DPDK

DATA PLANE DEVELOPMENT KIT

Getting Started Guide for FreeBSD

Release 16.07.2

November 30, 2016

CONTENTS

CHAPTER

ONE

INTRODUCTION

This document contains instructions for installing and configuring the Data Plane Development Kit (DPDK) software. It is designed to get customers up and running quickly and describes how to compile and run a DPDK application in a FreeBSD application (bsdapp) environment, without going deeply into detail.

For a comprehensive guide to installing and using FreeBSD, the following handbook is available from the FreeBSD Documentation Project: FreeBSD Handbook.

Note: The DPDK is now available as part of the FreeBSD ports collection. Installing via the ports collection infrastructure is now the recommended way to install the DPDK on FreeBSD, and is documented in the next chapter, *Installing DPDK from the Ports Collection*.

1.1 Documentation Roadmap

The following is a list of DPDK documents in the suggested reading order:

- **Release Notes** : Provides release-specific information, including supported features, limitations, fixed issues, known issues and so on. Also, provides the answers to frequently asked questions in FAQ format.
- **Getting Started Guide** (this document): Describes how to install and configure the DPDK; designed to get users up and running quickly with the software.
- **Programmer's Guide**: Describes:
  - The software architecture and how to use it (through examples), specifically in a Linux* application (linuxapp) environment
  - The content of the DPDK, the build system (including the commands that can be used in the root DPDK Makefile to build the development kit and an application) and guidelines for porting an application
  - Optimizations used in the software and those that should be considered for new development

  A glossary of terms is also provided.
- **API Reference**: Provides detailed information about DPDK functions, data structures and other programming constructs.

- **Sample Applications User Guide**: Describes a set of sample applications. Each chapter describes a sample application that showcases specific functionality and provides instructions on how to compile, run and use the sample application.

CHAPTER

TWO

INSTALLING DPDK FROM THE PORTS COLLECTION

The easiest way to get up and running with the DPDK on FreeBSD is to install it from the ports collection. Details of getting and using the ports collection are documented in the FreeBSD Handbook.

Note: Testing has been performed using FreeBSD 10.0-RELEASE (x86_64) and requires the installation of the kernel sources, which should be included during the installation of FreeBSD.

2.1 Installing the DPDK FreeBSD Port

On a system with the ports collection installed in `/usr/ports`, the DPDK can be installed using the commands:

```
cd /usr/ports/net/dpdk

make install
```

After the installation of the DPDK port, instructions will be printed on how to install the kernel modules required to use the DPDK. A more complete version of these instructions can be found in the sections *Loading the DPDK contigmem Module* and *Loading the DPDK nic_uio Module*. Normally, lines like those below would be added to the file `/boot/loader.conf`.

```
# Reserve 2 x 1G blocks of contiguous memory using contigmem driver:
hw.contigmem.num_buffers=2
hw.contigmem.buffer_size=1073741824
contigmem_load="YES"

# Identify NIC devices for DPDK apps to use and load nic_uio driver:
hw.nic_uio.bdfs="2:0:0,2:0:1"
nic_uio_load="YES"
```

2.2 Compiling and Running the Example Applications

When the DPDK has been installed from the ports collection it installs its example applications in `/usr/local/share/dpdk/examples` - also accessible via symlink as `/usr/local/share/examples/dpdk`. These examples can be compiled and run as described in *Compiling and Running Sample Applications*. In this case, the required environmental variables should be set as below:

- `RTE_SDK=/usr/local/share/dpdk`
- `RTE_TARGET=x86_64-native-bsdapp-clang`

Note: To install a copy of the DPDK compiled using gcc, please download the official DPDK package from http://dpdk.org/ and install manually using the instructions given in the next chapter, *Compiling the DPDK Target from Source*

An example application can therefore be copied to a user's home directory and compiled and run as below:

```
export RTE_SDK=/usr/local/share/dpdk

export RTE_TARGET=x86_64-native-bsdapp-clang

cp -r /usr/local/share/dpdk/examples/helloworld .

cd helloworld/

gmake
  CC main.o
  LD helloworld
  INSTALL-APP helloworld
  INSTALL-MAP helloworld.map

sudo ./build/helloworld -c F -n 2

EAL: Contigmem driver has 2 buffers, each of size 1GB
EAL: Sysctl reports 8 cpus
EAL: Detected lcore 0
EAL: Detected lcore 1
EAL: Detected lcore 2
EAL: Detected lcore 3
EAL: Support maximum 64 logical core(s) by configuration.
EAL: Detected 4 lcore(s)
EAL: Setting up physically contiguous memory...
EAL: Mapped memory segment 1 @ 0x802400000: len 1073741824
EAL: Mapped memory segment 2 @ 0x842400000: len 1073741824
EAL: WARNING: clock_gettime cannot use CLOCK_MONOTONIC_RAW and HPET
     is not available - clock timings may be less accurate.
EAL: TSC frequency is ~3569023 KHz
EAL: PCI scan found 24 devices
EAL: Master core 0 is ready (tid=0x802006400)
EAL: Core 1 is ready (tid=0x802006800)
EAL: Core 3 is ready (tid=0x802007000)
EAL: Core 2 is ready (tid=0x802006c00)
EAL: PCI device 0000:01:00.0 on NUMA socket 0
EAL:   probe driver: 8086:10fb rte_ixgbe_pmd
EAL:   PCI memory mapped at 0x80074a000
EAL:   PCI memory mapped at 0x8007ca000
EAL: PCI device 0000:01:00.1 on NUMA socket 0
EAL:   probe driver: 8086:10fb rte_ixgbe_pmd
EAL:   PCI memory mapped at 0x8007ce000
EAL:   PCI memory mapped at 0x80084e000
EAL: PCI device 0000:02:00.0 on NUMA socket 0
EAL:   probe driver: 8086:10fb rte_ixgbe_pmd
EAL:   PCI memory mapped at 0x800852000
EAL:   PCI memory mapped at 0x8008d2000
EAL: PCI device 0000:02:00.1 on NUMA socket 0
EAL:   probe driver: 8086:10fb rte_ixgbe_pmd
EAL:   PCI memory mapped at 0x801b3f000
EAL:   PCI memory mapped at 0x8008d6000
hello from core 1
hello from core 2
hello from core 3
```

```
hello from core 0
```

Note: To run a DPDK process as a non-root user, adjust the permissions on the `/dev/contigmem` and `/dev/uio device` nodes as described in section *Running DPDK Applications Without Root Privileges*

Note: For an explanation of the command-line parameters that can be passed to an DPDK application, see section *Running a Sample Application*.

CHAPTER

THREE

COMPILING THE DPDK TARGET FROM SOURCE

3.1 System Requirements

The DPDK and its applications require the GNU make system (gmake) to build on FreeBSD. Optionally, gcc may also be used in place of clang to build the DPDK, in which case it too must be installed prior to compiling the DPDK. The installation of these tools is covered in this section.

Compiling the DPDK requires the FreeBSD kernel sources, which should be included during the installation of FreeBSD on the development platform. The DPDK also requires the use of FreeBSD ports to compile and function.

To use the FreeBSD ports system, it is required to update and extract the FreeBSD ports tree by issuing the following commands:

```
portsnap fetch
portsnap extract
```

If the environment requires proxies for external communication, these can be set using:

```
setenv http_proxy <my_proxy_host>:<port>
setenv ftp_proxy <my_proxy_host>:<port>
```

The FreeBSD ports below need to be installed prior to building the DPDK. In general these can be installed using the following set of commands:

```
cd /usr/ports/<port_location>

make config-recursive

make install

make clean
```

Each port location can be found using:

```
whereis <port_name>
```

The ports required and their locations are as follows:

- dialog4ports: `/usr/ports/ports-mgmt/dialog4ports`
- GNU make(gmake): `/usr/ports/devel/gmake`
- coreutils: `/usr/ports/sysutils/coreutils`

For compiling and using the DPDK with gcc, the compiler must be installed from the ports collection:

- gcc: version 4.8 is recommended `/usr/ports/lang/gcc48`. Ensure that `CPU_OPTS` is selected (default is OFF).

When running the make config-recursive command, a dialog may be presented to the user. For the installation of the DPDK, the default options were used.

Note: To avoid multiple dialogs being presented to the user during make install, it is advisable before running the make install command to re-run the make config-recursive command until no more dialogs are seen.

3.2 Install the DPDK and Browse Sources

First, uncompress the archive and move to the DPDK source directory:

```
unzip DPDK-<version>.zip
cd DPDK-<version>

ls
app/ config/ examples/ lib/ LICENSE.GPL LICENSE.LGPL Makefile
mk/ scripts/ tools/
```

The DPDK is composed of several directories:

- lib: Source code of DPDK libraries
- app: Source code of DPDK applications (automatic tests)
- examples: Source code of DPDK applications
- config, tools, scripts, mk: Framework-related makefiles, scripts and configuration

3.3 Installation of the DPDK Target Environments

The format of a DPDK target is:

```
ARCH-MACHINE-EXECENV-TOOLCHAIN
```

Where:

- `ARCH` is: `x86_64`
- `MACHINE` is: `native`
- `EXECENV` is: `bsdapp`
- `TOOLCHAIN` is: `gcc` | `clang`

The configuration files for the DPDK targets can be found in the DPDK/config directory in the form of:

```
defconfig_ARCH-MACHINE-EXECENV-TOOLCHAIN
```

Note: Configuration files are provided with the `RTE_MACHINE` optimization level set. Within the configuration files, the `RTE_MACHINE` configuration value is set to native, which means that the compiled software is tuned for the platform on which it is built. For more information on this setting, and its possible values, see the *DPDK Programmers Guide*.

To make the target, use `gmake install T=<target>`.

For example to compile for FreeBSD use:

```
gmake install T=x86_64-native-bsdapp-clang
```

Note: If the compiler binary to be used does not correspond to that given in the TOOLCHAIN part of the target, the compiler command may need to be explicitly specified. For example, if compiling for gcc, where the gcc binary is called gcc4.8, the command would need to be `gmake install T=<target> CC=gcc4.8`.

3.4 Browsing the Installed DPDK Environment Target

Once a target is created, it contains all the libraries and header files for the DPDK environment that are required to build customer applications. In addition, the test and testpmd applications are built under the build/app directory, which may be used for testing. A kmod directory is also present that contains the kernel modules to install:

```
ls x86_64-native-bsdapp-gcc

app build include kmod lib Makefile
```

3.5 Loading the DPDK contigmem Module

To run a DPDK application, physically contiguous memory is required. In the absence of non-transparent superpages, the included sources for the contigmem kernel module provides the ability to present contiguous blocks of memory for the DPDK to use. The contigmem module must be loaded into the running kernel before any DPDK is run. The module is found in the kmod sub-directory of the DPDK target directory.

The amount of physically contiguous memory along with the number of physically contiguous blocks to be reserved by the module can be set at runtime prior to module loading using:

```
kenv hw.contigmem.num_buffers=n
kenv hw.contigmem.buffer_size=m
```

The kernel environment variables can also be specified during boot by placing the following in `/boot/loader.conf`:

```
hw.contigmem.num_buffers=n hw.contigmem.buffer_size=m
```

The variables can be inspected using the following command:

```
sysctl -a hw.contigmem
```

Where n is the number of blocks and m is the size in bytes of each area of contiguous memory. A default of two buffers of size 1073741824 bytes (1 Gigabyte) each is set during module load if they are not specified in the environment.

The module can then be loaded using kldload (assuming that the current directory is the DPDK target directory):

```
kldload ./kmod/contigmem.ko
```

It is advisable to include the loading of the contigmem module during the boot process to avoid issues with potential memory fragmentation during later system up time. This can be

achieved by copying the module to the /boot/kernel/ directory and placing the following into /boot/loader.conf:

```
contigmem_load="YES"
```

Note: The contigmem_load directive should be placed after any definitions of hw.contigmem.num_buffers and hw.contigmem.buffer_size if the default values are not to be used.

An error such as:

```
kldload: can't load ./x86_64-native-bsdapp-gcc/kmod/contigmem.ko:
         Exec format error
```

is generally attributed to not having enough contiguous memory available and can be verified via dmesg or /var/log/messages:

```
kernel: contigmalloc failed for buffer <n>
```

To avoid this error, reduce the number of buffers or the buffer size.

3.6 Loading the DPDK nic_uio Module

After loading the contigmem module, the nic_uio module must also be loaded into the running kernel prior to running any DPDK application. This module must be loaded using the kldload command as shown below (assuming that the current directory is the DPDK target directory).

```
kldload ./kmod/nic_uio.ko
```

Note: If the ports to be used are currently bound to a existing kernel driver then the hw.nic_uio.bdfs sysctl value will need to be set before loading the module. Setting this value is described in the next section below.

Currently loaded modules can be seen by using the kldstat command and a module can be removed from the running kernel by using kldunload <module_name>.

To load the module during boot, copy the nic_uio module to /boot/kernel and place the following into /boot/loader.conf:

```
nic_uio_load="YES"
```

Note: nic_uio_load="YES" must appear after the contigmem_load directive, if it exists.

By default, the nic_uio module will take ownership of network ports if they are recognized DPDK devices and are not owned by another module. However, since the FreeBSD kernel includes support, either built-in, or via a separate driver module, for most network card devices, it is likely that the ports to be used are already bound to a driver other than nic_uio. The following sub-section describe how to query and modify the device ownership of the ports to be used by DPDK applications.

3.6.1 Binding Network Ports to the nic_uio Module

Device ownership can be viewed using the pciconf -l command. The example below shows four Intel® 82599 network ports under `if_ixgbe` module ownership.

```
pciconf -l
ix0@pci0:1:0:0: class=0x020000 card=0x00038086 chip=0x10fb8086 rev=0x01 hdr=0x00
ix1@pci0:1:0:1: class=0x020000 card=0x00038086 chip=0x10fb8086 rev=0x01 hdr=0x00
ix2@pci0:2:0:0: class=0x020000 card=0x00038086 chip=0x10fb8086 rev=0x01 hdr=0x00
ix3@pci0:2:0:1: class=0x020000 card=0x00038086 chip=0x10fb8086 rev=0x01 hdr=0x00
```

The first column constitutes three components:

1. Device name: `ixN`
2. Unit name: `pci0`
3. Selector (Bus:Device:Function): `1:0:0`

Where no driver is associated with a device, the device name will be `none`.

By default, the FreeBSD kernel will include built-in drivers for the most common devices; a kernel rebuild would normally be required to either remove the drivers or configure them as loadable modules.

To avoid building a custom kernel, the `nic_uio` module can detach a network port from its current device driver. This is achieved by setting the `hw.nic_uio.bdfs` kernel environment variable prior to loading `nic_uio`, as follows:

```
hw.nic_uio.bdfs="b:d:f,b:d:f,..."
```

Where a comma separated list of selectors is set, the list must not contain any whitespace.

For example to re-bind `ix2@pci0:2:0:0` and `ix3@pci0:2:0:1` to the `nic_uio` module upon loading, use the following command:

```
kenv hw.nic_uio.bdfs="2:0:0,2:0:1"
```

The variable can also be specified during boot by placing the following into `/boot/loader.conf`, before the previously-described `nic_uio_load` line - as shown:

```
hw.nic_uio.bdfs="2:0:0,2:0:1"
nic_uio_load="YES"
```

3.6.2 Binding Network Ports Back to their Original Kernel Driver

If the original driver for a network port has been compiled into the kernel, it is necessary to reboot FreeBSD to restore the original device binding. Before doing so, update or remove the `hw.nic_uio.bdfs` in `/boot/loader.conf`.

If rebinding to a driver that is a loadable module, the network port binding can be reset without rebooting. To do so, unload both the target kernel module and the `nic_uio` module, modify or clear the `hw.nic_uio.bdfs` kernel environment (kenv) value, and reload the two drivers - first the original kernel driver, and then the `nic_uio driver`. Note: the latter does not need to be reloaded unless there are ports that are still to be bound to it.

Example commands to perform these steps are shown below:

```
kldunload nic_uio
kldunload <original_driver>

# To clear the value completely:
```

```
kenv -u hw.nic_uio.bdfs

# To update the list of ports to bind:
kenv hw.nic_uio.bdfs="b:d:f,b:d:f,..."

kldload <original_driver>

kldload nic_uio  # optional
```

CHAPTER

FOUR

COMPILING AND RUNNING SAMPLE APPLICATIONS

The chapter describes how to compile and run applications in a DPDK environment. It also provides a pointer to where sample applications are stored.

4.1 Compiling a Sample Application

Once a DPDK target environment directory has been created (such as `x86_64-native-bsdapp-clang`), it contains all libraries and header files required to build an application.

When compiling an application in the FreeBSD 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 FreeBSD, this is the `x86_64-native-bsdapp-clang` or `x86_64-native-bsdapp-gcc` directory.

The following is an example of creating the `helloworld` application, which runs in the DPDK FreeBSD environment. While the example demonstrates compiling using gcc version 4.8, compiling with clang will be similar, except that the `CC=` parameter can probably be omitted. The `helloworld` example may be found in the `${RTE_SDK}/examples` directory.

The directory contains the `main.c` file. This file, when combined with the libraries in the DPDK target environment, calls the various functions to initialize the DPDK environment, then launches an entry point (dispatch application) for each core to be utilized. By default, the binary is generated in the build directory.

```
setenv RTE_SDK /home/user/DPDK
cd $(RTE_SDK)
cd examples/helloworld/
setenv RTE_SDK $HOME/DPDK
setenv RTE_TARGET x86_64-native-bsdapp-gcc

gmake CC=gcc48
  CC main.o
  LD helloworld
  INSTALL-APP helloworld
  INSTALL-MAP helloworld.map

ls build/app
  helloworld helloworld.map
```

Note: In the above example, `helloworld` was in the directory structure of the DPDK. However, it could have been located outside the directory structure to keep the DPDK structure

intact. In the following case, the `helloworld` application is copied to a new directory as a new starting point.

```
setenv RTE_SDK /home/user/DPDK
cp -r $(RTE_SDK)/examples/helloworld my_rte_app
cd my_rte_app/
setenv RTE_TARGET x86_64-native-bsdapp-gcc

gmake CC=gcc48
  CC main.o
  LD helloworld
  INSTALL-APP helloworld
  INSTALL-MAP helloworld.map
```

4.2 Running a Sample Application

1. The `contigmem` and `nic_uio` modules must be set up prior to running an application.
2. Any ports to be used by the application must be already bound to the `nic_uio` module, as described in section *Binding Network Ports to the nic_uio Module*, prior to running the application. The application is linked with the DPDK target environment's Environment Abstraction Layer (EAL) library, which provides some options that are generic to every DPDK application.

The following is the list of options that can be given to the EAL:

```
./rte-app -c COREMASK [-n NUM] [-b <domain:bus:devid.func>] \
          [-r NUM] [-v] [--proc-type <primary|secondary|auto>]
```

Note: EAL has a common interface between all operating systems and is based on the Linux notation for PCI devices. For example, a FreeBSD device selector of `pci0:2:0:1` is referred to as `02:00.1` in EAL.

The EAL options for FreeBSD are as follows:

- `-c COREMASK`: A hexadecimal bit mask of the cores to run on. Note that core numbering can change between platforms and should be determined beforehand.
- `-n NUM`: Number of memory channels per processor socket.
- `-b <domain:bus:devid.func>`: Blacklisting of ports; prevent EAL from using specified PCI device (multiple `-b` options are allowed).
- `--use-device`: Use the specified Ethernet device(s) only. Use comma-separate `[domain:]bus:devid.func` values. Cannot be used with `-b` option.
- `-r NUM`: Number of memory ranks.
- `-v`: Display version information on startup.
- `--proc-type`: The type of process instance.

Other options, specific to Linux and are not supported under FreeBSD are as follows:

- `socket-mem`: Memory to allocate from hugepages on specific sockets.
- `--huge-dir`: The directory where hugetlbfs is mounted.
- `--file-prefix`: The prefix text used for hugepage filenames.

- `-m MB`: Memory to allocate from hugepages, regardless of processor socket. It is recommended that `--socket-mem` be used instead of this option.

The `-c` option is mandatory; the others are optional.

Copy the DPDK application binary to your target, then run the application as follows (assuming the platform has four memory channels, and that cores 0-3 are present and are to be used for running the application):

```
./helloworld -c f -n 4
```

Note: The `--proc-type` and `--file-prefix` EAL options are used for running multiple DPDK processes. See the "Multi-process Sample Application" chapter in the *DPDK Sample Applications User Guide and the DPDK Programmers Guide* for more details.

4.3 Running DPDK Applications Without Root Privileges

Although applications using the DPDK use network ports and other hardware resources directly, with a number of small permission adjustments, it is possible to run these applications as a user other than "root". To do so, the ownership, or permissions, on the following file system objects should be adjusted to ensure that the user account being used to run the DPDK application has access to them:

- The userspace-io device files in `/dev`, for example, `/dev/uio0`, `/dev/uio1`, and so on
- The userspace contiguous memory device: `/dev/contigmem`

Note: Please refer to the DPDK Release Notes for supported applications.
