Pktgen:/> enable 0 range
Pktgen:/> range 0 size 128 128 128 1
Pktgen:/> range 0 src mac 68:05:ca:a6:0b:1c 68:05:ca:a6:0b:1c 68:05:ca:a6:0b:1c
00:00:00:00:00:01
Pktgen:/> range 0 dst mac 68:05:ca:a6:0a:b0 68:05:ca:a6:0a:b0 68:05:ca:a6:0a:b0
00:00:00:00:00:01
Pktgen:/> range 0 src ip 192.168.103.101 192.168.103.101 192.168.103.101 0.0.0.1
Pktgen:/> range 0 dst ip 192.168.103.102 192.168.103.102 192.168.103.102 0.0.0.1
Pktgen:/> range 0 dst port 2000 2001 2002 1
Pktgen:/> range 0 src port 3000 3001 3002 1
Pktgen:/> start 0
Pktgen:/>
```

5.5 Example testpmd Configuration

```
/usr/src/dpdk_latest/dpdk-20.11# sudo /usr/local/bin/dpdk-testpmd -l 2-11 -n 4 -w
18:00.0 --file-prefix testpmd18000 --socket-mem=1024,0 --proc-type=auto -- --nb-
cores=4 --rxq=4 --txq=4 -i --forward-mode=mac --eth-peer=0,68:05:ca:a6:0b:1c

EAL: Detected 88 lcore(s)
EAL: Detected 2 NUMA nodes
EAL: Auto-detected process type: PRIMARY
EAL: Multi-process socket /var/run/dpdk/testpmd18000/mp_socket
EAL: Selected IOVA mode 'PA'
EAL: Probing VFIO support...
EAL: PCI device 0000:18:00.0 on NUMA socket 0
EAL:   probe driver: 8086:1592 net_ice
Interactive-mode selected
Set mac packet forwarding mode
testpmd: create a new mbuf pool <mbuf_pool_socket_0>: n=219456, size=2176, socket=0
testpmd: preferred mempool ops selected: ring_mp_mc

Warning! port-topology=paired and odd forward ports number, the last port will pair
with itself.

Configuring Port 0 (socket 0)
Port 0: 68:05:CA:A6:0A:B0
Checking link statuses...
Done

testpmd> start
mac packet forwarding - ports=1 - cores=4 - streams=4 - NUMA support enabled, MP
allocation mode: native
Logical Core 3 (socket 0) forwards packets on 1 streams:
  RX P=0/Q=0 (socket 0) -> TX P=0/Q=0 (socket 0) peer=68:05:CA:A6:0B:1C
Logical Core 4 (socket 0) forwards packets on 1 streams:
  RX P=0/Q=1 (socket 0) -> TX P=0/Q=1 (socket 0) peer=68:05:CA:A6:0B:1C
Logical Core 5 (socket 0) forwards packets on 1 streams:
  RX P=0/Q=2 (socket 0) -> TX P=0/Q=2 (socket 0) peer=68:05:CA:A6:0B:1C
Logical Core 6 (socket 0) forwards packets on 1 streams:
  RX P=0/Q=3 (socket 0) -> TX P=0/Q=3 (socket 0) peer=68:05:CA:A6:0B:1C

mac packet forwarding packets/burst=32
nb forwarding cores=4 - nb forwarding ports=1
port 0: RX queue number: 4 Tx queue number: 4
Rx offloads=0x0 Tx offloads=0x10000
RX queue: 0
RX desc=1024 - RX free threshold=32
RX threshold registers: pthresh=8 hthresh=8  wthresh=0
RX Offloads=0x0
TX queue: 0
TX desc=1024 - TX free threshold=32
TX threshold registers: pthresh=32 hthresh=0  wthresh=0
TX offloads=0x10000 - TX RS bit threshold=32

testpmd> show port stats all

  ######################## NIC statistics for port 0  ########################
  RX-packets: 322932478810 RX-missed: 558861443  RX-bytes:  18446743441590657960
  RX-errors:  0
```

```
  RX-nombuf:  0
  TX-packets: 302564215177 TX-errors: 0          TX-bytes:  18446743108765253980

  Throughput (since last show)
  Rx-pps:     35183210
  Tx-pps:     32640093
  ############################################################################

testpmd> stop
Telling cores to stop...
Waiting for lcores to finish...

  ------- Forward Stats for RX Port= 0/Queue= 0 -> TX Port= 0/Queue= 0 -------
  RX-packets: 155985410964   TX-packets: 146417324571   TX-dropped: 9568086393

  ------- Forward Stats for RX Port= 0/Queue= 1 -> TX Port= 0/Queue= 1 -------
  RX-packets: 6224427006     TX-packets: 4972240065     TX-dropped: 1252186941

  ------- Forward Stats for RX Port= 0/Queue= 2 -> TX Port= 0/Queue= 2 -------
  RX-packets: 4981979331     TX-packets: 4973665526     TX-dropped: 8313805

  ------- Forward Stats for RX Port= 0/Queue= 3 -> TX Port= 0/Queue= 3 -------
  RX-packets: 155943833856   TX-packets: 146389491787   TX-dropped: 9554342069

  ---------------------- Forward statistics for port 0  ----------------------
  RX-packets: 323135651965   RX-dropped: 561082081     RX-total: 323696734046
  TX-packets: 302752721949   TX-dropped: 20382929208   TX-total: 323135651157
  ----------------------------------------------------------------------------

  ++++++++++++++++ Accumulated forward statistics for all ports++++++++++++++++
  RX-packets: 323135651965   RX-dropped: 561082081     RX-total: 323696734046
  TX-packets: 302752721949   TX-dropped: 20382929208   TX-total: 323135651157
  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Done.
```

5.6 Running a Sample Application L3fwd Server

Following is a Layer 3 forwarding application in which the forwarding decision is based on information read from the input packet. For more information, see:

https://doc.dpdk.org/guides/sample_app_ug/l3_forward.html

1. Set the required environmental variable, compile target, and build application.

```
### l3fwd server
export RTE_SDK=/usr/src/dpdk_latest/dpdk-20.11
export RTE_TARGET=x86_64-native-linuxapp-gcc
export DESTDIR=install
cd $RTE_SDK

cd $RTE_SDK/examples/l3fwd
```

2. Build the application.

```
make -j 8
cd $RTE_SDK
```

4 cores command line option:

```
/usr/src/dpdk_latest/dpdk-next-net-intel/examples/l3fwd/build# ./l3fwd -l
22,24,26,28 -n 6 -- -p 0x0 --config="(0,0,22),(0,1,24),(0,2,26),(0,3,28)"
```

or:

```
./l3fwd/build/l3fwd -l 3-6 -n 4 -w 18:00.0 -- -p 0x1 --parse-ptype --config
"(0,0,3),(0,1,4),(0,2,5),(0,3,6)" -P
```

2 cores command line option:

```
./l3fwd/build/l3fwd -l 3-4 -n 4 -w 18:00.0 -- -p 0x1 --parse-ptype --config
"(0,0,3),(0,1,4)" -P
```

1 core command line option:

```
./l3fwd/build/l3fwd -l 3 -n 4 -w 18:00.0 -- -p 0x1 --parse-ptype --config
"(0,0,3)" -P
```

Note: The Ethernet interface used in **L3fwd** is on NUMA Node 0. Refer to Section 4.4 to gather numa_node info to appropriately position cores with the Ethernet interface.

In this command:

- The **-l** option enables cores 22,24,26,28.
- The **-p** option enables port 0,
- The **-config** option enables one queue on each port and maps each (port,queue) pair to a specific core.

3. Example **l3fwd** output

```
/usr/src/dpdk_latest/dpdk-next-net-intel/examples/l3fwd/build# ./l3fwd -l
22,24,26,28 -n 6 -- -p 0x0 --config="(0,0,22),(0,1,24),(0,2,26),(0,3,28)"
EAL: Detected 88 lcore(s)
EAL: Detected 2 NUMA nodes
EAL: Multi-process socket /var/run/dpdk/rte/mp_socket
EAL: Selected IOVA mode 'PA'
EAL: Probing VFIO support...
EAL: PCI device 0000:00:04.0 on NUMA socket 0
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:00:04.1 on NUMA socket 0
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:00:04.2 on NUMA socket 0
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:00:04.3 on NUMA socket 0
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:00:04.4 on NUMA socket 0
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:00:04.5 on NUMA socket 0
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:00:04.6 on NUMA socket 0
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:00:04.7 on NUMA socket 0
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:3d:00.0 on NUMA socket 0
EAL:   probe driver: 8086:37d2 net_i40e
EAL: PCI device 0000:3d:00.1 on NUMA socket 0
EAL:   probe driver: 8086:37d2 net_i40e
```

```
EAL: PCI device 0000:80:04.0 on NUMA socket 1
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:80:04.1 on NUMA socket 1
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:80:04.2 on NUMA socket 1
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:80:04.3 on NUMA socket 1
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:80:04.4 on NUMA socket 1
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:80:04.5 on NUMA socket 1
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:80:04.6 on NUMA socket 1
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:80:04.7 on NUMA socket 1
EAL:   probe driver: 8086:2021 rawdev_ioat
EAL: PCI device 0000:86:00.0 on NUMA socket 1
EAL:   probe driver: 8086:10c9 net_e1000_igb
EAL: PCI device 0000:86:00.1 on NUMA socket 1
EAL:   probe driver: 8086:10c9 net_e1000_igb
EAL: PCI device 0000:af:00.0 on NUMA socket 1
EAL:   probe driver: 8086:1592 net_ice
ice_load_pkg_type(): Active package is: 1.3.4.0, ICE OS Default Package
ice_init_proto_xtr(): Protocol extraction is not supported
LPM or EM none selected, default LPM on
Initializing port 0 ... Creating queues: nb_rxq=4 nb_txq=4... Port 0 modified RSS
hash function based on hardware support,requested:0xa38c configured:0x2288
 Address:68:05:CA:5C:CF:A8, Destination:02:00:00:00:00:00, Allocated mbuf pool on
socket 1
LPM: Adding route 192.18.0.0 / 24 (0)
LPM: Adding route 192.18.1.0 / 24 (1)
LPM: Adding route 192.18.2.0 / 24 (2)
LPM: Adding route 192.18.3.0 / 24 (3)
LPM: Adding route 192.18.4.0 / 24 (4)
LPM: Adding route 192.18.5.0 / 24 (5)
LPM: Adding route 192.18.6.0 / 24 (6)
LPM: Adding route 192.18.7.0 / 24 (7)
LPM: Adding route 2001:200:: / 48 (0)
LPM: Adding route 2001:200:0:0:1:: / 48 (1)
LPM: Adding route 2001:200:0:0:2:: / 48 (2)
LPM: Adding route 2001:200:0:0:3:: / 48 (3)
LPM: Adding route 2001:200:0:0:4:: / 48 (4)
LPM: Adding route 2001:200:0:0:5:: / 48 (5)
LPM: Adding route 2001:200:0:0:6:: / 48 (6)
LPM: Adding route 2001:200:0:0:7:: / 48 (7)
txq=22,0,1 txq=24,1,1 txq=26,2,1 txq=28,3,1

Initializing rx queues on lcore 22 ... rxq=0,0,1
Initializing rx queues on lcore 24 ... rxq=0,1,1
Initializing rx queues on lcore 26 ... rxq=0,2,1
Initializing rx queues on lcore 28 ... rxq=0,3,1

Checking link
status.....................................................................
...............done
Port 0 Link Down
```

```
L3FWD: entering main loop on lcore 24
L3FWD:  -- lcoreid=24 portid=0 rxqueueid=1
L3FWD: entering main loop on lcore 26
L3FWD:  -- lcoreid=26 portid=0 rxqueueid=2
L3FWD: entering main loop on lcore 22
L3FWD:  -- lcoreid=22 portid=0 rxqueueid=0
L3FWD: entering main loop on lcore 28
L3FWD:  -- lcoreid=28 portid=0 rxqueueid=3
```

6.0 Example Virtual Function Setup with DPDK

Following is an example setup and configuration of *dpdk-20.11 iavf* PMD on an E810 VF with the 4.0.1:

Note: This example assumes that the PF interface is enp175s0f0.

1. Confirm **iommu** (or Intel VT-d) is enabled in the BIOS.

   Note: **iommu** enables mapping of virtual memory addresses to physical addresses. In vi */etc/default/grub* file, turn on **iommu** and set it to pass-through mode:

   ```
   intel_iommu=on
   iommu=pt
   ```

   Be sure to update grub file, then reboot.

1. Download the kernel *iavf* driver and uncompress.

   ```
   tar xzvf iavf-4.0.1
   cd iavf-4.0.1/src/
   ```

2. Compile and install the *iavf* driver.

   ```
   make -j 8
   make install
   modprobe iavf
   ```

3. Create four VFs on the interface.

   ```
   echo 4 > /sys/class/net/enp24s0/device/sriov_numvfs
   ```

4. Bring up the PF interface.

   ```
   ip link set up enp24s0
   ```

5. Add MAC Addresses to the VFs. (This step is optional.)

   ```
   ip link set enp24s0v0 vf 0 mac 68:05:ca:a6:0a:b1
   ip link set enp24s0v1 vf 1 mac 68:05:ca:a6:0a:b2
   ip link set enp24s0v2 vf 2 mac 68:05:ca:a6:0a:b3
   ip link set enp24s0v3 vf 3 mac 68:05:ca:a6:0a:b4
   ```

6. Gather required configuration.

   ```
   ifconfig, ethtool -i,
   cat /sys/class/net/ethernet_interface/device/ numa_node,
   cat /sys/class/net/ethernet_interface/device/device, numactl -hardware, and dpdk-
   setup.sh

   # enp24s0:_pci_address/bus-info: 0000:18:00.0
   # enp24s0:_mac_address/ether: 68:05:ca:a6:0a:b0

   # enp24s1:_pci_address/bus-info: 0000:18:01.0
   # enp24s1:_mac_address/ether: 68:05:ca:a6:0a:b1

   # enp24s1f1:_pci_address/bus-info: 0000:18:01.1
   # enp24s1f1:_mac_address/ether: 68:05:ca:a6:0a:b2

   # enp24s2f2:_pci_address/bus-info: 0000:18:01.2
   # enp24s2f2:_mac_address/ether: 68:05:ca:a6:0a:b3
   ```

```
# enp24s2f3:_pci_address/bus-info: 0000:18:01.3
# enp24s2f3:_mac_address/ether: 68:05:ca:a6:0a:b4
```

Use the `get_config.sh` from Section 7.1 to get the PCI address.

```
root@nd-wolfpass-39:/usr/src# ./get_config.sh
No output file name supplied, configuration_details.txt will be used
*****************************************************
Configuration for nd-wolfpass-39

enp175s1f0_device_id=0x1889
enp175s1f0_firmware=
enp175s1f0_pci_address=0000:af:01.0
enp175s1f0_mac_address=68:05:ca:8d:c1:00
enp175s1f0_numa_node=1
enp175s1f0_ip_address=NA

enp175s1f1_device_id=0x1889
enp175s1f1_firmware=
enp175s1f1_pci_address=0000:af:01.1
enp175s1f1_mac_address=68:05:ca:8d:c1:01
enp175s1f1_numa_node=1
enp175s1f1_ip_address=NA

enp24s0f1_device_id=0x1593
enp24s0f1_firmware=0x80000ec4
enp24s0f1_pci_address=0000:18:00.1
enp24s0f1_mac_address=68:05:ca:9a:bd:dd
enp24s0f1_numa_node=0
enp24s0f1_ip_address=NA

enp175s1f2_device_id=0x1889
enp175s1f2_firmware=
enp175s1f2_pci_address=0000:af:01.2
enp175s1f2_mac_address=68:05:ca:8d:c1:02
enp175s1f2_numa_node=1
enp175s1f2_ip_address=NA

enp24s0f2_device_id=0x1593
enp24s0f2_firmware=0x80000ec4
enp24s0f2_pci_address=0000:18:00.2
enp24s0f2_mac_address=68:05:ca:9a:bd:de
enp24s0f2_numa_node=0
enp24s0f2_ip_address=NA

enp24s0f0_device_id=0x1593
enp24s0f0_firmware=0x80000ec4
enp24s0f0_pci_address=0000:18:00.0
enp24s0f0_mac_address=68:05:ca:9a:bd:dc
enp24s0f0_numa_node=0
enp24s0f0_ip_address=NA

enp175s0f0_device_id=0x1592
enp175s0f0_firmware=0x80000ec6
enp175s0f0_pci_address=0000:af:00.0
enp175s0f0_mac_address=68:05:ca:8d:b7:c0
enp175s0f0_numa_node=1
```

```
enp175s0f0_ip_address=192.168.43.39

enp175s1f3_device_id=0x1889
enp175s1f3_firmware=
enp175s1f3_pci_address=0000:af:01.3
enp175s1f3_mac_address=68:05:ca:8d:c1:03
enp175s1f3_numa_node=1
enp175s1f3_ip_address=NA

enp24s0f3_device_id=0x1593
enp24s0f3_firmware=0x80000ec4
enp24s0f3_pci_address=0000:18:00.3
enp24s0f3_mac_address=68:05:ca:9a:bd:df
enp24s0f3_numa_node=0
enp24s0f3_ip_address=NA

NUMA node(s):         2
NUMA node0 CPU(s):    0-17,36-53
NUMA node1 CPU(s):    18-35,54-71
```

7. Bind two virtual function interfaces to *igb_uio*.

```
/$RTE_SDK/install/sbin/dpdk-devbind -b igb_uio 18:01.0 18:01.1
```

8. Start the **testpmd** application on the VF.

```
root@nd-wolfpass-39:/usr/src/dpdk/dpdk/install/bin# ./testpmd -l 18-36 -n 4 -w
18:01.0 --file-prefix testpmdaf010 --socket-mem=0,1024 --proc-type=auto -- --nb-
cores=4 --rxq=4 --txq=4 -i --forward-mode=mac --eth-peer=0,68:05:ca:a6:0b:1c
EAL: Detected 72 lcore(s)
EAL: Detected 2 NUMA nodes
EAL: Auto-detected process type: PRIMARY
EAL: Multi-process socket /var/run/dpdk/testpmdaf010/mp_socket
EAL: No available hugepages reported in hugepages-1048576kB
EAL: Probing VFIO support...
EAL: VFIO support initialized
EAL: PCI device 0000:af:01.0 on NUMA socket 1
EAL:   probe driver: 8086:1889 net_iavf
Interactive-mode selected
Set mac packet forwarding mode
testpmd: create a new mbuf pool <mbuf_pool_socket_1>: n=291456, size=2176,
socket=1
testpmd: preferred mempool ops selected: ring_mp_mc
testpmd: create a new mbuf pool <mbuf_pool_socket_0>: n=291456, size=2176,
socket=0
testpmd: preferred mempool ops selected: ring_mp_mc

Warning! port-topology=paired and odd forward ports number, the last port will
pair with itself.

Configuring Port 0 (socket 1)

Port 0: link state change event

Port 0: link state change event

Port 0: link state change event

Port 0: link state change event
```

```
Port 0: link state change event

Port 0: link state change event

Port 0: link state change event

Port 0: link state change event
Port 0: 68:05:CA:8D:C1:00
Checking link statuses...
Done
```

7.0 Sample Scripts

7.1 Sample Script to Gather Information

Following is a sample script that gathers the required information to be used by the **pktgen** and **testpmd** applications:

```
## Get netdev id, pci addresses, mac address and numa node for E810 devices
### E810 devices (E810-CQDA1/CQDA2:0x1592 E810-XXVDA4:0x1593 iavf:0x1889)
### Script to gather information

################################
#!/bin/bash echo
if [ -z "$1" ] then
echo "No output file name supplied, configuration_details.txt will be used"
outfile=configuration_details.txt
else
outfile=$1
fi

echo "******************************************************"
readarray interfaces < <(find /sys/class/net -mindepth 1 -maxdepth 1 ! -name lo ! name
vir* ! -name br -printf "%P " -execdir cat {}/device/device \; | grep -e 159 -e 1889
| awk '{ print $1 }')

echo -n "configuration for " > $outfile hostname >> $outfile
echo >> $outfile
#for i in "${interfaces[@]// /}" ; do echo -n $i; ethtool -i $i | grep bus-info | awk
'{ print "_pci_address="$2 }' ; done >> $outfile
#echo >> $outfile

for i in "${interfaces[@]// /}" ; do
   find /sys/class/net -mindepth 1 -maxdepth 1 -name $i -printf "%P_device_id="
   -execdir cat {}/device/device \; >> $outfile

   echo -n $i >> $outfile; ethtool -i $i | grep firmware-version | awk '{ print \
   "_firmware="$3 }'>> $outfile
   echo -n $i >> $outfile; ethtool -i $i | grep bus-info | awk '{ print
   "_pci_address="$2 }'>> $outfile

    find /sys/class/net -mindepth 1 -maxdepth 1 -name $i -printf "%P_mac_address="
    -execdir cat {}/address \; >> $outfile

find /sys/class/net -mindepth 1 -maxdepth 1 -name $i -printf "%P_numa_node="
-execdir cat {}/device/numa_node \; >> $outfile

ipaddr=$(ip -f inet -o addr show dev $i|cut -d\-f7|cut -d/ -f1)

if [ -z "$ipaddr" ]
   then
      echo$i"_ip_address=NA" >> $outfile
   else
      echo -n $i >> $outfile;echo "_ip_address="$ipaddr >> $outfile

done
```

7.2 pktgen Configuration Script

Create *cvl.test.cfg* for **pktgen** in *cfg/* directory - change PCI addresses as needed.

```
vim $RTE_SDK/pktgen-dpdk/cfg/cvl.test.cfg description = 'A Pktgen default simple
configuration'

# Setup configuration setup = {
'exec': (
'sudo', '-E'
        ),

   'devices': (
   '86:00.0', '86:00.1'
   ),
   # UIO module type, igb_uio, vfio-pci or uio_pci_generic 'uio': 'vfio-pci'
   }

# Run command and options run = {
'exec': (
'sudo', '-E'
        ),

      # Application name and use app_path to help locate the app 'app_name': 'pktgen',

      # using (sdk) or (target) for specific variables
      # add (app_name) of the application
      # Each path is tested for the application
      'app_path': (
         './app/%(target)s/%(app_name)s',
         '%(sdk)s/%(target)s/app/%(app_name)s',
            ),

'cores': '21-37',
'nrank': '6',
'proc': 'auto',
'log': '7',
'prefix': 'pg8600x', 'whitelist': (
'86:00.0', '86:00.1'
),
'opts': (
'-T',
'-P',
   ),

'map': (
   '[22/24/26/28:23/25/27/29].0',
      '[30/32/34/36:31/33/35/37].1' ),

'theme': (
'   themes/black-yellow.theme',
   'test/cvl.test.txt',
   )
}
```

7.3 pktgen Configuration for Multiple Traffic Streams

```
vim $RTE_SDK/pktgen-dpdk/test/cvl.test.txt set 0 proto tcp
set 0 size 128
set 0 src mac 68:05:ca:a6:0b:1c
set 0 dst mac 68:05:ca:a6:0a:b0
set 0 src ip 192.168.103.101/24
set 0 dst ip 192.168.103.102
enable 0 range
range 0 size 128 128 128 1
range 0 src mac 68:05:ca:a6:0b:1c 68:05:ca:a6:0b:1c 68:05:ca:a6:0b:1c
00:00:00:00:00:01
range 0 dst mac 68:05:ca:a6:0a:b0 68:05:ca:a6:0a:b0 68:05:ca:a6:0a:b0
00:00:00:00:00:01
range 0 src ip 192.168.103.101 192.168.103.101 192.168.103.101 0.0.0.1
range 0 dst ip 192.168.103.102 192.168.103.102 192.168.103.102 0.0.0.1
range 0 dst port 2000 2000 2000 1
range 0 src port 3000 3000 3016 1
start 0
```

NOTE: ***This page intentionally left blank.***

LEGAL