Getting Started with DPDK and UHD

Contents

AN-500 by Nate Temple, Alex Williams, Wade Fife, Matt Prost, and Michael Dickens

This application note walks through the process to get started with the Data Plane Development Kit (DPDK) driver within UHD.

Up until now, UHD's only support for networked devices was backed by the kernel's sockets implementation. Every call to `send()` or `recv()` would cause a context switch and invite the kernel's scheduler to replace our thread with something else. Because the typical scheduler is optimized to distribute CPU time fairly across multiple loads, the timing-critical threads might sporadically be hit with sleeping time, and the thread might be migrated off its current CPU and forced to run on another. The overhead and random latency spikes make it difficult to enable reliable real-time streaming at higher rates.

DPDK is a high-speed packet processing framework that enables a kernel bypass for network drivers. By putting the entire driver in user space, avoiding context switches, and pinning I/O threads to cores, UHD and DPDK combine to largely prevent the latency spikes induced by the scheduler. In addition, the overall overhead for packet processing lowers.

DPDK is supported on the following USRP devices:

- E320
- N300 / N310
- N320 / N321
- X300 / X310
- X410
- X440

DPDK is supported on many Intel and Mellanox based 10Gb and 100Gb NICs and lots of other NICs. Below is a list of NICs Ettus Research has tested. For a full list of NICs supported by DPDK, please see the DPDK manual.

- Intel X520-DA1 (1x10Gb)
- Intel X520-DA2 (2x10Gb)
- Intel X710-DA2 (2x10Gb)
- Intel X710-DA4 (4x10Gb)
- Intel XL710-QDA2 (2x40Gb breakout to 4x10Gb)
- Intel E810-CQDA1 (1x100Gb and 1x4x10Gb)
- Intel E810-CQDA2 (1x100Gb and 2x4x10Gb; note that this NIC does not support 2x100Gb)
- Intel E810-2CQDA2 (1x100Gb and 2x4x10Gb; 2x100Gb with some work)
- Mellanox MCX4121A-ACAT ConnectX-4 Lx (2x10Gb)
- Mellanox MCX515A-CCAT ConnectX-5 EN (2x100Gb or 2x10Gb)
- Mellanox MCX516A-CCAT ConnectX-5 EN (2x100Gb or 2x10Gb)
- Mellanox MCX516A-CDAT ConnectX-5 Ex EN (2x100Gb or 2x10Gb)
- Mellanox MCX623106AN-CDAT ConnectX-6 Dx EN (2x100Gb or 2x10Gb)
- NI Dual 100 Gigabit Ethernet PCIe Interface Kit (PN 788216-01)

- DPDK: https://www.dpdk.org/
    - https://doc.dpdk.org/guides-17.11
    - https://doc.dpdk.org/guides-18.11
    - https://doc.dpdk.org/guides-19.11
    - https://doc.dpdk.org/guides-20.11
    - https://doc.dpdk.org/guides-21.11
    - https://doc.dpdk.org/guides-22.11
    - https://doc.dpdk.org/guides-23.11
    - https://doc.dpdk.org/guides-24.11
- UHD Manual: https://files.ettus.com/manual/page_dpdk.html

- UHD 3.x requires DPDK 17.11
- UHD 4.0 and 4.1 require DPDK 18.11
- UHD 4.2 to 4.7 can use any version of DPDK from 18.11 to 21.11
- UHD 4.8 to 4.9 can use any version of DPDK from 18.11 to 24.11

We recommend installing DPDK via the system-provided installer; for example with Ubuntu:

```
sudo apt install dpdk dpdk-dev
```

While it is possible to install DPDK from source, we recommend using the system-provided install unless there is a very good reason to not do so. DPDK can be challenging to *correctly* build from source though more recent versions use meson and ninja along with default settings needed by UHD and hence are relatively simple to build and install. If you require installing DPDK from source, the install guide for various versions is noted below:

- DPDK 17.11
- DPDK 18.11
- DPDK 19.11
- DPDK 20.11
- DPDK 21.11
- DPDK 22.11
- DPDK 23.11
- DPDK 24.11

DPDK releases come three times per year in April (version X.04), July (version X.07), and November (version X.11). The X.11 version is more of a formal release. While any *can* be used with UHD, we recommend using just the formal releases.

NOTE: It is sometimes necessary to use NIC device drivers because of improved feature sets or just getting a NIC to work; the system provided NIC drivers are too old. In this case: (1) remove the system provided DPDK and DPDK-DEV and whatever UHD install is in use; you will need to build these from source. (2) Download and install the new NIC drivers, NVM, and anything else that the NIC needs to be updated to the version needed. (3) Download and install DPDK from source. (4) Downlaod and install UHD from source.

Once the `dpdk` and `dpdk-dev` packages are installed, UHD will locate them during a build and you should see DPDK in the enabled components lists when running `cmake`.

NOTE that in general UHD installed from PPA or system packages *does not* include support for DPDK, and even if DPDK is installed alongside these UHD it will not be used. In order to get UHD with DPDK support, UHD generally has to be built from source.

Edit your grub configuration file, `/etc/default/grub` and add the follow parameters to `GRUB_CMDLINE_LINUX_DEFAULT`:

```
iommu=pt intel_iommu=on hugepages=2048
```

On a vanilla Ubuntu system it should look like this:

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash iommu=pt intel_iommu=on hugepages=2048"
```

Close `/etc/default/grub` and at the command prompt, update your grub configuration with the command:

```
sudo update-grub
```

For these settings to take effect, reboot your host machine.

You must note the MAC addresses for your NICs before proceeding.

The MAC addresses for your NICs can be found by running the command:

```
ip a
```

You must then create a UHD configuration file.

For UHD 3 the location is `/root/.uhd/uhd.conf`.

```
sudo su
mkdir -p /root/.uhd
nano /root/.uhd/uhd.conf
```

For UHD 4 the location is `/root/.config/uhd.conf`.

```
sudo su
mkdir -p /root/.config
nano /root/.config/uhd.conf
```

An example `uhd.conf` file is listed below. Note that field names in UHD 3.x are slightly different from UHD 4.0.

You should update the following fields for your configuration from this example:

- Update the MAC address variables, `dpdk-mac`, to match your NIC
- Update the `dpdk-driver` if the location is different on your system. `/usr/lib/x86_64-linux-gnu/dpdk-17.11-drivers/` is the default location on Ubuntu 18.04.x when `dpdk` is installed via `apt`.
- Update the `dpdk-corelist` and `dpdk-io-cpu` fields. In this example, a two port NIC is used. There should be one core for the main `dpdk` thread (in this example `core #2`), and then separate cores assigned to each NIC (in this example `core #3` for the first port on the NIC, `core #4` for the second port on the NIC)
- Update the `dpdk-ipv4` fields to your desired IP range.
    - `192.168.30.2`, `192.168.40.2` on a default X3xx system
    - `192.168.10.2`, `192.168.20.2` on a default N3xx system
    - `192.168.10.2` on a default E320 system

```
[use_dpdk=1]
dpdk-mtu=9000
dpdk-driver=/usr/lib/x86_64-linux-gnu/dpdk-17.11-drivers/
dpdk-corelist=2,3,4
dpdk-num-mbufs=4095
dpdk-mbufs-cache-size=315

[dpdk-mac=aa:bb:cc:dd:ee:f1]
dpdk-io-cpu = 3
dpdk-ipv4 = 192.168.10.1/24

[dpdk-mac=aa:bb:cc:dd:ee:f2]
dpdk-io-cpu = 4
dpdk-ipv4 = 192.168.20.1/24
```

Note: Additional information on the UHD configuration file can be found here:
https://files.ettus.com/manual_archive/v3.15.0.0/html/page_dpdk.html#dpdk_nic_config

An example `uhd.conf` file is listed below. Note that the field names in UHD 4.x are slightly different from UHD 3.x.

You must verify and/or update the following fields for your configuration from this example:

- The MAC address variables, `dpdk_mac`, to match your NIC(s) link(s); note that the MAC address info *must be lowercase*
- The `dpdk_driver` if the location is different on your system. `/usr/local/lib/` is the default location on when DPDK is built and installed from source.
- The `dpdk_corelist` and `dpdk_lcore` fields. In this example, a two port NIC and both links are used. There must be one core for the main `dpdk` thread (in this example `core #2` because it is not otherwise used in the file) and then separate cores assigned to each NIC link (in this example `core #3` for the one of the NIC links and `core #4` for the other NIC link).
- The `dpdk_ipv4` fields to your desired IP range(s).
    - `192.168.30.2`, `192.168.40.2` on a default X3xx system
    - `192.168.10.2`, `192.168.20.2` on a default N3xx system
    - `192.168.10.2` on a default E320 system
    - `192.168.10.2` and `192.168.20.2` on a default X4xx systems

```
[use_dpdk=1]
dpdk_mtu=9000
dpdk_driver=/usr/lib/x86_64-linux-gnu/dpdk/pmds-20.0/
dpdk_corelist=2,3,4
dpdk_num_mbufs=4096
dpdk_num_desc=4096
dpdk_mbuf_cache_size=315

[dpdk_mac=aa:bb:cc:dd:ee:f1]
dpdk_lcore=3
dpdk_ipv4=192.168.10.1/24

[dpdk_mac=aa:bb:cc:dd:ee:f2]
dpdk_lcore=4
dpdk_ipv4=192.168.20.1/24
```

Notes:

- Additional information on the UHD DPDK configuration file is in the UHD manual.
- The number of cores listed must be used exactly within the file.
- The number of NIC link(s) listed must exactly match those specified in the UHD instantiation args.
- A simple way to move between different DPDK configurations is to create different files and then symlink from the desired file to `uhd.conf`

The process for this step is different for Intel and Mellanox NICs and is detailed in individual sections below.

The Intel based NICs will use the `vfio-pci` driver which must be loaded:

```
sudo modprobe vfio-pci
```

Next, you will need to rebind the NIC to the `vfio-pci` drivers.

First, identify the PCI address your NIC is at via the following command, noting that for older DPDK the command does not contain the trailing `.py`

```
dpdk-devbind.py -s
```

Note the PCI address that your NIC is connected to for the next step.

Before the next step, you will need to turn off the NIC first before doing the rebind.

In Ubuntu under `System` -> `Network` -> click the switches to `off` for the 10Gb ports, then run the `dpdk-devbind` commands:

Note: Your PCI address will likely be different than `02:00.0` as shown in the example below.

```
sudo dpdk-devbind.py --bind=vfio-pci 02:00.0
sudo dpdk-devbind.py --bind=vfio-pci 02:00.1
```

You should now see the NICs listed under DPDK devices

```
# dpdk-devbind.py -s

Network devices using DPDK-compatible driver
============================================
0000:02:00.0 '82599ES 10-Gigabit SFI/SFP+ Network Connection 10fb' drv=vfio-pci unused=ixgbe
0000:02:00.1 '82599ES 10-Gigabit SFI/SFP+ Network Connection 10fb' drv=vfio-pci unused=ixgbe
```

Note: More info can be found here on the rebinding process:
https://doc.dpdk.org/guides-24.11/linux_gsg/linux_drivers.html#binding-and-unbinding-network-ports-to-from-the-kernel-modules

The NVIDIA Mellanox ("Mellanox") NICs do *not* require rebinding using the `vfio-pci` driver. Mellanox provides system IC drivers and additional drivers for DPDK to handle the binding to and from VFIO-PCI. In general the Mellanox system-provided NIC and DPDK drivers should work and you should not need to install Mellanox drivers.

If for some reason your host OS does not provide Mellanox DPDK drivers, then these can be installed manually. For example:

```
sudo apt install librte-pmd-mlx5
sudo modprobe -a ib_uverbs mlx5_core mlx5_ib
```

If for some reason you are running into issues using a Mellanox NIC, then you can download and install the latest Mellanox drivers from the Mellanox website (https://network.nvidia.com/products/infiniband-drivers/linux/mlnx_ofed/).

The MLX5 poll mode driver library (librte_pmd_mlx5) in DPDK provides support for Mellanox ConnextX-4 through -7 NICs. For DPDK 20 and older only this driver must be enabled manually with the build option `CONFIG_RTE_LIBRTE_MLX5_PMD=y` when building DPDK. We recommend DPDK 21 and newer when possible; these versions are easy to build from source using `meson` and `ninja`.

UHD based application (including GNU Radio flowgraphs) can now be ran using a DPDK transport by passing in the Device Argument: `use_dpdk=1`.

Important Note: In order for UHD to use DPDK, the UHD application *must* be ran as the `root` user; this can be done either by prepending `sudo` to the command or becoming the roort user via `sudo su`.

For example, running the `benchmark_rate` utility after issuing `sudo su` (one can also run this command simply as `sudo /usr/local/lib/uhd/examples/benchmark_rate ....`:

```
# cd /usr/local/lib/uhd/examples

# ./benchmark_rate --rx_rate 125e6 --rx_subdev "A:0 B:0" --rx_channels 0,1 --tx_rate 125e6 --tx_subdev "A:0 B:0" --tx_channels 0,1 --args "ad

[INFO] [UHD] linux; GNU C++ version 7.3.0; Boost_106501; UHD_3.14.0.HEAD-0-gabf0db4e
EAL: Detected 8 lcore(s)
EAL: Some devices want iova as va but pa will be used because.. EAL: IOMMU does not support IOVA as VA
EAL: No free hugepages reported in hugepages-1048576kB
EAL: Probing VFIO support...
EAL: VFIO support initialized
EAL: PCI device 0000:02:00.0 on NUMA socket -1
EAL:   Invalid NUMA socket, default to 0
EAL:   probe driver: 8086:10fb net_ixgbe
EAL:   using IOMMU type 1 (Type 1)
EAL: Ignore mapping IO port bar(2)
EAL: PCI device 0000:02:00.1 on NUMA socket -1
EAL:   Invalid NUMA socket, default to 0
EAL:   probe driver: 8086:10fb net_ixgbe
EAL: Ignore mapping IO port bar(2)
PMD: ixgbe_dev_link_status_print():  Port 0: Link Down
EAL: Port 0 MAC: aa bb cc dd ee f1
EAL: Port 0 UP: 1
PMD: ixgbe_dev_link_status_print():  Port 1: Link Down
EAL: Port 1 MAC: aa bb cc dd ee f2
EAL: Port 1 UP: 1
EAL: Init DONE!
EAL: Starting I/O threads!
USER2: Thread 1 started
[00:00:00.000003] Creating the usrp device with: addr=192.168.10.2,second_addr=192.168.20.2,mgmt_addr=10.2.1.19,master_clock_rate=125e6,use_d
[INFO] [MPMD] Initializing 1 device(s) in parallel with args: mgmt_addr=10.2.1.19,type=n3xx,product=n310,serial=313ABDA,claimed=False,addr=19
[INFO] [MPM.PeriphManager] init() called with device args 'product=n310,time_source=internal,master_clock_rate=125e6,clock_source=internal,us
[INFO] [0/DmaFIFO_0] Initializing block control (NOC ID: 0xF1F0D00000000004)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1344 MB/s)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1341 MB/s)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1348 MB/s)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1347 MB/s)
[INFO] [0/Radio_0] Initializing block control (NOC ID: 0x12AD100000011312)
[INFO] [0/Radio_1] Initializing block control (NOC ID: 0x12AD100000011312)
[INFO] [0/DDC_0] Initializing block control (NOC ID: 0xDDC0000000000000)
[INFO] [0/DDC_1] Initializing block control (NOC ID: 0xDDC0000000000000)
[INFO] [0/DUC_0] Initializing block control (NOC ID: 0xD0C0000000000002)
[INFO] [0/DUC_1] Initializing block control (NOC ID: 0xD0C0000000000002)
Using Device: Single USRP:
  Device: N300-Series Device
  Mboard 0: ni-n3xx-313ABDA
  RX Channel: 0
    RX DSP: 0
    RX Dboard: A
    RX Subdev: Magnesium
  RX Channel: 1
    RX DSP: 0
    RX Dboard: B
    RX Subdev: Magnesium
  TX Channel: 0
    TX DSP: 0
    TX Dboard: A
    TX Subdev: Magnesium
  TX Channel: 1
```

```
    TX DSP: 0
    TX Dboard: B
    TX Subdev: Magnesium

[00:00:03.728707] Setting device timestamp to 0...
[INFO] [MULTI_USRP]     1) catch time transition at pps edge
[INFO] [MULTI_USRP]     2) set times next pps (synchronously)
[00:00:05.331920] Testing receive rate 125.000000 Msps on 2 channels
[00:00:05.610789] Testing transmit rate 125.000000 Msps on 2 channels
[00:00:15.878071] Benchmark complete.


Benchmark rate summary:
  Num received samples:     2557247854
  Num dropped samples:      0
  Num overruns detected:    0
  Num transmitted samples:  2504266704
  Num sequence errors (Tx): 0
  Num sequence errors (Rx): 0
  Num underruns detected:   0
  Num late commands:        0
  Num timeouts (Tx):        0
  Num timeouts (Rx):        0


Done!
```

Perform the general host performance tuning tips and tricks.

If you experience `Overflows` at higher data rates, adding the device argument `num_recv_frames=512` can help.

If you're streaming data at the full master clock rate, and there is no interpolation or decimation being performed on the FPGA, you can skip the DUC and DDC blocks within the FPGA with the following parameters:

```
skip_ddc=1
skip_duc=1
```

If you're streaming two transmit channels at full rate (200e6) on the X3xx platform, you should additionally set the following device arg:

```
enable_tx_dual_eth=1
```

Isolating the cores that are used for DPDK can improve performance. This can be done by adding the `isolcpus` parameter to your `/etc/default/grub` file in the `GRUB_CMDLINE_LINUX_DEFAULT=""` (the "`GRUB_CONFIG`"). For example, to isolate cores 2 through and including 4, add this entry:

```
isolcpus=2-4
```

NOTE: After saving the `GRUB_CONFIG` file, execute `sudo update-grub` and then reboot the computer for the changes to take effect. It is OK to isolate core used by DPDK via this method.

Disabling system interrupts can improve the jitter and performance generally by 1-3%. This can be done by adding the parameters `nohz_full` and `rcu_nocbs` to your `GRUB_CONFIG`. For example, to disable system interrupts on cores 2 through and including 4, add this entry:

```
nohz_full=2-4 rcu_nocbs=2-4
```

NOTE: After saving the `GRUB_CONFIG` file, execute `sudo update-grub` and then reboot the computer for the changes to take effect. It is OK to isolate core used by DPDK via this method.

If you're streaming on multiple channels simultaneously, you can create multiple streamer objects on separate threads. This can be accomplished with the `benchmark_rate` example by using the parameter `--multi_streamer`.

In UHD 4, streaming thread priorities can be elevated with the `uhd::set_thread_priority_safe()` function call. This can be accomplished with the `benchmark_rate` example by using parameter `--priority high`. Note that if the Thread Schedule Priority has not been enabled for the current user then this parameter requires `sudo` to work.

Beyond elevating streaming thread priority, one can also increase the `nice` priority level to maximum to increase the amount of CPU time for the process and its thread by prepending the following to the command being issued:

```
sudo nice -n -20
```

With Linux one can limit the cores being used by the threads in a process via a `taskset` prepended to a command being executed. When combining with the other techniques listed here, one can fairly well constrain a process and its thread to specific cores and give them maximum CPU time. Note that when using DPDK the MAC cores specified in `uhd.conf` will already be included as part of the `taskset` (but those cores will not be isolated nor have system interrupts disabled on them unless using that setting as noted above); it generally won't hurt to include the DPDK cores as part of the overall `taskset` but it's not required. For example, to limit process/thread execution to cores 2 through and including 4, one would prepend the following (note that `sudo` is *not* required):

```
taskset -c "2-4"
```

NOTE: For best performance do *not* include the cores used by DPDK in the `dpdk_corelist` argument in the `taskset`.

The Linux kernel spawns a number of processes and threads that spend most of their time sleeping; one cannot stop these processes/threads. That said, in a typical Linux install (for example Ubuntu) there will be a graphical desktop (e.g., `gdm`) and various daemons started up by user request (e.g., `containerd`, `docker`, `snapd`). Most of these daemons can be controlled by `systemctl`; some must be manually quit. Any process that runs regularly stands a chance of interrupting the UHD and DPDK threads, and thus stopping, quitting, or disabling those processes can increase performance. For example, the following commands stop various daemons that execute regularly on Ubuntu; these need to be executed with `sudo` and some might not

exist on your specific system and for these the command will do nothing so it's OK to run it:

```
systemctl stop containerd containerd.socket
systemctl stop docker docker.socket
systemctl stop dbus dbus.socket
systemctl stop snapd snapd.socket
systemctl stop udev systemd-udevd-control.socket systemd-udevd-kernel.socket
```

The following command stops the main Ubuntu desktop GUI, so running it will render the system accessible only via networking (e.g., `ssh`), so make sure network access is enabled before executing this command:

```
systemctl stop gdm
```

In some applications which require the highest possible CPU performance per core, disabling hyper-threading can provide roughly a 10% increase in core performance, at the cost of having fewer core threads. Hyper-threading is disabled within the BIOS and how to do this varies by motherboard manufacturer. With other techniques listed here, disabling hyper-threading should only be done as a last resort to eek absolute maximum performance from the CPU.

- Performance report from Intel on DPDK 17.11: https://fast.dpdk.org/doc/perf/DPDK_17_11_Intel_NIC_performance_report.pdf
- How to get best performance with NICs on Intel platforms: https://doc.dpdk.org/guides/linux_gsg/nic_perf_intel_platform.html

On Multi-CPU systems each CPU is a NUMA node. The NUMA node contains all of the cores on the CPU. For best performance when using CPU/core isolation ("isocpus" per AAA above) make sure that all of the cores are in the same NUMA node.

With Linux kernels 5.10 and beyond, we have observed periodic underruns on systems that otherwise have no issues. These Linux kernel versions are the default for Ubuntu 20.04.3 LTS and later. The underrun issue is due to the `RT_RUNTIME_SHARE` feature being disabled by default in these versions of the Linux kernel (shown as `NO_RT_RUNTIME_SHARE`). The following procedure can be used to enable this feature. This process was tested on Linux kernel version 5.13; the procedure may be slightly different on other kernel versions. To determine the Linux kernel version of your system, in a terminal issue the command `uname -r`.

```
$ sudo -s
$ cd /sys/kernel/debug/sched
$ cat features | tr ' ' '\n' | grep RUNTIME_SHARE
NO_RT_RUNTIME_SHARE
$ echo RT_RUNTIME_SHARE > features
$ cat features | tr ' ' '\n' | grep RUNTIME_SHARE
RT_RUNTIME_SHARE
```