NVIDIA BlueField DPU Scalable Function

User Guide

MLNX-15-060491 _v1.1 | October 2021

Table of Contents

Chapter 1. Introduction

Scalable functions (SFs), or sub-functions, are very similar to virtual functions (VFs) which are part of a Single Root I/O Virtualization (SR-IOV) solution. I/O virtualization is one of the key features used in data centers today. It improves the performance of enterprise servers by giving virtual machines direct access to hardware I/O devices. The SR-IOV specification allows one PCI Express (PCIe) device to present itself to the host as multiple distinct "virtual" devices. This is done with a new PCIe capability structure added to a traditional PCIe function (i.e., a physical function or PF).

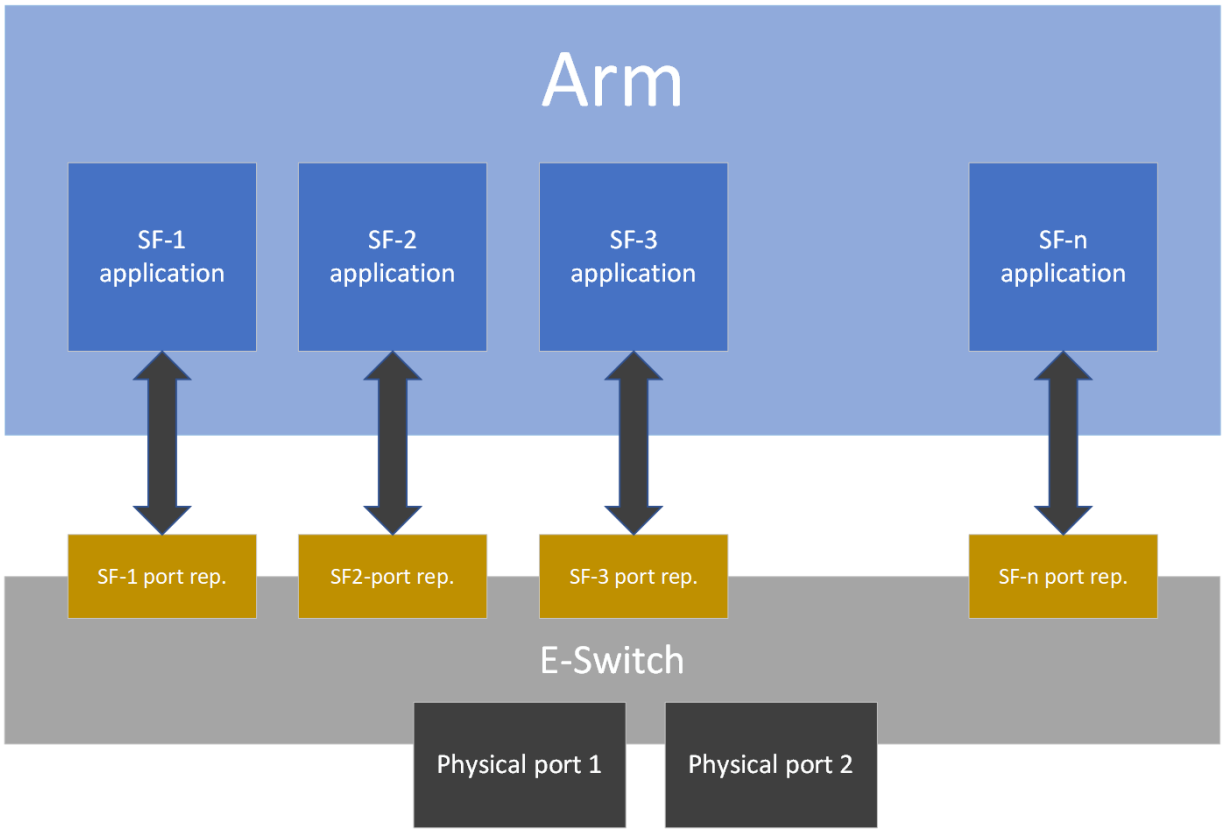
The PF provides control over the creation and allocation of new VFs. VFs share the device's underlying hardware and PCIe. A key feature of the SR-IOV specification is that VFs are very lightweight so that many of them can be implemented in a single device.

To utilize the capabilities of VF in the BlueField, SFs are used. SFs allow support for a larger number of functions than VFs, and more importantly, they allow running multiple services concurrently on the DPU.

An SF is a lightweight function which has a parent PCIe function on which it is deployed. The SF, therefore, has access to the capabilities and resources of its parent PCIe function and has its own function capabilities and its own resources. This means that an SF would also have its own dedicated queues (i.e., txq, rxq).

SFs co-exist with PCIe SR-IOV virtual functions (on the host) but also do not require enabling PCIe SR-IOV.

SFs support E-Switch representation offload like existing PF and VF representors. An SF shares PCIe-level resources with other SFs and/or with its parent PCIe function.


Arm
SF-1 application
SF-2 application
SF-3 application
SF-n application
SF-1 port rep.
SF2-port rep.
SF-3 port rep.
SF-n port rep.
E-Switch
Physical port 1
Physical port 2

Chapter 2. Prerequisites

Please refer to the DOCA Installation Guide for details on how to install BlueField related software.

- The minimum firmware version required for cloud configuration is 16.29.1040 or higher.
- The minimum firmware version required for non-cloud, physical, or server BlueField configuration is 16.31.0238 or higher.
- Set SFs to "trusted" on the device using the following command on the Arm side:

```
mlxreg -d /dev/mst/mt41686_pciconf0 --reg_id 0xc007 --reg_len 0x40 --indexes
 "0x0.0:32=0x80000000" --yes --set "0x4.0:32=0x1"
```

Note: This command sets all SFs to "trusted".

Chapter 3. SF Configuration

To use a subfunction, a 3-step setup sequence must be followed first:

1. Create.
2. Configure.
3. Deploy.

These steps may be performed in the following two methods:

- Using mlnx-sf script
- Using mlxdevm tool (legacy)

3.1. Configuration Using mlnx-sf Script

The mlnx-sf script creates, configures, and deploys an SF using one command:

```
mlnx-sf --action create --device <pci_address> --sfnum <sfnum> --hwaddr
 <mac_address>
```

For example:

```
mlnx-sf --action create --device 0000:03:00.0 --sfnum 9 --hwaddr 02:25:f2:8d:a2:4c
```

- `--action create` – creates the SF (there is a "show" action as well)
- `--device <pci_address>` – links the created SF to a parent PCIe device
- `--sfnum <sfnum>` – gives the SF a unique SF number
- `--hwaddr <mac_address>` – configures the MAC address of the SF

Useful commands:

- During configuration or after it is complete, you may use the command `mlnx-sf --action show` to display the attached auxiliary device to each SF, as that is needed during application runtime. For example (only a single device is shown):

```
SF Index: pci/0000:03:00.0/229409
  Parent PCI dev: 0000:03:00.0
  Representor netdev: en3f0pf0sf70
  Function HWADDR: 00:01:01:01:01:70
  Auxiliary device: mlx5_core.sf.4
    netdev: enp3s0f0s70
    RDMA dev: mlx5_4
```

- To delete the SF port representor, run:

```
mlnx-sf --action delete --sfindex pci/<pci_address>/<pasre_dev>
```

  For example:

```
mlnx-sf --action delete --sfindex pci/0000:03:00.0/229409
```

- Please use the command "`mlnx-sf --help`" for more details

3.2. Configuration Using mlxdevm Tool

1. Create the SF.

   SFs are managed using the `mlxdevm` tool supplied with iproute2 package. The tool is found at `/opt/mellanox/iproute2/sbin/mlxdevm`.

   An SF is created using the mlxdevm interface. The SF is added by adding a port of "`pcisf`" flavor.

   To create an SF port representor, run:

```
/opt/mellanox/iproute2/sbin/mlxdevm port add pci/<pci_address> flavour pcisf
 pfnum <corresponding pfnum> sfnum <sfnum>
```

   Note: Each SF must have a unique number (<sfnum>).

   For example:

```
/opt/mellanox/iproute2/sbin/mlxdevm port add pci/0000:03:00.0 flavour pcisf pfnum
 0 sfnum 8
```

   Output:

```
Pci/0000:30:00.0/229409: type eth netdev eth0 flavour pcisf controller 0 pfnum 0
 sfnum 8
   function:
hw_addr 00:00:00:00:00:00 state inactive opstate detached roce true max_uc_macs
 1024
```

   The highlighted number (229409) will be required to complete the next two steps (i.e., configuration and deployment).

   "`pci/0000:03:00.0/229409`" represents SF port representor that is created. That is, "`pci/0000:03:00.0`" is the parent port of the SF port representor 229409.

2. Configure the SF.

   A subfunction representor (SF port representor) is created but it is not active yet. Users may use the e-switch port representor to configure settings such as adding the SF port representor to an OVS bridge, adding TC rules, etc. Users may also configure the hardware address (e.g., MAC address) of the SF while it is inactive.

To set the MAC address, run:

```
/opt/mellanox/iproute2/sbin/mlxdevm port function set pci/<pci_address>/
<pasre_dev>  hw_addr <MAC address>
```

For example:

```
/opt/mellanox/iproute2/sbin/mlxdevm port function set pci/0000:03:00.0/229409
 hw_addr 00:00:00:00:08:0
```

Note: Other SF capabilities (e.g., disabling RoCE for the SF) must be set before activating the SF (optional).

To verify that the MAC address has been configured as expected, run the following command:

```
/opt/mellanox/iproute2/sbin/mlxdevm port show pci/<pci_address>/<pasre_dev>
```

For example:

```
/opt/mellanox/iproute2/sbin/mlxdevm port show pci/0000:30:00.0/229409
```

Output:

```
pci/0000:30:00.0/229409: type eth netdev en3f0pf0sf8 eth0 flavour pcisf
 controller 0 pfnum 0 sfnum 8
   function:
 hw_addr 00:00:00:00:08:08 state inactive opstate detached roce true max_uc_macs
 1024
```

3. Deploy the SF.

   Once an SF is configured, it must be activated for it to be used.

   To activate the SF, run:

   ```
   /opt/mellanox/iproute2/sbin/mlxdevm port function set pci/< pci_address > /
   <pasre_dev> state active
   ```

   For example:

   ```
   /opt/mellanox/iproute2/sbin/mlxdevm port function set pci/0000:03:00.0/229409
    state active
   ```

   Note: To verify that the SF state is active, you may run either the command `mlnx-sf --action show`, or:

   ```
   /opt/mellanox/iproute2/sbin/mlxdevm port show pci/<pci_address> /
   <pasre_dev>
   ```

   To unbind the SF from the default config driver and bind the actual SF driver, run:

   ```
   echo mlx5_core.sf.<next_serial> >  /sys/bus/auxiliary/drivers/mlx5_core.sf_cfg/
   unbind
   echo mlx5_core.sf.<next_serial>  > /sys/bus/auxiliary/drivers/mlx5_core.sf/bind
   ```

   Note: See the useful commands listed below to know where to obtain the `<next_serial>` value.

   For example:

   ```
   echo mlx5_core.sf.8  > /sys/bus/auxiliary/drivers/mlx5_core.sf_cfg/unbind
   echo mlx5_core.sf.8  > /sys/bus/auxiliary/drivers/mlx5_core.sf/bind
   ```

   Useful commands:

   - To see the available sub-functions, run:

     ```
     ls /sys/bus/auxiliary/devices/mlx5_core.sf.*
     ```

For example, if you run the command before creating the SF (`port add` command), then the output would be as follows:

```
/sys/bus/auxiliary/devices/mlx5_core.sf.2:
driver infiniband infiniband_mad infiniband_vrebs mlx5_core.eth2
 mlx5_core.rdma.2 net power sfnum subsystem uevent

/sys/bus/auxiliary/devices/mlx5_core.sf.3:
driver infiniband infiniband_mad infiniband_vrebs mlx5_core.eth2
 mlx5_core.rdma.2 net power sfnum subsystem uevent

/sys/bus/auxiliary/devices/mlx5_core.sf.4:
driver infiniband infiniband_mad infiniband_vrebs mlx5_core.eth2
 mlx5_core.rdma.2 net power sfnum subsystem uevent

/sys/bus/auxiliary/devices/mlx5_core.sf.5:
driver infiniband infiniband_mad infiniband_vrebs mlx5_core.eth2
 mlx5_core.rdma.2 net power sfnum subsystem uevent

/sys/bus/auxiliary/devices/mlx5_core.sf.6:
driver  power sfnum subsystem uevent

/sys/bus/auxiliary/devices/mlx5_core.sf.7:
driver  power sfnum subsystem uevent

/sys/bus/auxiliary/devices/mlx5_core.sf.8:
driver infiniband infiniband_mad infiniband_vrebs mlx5_core.eth2
 mlx5_core.rdma.2 net power sfnum subsystem uevent
```

After creating the SF, the output would be:

```
/sys/bus/auxiliary/devices/mlx5_core.sf.2:
driver infiniband infiniband_mad infiniband_vrebs mlx5_core.eth2
 mlx5_core.rdma.2 net power sfnum subsystem uevent

/sys/bus/auxiliary/devices/mlx5_core.sf.3:
driver infiniband infiniband_mad infiniband_vrebs mlx5_core.eth2
 mlx5_core.rdma.2 net power sfnum subsystem uevent

/sys/bus/auxiliary/devices/mlx5_core.sf.4:
driver infiniband infiniband_mad infiniband_vrebs mlx5_core.eth2
 mlx5_core.rdma.2 net power sfnum subsystem uevent

/sys/bus/auxiliary/devices/mlx5_core.sf.5:
driver infiniband infiniband_mad infiniband_vrebs mlx5_core.eth2
 mlx5_core.rdma.2 net power sfnum subsystem uevent

/sys/bus/auxiliary/devices/mlx5_core.sf.6:
driver  power sfnum subsystem uevent

/sys/bus/auxiliary/devices/mlx5_core.sf.7:
driver  power sfnum subsystem uevent

/sys/bus/auxiliary/devices/mlx5_core.sf.8:
driver infiniband infiniband_mad infiniband_vrebs mlx5_core.eth2
 mlx5_core.rdma.2 net power sfnum subsystem uevent
```

Note that the `<next_serial>` number is 8 for the created SF.

- To see the sfnum of each sub-function, run:

```
cat /sys/bus/auxiliary/devices/mlx5_core.sf.<next_serial>/sfnum
```

For example:

```
cat /sys/bus/auxiliary/devices/mlx5_core.sf.8/sfnum
```

Output:

```
cat /sys/bus/auxiliary/devices/mlx5_core.sf.8
```

```
8
```

- To remove an SF, you must first make its state inactive and only then remove the SF representor.

  To make the SF's state inactive, run:

  ```
  opt/mellanox/iproute2/sbin/mlxdevm port function set pci/<pci_address>/
  <pasre_dev> state inactive
  ```

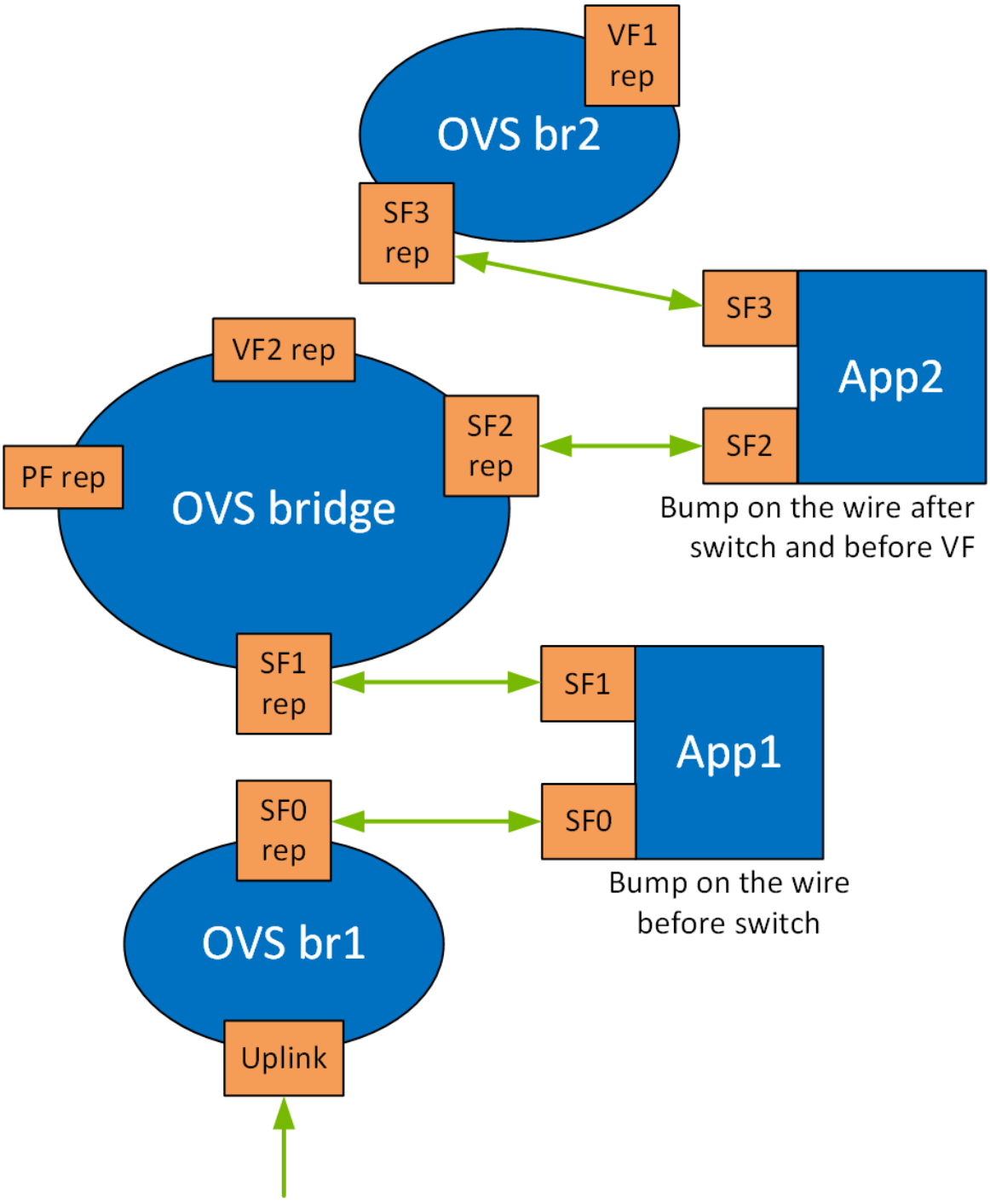
  To delete the SF port representor, run:

  ```
  /opt/mellanox/iproute2/sbin/mlxdevm port del pci/<pci_address>/<pasre_dev>
  ```

  For example:

  ```
  /opt/mellanox/iproute2/sbin/mlxdevm port function set pci/0000:03:00.0/229409
   state inactive
  /opt/mellanox/iproute2/sbin/mlxdevm/devlink port del pci/0000:03:00.0/229409
  ```

4. Use the SF.



Running the application on the DPU requires OVS configuration. By creating SFs, an SF representor for the OVS is also created and named `en3f0pf*sf*`. Therefore, each representor needs to be connected to the correct OVS bridge.

Note: Two SFs related to the same PCIe are necessary for the configuration in the illustration.

The following example configures 2 SFs and adds their representors to the OVS.

a). Create, configure, and deploy the SFs. Run:

```
mlnx-sf --action create --device 0000:03:00.0 --sfnum 4 --hwaddr
 02:25:f2:8d:a2:4c
mlnx-sf --action create --device 0000:03:00.0 --sfnum 5 --hwaddr
 02:25:f2:8d:a2:5c
```

Using `ifconfig`, you may see that there are 2 added network interfaces: `en3f0pf0sf4` and `en3f0pf0sf5` for the two respective SF port representors.

b). Add the port representors to the OVS bridges:

```
ovs-vsctl add-port sf_bridge1 en3f0pf0sf4
ovs-vsctl add-port sf_bridge2 en3f0pf0sf5
```

The OVS bridges after adding the SF representors:

```
Bridge sf_bridge1
     Port p0
         Interface p0
     Port sf_bridge1
         Interface sf_bridge1
             type: internal
     Port en3f0pf0sf4
         Interface en3f0pf0sf4
Bridge sf_bridge2
     Port sf_bridge2
         Interface sf_bridge2
             type: internal
     Port en3f0pf0sf5
         Interface en3f0pf0sf5
     Port pf0hpf
         Interface pf0hpf
ovs_version: "2.14.1"
```

Note: The interface might be down by default. Remember to `ifconfing` the interface to "up" status.

Note: When deleting the SF port representor, you must also de-attach it from the bridge it is connected to using the command `ovs-vsctl port-del en3f0pf0sf*`. Otherwise, the port representor will still be connected to the bridge but would not be recognizable.

To run the application, use the following command to initialize the subfunctions during runtime:

```
*Executable_binary* -a auxiliary:mlx5_core.sf.* -a auxiliary:mlx5_core.sf.*
```

For example:

```
doca_app_rec -a 0000:03:00.0,class=regex -a auxiliary:mlx5_core.sf.4,sft_en=1 -a
 auxiliary:mlx5_core.sf.5,sft_en=1 -v - [application_flags]
```

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. 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