Intel® Data Plane Development Kit (Intel® DPDK) for FreeBSD*

Getting Started Guide

June 2014

Reference Number: 330041-002

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A “Mission Critical Application” is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined”. Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548- 4725, or go to: http://www.intel.com/design/literature.htm

Any software source code reprinted in this document is furnished for informational purposes only and may only be used or copied and no license, express or implied, by estoppel or otherwise, to any of the reprinted source code is granted by this document.

Code Names are only for use by Intel to identify products, platforms, programs, services, etc. (“products”) in development by Intel that have not been made commercially available to the public, i.e., announced, launched or shipped. They are never to be used as “commercial” names for products. Also, they are not intended to function as trademarks.

Contents

Revision History

Date	Revision	Description
January 2014	1.0	Initial release. Supports public software release R1.6.0.
June 2014	1.1	Supports public software release R1.7.0 and updated to include FreeBSD* 10

§

1 Introduction

This document contains instructions for installing and configuring the Intel® Data Plane Development Kit (Intel® DPDK) software. It is designed to get customers up and running quickly. The document describes how to compile and run an Intel® 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:

http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/index.html

1.1 Documentation Roadmap

The following is a list of Intel® 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 Intel® 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 Intel® DPDK, the build system (including the commands that can be used in the root Intel® 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 Intel® 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.

Note: These documents are available for download as a separate documentation package at the same location as the Intel® DPDK code package.

§

2 System Requirements

This chapter describes the packages required to compile the Intel® DPDK.

2.1 Compilation of the Intel® DPDK

Note: The Intel® DPDK and its applications requires the GNU make system (`gmake`) and the GNU Compiler Collection (`gcc`) to build on FreeBSD*. The installation of these tools is covered in this section.

Required Tools:

Note: Testing has been performed using FreeBSD* 9.2-RELEASE (x86_64), FreeBSD* 10.0-RELEASE (x86_64) and requires the installation of the kernel sources, which should be included during the installation of FreeBSD*. The Intel® 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:

```
root@host:~ # portsnap fetch
root@host:~ # portsnap extract
```

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

```
root@host:~ # setenv http_proxy <my_proxy_host>:<port>>
root@host:~ # setenv ftp_proxy <my_proxy_host>:<port>
```

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

1. cd /usr/ports/<port_location>
2. make config-recursive
3. make install
4. make clean

Each port location can be found using:

```
user@host:~ # whereis <port_name>
```

The ports required and their locations are as follows:

- dialog4ports
  - `/usr/ports/ports-mgmt/dialog4ports`
- `gcc`: version 4.8 is recommended
  - `/usr/ports/lang/gcc48`
  - Ensure that CPU_OPTS is selected (default is OFF)
- GNU make(`gmake`)
  - Installed automatically with `gcc48`

- `coreutils`
  - `/usr/ports/sysutils/coreutils`
- `libexecinfo` (Not required for FreeBSD* 10)
  - `/usr/src/contrib/libexecinfo`

When running the `make config-recursive` command, a dialog may be presented to the user. For the installation of the Intel® 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.

2.2 Running Intel® DPDK Applications

To run an Intel® 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 Intel® DPDK to use. Section 3.4, “Loading the Intel® DPDK contigmem Module” on page 8 for details on the loading of this module.

2.2.1 Using Intel® DPDK contigmem Module

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

```
root@host:~ # kenv hw.contigmem.num_buffers=n
root@host:~ # 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:

```
root@host:~ # 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.

Note: The `/boot/loader.conf` file may not exist, but can be created as a root user and should be given permissions as follows:

```
root@host:~ # chmod 644 /boot/loader.conf
```

§

3 Compiling the Intel® DPDK Target from Source

3.1 Install the Intel® DPDK and Browse Sources

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

```
user@host:~ # unzip DPDK-<version>zip
user@host:~ # cd DPDK-<version>
user@host:~ /DPDK # ls
app/ config/ examples/ lib/ LICENSE.GPL LICENSE.LGPL Makefile mk/ scripts/ tools/
```

The Intel® DPDK is composed of several directories:

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

3.2 Installation of the Intel® DPDK Target Environments

The format of an Intel® DPDK target is:

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

Where:

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

The configuration files for the Intel® 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 *Intel® DPDK Programmer’s Guide*.

To install and make the target, use `gmake install T=<target> CC=gcc48`.

For example to compile for FreeBSD* use:

```
gmake install T=x86_64-native-bsdapp-gcc CC=gcc48
```

To prepare a target without building it, for example, if the configuration changes need to be made before compilation, use the `gmake config T=<target>` command:

```
gmake config T=x86_64-native-bsdapp-gcc CC=gcc48
```

To build after configuration, change directory to `./x86_64-native-bsdapp-gcc` and use:

```
gmake CC=gcc48
```

3.3 Browsing the Installed Intel® DPDK Environment Target

Once a target is created, it contains all the libraries and header files for the Intel® 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:

```
user@host:~ /DPDK # ls x86_64-native-bsdapp-gcc
app     build   hostapp        include        kmod    lib        Makefile
```

3.4 Loading the Intel® DPDK contigmem Module

To run any Intel® DPDK application, the `contigmem` module must be loaded into the running kernel. The module is found in the `kmod` sub-directory of the Intel® DPDK target directory. The module can be loaded using `kldload` (assuming that the current directory is the Intel® 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.

3.5 Loading the Intel® DPDK nic_uio Module

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

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

Note: Currently loaded modules can be seen by using the kldstat command. A module can be removed from the running kernel by using kldunload <module_name>. While the nic_uio module can be loaded during boot, the module load order cannot be guaranteed and in the case where only some ports are bound to nic_uio and others remain in use by the original driver, it is necessary to load nic_uio after booting into the kernel, specifically after the original driver has been loaded.

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.

3.6 Binding Network Ports to the nic_uio Module

By default, the nic_uio module will take ownership of network ports if they are recognized Intel® DPDK devices and are not owned by another 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.

```
user@host:~ # 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:` 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`:

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

To restore the original device binding, it is necessary to reboot FreeBSD* if the original driver has been compiled into the kernel.

For example to rebind some or all ports to the original driver:

Update or remove the `hw.nic_uio.bdfs` entry in `/boot/loader.conf` if specified there for persistency, then;

```
reboot
```

If rebinding to a driver that is a loadable module, the network port binding can be reset without rebooting. This requires the unloading of the `nic_uio` module and the original driver.

Update or remove the `hw.nic_uio.bdfs` entry from `/boot/loader.conf` if specified there for persistency.

```
kldunload nic_uio
kldunload <original_driver>
kenv -u hw.nic_uio.bdfs, to remove all network ports from nic_uio and undefined
this system variable
```

OR

```
kenv hw.nic_uio.bdfs="b:d:f,b:d:f…" (to update nic_uio ports) kldload
<original_driver>
```

```
kldload nic_uio (if updating the list of associated network ports)
```

§

4 Compiling and Running Sample Applications

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

4.1 Compiling a Sample Application

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

When compiling an application in the FreeBSD* environment on the Intel® DPDK, the following variables must be exported:

- RTE_SDK - Points to the Intel® DPDK installation directory.
- RTE_TARGET - Points to the Intel® DPDK target environment directory. For FreeBSD*, this is the x86_64-native-bsdapp-gcc directory.

The following is an example of creating the helloworld application, which runs in the Intel® DPDK FreeBSD* environment. This 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 Intel® DPDK target environment, calls the various functions to initialize the Intel® 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.

```
user@host:~/DPDK$ cd examples/helloworld/
user@host:~/DPDK/examples/helloworld$ setenv RTE_SDK $HOME/DPDK
user@host:~/DPDK/examples/helloworld$ setenv RTE_TARGET x86_64-native-bsdapp-gcc
user@host:~/DPDK/examples/helloworld$ gmake CC=gcc48
CC main.o
LD helloworld
INSTALL-APP helloworld
INSTALL-MAP helloworld.map
user@host:~/DPDK/examples/helloworld$ ls build/app
helloworld helloworld.map
```

Note: In the above example, helloworld was in the directory structure of the Intel® DPDK. However, it could have been located outside the directory structure to keep the Intel® DPDK structure intact. In the following case, the helloworld application is copied to a new directory as a new starting point.

```
user@host:~$ setenv RTE_SDK /home/user/DPDK
user@host:~$ cp -r $(RTE_SDK)/examples/helloworld my_rte_app
user@host:~$ cd my_rte_app/
user@host:~$ setenv RTE_TARGET x86_64-native-bsdapp-gcc
user@host:~/my_rte_app$ gmake CC=gcc48
CC main.o
LD helloworld
INSTALL-APP helloworld
INSTALL-MAP helloworld.map
```

4.2 Running a Sample Application

Caution: The `contigmem` and `nic_uio` modules must be set up prior to running an application.

Caution: Any ports to be used by the application must be already bound to the nic_uio module, as described in section Section 3.6, "Binding Network Ports to the nic_uio Module," prior to running the application. The application is linked with the Intel® DPDK target environment's Environment Abstraction Layer (EAL) library, which provides some options that are generic to every Intel® 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>]
[-m MB] [-r NUM] [-v] [--file-prefix] [--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. The device and function separator used is a ":" rather than "." as seen with `pciconf` on FreeBSD*. 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 and the -n options are mandatory; the others are optional.

Copy the Intel® 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):

```
root@target:~$ ./helloworld -c f -n 4
```

Note: The --proc-type and --file-prefix EAL options are used for running multiple Intel® DPDK processes. See the “Multi-process Sample Application” chapter in the *Intel® DPDK Sample Applications User Guide and the Intel® DPDK Programmer’s Guide* for more details.

4.3 Running Intel® DPDK Applications Without Root Privileges

Although applications using the Intel® 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 Intel® 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 Intel® DPDK Release Notes for supported applications.

§