NVIDIA DOCA Switch

Application Guide

MLNX-15-060571 _v2.2.0 | October 2023

Table of Contents

Chapter 1. Introduction

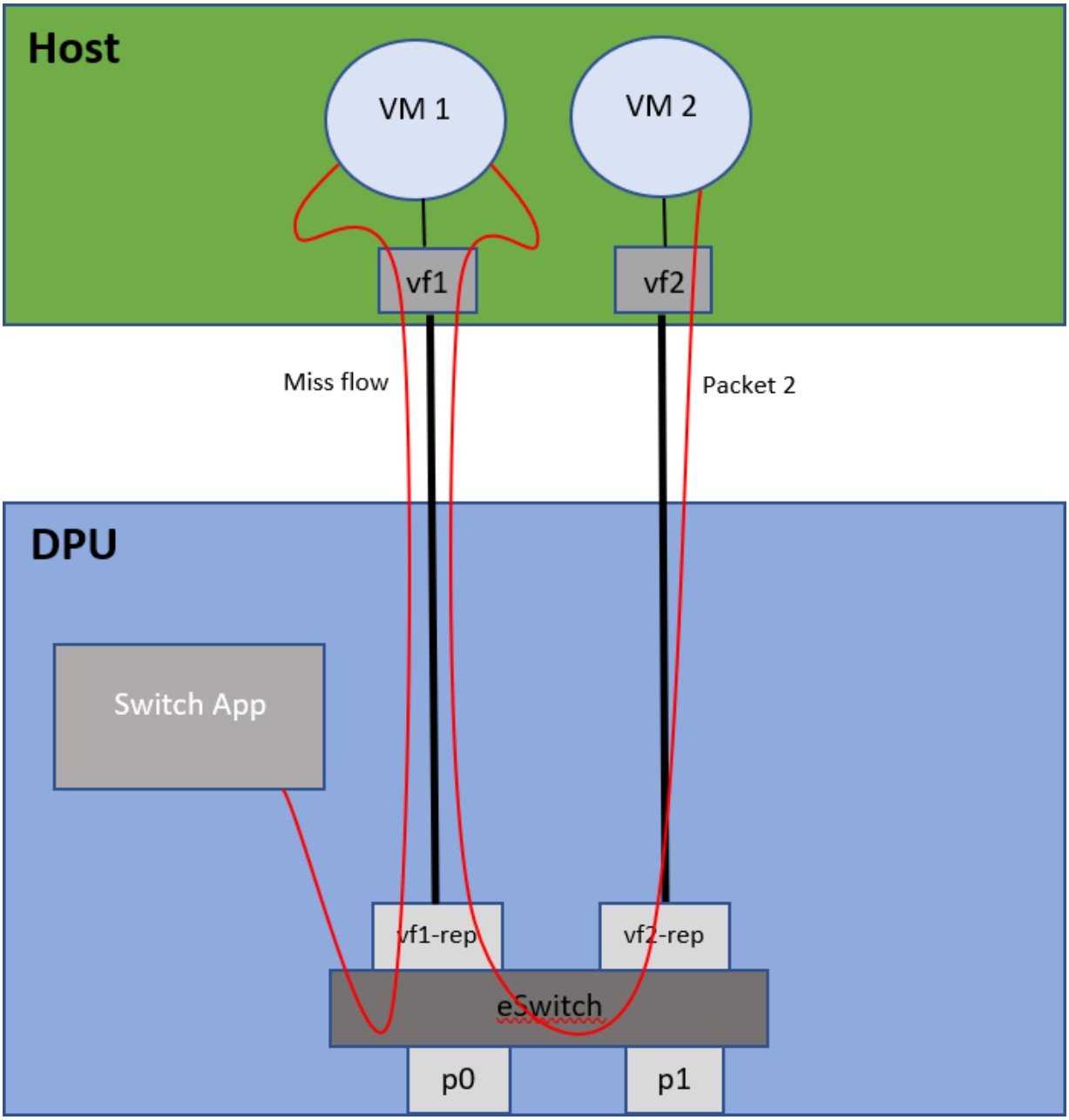
DOCA Switch is a network application that leverages the DPU's hardware capability for internal switching between representor ports on the DPU.

DOCA Switch is based on the DOCA Flow library. As such, it exposes a command line interface which receives DOCA Flow like commands to allow adding rules in real time.

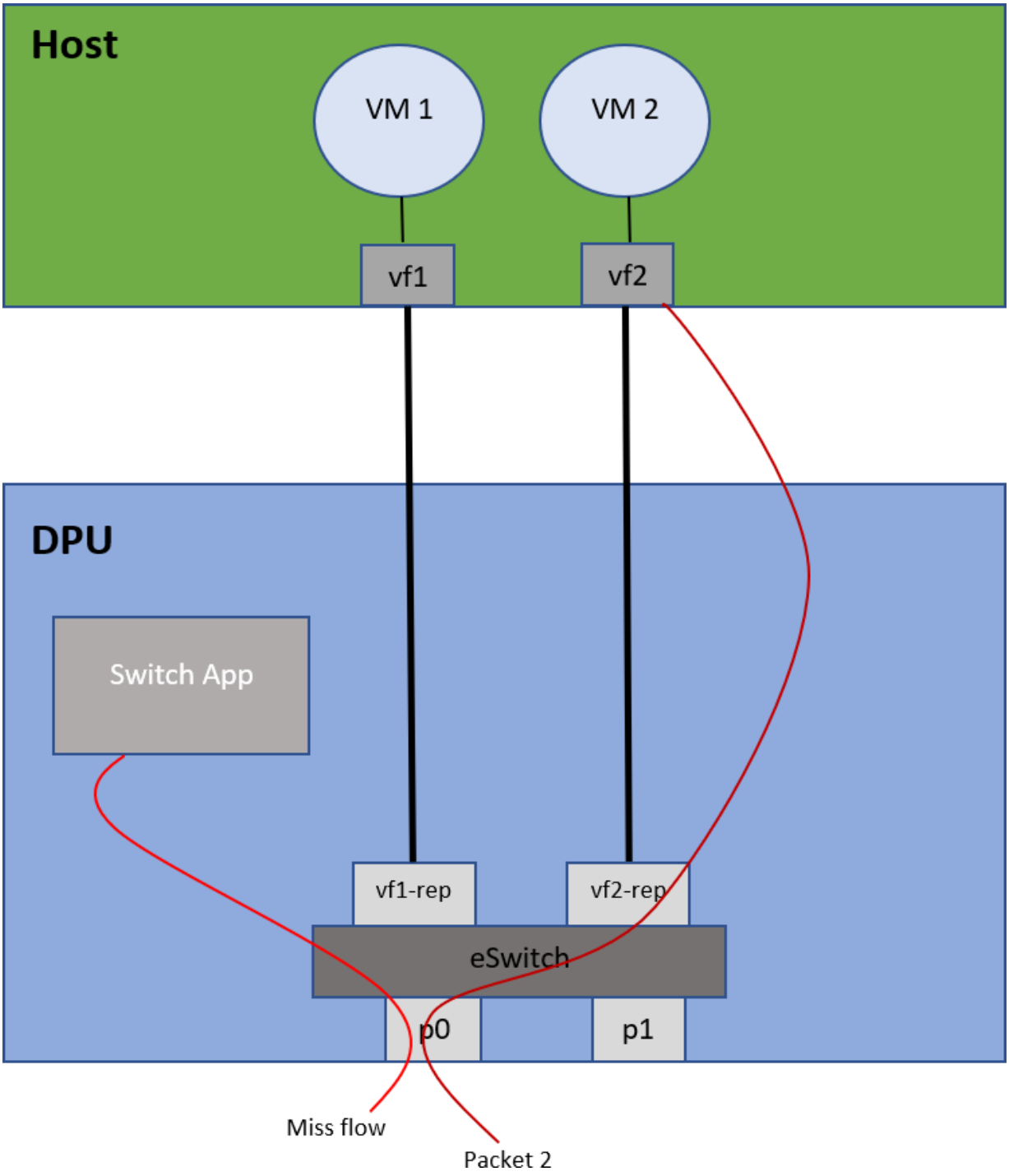
Chapter 2. System Design

DOCA Switch is designed to run on the DPU as a standalone application (all network traffic goes directly through it).

Traffic flows between two VMs on the host:



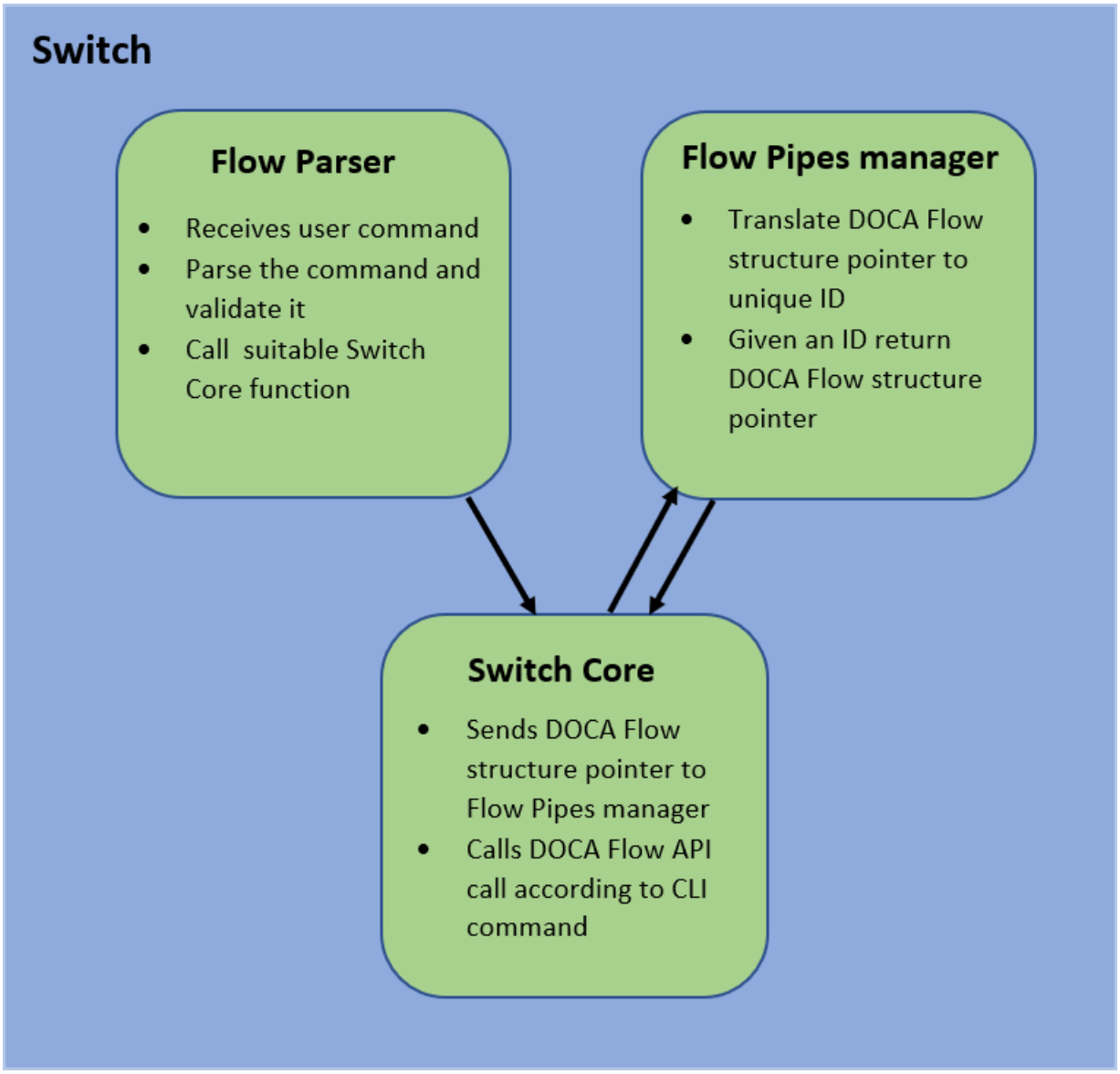
Traffic flow from a physical port to a VM on the host:


Host
VM 1
VM 2
vf1
vf2
DPU
Switch App
vf1-rep
vf2-rep
eSwitch
p0
p1
Miss flow
Packet 2

Chapter 3. Application Architecture

DOCA Switch is based on 3 modules:

- Command line interface – receives pre-defined DOCA Flow-like commands and parses them
- Flow pipes manger – generates a unique identification number for each DOCA Flow structure created
- Switch core – combines all modules together and calls necessary DOCA Flow API



Port initialization cannot be made dynamically. All ports must be defined when running the application with standard DPDK flags.

- When adding a pipe or an entry, the user must run commands to create the relevant structs beforehand
- Optional parameters must be specified by the user in the command line; otherwise, `NULL` is used
- After a pipe or an entry is created successfully, the relevant ID is printed for future use

Available commands:

- `create pipe port_id=[port_id][,<optional_parameters>]`

  Available optional parameters:

- `name=<pipe-name>`
- `root_enable=[1|0]`
- `monitor=[1|0]`
- `match_mask=[1|0]`
- `fwd=[1|0]`
- `fwd_miss=[1|0]`
- `type=[basic|control]`

- `add entry pipe_id=<pipe_id>,pipe_queue=<pipe_queue>[,<optional_parameters>]`

  Available optional parameters:

  - `monitor=[1|0]`
  - `fwd=[1|0]`

- `add control_pipe entry priority=<priority>,pipe_id=<pipe_id>,pipe_queue=<pipe_queue>[,<optional_parameters>]`

  Available optional parameters:

  - `match_mask=[1|0]`
  - `fwd=[1|0]`

- `destroy pipe port_id=[port_id],pipe_id=<pipe_id>`
- `rm entry pipe_queue=<pipe_queue>,entry_id=[entry_id]`
- `port pipes flush port_id=[port_id]`
- `port pipes dump port_id=[port_id],file=[file_name]`
- `query entry_id=[entry_id]`
- `create [struct] [field=value,…]`

  - Struct options: `pipe_match`, `entry_match`, `match_mask`, `actions`, `monitor`, `fwd`, `fwd_miss`

    - Match struct fields:

Fields	Field Options
`flags`	
`port_meta` (source port)	According to the number of physical ports
`out_src_mac`	
`out_dst_mac`	
`out_eth_type`	
`out_vlan_id`	
`out_src_ip_type`	`ipv4`, `ipv6`
`out_src_ip_addr`	

Fields	Field Options
out_dst_ip_type	ipv4, ipv6
out_dst_ip_addr	
out_l4_type	tcp, udp, gre
out_tcp_flags	FIN, SYN, RST, PSH, ACK, URG, ECE, CWR
out_src_port	
out_dst_port	
tun_type	
vxlan-tun_id	
gre_key	
gtp_teid	
in_src_mac	
in_dst_mac	
in_eth_type	
in_vlan_id	
in_src_ip_type	ipv4, ipv6
in_src_ip_addr	
in_dst_ip_type	ipv4, ipv6
in_dst_ip_addr	
in_l4_type	tcp, udp
in_tcp_flags	FIN, SYN, RST, PSH, ACK, URG, ECE, CWR
in_src_port	
in_dst_port	

- Actions struct fields:

Fields	Field Options
decap	true, false
mod_src_mac	
mod_dst_mac	
mod_src_ip_type	ipv4, ipv6
mod_src_ip_addr	
mod_dst_ip_type	ipv4, ipv6
mod_dst_ip_addr	
mod_src_port	
mod_dst_port	
dec_ttl	true, false
has_encap	true, false
encap_src_mac	
encap_dst_mac	

Fields	Field Options
encap_src_ip_type	ipv4, ipv6
encap_src_ip_addr	
encap_dst_ip_type	ipv4, ipv6
encap_dst_ip_addr	
encap_tup_type	vxlan, gtpu, gre
encap_vxlan-tun_id	
encap_gre_key	
encap_gtp_teid	

- FWD struct fields:

Fields	Field Options
type	rss, port, pipe, drop
rss_flags	
rss_queues	
num_of_queues	
rss_mark	
port_id	
next_pipe_id	

- Monitor struct fields:
  - flags
  - id
  - cir
  - cbs
  - aging

Consider that the physical port number (only one physical port is supported) will always be 0 and all representor ports are numbered from 1 to N where N is the number of representors being used. For example:

- Physical port ID: 0
- VF0 representor port ID: 1
- VF1 representor port ID: 2
- VF2 representor port ID: 3

The following is an example of creating a pipe and adding one entry into it:

1. Pipe is configured on port ID 0 (physical port).
2. Entry is configured to forward all traffic directly into port ID 1 (VF0).
3. When the forwarding rule is no longer needed, the entry is deleted.

4. Ultimately, both entries are deleted, each according to the unique random ID it was given:

```
create fwd type=port,port_id=0xffff
create pipe port_id=0,name=p0_to_vf1,root_enable=1,fwd=1
create fwd type=port,port_id=1
add entry pipe_queue=0,fwd=1,pipe_id=1012
          ....
rm entry pipe_queue=0,entry_id=447
```

Chapter 4. DOCA Libraries

This application leverages the DOCA Flow library.

Chapter 5. Configuration Flow

1. Parse application argument.
   a). Initialize the arg parser resources and register DOCA general parameters.
   ```
   doca_argp_init();
   ```
   b). Register application parameters.
   ```
   register_switch_params();
   ```
   c). Parse application flags.
   ```
   doca_argp_start();
   ```
2. Count total number of ports.
   ```
   switch_ports_count();
   ```
   a). Check how many ports are entered when running the application.
3. Initialize DPDK ports and queues.
   ```
   dpdk_queues_and_ports_init();
   ```
4. Initialize DOCA Switch.
   ```
   switch_init();
   ```
   a). Initialize DOCA Flow.
   b). Create port pairs.
   c). Create Flow Pipes Manger module
   d). Register an action for each relevant CLI command.
5. Initialize Flow Parser.
   ```
   flow_parser_init();
   ```
   a). Reset all internal Flow Parser structures.
   b). Start the command line interface.
   c). Receive user commands, parse them, and call the required DOCA Flow API command.
   d). Close the interactive shell once a "quit" command is entered.
6. Clean Flow Parser resources.
   ```
   flow_parser_cleanup();
   ```
7. Destroy DOCA Switch resources.
   ```
   switch_destroy();
   ```
   a). Destroy Flow Pipes Manager resources.
8. Destroy DOCA Flow.
   ```
   switch_destroy();
   ```

9. Destroy DPDK ports and queues.
```
dpdk_queues_and_ports_fini();
```
10. DPDK finish.
```
dpdk_fini();
```
    a). Call `rte_eal_destroy()` to destroy initialized EAL resources.
11. Arg parser destroy.
```
doca_argp_destroy();
```

Chapter 6. Running the Application

1. Refer to the following documents:
   - NVIDIA DOCA Installation Guide for Linux for details on how to install BlueField-related software.
   - NVIDIA DOCA Troubleshooting Guide for any issue you may encounter with the installation, compilation, or execution of DOCA applications.
   - NVIDIA DOCA Applications Overview for additional compilation instructions and development tips for the DOCA applications.
2. The DOCA Switch example binary is located under `/opt/mellanox/doca/applications/switch/bin/doca_switch`. To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/
meson build
ninja -C build
```

3. To build only the Switch application:

   a). Edit the following flags in `/opt/mellanox/doca/applications/meson_options.txt`:

   - Set `enable_all_applications` to `false`
   - Set `enable_switch` to `true`

   b). Run the commands in step 2.

   Note: `doca_switch` will be created under `./build/switch/src/`.

   Application usage:

```
Usage: doca_switch [DPDK Flags] -- [DOCA Flags]

DOCA Flags:
  -h, --help                          Print a help synopsis
  -v, --version                       Print program version information
  -l, --log-level                     Set the log level for the program
 <CRITICAL=20, ERROR=30, WARNING=40, INFO=50, DEBUG=60>
```

   Note: For additional information on the app, use `-h`:

```
/opt/mellanox/doca/applications/switch/bin/doca_switch -h
```

4. CLI example for running the app on BlueField with 3 VF representors (`dv_flow_en=2` is necessary to run the application with hardware steering):

```
/opt/mellanox/doca/applications/switch/bin/doca_switch -a
 03:00.0,representor=[0-2],dv_flow_en=2 -- -l 30
```

Chapter 7. Arg Parser DOCA Flags

Refer to NVIDIA DOCA Arg Parser Programming Guide for more information.

Flag Type	Short Flag	Long Flag/ JSON Key	Description	JSON Content
General flags	`l`	`log-level`	Sets the log level for the application: ▶ CRITICAL=20 ▶ ERROR=30 ▶ WARNING=40 ▶ INFO=50 ▶ DEBUG=60	`"log-level": 60`
	`v`	`version`	Print program version information	N/A
	`h`	`help`	Print a help synopsis	N/A

Chapter 8. References

- `/opt/mellanox/doca/applications/switch/src`

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

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