NVIDIA DOCA DPACC Compiler

User Guide

MLNX-15-060563 _v1.5.2 | July 2023

Table of Contents

Chapter 1. Introduction 1
1.1. Glossary 1
1.2. Offloading Work on DPA 2
1.3. Writing DPA Applications 2
1.3.1. Restrictions on DPA Code 2
1.3.2. DPA RPC Functions 2
1.3.3. DPA Kernels 3
1.3.4. Characteristics of DPA Kernels 3
1.3.5. Handling User-defined Data Types 3
1.3.6. Characteristics of Annotated Types 3
Chapter 2. Prerequisites 4
2.1. Supported Platforms 4
Chapter 3. Description 5
3.1. DPACC Inputs and Outputs 5
3.1.1. DPA Program 6
3.1.2. DPA Object 6
3.2. Modes of Operation 6
3.2.1. Compile-and-link Mode 7
3.2.2. Compile-only Mode 7
Chapter 4. Execution 9
4.1. Mandatory Arguments 9
4.2. Commonly Used Arguments 9
4.3. Incorrect Usage 10
4.4. Examples 10
4.4.1. Link with Device Libraries 10
4.4.2. Include Headers 11
4.5. DPA Compiler Usage 11
4.5.1. Compiler Driver Command Line Options 11
4.5.2. Linker Command Line Options 11
4.5.3. Objdump Command Line Options 11
4.5.4. Archiver Command Line Options 12
4.5.5. NM Tool Command Line Options 12
4.5.6. Common Compiler Options 12
4.5.7. Common Linker Options 12
4.5.8. Debugging Options 12

4.5.9. Miscellaneous Notes..... 12

Chapter 1. Introduction

DPACC is a high-level compiler for the DPA processor. It compiles code targeted for the DPA processor into an executable and generates a DPA program.

The DPA program is a host library with interfaces encapsulating the DPA executable. This DPA program is linked with the host application to generate a host executable. The host executable can invoke the DPA code through FlexIO runtime API.

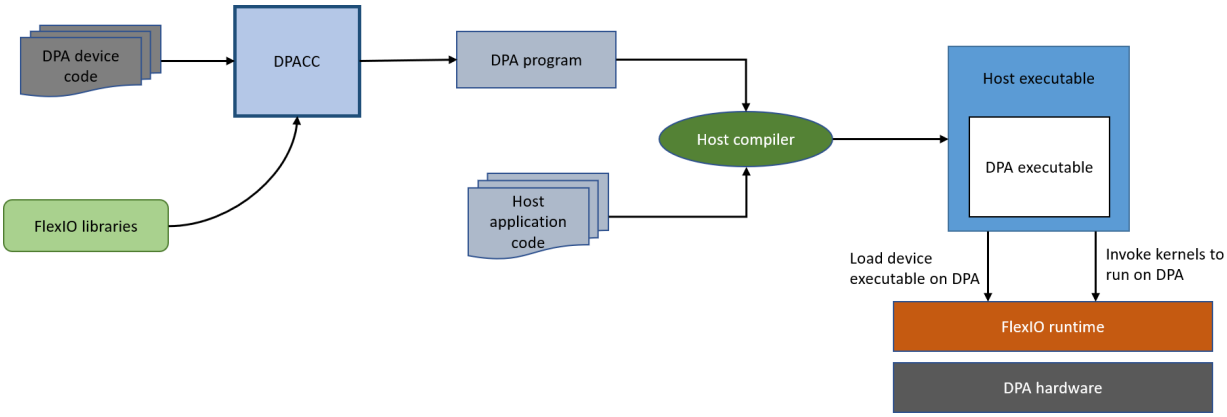
Producing a DPA program involves steps such as input file preprocessing, validation, interface generation, compilation and linking. DPACC hides these intricate details and provides a one-step solution to enable a seamless programming experience.

DPACC uses dpa-clang to compile code targeted for DPA. dpa-clang is part of the DPA toolchain package which is an LLVM-based cross-compiling bare-metal toolchain. It provides Clang compiler, LLD linker targeting DPA architecture, and other utility tools.

1.1. Glossary

Term	Definition
Device	DPA simulator/hardware
Host	CPU that launches the device code to run on the DPA
Device function	Any C function that runs on the DPA device
Kernel	Device function that is the point of entry from the host when offloading any work on DPA
Host compiler	Compiler used to compile the code targeting the host CPU
Host stubs	Interfaces (functions and data structures) used for argument marshalling and loading of the DPA executable
Device compiler	Compiler used to compile code targeting the DPA
DPA program	Host library that encapsulates the DPA device executable (ELF) and host stubs which are used to access the DPA executable with

1.2. Offloading Work on DPA



To offload tasks on the DPA, the following things are required:

- DPA device code – C programs, targeted to run on the DPA. DPA device code may contain one or more kernel entry functions.
- Host application code – the host application is responsible for initializing the device using appropriate FlexIO runtime calls and invoking kernels in the DPA executable. Kernel registration and interface with FlexIO runtime is managed by DPACC.
- Runtime – FlexIO runtime libraries and headers supplied to DPACC through the appropriate options

The generated DPA program, when linked with host application, results in an executable containing both the host and DPA executables. Running this executable loads the DPA executable to the device memory.

1.3. Writing DPA Applications

DPA can be programmed using the FlexIO API.

DPA device code is a C code with some restrictions and special definitions.

1.3.1. Restrictions on DPA Code

Use of thread local storage is not allowed for any variables.

1.3.2. DPA RPC Functions

A remote procedure call function is a synchronous call that triggers work in DPA and waits for its completion. It is annotated with a `__dpa_rpc__` attribute. For more information, please refer to NVIDIA DOCA FlexIO SDK Programming Guide.

1.3.3. DPA Kernels

A kernel function is a device function meant to be called from the host code. Kernels are annotated with a `__dpa_global__` attribute.

1.3.4. Characteristics of DPA Kernels

- Kernels cannot explicitly return a value. They must have `void` return type.
- Kernels cannot accept pointers and arrays as arguments
- Kernels cannot accept a variable number of arguments
- Inline specifier is not allowed on kernel functions

1.3.5. Handling User-defined Data Types

User-defined data types, when used as kernel arguments, require special handling. They must be annotated with a `__dpa_global__` attribute.

If the user-defined data type is `typedef`'d, the `typedef` statement must be annotated with a `__dpa_global__` attribute along the data type itself.

1.3.6. Characteristics of Annotated Types

- They must have a copy of the definition in all translation units where they are used as kernel arguments
- They cannot have pointers, variable length arrays, and flexible arrays as members
- Fixed-size arrays as C structure members are supported
- These characteristics apply recursively to any user-defined/`typedef`'d types that are members of an annotated type

DPACC processes all kernels and annotated data structures and generates host and device interfaces to facilitate the kernel launch.

Chapter 2. Prerequisites

Package	Instructions
Host compiler	Compiler specified through `hostcc` option. Both `gcc` and `clang` are supported.
Device compiler	Compiler which supports DPA target specified through `devicecc` option. The preferred device compiler is "DPA compiler". Installing DPACC package also installs DPA compiler binaries: `dpa-clang`, `dpa-ar`, `dpa-nm` and `dpa-objdump`.
FlexIO SDK and C library	Available as part of the DOCA software package. DPA toolchain does not provide C library and corresponding headers. Users are expected to use the C library for DPA from the FlexIO SDK.

2.1. Supported Platforms

Architecture	Operating Systems
x86_64	Ubuntu 20.04
	Ubuntu 22.04
	CentOS 8.2
	RHEL 8.2
arm64	Ubuntu 20.04
	Ubuntu 22.04

Chapter 3. Description

3.1. DPACC Inputs and Outputs

DPACC can produce DPA programs in a single command by accepting all source files as input.

Additionally, DPACC offers the flexibility of producing DPA object files from individual source files.

DPA object files contain both host stub objects and device objects. These DPA object files can later be given to DPACC as input to produce the DPA library.

Phase	Option Name	Default Output File Name
Compile input device code files to DPA object files	`--compile` or `-c`	`.dpa.o` appended to the name of each input source file
Compile and link the input device code files/ DPA object files, and produce a DPA program	No specific option	No default name, output file name must be specified

DPACC can accept following two file types as input:

File Extension	File Type	Description
`.c`	C source file	DPA device code
`.dpa.o`	DPA object file	Object file generated by DPACC, containing both host and device objects

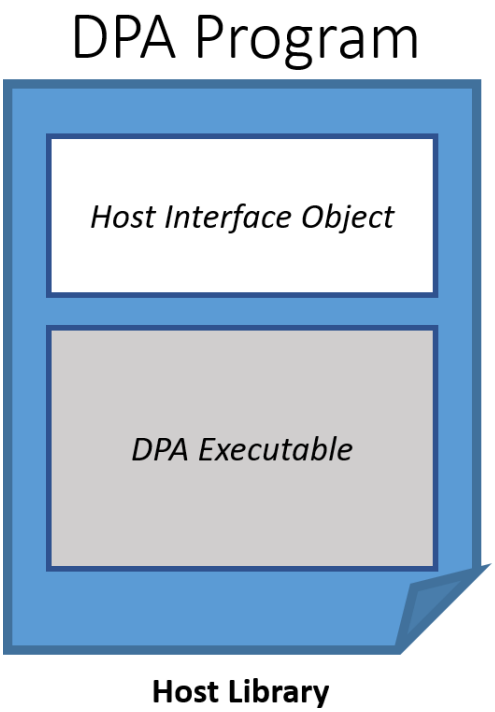
Based on the mode of operations, DPACC can generate the two following output files:

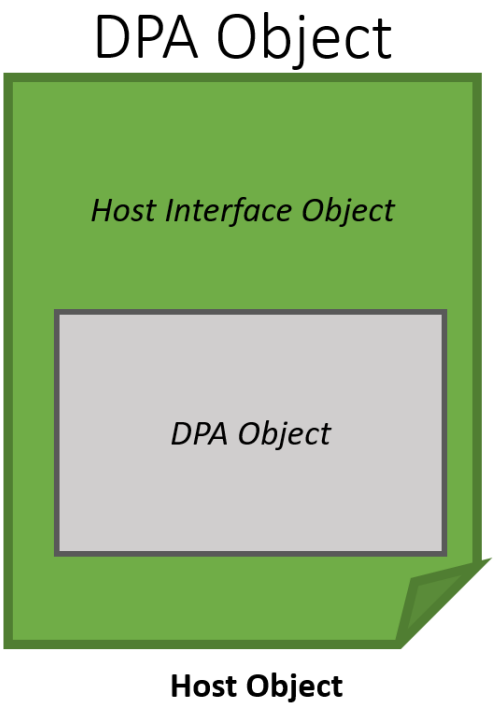
Output File Type	Input Files
DPA program	C source files and/or DPA object files
DPA object file	C source files

3.1.1. DPA Program

DPACC produces a DPA program in compile-and-link mode. A DPA program is a library built for the host machine. This library contains a single object file which comprises of the host stubs which facilitate invoking a kernel from the host application.

Additionally, the device executable which is built from the input device code files to DPACC is embedded into a specific section inside this library. On linking this library with the host application and running the resulting executable, the FlexIO runtime along with the host stubs will load this executable onto the DPA memory.

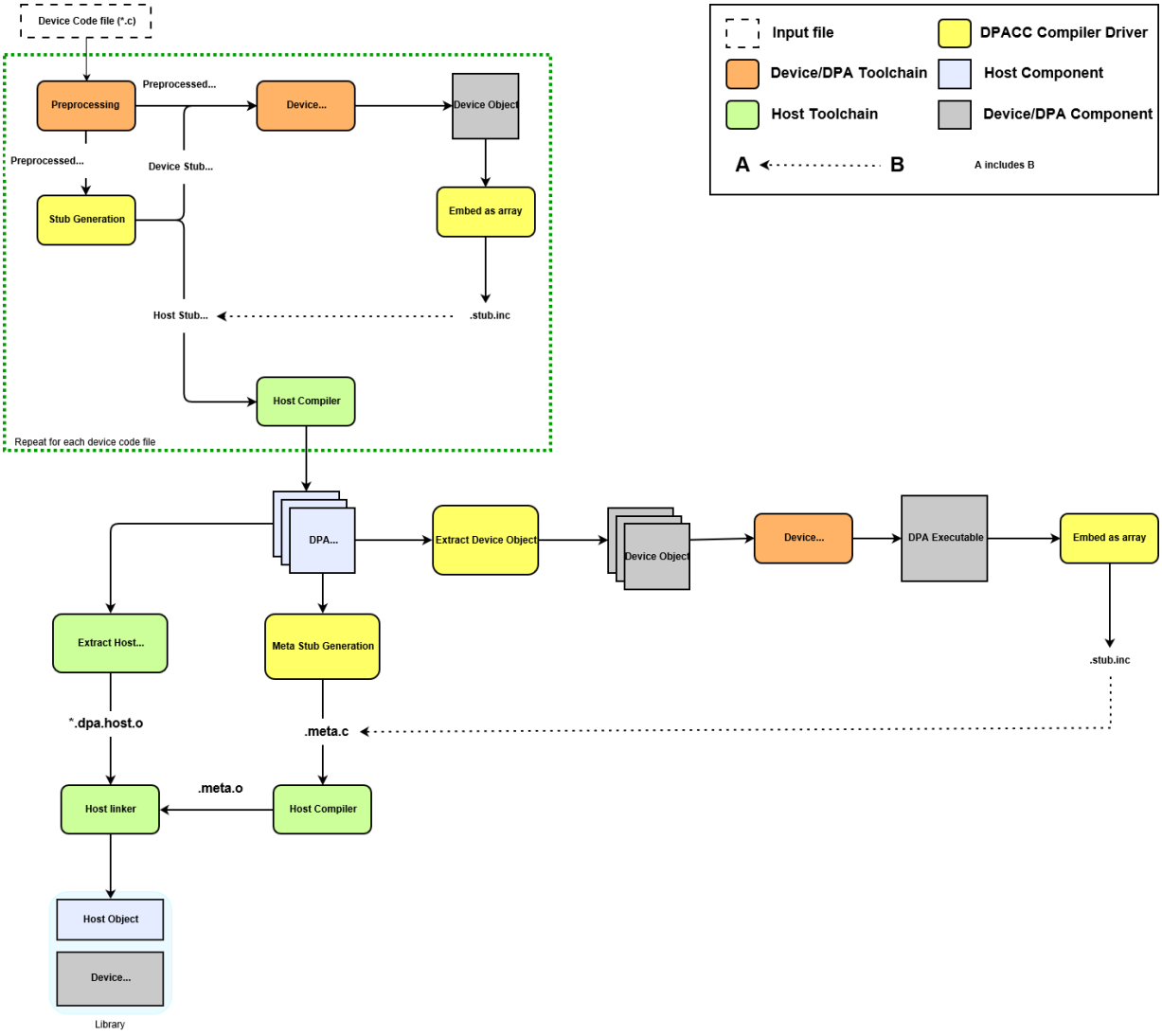




3.1.2. DPA Object

DPACC produces DPA object files in compile-only mode. A DPA object is an object file for the host machine. In a DPA object, the device object generated by compiling the input device code file is placed inside a specific section of the generated host stubs object. This process is repeated for each input file.

3.2. Modes of Operation



3.2.1. Compile-and-link Mode

This is a one-step mode that accepts C source files or DPA object files and produces the DPA program. Specifying the output library name is mandatory in this mode.

Example commands:

```
$ dpacc in1.c in2.c -o myLib1.a -hostcc=gcc          # Takes C sources to produce
 myLib1.a library
$ dpacc in3.dpa.o in4.dpa.o -o myLib2.a -hostcc=gcc # Takes DPA object files to
 produce myLib2.a library
$ dpacc in1.c in3.dpa.o -o myLib3.a -hostcc=gcc     # Takes C source and DPA object
 to produce myLib3.a library
```

3.2.2. Compile-only Mode

This mode accepts C source code and produces `.dpa.o` object files. These files can be given to DPACC to produce the DPA program. The mode is invoked by the `--compile` or `-c` option.

The user can explicitly provide the output object file name using the `--output-file` or `-o` option.

Example commands:

```
$ dpacc -c input1.c -hostcc=gcc                  # Produces input1.dpa.o
$ dpacc -c input3.c input4.c -hostcc=gcc         # Produces input3.dpa.o and
 input4.dpa.o
$ dpacc -c input2.c -o myObj.dpa.o -hostcc=gcc  # Produces myObj.dpa.o
```

Chapter 4. Execution

To execute DOCA DPACC Compiler:

```
Usage: dpacc <list-of-input-files> -hostcc=<path> [other options]
Helper Flags:
  -h, --help                          Print help information about DPACC
  -V, --version                       Print DPACC version information
  -v, --verbose                       List the compilation commands generated by
 this invocation while also executing every command in verbose mode
  -dryrun, --dryrun                   Only list the compilation commands generated
 by DPACC, without executing them
  -keep, --keep                       Keep all intermediate files that are generated
 during internal compilation steps in the current directory
  -keep-dir, --keep-dir               Keep all intermediate files that are generated
 during internal compilation steps in the given directory
  -optf, --options-file <file>,...    Include command line options from the
 specified file
```

4.1. Mandatory Arguments

Flag	DPACC Mode	Description
List of one or more input files	All	List of C source files or DPA object file names. Specifying at least one input file is mandatory. A file with an unknown extension is treated as a DPA object file.
`-hostcc, --hostcc <path>`	All	Specify the host compiler. This is typically the native compiler present on the host system.
`-o, --output-file <file>`	Compile-and-link	Specify name and location of the output archive (DPA program)

4.2. Commonly Used Arguments

Flag	Description
`-devicecc-options, --devicecc-options <options>,...`	Specify the list of options to pass to the device compiler
`-devicelink-options, --devicelink-options <options>,...`	Specify the list of options to pass during device linking stage. Typically, these include FlexIO libraries and DPA linker scripts.

Flag	Description
`-I, --common-include-path <path>,...`	Specify include search paths common to host and device code compilation. Typically, these are FlexIO headers.
`-devicecc, --devicecc <path>`	Specify the device compiler. By default, DPACC invokes dpa-clang.
`-o, --output-file <file>`	Specify name and location of the output file. ▶ Compile-only mode: Name of the output DPA object file. If not specified, `.dpa.o` is generated for each `.c` file. ▶ Compiler-and-link mode: Name of the output archive. This is a mandatory option in compiler-and-link mode
`-hostcc-options, --hostcc-options <options>,...`	Specify the list of options to pass to the host compiler

4.3. Incorrect Usage

- The `devicecc-options` option allows passing any option to the device compiler. However, passing options that prevent compilation of the input file may lead to unexpected behavior.

  For example: `-devicecc-options="-version"` makes the device compiler print the version and not process input files.
- Incompatible options which affect the kernel argument sizes during DPACC invocation and host application compilation may lead to undefined behavior during execution

  For example: Passing `-hostcc-options="-fshort-enums"` to DPACC and missing this option when building the host application

4.4. Examples

This section provides some common use cases of DPACC and showcases the `dpacc` command.

4.4.1. Link with Device Libraries

This example specifies the names and paths of the libraries using `devicelink-options`:

```
dpacc input.c -hostcc=gcc -o libInput.a -devicelink-options="-L <path-to-library> -l<libName>"
```

4.4.2. Include Headers

This example includes headers for device compilation using `devicecc-options` and host compilation using `hostcc-options`. You can also specify headers for any compilation on both the host and device side using the `-I` option.

```
dpacc input.c -hostcc=gcc -o libInput.a -I <common-headers-path> -devicecc-
options="-I <device-headers-path>" -hostcc-options="-I <host-headers-path>"
```

4.5. DPA Compiler Usage

dpa-clang is a compiler driver for accessing the Clang/LLVM compiler, assembler, and linker. The user is expected to invoke these tools only using the dpa-clang compiler driver.

dpa-clang is also a compiler frontend for C language.

Refer to the following resources for detailed user guide and command line references:

- Clang user manual
- Clang command line reference
- Target dependent options

4.5.1. Compiler Driver Command Line Options

DPA compiler provides a Clang compiler binary, dpa-clang. It accepts C code files or object files and generates an output according to different usage modes.

```
dpa-clang <list-of-input-files> [other options]
```

4.5.2. Linker Command Line Options

Note: Link time optimization (LTO) is not supported by DPA toolchain LLD.

LLD is the default linker provided in the DPA toolchain. Linker is invoked through the compiler driver binary,dpa-clang. Invoking the linker directly may lead to unexpected errors.

Linker related options are passed to through the compiler driver.

```
dpa-clang <list-of-input-files> [other options]
```

For more information, please refer to the LLD command line reference.

4.5.3. Objdump Command Line Options

The dpa-objdump utility prints the contents of object files and final linked images named on the command line.

For more information, please refer to the Objdump command line reference.

4.5.4. Archiver Command Line Options

dpa-ar is a Unix ar compatible archiver.

For more information, please refer to the Archiver command line reference.

4.5.5. NM Tool Command Line Options

The dpa-nm utility lists the names of symbols from object files, and archives.

For more information, please refer to the NM tool command line reference.

4.5.6. Common Compiler Options

Flag	Description
`-mcpu=nv-dpa-bf3`	Option to choose micro-architecture and ABI for DPA processor. This is the default option.
`-mrelax/-mno-relax`	Option to enable/disable linker relaxations
`-isystem <dir>`	Option to include Libc header files present in <dir> in FlexIO SDK as system headers
`-I <dir>`	Option to include header files present in <dir>

4.5.7. Common Linker Options

Flag	Description
`-L <path-to-library> -l<library-name>`	Option to link against libraries

Note: Linker options are provided through the compiler driver dpa-clang.

Note: The LLD linker script is honored in addition to the default configuration rather than replacing the whole configuration like in GNU ld. Hence, additional options may be required to override some default behaviors.

4.5.8. Debugging Options

Flag	Description
`-fdebug-macro`	Option to emit macro debugging information. This option enables macro-debugging similar to GCC option `-g3`.

4.5.9. Miscellaneous Notes

- DPA Toolchain provides the ability to invoke GNU linker instead of the default LLD linker via `--fuse-ld=<arg>` option in command line reference. However, it is discouraged to do so and not tested.

- Objects produced by LLD are not compatible with those generated by any other linker.
- DPA processor does not have native floating-point support and dpa-clang generates software emulated routines for floating point operations. Note that using floating point operations will have severe performance impact. Code generation for these routines will disabled by default in the next release with a command line option to enable it if needed.

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation nor any of its direct or indirect subsidiaries and affiliates (collectively: "NVIDIA") make no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assume no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA, the NVIDIA logo, and Mellanox are trademarks and/or registered trademarks of Mellanox Technologies Ltd. and/or NVIDIA Corporation in the U.S. and in other countries. The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a world¬wide basis. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2023 NVIDIA Corporation & affiliates. All rights reserved.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com