Intel® Data Plane Development Kit (Intel® DPDK)

Getting Started Guide

September 2013

Reference Number: 326002-004

Revision History

Date	Revision	Description
September 2013	-004	Supports public software release 1.5.0
August 2013	-003	Supports public software release 1.4.1
June 2013	-002	Supports public software release 1.3.1
November 2012	-001	Supports public software release 1.2.3

Contents

§ §

1.0 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 Linux* application (linuxapp) environment, without going deeply into detail.

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.

§ §

2.0 System Requirements

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

Note: If the Intel® DPDK is being used on an Intel® Communications Chipset 89xx Series platform, please consult the *Intel® Communications Chipset 89xx Series Software for Linux* Getting Started Guide*.

2.1 BIOS Setting Prerequisite

For the majority of platforms, no special BIOS settings are needed to use basic Intel® DPDK functionality. However, for additional HPET timer and power management functionality, BIOS setting changes may be needed. Consult Chapter 5.0, “Enabling Additional Functionality” for more information on the required changes.

2.2 Compilation of the Intel® DPDK

Required Tools:

Note: Testing has been performed using Fedora* 18. The setup commands and installed packages needed on other systems may be different. For details on other Linux distributions and the versions tested, please consult the *Intel® DPDK Release Notes*.

- GNU `make`
- coreutils: `cmp`, `sed`, `grep`, `arch`
- `gcc`: versions 4.5.x or later is recommended. On some distributions, some specific compiler flags and linker flags are enabled by default and affect performance (`-fstack-protector`, for example). Please refer to the documentation of your distribution and to `gcc -dumpspecs`.
- libc headers (`glibc-devel.i686` / `libc6-dev-i386`; `glibc-devel.x86_64` for 64-bit compilation)
- Linux kernel headers or sources required to build kernel modules. (kernel-devel.x86_64)
- Additional packages required for 32-bit compilation on 64-bit systems are: `glibc.i686`, `libgcc.i686`, `libstdc++.i686` and `glibc-devel.i686`
- Python, version 2.6 or 2.7, to use various helper scripts included in the Intel® DPDK package

Optional Tools:

- Intel® C++ Compiler (`icc`). For installation, additional libraries may be required. See the `icc` Installation Guide found in the `Documentation` directory under the compiler installation. This release has been tested using version 12.1.
- libpcap headers and libraries (libpcap-devel) to compile and use the libpcap-based poll-mode driver. This driver is disabled by default and can be enabled by setting `CONFIG_RTE_LIBRTE_PMD_PCAP=y` in the build time config file.

2.3 Running Intel® DPDK Applications

To run an Intel® DPDK application, some customization may be required on the target machine.

2.3.1 System Software

Required:

- Kernel version >= 2.6.33

  The kernel version in use can be checked using the command:

  ```
  uname -r
  ```

  For details of the patches needed to use the Intel® DPDK with earlier kernel versions, see the Intel® DPDK FAQ included in the *Intel® DPDK Release Notes*. Note also that Redhat* Linux 6.2 and 6.3 uses a 2.6.32 kernel that already has all the necessary patches applied.

- `glibc` >= 2.7 (for features related to `cpuset`)

  The version can be checked using the `ldd --version` command. A sample output is shown below:

  ```
  # ldd --version
  ```

```
ldd (GNU libc) 2.14.90
Copyright (C) 2011 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Written by Roland McGrath and Ulrich Drepper.
```

- Kernel configuration

  In the Fedora OS and other common distributions, such as Ubuntu*, or RedHat Enterprise Linux, the vendor supplied kernel configurations can be used to run most Intel® DPDK applications.

  For other kernel builds, options which should be enabled for Intel® DPDK include:

  — `UIO` support,

  — `HUGETLBFS`

  — `PROC_PAGE_MONITOR` support.

  — `HPET` and `HPET_MMAP` configuration options should also be enabled if HPET support is required. See Section 5.1“High Precision Event Timer (HPET) Functionality” for more details.

2.3.2 Use of Hugepages in the Linux Environment

Hugepage support is required for the large memory pool allocation used for packet buffers (the `HUGETLBFS` option must be enabled in the running kernel as indicated in Section 2.3). 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.

2.3.2.1 Reserving Hugepages for Intel® DPDK Use

The allocation of hugepages should be done at boot time or as soon as possible after system boot to prevent memory from being fragmented in physical memory. To reserve hugepages at boot time, a parameter is passed to the Linux kernel on the kernel command line.

For 2 MB pages, just pass the `hugepages` option to the kernel. For example, to reserve 1024 pages of 2 MB, use:

```
hugepages=1024
```

For other hugepage sizes, for example 1G pages, the size must be specified explicitly and can also be optionally set as the default hugepage size for the system. For example, to reserve 4G of hugepage memory in the form of four 1G pages, the following options should be passed to the kernel:

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

Note: The hugepage sizes that a CPU supports can be determined from the CPU flags. If `pse` exists, 2M hugepages are supported; if `pdpe1gb` exists, 1G hugepages are supported.

Note: For 64-bit applications, it is recommended to use 1 GB hugepages if the platform supports them.

In the case of a dual-socket NUMA system, the number of hugepages reserved at boot time is generally divided equally between the two sockets (on the assumption that sufficient memory is present on both sockets).

See the `Documentation/kernel-parameters.txt` file in your Linux source tree for further details of these and other kernel options.

Alternative:

For 2 MB pages, there is also the option of allocating hugepages 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
```

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

```
echo 1024 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
echo 1024 > /sys/devices/system/node/node1/hugepages/hugepages-2048kB/nr_hugepages
```

Note: For 1G pages, it is not possible to reserve the hugepage memory after the system has booted.

2.3.2.2 Using Hugepages with the Intel® DPDK

Once the hugepage memory is reserved, to make the memory available for Intel® DPDK use, perform the following steps:

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

The mount point can be made permanent across reboots, by adding the following line to the `/etc/fstab` file:

```
nodev   /mnt/huge hugetlbfs       defaults 0 0
```

For 1GB pages, the page size must be specified as a mount option:

```
nodev   /mnt/huge_1GB   hugetlbfs       pagesize=1GB 0 0
```

§ §

3.0 Compiling the Intel® DPDK Target from Source

Note: Parts of this process can also be done using the setup script described in Chapter 6.0 of this document.

3.1 Install the Intel® DPDK and Browse Sources

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

```
user@host:~$ unzip DPDK.L<version>.zip
user@host:~$ cd DPDK
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 application examples
- `config`, `tools`, `scripts`, `mk`: Framework-related makefiles, scripts and configuration

3.2 Installation of Intel® DPDK Target Environments

The format of a Intel® DPDK target is:

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

where:

- `ARCH` can be: `i686`, `x86_64`
- `MACHINE` can be: `default`
- `EXECENV` can be: `linuxapp`
- `TOOLCHAIN` can be: `gcc`, `icc`

The targets to be installed depend on the 32-bit and/or 64-bit packages and compilers installed on the host. Available targets can be found in the `DPDK/config` directory. The `defconfig_` prefix should not be used.

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*.

When using the Intel® C++ Compiler (`icc`), one of the following commands should be invoked for 64-bit or 32-bit use respectively. Notice that the shell scripts update the `$PATH` variable and therefore should not be performed in the same session. Also, verify the compiler's installation directory since the path may be different:

```
source /opt/intel/bin/iccvars.sh intel64
source /opt/intel/bin/iccvars.sh ia32
```

To install and make targets, use the `make install T=<target>` command in the top-level Intel® DPDK directory.

For example, to compile a 64-bit target using `icc`, run:

```
make install T=x86_64-default-linuxapp-icc
```

To compile a 32-bit build using `gcc`, the make command should be:

```
make install T=i686-default-linuxapp-gcc
```

To compile all 64-bit targets using `gcc`, use:

```
make install T=x86_64*gcc
```

To compile all 64-bit targets using both `gcc` and `icc`, use:

```
make install T=x86_64-*
```

Note: The wildcard operator (*) can be used to create multiple targets at the same time.

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

```
make config T=x86_64-default-linuxapp-gcc
```

Warning: The `igb_uio` module must be compiled with the same kernel as the one running on the target. If the Intel® DPDK is not being built on the target machine, the `RTE_KERNELDIR` environment variable should be used to point the compilation at a copy of the kernel version to be used on the target machine.

Once the target environment is created, the user may move to the target environment directory and continue to make code changes and re-compile. The user may also make modifications to the compile-time Intel® DPDK configuration by editing the `.config` file in the build directory. (This is a build-local copy of the `defconfig` file from the top-level config directory).

```
cd x86_64-default-linuxapp-gcc
vi .config
make
```

In addition, the `make clean` command can be used to remove any existing compiled files for a subsequent full, clean rebuild of the code.

3.3 Browsing the Installed Intel® DPDK Environment Target

Once a target is created it contains all 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. In the case of Linux, a `kmod` directory is also present that contains a module to install:

```
$ ls x86_64-default-linuxapp-gcc
app  build  hostapp  include  kmod  lib  Makefile
```

3.4 Loading the Intel® DPDK igb_uio Module

To run any Intel® DPDK application, the `igb_uio` module must be loaded into the running kernel. The module is found in the `kmod` sub-directory of the Intel® DPDK target directory. This module should be loaded using the `insmod` command as shown below (assuming that the current directory is the Intel® DPDK target directory). In many cases, the `uio` support in the Linux kernel is compiled as a module rather than as part of the kernel, so it is often necessary to load the `uio` module first:

```
sudo modprobe uio
sudo insmod kmod/igb_uio.ko
```

3.5 Binding and Unbinding Network Ports to/from the igb_uio Module

As of release 1.4, Intel® DPDK applications no longer automatically unbind all supported network ports from the kernel driver in use. Instead, all ports that are to be used by an Intel® DPDK application must be bound to the `igb_uio` module before the application is run. Any network ports under Linux* control will be ignored by the Intel® DPDK poll-mode drivers and cannot be used by the application.

Warning: The Intel® DPDK will, by default, no longer automatically unbind network ports from the kernel driver at startup. Any ports to be used by an Intel® DPDK application must be unbound from Linux* control and bound to the `igb_uio` module before the application is run.

To bind ports to the `igb_uio` module for Intel® DPDK use, and then subsequently return ports to Linux* control, a utility script called `pci_unbind.py` is provided in the `tools` subdirectory. This utility can be used to provide a view of the current state of the network ports on the system, and to bind and unbind those ports from the different kernel modules, including `igb_uio`. The following are some examples of how the script can be used. A full description of the script and its parameters can be obtained by calling the script with the `--help` or `--usage` options.

Warning: While any user can run the `pci_unbind.py` script to view the status of the network ports, binding or unbinding network ports requires root privileges.

To see the status of all network ports on the system:

```
root@host:DPDK# ./tools/pci_unbind.py --status

Network devices using IGB_UIO driver
====================================
0000:82:00.0 '82599EB 10-Gigabit SFI/SFP+ Network Connection' drv=igb_uio unused=ixgbe
0000:82:00.1 '82599EB 10-Gigabit SFI/SFP+ Network Connection' drv=igb_uio unused=ixgbe

Network devices using kernel driver
===================================
0000:04:00.0 'I350 Gigabit Network Connection' if=em0 drv=igb unused=igb_uio *Active*
0000:04:00.1 'I350 Gigabit Network Connection' if=eth1 drv=igb unused=igb_uio
0000:04:00.2 'I350 Gigabit Network Connection' if=eth2 drv=igb unused=igb_uio
0000:04:00.3 'I350 Gigabit Network Connection' if=eth3 drv=igb unused=igb_uio

Other network devices
=====================
<none>
```

To bind device eth1, `04:00.1`, to the `igb_uio` driver:

```
root@host:DPDK# ./tools/pci_unbind.py --bind=igb_uio 04:00.1
```

or, alternatively,

```
root@host:DPDK# ./tools/pci_unbind.py --bind=igb_uio eth1
```

To restore device `82:00.0` to its original kernel binding:

```
root@host:DPDK# ./tools/pci_unbind.py --bind=ixgbe 82:00.0
```

§ §

4.0 Compiling and Running Sample Applications

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

Note: Parts of this process can also be done using the setup script described in Chapter 6.0 of this document.

4.1 Compiling a Sample Application

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

When compiling an application in the Linux* 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.

The following is an example of creating the `helloworld` application, which runs in the Intel® DPDK Linux 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$ export RTE_SDK=$HOME/DPDK
user@host:~/DPDK/examples/helloworld$ export RTE_TARGET=x86_64-default-linuxapp-gcc
user@host:~/DPDK/examples/helloworld$ make
  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:~$ export RTE_SDK=/home/user/DPDK
user@host:~$ cp -r $(RTE_SDK)/examples/helloworld my_rte_app
user@host:~$ cd my_rte_app/
user@host:~$ export RTE_TARGET=x86_64-default-linuxapp-gcc
user@host:~/my_rte_app$ make
  CC main.o
  LD helloworld
  INSTALL-APP helloworld
  INSTALL-MAP helloworld.map
```

4.2 Running a Sample Application

Warning: The UIO drivers and hugepages must be setup prior to running an application.

Warning: Any ports to be used by the application must be already bound to the `igb_uio` module, as described in section Section 3.5, prior to running the application.

The application is linked with the Intel® DPDK target environment's Environmental 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>] [--socket-mem=MB,...]
[-m MB] [-r NUM] [-v] [--file-prefix] [--proc-type <primary|secondary|auto>]
```

The EAL options are as follows:

- `-c COREMASK`: An 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
- `--socket-mem`: Memory to allocate from hugepages on specific sockets
- `-m MB`: Memory to allocate from hugepages, regardless of processor socket. It is recommended that --socket-mem be used instead of this option.
- `-r NUM`: Number of memory ranks
- `-v`: Display version information on startup
- `--huge-dir`: The directory where hugetlbfs is mounted
- `--file-prefix`: The prefix text used for hugepage filenames
- `--proc-type`: The type of process instance

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 per processor socket, and that cores 0-3 are present and are to be used for running the application):

```
user@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.

4.2.1 Logical Core use by Applications

The `coremask` parameter is always mandatory for Intel® DPDK applications. Each bit of the mask corresponds to the equivalent logical core number as reported by Linux. Since these logical core numbers, and their mapping to specific cores on specific NUMA sockets, can vary from platform to platform, it is recommended that the core layout for each platform be considered when choosing the `coremask` to use in each case.

On initialization of the EAL layer by an Intel® DPDK application, the logical cores to be used and their socket location are displayed. This information can also be determined for all cores on the system by examining the `/proc/cpuinfo` file, for example, by running `cat /proc/cpuinfo`. The `physical id` attribute listed for each processor indicates the CPU socket to which it belongs. This can be useful when using other processors to understand the mapping of the logical cores to the sockets.

Note: A more graphical view of the logical core layout may be obtained using the `lstopo` Linux utility. On Fedora* 18, this may be installed and run using the following command:

```
sudo yum install hwloc
./lstopo
```

Warning: The logical core layout can change between different board layouts and should be checked before selecting an application `coremask`.

4.2.2 Hugepage Memory Use by Applications

When running an application, it is recommended to use the same amount of memory as that allocated for hugepages. This is done automatically by the Intel® DPDK application at startup, if no `-m` or `--socket-mem` parameter is passed to it when run.

If more memory is requested by explicitly passing a `-m` or `--socket-mem` value, the application fails. However, the application itself can also fail if the user requests less memory than the reserved amount of hugepage-memory, particularly if using the `-m` option. The reason is as follows. Suppose the system has 1024 reserved 2 MB pages in socket 0 and 1024 in socket 1. If the user requests 128 MB of memory, the 64 pages may not match the constraints:

- The hugepage memory by be given to the application by the kernel in socket 1 only. In this case, if the application attempts to create an object, such as a ring or memory pool in socket 0, it fails. To avoid this issue, it is recommended that the `--socket-mem` option be used instead of the `-m` option.
- These pages can be located anywhere in physical memory, and, although the Intel® DPDK EAL will attempt to allocate memory in contiguous blocks, it is possible that the pages will not be contiguous. In this case, the application is not able to allocate big memory pools.

The `socket-mem` option can be used to request specific amounts of memory for specific sockets. This is accomplished by supplying the `--socket-mem` flag followed by amounts of memory requested on each socket, for example, supply `--socket-mem=0,512` to try and reserve 512 MB for socket 1 only. Similarly, on a four socket system, to allocate 1 GB memory on each of sockets 0 and 2 only, the parameter `--socket-mem=1024,0,1024` can be used. No memory will be reserved on any CPU socket that is not explicitly referenced, for example, socket 3 in this case. If the Intel® DPDK cannot allocate enough memory on each socket, the EAL initialization fails.

4.3 Additional Sample Applications

Additional sample applications are included in the `${RTE_SDK}/examples` directory. These sample applications may be built and run in a manner similar to that described in earlier sections in this manual. In addition, see the *Intel® DPDK Sample Applications User Guide* for a description of the application, specific instructions on compilation and execution and some explanation of the code.

4.4 Additional Test Applications

In addition, there are two other applications that are built when the libraries are created. The source for these are in the `DPDK/app` directory and are called `test` and `testpmd`. Once the libraries are created, they can be found in the `build/app` directory.

- The `test` application provides a variety of specific tests for the various functions in the Intel® DPDK.

- The `testpmd` application provides a number of different packet throughput tests and examples of features such as how to use the Flow Director found in the Intel® 82599 10 Gigabit Ethernet Controller.

§ §

5.0 Enabling Additional Functionality

5.1 High Precision Event Timer (HPET) Functionality

5.1.1 BIOS Support

The High Precision Timer (HPET) must be enabled in the platform BIOS if the HPET is to be used. Otherwise, the Time Stamp Counter (TSC) is used by default. The BIOS is typically accessed by pressing F2 while the platform is starting up. The user can then navigate to the HPET option. On the Crystal Forest platform BIOS, the path is: **Advanced** -> **PCH-IO Configuration** -> **High Precision Timer** -> (Change from Disabled to Enabled if necessary).

On a system that has already booted, the following command can be issued to check if HPET is enabled:

```
# grep hpet /proc/timer_list
```

If no entries are returned, HPET must be enabled in the BIOS (as per the instructions above) and the system rebooted.

5.1.2 Linux Kernel Support

The Intel® DPDK makes use of the platform HPET timer by mapping the timer counter into the process address space, and as such, requires that the `HPET_MMAP` kernel configuration option be enabled.

Warning: On Fedora*, and other common distributions such as Ubuntu*, the `HPET_MMAP` kernel option is not enabled by default. To recompile the Linux kernel with this option enabled, please consult the distribution’s documentation for the relevant instructions.

5.1.3 Enabling HPET in the Intel® DPDK

By default, HPET support is disabled in the Intel® DPDK build configuration files. To use HPET, the `CONFIG_RTE_LIBEAL_USE_HPET` setting should be changed to “y”, which will enable the HPET settings at compile time.

For an application to use the `rte_get_hpet_cycles`() and `rte_get_hpet_hz`() API calls, and optionally to make the HPET the default time source for the `rte_timer` library, the new `rte_eal_hpet_init`() API call should be called at application initialization. This API call will ensure that the HPET is accessible, returning an error to the application if it is not, for example, if `HPET_MMAP` is not enabled in the kernel. The application can then determine what action to take, if any, if the HPET is not available at run-time.

Note: For applications that require timing APIs, but not the HPET timer specifically, it is recommended that the `rte_get_timer_cycles`() and `rte_get_timer_hz`() API calls be used instead of the HPET-specific APIs. These generic APIs can work with either TSC or HPET time sources, depending on what is requested by an application call to `rte_eal_hpet_init()`, if any, and on what is available on the system at runtime.

5.2 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 Linux file system objects should be adjusted to ensure that the Linux user account being used to run the Intel® DPDK application has access to them:

- All directories which serve as hugepage mount points, for example, `/mnt/huge`
- The userspace-io device files in `/dev`, for example, `/dev/uio0`, `/dev/uio1`, and so on
- If the HPET is to be used, `/dev/hpet`

Note: On some Linux installations, `/dev/hugepages` is also a hugepage mount point created by default.

5.3 Power Management and Power Saving Functionality

Enhanced Intel SpeedStep® Technology must be enabled in the platform BIOS if the power management feature of Intel® DPDK is to be used. Otherwise, the `sys` file folder `/sys/devices/system/cpu/cpu0/cpufreq` will not exist, and the CPU frequency-based power management cannot be used. Consult the relevant BIOS documentation to determine how these settings can be accessed.

For example, on some Intel reference platform BIOS variants, the path to Enhanced Intel SpeedStep® Technology is:

Advanced->Processor Configuration->Enhanced Intel SpeedStep® Tech

In addition, C3 and C6 should be enabled as well for power management. The path of C3 and C6 on the same platform BIOS is:

Advanced->Processor Configuration->Processor C3

Advanced->Processor Configuration-> Processor C6

5.4 Using Linux Core Isolation to Reduce Context Switches

While the threads used by an Intel® DPDK application are pinned to logical cores on the system, it is possible for the Linux scheduler to run other tasks on those cores also. To help prevent additional workloads from running on those cores, it is possible to use the `isolcpus` Linux kernel parameter to isolate them from the general Linux scheduler. For example, if Intel® DPDK applications are to run on logical cores 2, 4 and 6, the following should be added to the kernel parameter list:

```
isolcpus=2,4,6
```

5.5 Loading the Intel® DPDK KNI Kernel Module

To run the Intel® DPDK Kernel NIC Interface (KNI) sample application, an extra kernel module (the `kni` module) must be loaded into the running kernel. The module is found in the `kmod` sub-directory of the Intel® DPDK target directory. Similar to the loading of the `igb_uio` module, this module should be loaded using the `insmod` command as shown below (assuming that the current directory is the Intel® DPDK target directory):

```
#insmod kmod/rte_kni.ko
```

Note: See the “Kernel NIC Interface Sample Application” chapter in the *Intel® DPDK Sample Applications User Guide* for more details.

5.6 Using Linux IOMMU Pass-Through to Run Intel® DPDK with Intel® VT-d

To enable Intel® VT-d in a Linux kernel, a number of kernel configuration options must be set. These include:

- `IOMMU_SUPPORT`
- `IOMMU_API`
- `INTEL_IOMMU`

In addition, to run the Intel® DPDK with Intel® VT-d, the `iommu=pt` kernel parameter must be used. This results in pass-through of the DMAR (DMA Remapping) 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 too. This ensures that the Intel IOMMU is being initialized as expected.

§ §

6.0 Quick Start Setup Script

The `setup.sh` script, found in the `tools` subdirectory, allows the user to perform the following tasks:

- Build the Intel® DPDK libraries
- Insert and remove the Intel® DPDK UIO kernel module
- Create and delete hugepages for NUMA and UMA cases
- View network port status and reserve ports for Intel® DPDK application use
- Run the `test` and `testpmd` applications
- Look at hugepages in the meminfo
- List hugepages in `/mnt/huge`
- Remove built Intel® DPDK libraries

Once these steps have been completed for one of the EAL targets, the user may compile their own application that links in the EAL libraries to create the Intel® DPDK image.

6.1 Script Organization

The `setup.sh` script is logically organized into a series of steps that a user performs in sequence. Each step provides a number of options that guide the user to completing the desired task. The following is a brief synopsis of each step.

Step 1: Build DPDK Libraries

Initially, the user must select an Intel® DPDK target to choose the correct target type and compiler options to use when building the libraries.

The user must have all libraries, modules, updates and compilers installed in the system prior to this, as described in the earlier chapters in this Getting Started Guide.

Step 2: Setup Environment

The user configures the Linux* environment to support the running of Intel® DPDK applications. Hugepages can be set up for NUMA or non-NUMA systems. Any existing hugepages will be removed. The Intel® DPDK kernel module that is needed can also be inserted in this step, and network ports may be bound to this module for Intel® DPDK application use.

Step 3: Run an Application

The user may run the `test` application once the other steps have been performed. The `test` application allows the user to run a series of functional tests for the Intel® DPDK. The `testpmd` application, which supports the receiving and sending of packets, can also be run.

Step 4: Examining the System

This step provides some tools for examining the status of hugepage mappings.

Step 5: System Cleanup

The final step has options for restoring the system to its original state.

6.2 Use Cases

The following are some example of how to use the `setup.sh` script. The script should be run using the `source` command. Some options in the script prompt the user for further data before proceeding.

Warning: The `setup.sh` script should be run with root privileges.

```
user@host:~/rte$ source tools/setup.sh
------------------------------------------------------------------------------
RTE_SDK exported as /home/user/rte
------------------------------------------------------------------------------
----------------------------------------------------------
Step 1: Select the DPDK environment to build
----------------------------------------------------------
[1] i686-default-linuxapp-gcc
[2] i686-default-linuxapp-icc
[3] x86_64-default-linuxapp-gcc
[4] x86_64-default-linuxapp-icc

----------------------------------------------------------
Step 2: Setup linuxapp environment
----------------------------------------------------------
[5] Insert IGB UIO module
[6] Insert KNI module
[7] Setup hugepage mappings for non-NUMA systems
[8] Setup hugepage mappings for NUMA systems
[9] Display current Ethernet device settings
[10] Bind Ethernet device to IGB UIO module

----------------------------------------------------------
Step 3: Run test application for linuxapp environment
----------------------------------------------------------
[11] Run test application ($RTE_TARGET/app/test)
[12] Run testpmd application in interactive mode ($RTE_TARGET/app/testpmd)
```

```
----------------------------------------------------------
Step 4: Other tools
----------------------------------------------------------
[13] List hugepage info from /proc/meminfo

----------------------------------------------------------
Step 5: Uninstall and system cleanup
----------------------------------------------------------
[14] Uninstall all targets
[15] Unbind NICs from IGB UIO driver
[16] Remove IGB UIO module
[17] Remove KNI module
[18] Remove hugepage mappings

[19] Exit Script

Option:
```

The following selection demonstrates the creation of the x86_64-default-linuxapp-gcc Intel® DPDK library.

```
Option: 3

================== Installing x86_64-default-linuxapp-gcc
Configuration done
== Build scripts
== Build scripts/testhost
HOSTCC testhost.o

...

Build complete
------------------------------------------------------------------------------
RTE_TARGET exported as x86_64-default-linuxapp-gcc
------------------------------------------------------------------------------
```

The following selection demonstrates the starting of the Intel® DPDK UIO driver.

```
Option: 5

Unloading any existing DPDK UIO module
Loading DPDK UIO module
```

The following selection demonstrates the creation of hugepages in a NUMA system. 1024 2 Mbyte pages are assigned to each node. The result is that the application should use `-m 4096` for starting the application to access both memory areas (this is done automatically if the `-m` option is not provided).

Note: If prompts are displayed to remove temporary files, type 'y'.

```
Option: 8

Removing currently reserved hugepages
nmounting /mnt/huge and removing directory

Input the number of 2MB pages for each node
Example: to have 128MB of hugepages available per node,
enter '64' to reserve 64 * 2MB pages on each node
Number of pages for node0: 1024
Number of pages for node1: 1024
Reserving hugepages
Creating /mnt/huge and mounting as hugetlbfs
```

The following selection demonstrates the launch of the `test` application to run on a single core.

```
Option: 9

Enter hex bitmask of cores to execute test app on
Example: to execute app on cores 0 to 7, enter 0xff
bitmask: 0x01
Launching app
EAL: coremask set to 1
EAL: Detected lcore 0 on socket 0

...

EAL: Master core 0 is ready (tid=1b2ad720)
RTE>>
```

6.3 Applications

Once the user has run the `setup.sh` script, built one of the EAL targets and set up hugepages (if using one of the Linux EAL targets), the user can then move on to building and running their application or one of the examples provided.

The examples in the `/examples` directory provide a good starting point to gain an understanding of the operation of the Intel® DPDK. The following command sequence shows how the `helloworld` sample application is built and run. As recommended in Section 4.2.1, "Logical Core use by Applications" on page 13, the logical core layout of the platform should be determined when selecting a core mask to use for an application.

```
rte@rte-desktop:~/rte/examples$ cd helloworld/

rte@rte-desktop:~/rte/examples/helloworld$ make
CC main.o
LD helloworld
INSTALL-APP helloworld
INSTALL-MAP helloworld.map

rte@rte-desktop:~/rte/examples/helloworld$ sudo ./build/app/helloworld -c 0xf -n 3
[sudo] password for rte:
EAL: coremask set to f
EAL: Detected lcore 0 as core 0 on socket 0
EAL: Detected lcore 1 as core 0 on socket 1
EAL: Detected lcore 2 as core 1 on socket 0
EAL: Detected lcore 3 as core 1 on socket 1
EAL: Setting up hugepage memory...
EAL: Ask a virtual area of 0x200000 bytes
EAL: Virtual area found at 0x7f0add800000 (size = 0x200000)
EAL: Ask a virtual area of 0x3d400000 bytes
EAL: Virtual area found at 0x7f0aa0200000 (size = 0x3d400000)
EAL: Ask a virtual area of 0x400000 bytes
EAL: Virtual area found at 0x7f0a9fc00000 (size = 0x400000)
EAL: Ask a virtual area of 0x400000 bytes
EAL: Virtual area found at 0x7f0a9f600000 (size = 0x400000)
EAL: Ask a virtual area of 0x400000 bytes
EAL: Virtual area found at 0x7f0a9f000000 (size = 0x400000)
EAL: Ask a virtual area of 0x800000 bytes
EAL: Virtual area found at 0x7f0a9e600000 (size = 0x800000)
EAL: Ask a virtual area of 0x800000 bytes
EAL: Virtual area found at 0x7f0a9dc00000 (size = 0x800000)
EAL: Ask a virtual area of 0x400000 bytes
EAL: Virtual area found at 0x7f0a9d600000 (size = 0x400000)
EAL: Ask a virtual area of 0x400000 bytes
EAL: Virtual area found at 0x7f0a9d000000 (size = 0x400000)
```

```
EAL: Ask a virtual area of 0x400000 bytes
EAL: Virtual area found at 0x7f0a9ca00000 (size = 0x400000)
EAL: Ask a virtual area of 0x200000 bytes
EAL: Virtual area found at 0x7f0a9c600000 (size = 0x200000)
EAL: Ask a virtual area of 0x200000 bytes
EAL: Virtual area found at 0x7f0a9c200000 (size = 0x200000)
EAL: Ask a virtual area of 0x3fc00000 bytes
EAL: Virtual area found at 0x7f0a5c400000 (size = 0x3fc00000)
EAL: Ask a virtual area of 0x200000 bytes
EAL: Virtual area found at 0x7f0a5c000000 (size = 0x200000)
EAL: Requesting 1024 pages of size 2MB from socket 0
EAL: Requesting 1024 pages of size 2MB from socket 1
EAL: Master core 0 is ready (tid=de25b700)
EAL: Core 1 is ready (tid=5b7fe700)
EAL: Core 3 is ready (tid=5a7fc700)
EAL: Core 2 is ready (tid=5affd700)
hello from core 1
hello from core 2
hello from core 3
hello from core 0
```

§ §