SmartSSD Computational Storage Drive

Installation and User Guide

UG1382 (v1.0) November 10, 2020

XILINX®

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
11/10/2020 Version 1.0	
Initial release.	N/A

Table of Contents

Revision History 2

Chapter 1: Introduction 5

Chapter 2: Installation 6

- Overview 6
- Card Installation Procedures 8
- Software Installation 10
- Card Bring-Up and Validation 16
- Next Steps 26
- Troubleshooting 26

Chapter 3: Platform Notes 30

- Overview 30
- Development Target Shell 31
- Platform Naming and Life Cycle 32
- U.2 Platform 36

Appendix A: Changing XRT and Target Platform Versions 40

- RedHat and CentOS 40
- Ubuntu 41

Appendix B: Creating a Vault Repository for CentOS 43

Appendix C: Accessing the SmartSSD CSD through a Virtual Machine 45

Appendix D: Hot Plug Support on the SmartSSD CSD 46

- Managed Hot Add and Removal for the SmartSSD CSD 46
- Surprise Add Support on the SmartSSD CSD 48

Appendix E: Miscellaneous Information and Settings 50

Appendix F: FPGA-SSD DNA Pairing **52**

Appendix G: Accessing FPGA DNA Primitive from Dynamic Region **54**

Appendix H: Additional Resources and Legal Notices **57**

Xilinx Resources 57

Documentation Navigator and Design Hubs 57

References 57

Please Read: Important Legal Notices 58

XILINX

Chapter 1

Introduction

The Samsung SmartSSD® Computational Storage Drive (CSD), powered by Xilinx® FPGAs, is a PCI Express® compliant storage accelerator module that integrates a Xilinx FPGA and Samsung NVMe SSD (controller with storage media) together. It is designed to accelerate storage-intensive applications such as data compression, decompression, encryption, decryption, and filtering with standard NVMe SSD functions by enabling direct PCIe® peer-to-peer (P2P) transfers from NVMe SSD to FPGA-DDR (global memory).

This guide provides the installation as well as functional details for the SmartSSD CSD storage accelerator module.

Note: Module and card are used interchangeably throughout this document.

XILINX

Chapter 2

Installation

Overview

This chapter provides hardware and software installation procedures for the SmartSSD® CSD storage accelerator module and applies to the Vitis™ unified software platform release 2020.1 and later.

Different system configurations are available for running, developing, and debugging applications on your SmartSSD CSD accelerator module:

- **Running Applications:** To run accelerated applications, install a SmartSSD CSD module into a system as described in Card Installation Procedures along with the required deployment software to support running applications as described in Software Installation.
- **Developing Applications:** To develop FPGA accelerated applications, it is necessary to install the development software. See the Software Installation section which describes the installation procedure for both, a development target platform and the Vitis environment. This configuration need not have a SmartSSD CSD module installed and can be used for development along with debugging in emulation modes.
- **Running, Developing, and Debugging Applications:** By installing the SmartSSD CSD along with both the deployment and development software on a single machine, you can configure a system for developing and running accelerated applications. With the module installed, developers can debug applications in both emulation modes and on the hardware.

Minimum System Requirements

The following table lists the minimum system requirements for running a SmartSSD CSD storage accelerator module.

Table 1: **Minimum System Requirements**

Component	Requirements
PCI Express Subsystem	PCI Express 3.0 (or greater) compliant with NVMe U.2 bay available. • System must support memory mapped I/O above 4 GB This can be done by enabling Above 4G decoding in most standard BIOS. Above 4G decoding allows the user to enable or disable memory mapped I/O for a 64-bit PCIe device to 4G or greater address space.
BIOS Options	Above 4G decoding[1]
Bay Power Supply	25W
Operating System	Linux, 64-bit: • Ubuntu 16.04, 18.04 • CentOS 7.4, 7.5, 7.6, 7.7 • RHEL 7.4, 7.5, 7.6

OS Options	Reserve four PCIe bus numbers per U.2 slot for the SmartSSD CSD *Surprise Add* functionality. Use the following arguments to boot the kernel to support *Surprise Add*[1].
	`GRUB_CMDLINE_LINUX="pci=assign-busses,hpbussize=4"`
	`GRUB_CMDLINE_LINUX="realloc=on,hpmemsize=8G"`

Driver Installation	Linux inbox NVMe driver.
System Memory	For installations, a minimum of 4 GB plus application memory is required.
Internet Connection	Required for downloading drivers and utilities.
Hard Disk Space	Satisfy the minimum system requirements for your operating system.
Licensing	None required for application deployment. For the application development environment, see *Vitis Unified Software Platform Documentation: Application Acceleration Development* (UG1393).

Notes:

1. Reserving PCIe bus numbers to support the *Hot Plug Surprise Add* functionality is possible through either the BIOS or OS options.

Qualified Servers

For a list of servers on which the SmartSSD CSD module is fully qualified, see Xilinx Answer Record 75178.

Card Interfaces and Details

The SmartSSD CSD module is available in a passive cooling configuration and is designed for installation into a server where controlled air flow provides direct cooling to the module. The module includes the following interfaces.

- U.2 compliant PCIe Gen3x4 connector

Card Installation Procedures

To reduce the risk of fire, electric shock, or injury, always follow basic safety precautions.

CAUTION! *You must always use an ESD strap or other antistatic device when handling hardware.*

Safety Instructions

Safety Information

To ensure your personal safety and the safety of your equipment:

- Keep your work area and the computer/server clean and clear of debris.
- Before opening the computer/system cover, unplug the power cord.

Electrostatic Discharge Caution

Electrostatic discharge (ESD) can damage electronic components when they are improperly handled, and can result in total or intermittent failures. Always follow ESD-prevention procedures when removing and replacing components.

To prevent ESD damage:

- Use an ESD wrist or ankle strap and ensure that it makes skin contact. Connect the equipment end of the strap to an unpainted metal surface on the chassis.
- Avoid touching the module against your clothing. The wrist strap protects components from ESD on the body only.
- Handle the module by its bracket or edges only. Avoid touching the printed circuit board or the connectors.
- Put the module down only on an antistatic surface such as the bag supplied in your kit.
- If you are returning the module to Xilinx Product Support, place it back in its antistatic bag immediately.

Before You Begin

IMPORTANT! *SmartSSD CSD modules are delicate and sensitive electronic devices; equipment is to be installed by a qualified technician only. This equipment is intended for installation in a Restricted Access Location.*

Check for module compatibility with the system. Also check for proper system requirements such as power, bus type, and physical dimensions to support the module.

Installing the SmartSSD CSD

The following procedure is a guide for the SmartSSD CSD module installation. Consult your computer documentation for additional information.

***Note*:** The output results/logs provided throughout this document are for example purposes only. When executing any given command, your output may or may not exactly match the text.

If you encounter any issues during installation, see Troubleshooting and Known Issues.

The steps to install the SmartSSD CSD module are as follows.

1. Shut down the host computer and unplug the power cord.
2. Plug the SmartSSD CSD module into the U.2 PCIe Gen3x4 capable slot provided for NVMe drives.
3. Connect the power cord and turn on the computer.

   ***Note*:** Do not power-on a passively cooled module without adequate forced airflow across the module, otherwise the module can be damaged. This module can heat up after use in the server. Use caution when handling.
4. To verify that the device has been installed correctly, enter the following Linux command in the terminal:

   ```
   $ lspci | grep -i xilinx
   ```

   If the module is successfully installed with respect to lspci and found by the operating system, a message similar to the following listing five PCIe devices will be displayed.

   ```
   5e:00.0 PCI bridge: Xilinx Corporation Device 9134
   5f:00.0 PCI bridge: Xilinx Corporation Device 9234
   5f:01.0 PCI bridge: Xilinx Corporation Device 9434
   61:00.0 Memory controller: Xilinx Corporation Device 6987
   61:00.1 Memory controller: Xilinx Corporation Device 6988
   ```
5. To verify that the controller is properly enumerated, enter the following Linux command in the terminal:

   ```
   $ lspci | grep -i samsung
   ```

   If the PCIe device for the SSD controller is properly enumerated, a message similar to the following will be displayed.

   ```
   69:00.0 Non-Volatile memory controller: Samsung Electronics Co Ltd
   Device a824
   ```
6. If steps 4 and 5 are successful, enter the following Linux command in the terminal:

   ```
   $ lsblk
   ```

A message similar to the following will be displayed. If you do not see a message similar to the following, see Troubleshooting.

```
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sdb       8:16   0 238.5G  0 disk
sdb2      8:18   0     1K  0 part
sdb5      8:21   0   976M  0 part [SWAP]
sdb1      8:17   0 237.5G  0 part /
sda       8:0    0 931.5G  0 disk
sda2      8:2    0     1K  0 part
sda5      8:5    0  63.6G  0 part
sda1      8:1    0   868G  0 part
nvme0n1 259:0    0   3.5T  0 disk
```

Software Installation

This chapter details the procedures for installing the deployment/development software on RedHat/CentOS and Ubuntu operating systems. All software installations use standard Linux RPM and Linux DEB packages and require root access.

The deployment/development software consists of the following packages.

- **Xilinx® runtime (XRT):** XRT provides the libraries and drivers for an application to run on the SmartSSD CSD module.
- **Deployment/Development platform:** The deployment platform provides the base firmware needed to run pre-compiled applications. It cannot be used to compile or create new applications. To create new applications, install the development software detailed in Next Steps. While you can also install the development software on a machine with an installed module, doing so is not necessary to run applications.

Both the XRT and deployment/development platform installation packages can be downloaded from SmartSSD Computational Storage Device.

If you encounter any issues during installation, see Troubleshooting and Known Issues.

IMPORTANT! *Root access is required for all software and firmware installations.*

XRT and Deployment/Development Platform Installation Procedures on RedHat and CentOS

Use the following steps to download and install the XRT and deployment/development platform using a .rpm installation package.

For details on upgrading or downgrading the XRT and deployment/development platform, see Appendix A: Changing XRT and Target Platform Versions.

IMPORTANT! *The installation packages referenced here are updated regularly and the file names frequently change. If you copy and paste any commands from this user guide, be sure to update the placeholders in those commands to match the downloaded packages.*

1. Xilinx® runtime (XRT) installation requires extra packages for enterprise Linux (EPEL) and a related repository. The initial setup depends on whether you are using RedHat or CentOS.

   For Redhat:

   a. Open a terminal window and enter the following command:

   ```
   $ sudo yum-config-manager --enable rhel-7-server-optional-rpms
   ```

   This enables an additional repository on your system.

   b. Enter the following command to install EPEL:

   ```
   $ sudo yum install -y https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
   ```

   For CentOS:

   a. Enter the following command in a terminal window:

   ```
   $ sudo yum install epel-release
   ```

   This installs and enables the repository for Extra Packages for Enterprise Linux (EPEL).

2. Run the following two commands to install kernel headers and kernel development packages. Ensure that `uname` is surrounded by backticks (`` ` ``) and not single quotes (`'`):

   ```
   $ sudo yum install kernel-headers-`uname -r`
   $ sudo yum install kernel-devel-`uname -r`
   ```

   Note: If these yum commands fail because they cannot find packages matching your kernel version, set up a Vault repository. For more information, see Appendix B: Creating a Vault Repository for CentOS.

3. After the previous command completes, reboot your machine.

4. From SmartSSD Computational Storage Device, download the Xilinx runtime (XRT) installation package corresponding to your OS and version.

5. Install the XRT installation package by running the following command from within the directory where the installation packages reside. `<version>` is the latter part of the installation package file name.

   ```
   $ sudo yum install ./xrt_<version>.rpm
   ```

   This will install the XRT and its necessary dependencies. Follow the instructions when prompted throughout the installation.

6. From SmartSSD Computational Storage Device, download and unpack the deployment/development installation file based on your OS and version.

   Unpack the file into a single directory. The location of the directory is not important, however, the directory must not contain any other files.

7. Install the deployment/development packages. From within the directory where the installation packages were unpacked, run the following commands. This will install all the deployment/development packages.

```
Part of development package:
sudo yum install xilinx-u2-gen3x4-xdma-1-202020-1-dev*.rpm
sudo yum install xilinx-u2-gen3x4-xdma-1-202020-1-validate*.rpm
Part of both deployment & development package:
sudo yum install xilinx-u2-gen3x4-xdma-1-202020-1-base*.rpm
```

   The installation of the deployment/development partition and firmware are located in the `/opt/xilinx/firmware` directory and contains the named partition and firmware sub-directories. After installing the deployment/development packages you will see the following message:

```
Partition package installed successfully.
Please flash card manually by running below command:
sudo /opt/xilinx/xrt/bin/xbmgmt flash --update --shell <shell_name>
```

8. Flash the firmware to the module using the command displayed in the output of the previous step. It has the following format:

```
sudo /opt/xilinx/xrt/bin/xbmgmt flash --update --shell <shell_name>
```

   IMPORTANT! *When upgrading from `xilinx_samsung_u2x4_202010_1` platform to `xilinx_u2_gen3x4_xdma_1_202020_1` platform, use the following command:*

```
xbmgmt flash --shell --card 68:00.0 --primary /opt/xilinx/
firmware/u2/gen3x4-xdma/base/partition.xsabin --secondary /opt/
xilinx/firmware/u2/gen3x4-xdma/base/partition.xsabin --flash_type spi
```

   Here the `--card` option takes the `xclmgmt` BDF (Bus device function which can be found out by executing `lspci | grep -i xilinx`).

   If you have multiple modules installed on the server, you must run the `xbmgmt flash` command separately for each module.

   If the module/card has been upgraded, you will see a message similar to the following and no additional installation steps are necessary.

```
Status: shell is up-to-date Card(s) up-to-date and do not need to be
flashed.
```

   If you have multiple modules installed on the server, you must run the `xbmgmt flash` command separately for each module.

9. You will be asked to confirm the update. Type **y** and press the **Enter** key.

```
Status: shell needs updating
Current shell: <current_platform_name>
Shell to be flashed: <platform_to_be_flashed>
Are you sure you wish to proceed? [y/n]:
```

   Flashing will take up to 10 minutes.

> **IMPORTANT!** *Do not enter **Ctrl + c** in the terminal while the firmware is flashing because this can cause the module to become inoperable.*

The following message is the result of successfully flashing a new module. If the command returns `Card Not Found`, perform a cold reboot, and retry. Otherwise, see Troubleshooting.

```
Updating shell on card[0000:68:00.0]
Extracting bitstream from MCS data:
.........
Extracted 9275520 bytes from bitstream @0x0
Writing bitstream to flash 0:
.........
Extracting bitstream from MCS data:
.........
Extracted 9275520 bytes from bitstream @0x0
Writing bitstream to flash 1:
.........
Successfully flashed Card[0000:68:00.0]

1 Card(s) flashed successfully.
Cold reboot machine to load the new image on card(s).
```

10. Cold boot the machine to load the new firmware image on the FPGA.

> **IMPORTANT!** *Be sure to perform a cold boot to fully power off the machine and then power it on again. The image will not boot from flash if the machine is only rebooted (that is, warm reboot).*

The installation for deployment/development package is now complete. You can go directly to Card Bring-Up and Validation to validate the installation.

***Note*:** If your system has older version of XRT installed, it can be removed using, `$ yum remove xrt`.

XRT and Deployment/Development Platform Installation Procedures on Ubuntu

***Note*:** When installing XRT on Ubuntu, if the 2015 version of `pyopencl` is installed on your system, you must uninstall it. The XRT installation will install the 2019 version of `pyopencl` and will return an error if the 2015 version is installed. For more information, see Xilinx Answer Record 73055.

Use the following steps to download and install the XRT and deployment/development platform using a .deb installation package.

For details on upgrading or downgrading the XRT and deployment/development platform, see Appendix A: Changing XRT and Target Platform Versions.

> **IMPORTANT!** *The installation packages referenced here are updated regularly and the file names frequently change. If you copy and paste any commands from this user guide, be sure to update the placeholders in those commands to match the downloaded packages.*

1. Download the Xilinx runtime (XRT) installation package corresponding to your OS and version from SmartSSD Computational Storage Device.

2. Install the XRT installation package by running the following command from within the directory where the installation packages reside. `<version>` is the latter part of the installation package file name.

```
$ sudo apt install ./xrt_<version>.deb
```

   This will install the XRT along with any necessary dependencies. Follow the instructions when prompted throughout the installation.

3. Download and extract the deployment/development installation file based on your OS and version from the SmartSSD Computational Storage Device.

   Unpack the file into a single directory. The location of the directory is not important, however, the directory must not contain any other files.

4. Install the deployment/development packages. From within the directory where the installation packages were unpacked, run the following commands. This will install all deployment/development packages.

```
Part of development package:
sudo yum install xilinx-u2-gen3x4-xdma-1-202020-1-dev*.deb
sudo yum install xilinx-u2-gen3x4-xdma-1-202020-1-validate*.deb
Part of both deployment & development package:
sudo yum install xilinx-u2-gen3x4-xdma-1-202020-1-base*.deb
```

   The installation of the deployment/development partition and firmware are located in the `/opt/xilinx/firmware` directory and contain the named partition and firmware sub-directories. After installing the deployment/development packages you will see the following message:

```
Partition package installed successfully.
Please flash card manually by running below command:
sudo /opt/xilinx/xrt/bin/xbmgmt flash --update --shell <shell_name>
```

   Note: The `xbutil` is being deprecated and will not be supported in future releases. It is recommended to use the `xbmgmt` command for the SmartSSD CSD module.

5. Flash the firmware to the SmartSSD CSD module using the command displayed in the output of the previous step. It has the following format:

```
sudo /opt/xilinx/xrt/bin/xbmgmt flash --update --shell <shell_name>
```

   IMPORTANT! *When upgrading from `xilinx_samsung_u2x4_202010_1` platform to `xilinx_u2_gen3x4_xdma_1_202020_1` platform, use the following command:*

```
xbmgmt flash --shell --card 68:00.0 --primary /opt/xilinx/
firmware/u2/gen3x4-xdma/base/partition.xsabin --secondary /opt/
xilinx/firmware/u2/gen3x4-xdma/base/partition.xsabin --flash_type spi
```

   Here the `--card` option takes the `xclmgmt` BDF (Bus device function which can be found out by executing `lspci | grep -i xilinx`).

   If you have multiple modules installed on the server, you must run the `xbmgmt flash` command separately for each module.

If the module/card has been upgraded, you will see a message similar to the following and no additional installation steps are necessary.

```
Status: shell is up-to-date Card(s) up-to-date and do not need to be
flashed.
```

6. You will be asked to confirm the update. Type **y** and press the **Enter** key.

```
Status: shell needs updating
Current shell: <current_platform_name>
Shell to be flashed: <platform_to_be_flashed>
Are you sure you wish to proceed? [y/n]:
```

Flashing will take up to 10 minutes.

> **IMPORTANT!** *Do not enter **Ctrl + c** in the terminal while the firmware is flashing because this can cause the module to become inoperable.*

The following message is the result of successfully flashing a new module. If the command returns `Card Not Found`, perform a cold reboot, and retry. Otherwise, see Troubleshooting.

```
Updating shell on card[0000:68:00.0]
Extracting bitstream from MCS data:
.........
Extracted 9275520 bytes from bitstream @0x0
Writing bitstream to flash 0:
.........
Extracting bitstream from MCS data:
.........
Extracted 9275520 bytes from bitstream @0x0
Writing bitstream to flash 1:
.........
Successfully flashed Card[0000:68:00.0]

1 Card(s) flashed successfully.
Cold reboot machine to load the new image on card(s).
```

7. Cold boot the machine to load the new firmware image on the FPGA.

> **IMPORTANT!** *Be sure to perform a cold boot to fully power off the machine and then power it on again. The image will not boot from flash if the machine is only rebooted.*

The installation for deployment/development is now complete. You can go directly to Card Bring-Up and Validation to validate the installation.

***Note*:** If your system has older version of XRT installed, it can be removed using, `$ apt remove xrt`.

Card Bring-Up and Validation

After installing the XRT and deployment (or development) platform, the module installation can be verified using the following commands, which are explained in more detail in the following sections.

- `lspci`
- `lsblk`
- `FIO`
- `xbmgmt list`
- `xbmgmt flash --scan`
- `xbmgmt validate`
- `byte copy test kernel execution`

Both the `lspci` and `lsblk` Linux commands are used to validate the module as seen by the OS, as was done when installing the module.

The `FIO` command can be run after installing Flexible I/O software on your server machine and is used to check the access to NVMe SSD.

The `xbmgmt` utilities are included during the XRT package installation. These utilities include multiple commands to validate and identify the installed module(s) and report additional module details including DDR memory, PCIe®, platform name, and system information. This guide uses the `xbmgmt` utilities. See *Vitis Unified Software Platform Documentation: Application Acceleration Development* (UG1393) for a detailed list of commands.

Set the environment to use the utilities by running the following command. Note that the command is dependent on the command shell you are using.

Use the following command in csh shell:

```
$ source /opt/xilinx/xrt/setup.csh
```

Use the following command in bash shell:

```
$ source /opt/xilinx/xrt/setup.sh
```

Running lspci and lsblk

1. Enter the following command to see the Xilinx PCIe devices.

   ```
   $ sudo lspci -vd 10ee:
   ```

2. A message similar to the following is displayed if the module is successfully installed and found by the operating system.

```
5e:00.0 PCI bridge: Xilinx Corporation Device 9134 (prog-if 00 [Normal
decode])
    Flags: bus master, fast devsel, latency 0
    Bus: primary=5e, secondary=5f, subordinate=61, sec-latency=0
    I/O behind bridge: 00000000-00000fff
    Memory behind bridge: b8a00000-b8cfffff
    Prefetchable memory behind bridge: 0000383e00000000-0000383f040fffff
    Capabilities: [40] Power Management version 3
    Capabilities: [48] MSI: Enable- Count=1/1 Maskable- 64bit+
    Capabilities: [70] Express Upstream Port, MSI 00
    Capabilities: [100] Advanced Error Reporting
    Capabilities: [1c0] #19
    Kernel driver in use: pcieport
    Kernel modules: shpchp

5f:00.0 PCI bridge: Xilinx Corporation Device 9234 (prog-if 00 [Normal
decode])
    Flags: bus master, fast devsel, latency 0
    Bus: primary=5f, secondary=60, subordinate=60, sec-latency=0
    I/O behind bridge: 00000000-00000fff
    Memory behind bridge: b8a00000-b8cfffff
    Capabilities: [40] Power Management version 3
    Capabilities: [48] MSI: Enable- Count=1/1 Maskable- 64bit+
    Capabilities: [70] Express Downstream Port (Slot+), MSI 00
    Capabilities: [100] Access Control Services
    Capabilities: [1c0] #19
    Kernel driver in use: pcieport
    Kernel modules: shpchp

5f:01.0 PCI bridge: Xilinx Corporation Device 9434 (prog-if 00 [Normal
decode])
    Flags: bus master, fast devsel, latency 0
    Bus: primary=5f, secondary=61, subordinate=61, sec-latency=0
    I/O behind bridge: 00000000-00000fff
    Prefetchable memory behind bridge: 0000383e00000000-0000383f040fffff
    Capabilities: [40] Power Management version 3
    Capabilities: [70] Express Downstream Port (Slot-), MSI 00
    Capabilities: [100] Access Control Services
    Capabilities: [140] #19
    Kernel driver in use: pcieport
    Kernel modules: shpchp

61:00.0 Memory controller: Xilinx Corporation Device 6987
    Subsystem: Xilinx Corporation Device 1351
        Flags: bus master, fast devsel, latency 0
        Memory at c7200000000 (64-bit, prefetchable) [size=32M]
        Memory at c7204000000 (64-bit, prefetchable) [size=64K]
        Capabilities: [40] Power Management version 3
        Capabilities: [60] MSI-X: Enable- Count=33 Masked-
        Capabilities: [70] Express Endpoint, MSI 00
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [140] #19
        Capabilities: [180] Vendor Specific Information: ID=0040 Rev=0
Len=018 <?>
        Capabilities: [400] Access Control Services
        Capabilities: [480] Vendor Specific Information: ID=0020 Rev=0
Len=010 <?>
        Kernel driver in use: xclmgmt

61:00.1 Memory controller: Xilinx Corporation Device 6988
```

```
    Subsystem: Xilinx Corporation Device 1351
        Flags: bus master, fast devsel, latency 0
        Memory at c7202000000 (64-bit, prefetchable) [size=32M]
        Memory at c7204010000 (64-bit, prefetchable) [size=64K]
        Memory at c7000000000 (64-bit, prefetchable) [size=8G]
        Capabilities: [40] Power Management version 3
        Capabilities: [60] MSI-X: Enable+ Count=32 Masked-
        Capabilities: [70] Express Endpoint, MSI 00
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [400] Access Control Services
        Capabilities: [480] Vendor Specific Information: ID=0020 Rev=0
Len=010 <?>
        Kernel driver in use: xocl
        Kernel modules: xocl
```

3. Enter the following command to see the Samsung NVMe SSD PCIe device.

```
$ sudo lspci -vs <nvme_pcie_dev_id> :
Example: sudo lspci -vs 60:00.0
60:00.0 Non-Volatile memory controller: Samsung Electronics Co Ltd
Device a824 (prog-if 02 [NVM Express])
    Subsystem: Samsung Electronics Co Ltd Device a801
    Flags: bus master, fast devsel, latency 0, IRQ 33
    Memory at b8a00000 (64-bit, non-prefetchable) [size=32K]
    Expansion ROM at <ignored> [disabled]
    Capabilities: [40] Power Management version 3
    Capabilities: [70] Express Endpoint, MSI 00
    Capabilities: [b0] MSI-X: Enable+ Count=64 Masked-
    Capabilities: [100] Advanced Error Reporting
    Capabilities: [148] Device Serial Number 00-00-00-00-00-00-00-00
    Capabilities: [168] Alternative Routing-ID Interpretation (ARI)
    Capabilities: [178] #19
    Capabilities: [198] #26
    Capabilities: [1c0] #27
    Capabilities: [1e8] Single Root I/O Virtualization (SR-IOV)
    Capabilities: [3a4] #25
    Kernel driver in use: nvme
    Kernel modules: nvme
```

4. Enter the following command to see the properties of the Samsung NVMe SSD controller.

```
$ lsblk
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sdb       8:16   0 238.5G  0 disk
sdb2      8:18   0     1K  0 part
sdb5      8:21   0   976M  0 part [SWAP]
sdb1      8:17   0 237.5G  0 part /
sda       8:0    0 931.5G  0 disk
sda2      8:2    0     1K  0 part
sda5      8:5    0  63.6G  0 part
sda1      8:1    0   868G  0 part
nvme0n1 259:0    0   3.5T  0 disk
```

If the `lspci` and/or `lsblk` outputs do not match, see Troubleshooting.

Running FIO

Flexible I/O (FIO) is a disk I/O tool used to baseline SSD performance. After installing the software on the server machine (`-apt -get install fio`), FIO commands can be run as shown in the following examples. These example commands run each of the FIO command for one minute (runtime = 60). `FIO -help` can be used to understand or modify any of the FIO parameters as necessary.

- **Random-Write command:**

```
fio --name=rand-write --ioengine=libaio --iodepth=256 --rw=randwrite --
bs=4k --direct=1 --size=100% --numjobs=12 --runtime=60 --filename=/dev/
nvme0n1 --group_reporting=1
rand-write: (g=0): rw=randwrite, bs=4K-4K/4K-4K/4K-4K, ioengine=libaio,
iodepth=256
...
fio-2.2.10
Starting 12 processes
Jobs: 12 (f=12): [w(12)] [100.0% done] [0KB/2511MB/0KB /s] [0/643K/0
iops] [eta 00m:00s]
rand-write: (groupid=0, jobs=12): err= 0: pid=3894: Fri Mar 20 17:31:14
2020
  write: io=149439MB, bw=2490.5MB/s, iops=637551, runt= 60005msec
    slat (usec): min=0, max=2326, avg= 3.86, stdev= 2.45
    clat (usec): min=615, max=12844, avg=4809.97, stdev=425.89
     lat (usec): min=671, max=12850, avg=4813.91, stdev=425.88
    clat percentiles (usec):
     |  1.00th=[ 4384],  5.00th=[ 4704], 10.00th=[ 4768], 20.00th=[ 4768],
     | 30.00th=[ 4768], 40.00th=[ 4768], 50.00th=[ 4768], 60.00th=[ 4768],
     | 70.00th=[ 4768], 80.00th=[ 4768], 90.00th=[ 4832], 95.00th=[ 4832],
     | 99.00th=[ 8032], 99.50th=[ 8640], 99.90th=[ 9024], 99.95th=[ 9280],
     | 99.99th=[ 9536]
    bw (KB  /s): min=112856, max=315080, per=8.33%, avg=212543.43,
stdev=8157.99
    lat (usec) : 750=0.01%, 1000=0.01%
    lat (msec) : 2=0.03%, 4=0.06%, 10=99.89%, 20=0.01%
  cpu          : usr=11.96%, sys=25.69%, ctx=5205881, majf=0, minf=118287
  IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=0.1%,
>=64=100.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%,
>=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%,
>=64=0.1%
     issued    : total=r=0/w=38256283/d=0, short=r=0/w=0/d=0,
drop=r=0/w=0/d=0
     latency   : target=0, window=0, percentile=100.00%, depth=256

Run status group 0 (all jobs):
  WRITE: io=149439MB, aggrb=2490.5MB/s, minb=2490.5MB/s, maxb=2490.5MB/s,
mint=60005msec, maxt=60005msec

Disk stats (read/write):
  nvme0n1: ios=44/38193768, merge=0/0, ticks=0/182768396,
in_queue=201898004, util=100.00%
```

- **Random-Read command:**

```
fio --name=rand-read --ioengine=libaio --iodepth=256 --rw=randread --
bs=4k --direct=1 --size=100% --numjobs=12 --runtime=60 --filename=/dev/
nvme0n1 --group_reporting=1
rand-read: (g=0): rw=randread, bs=4K-4K/4K-4K/4K-4K, ioengine=libaio,
iodepth=256
...
fio-2.2.10
Starting 12 processes
Jobs: 12 (f=12): [r(12)] [100.0% done] [1798MB/0KB/0KB /s] [460K/0/0
iops] [eta 00m:00s]
rand-read: (groupid=0, jobs=12): err= 0: pid=3969: Fri Mar 20 17:33:55
2020
  read : io=174809MB, bw=2913.2MB/s, iops=745764, runt= 60007msec
    slat (usec): min=0, max=2070, avg= 3.05, stdev= 1.56
    clat (usec): min=37, max=88007, avg=4112.07, stdev=2197.35
     lat (usec): min=41, max=88010, avg=4115.20, stdev=2197.41
    clat percentiles (usec):
     |  1.00th=[  628],  5.00th=[ 1176], 10.00th=[ 1608], 20.00th=[ 2288],
     | 30.00th=[ 2864], 40.00th=[ 3440], 50.00th=[ 3984], 60.00th=[ 4512],
     | 70.00th=[ 5088], 80.00th=[ 5600], 90.00th=[ 6496], 95.00th=[ 7008],
     | 99.00th=[12992], 99.50th=[15424], 99.90th=[17024], 99.95th=[18048],
     | 99.99th=[29312]
    bw (KB  /s): min=106488, max=466448, per=8.33%, avg=248592.66,
stdev=76171.88
    lat (usec) : 50=0.01%, 100=0.01%, 250=0.05%, 500=0.40%, 750=1.25%
    lat (usec) : 1000=1.74%
    lat (msec) : 2=12.14%, 4=34.70%, 10=48.17%, 20=1.51%, 50=0.03%
    lat (msec) : 100=0.01%
  cpu          : usr=13.11%, sys=27.77%, ctx=21583274, majf=0, minf=158272
  IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=0.1%,
>=64=100.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%,
>=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%,
>=64=0.1%
     issued    : total=r=44751076/w=0/d=0, short=r=0/w=0/d=0,
drop=r=0/w=0/d=0
     latency   : target=0, window=0, percentile=100.00%, depth=256

Run status group 0 (all jobs):
   READ: io=174809MB, aggrb=2913.2MB/s, minb=2913.2MB/s, maxb=2913.2MB/s,
mint=60007msec, maxt=60007msec

Disk stats (read/write):
  nvme0n1: ios=44699555/0, merge=0/0, ticks=183464424/0,
in_queue=206110532, util=100.00%
```

- **Seq-Write command:**

```
fio --name=seq-write --ioengine=libaio --iodepth=64 --rw=write --bs=1024k
--direct=1 --size=100% --numjobs=12  --runtime=60 --filename=/dev/nvme0n1
--group_reporting=1
seq-write: (g=0): rw=write, bs=1M-1M/1M-1M/1M-1M, ioengine=libaio,
iodepth=64
...
fio-2.2.10
Starting 12 processes
Jobs: 12 (f=12): [W(12)] [100.0% done] [0KB/3193MB/0KB /s] [0/3192/0
iops] [eta 00m:00s]
seq-write: (groupid=0, jobs=12): err= 0: pid=4050: Fri Mar 20 17:35:20
```

```
2020
  write: io=191090MB, bw=3171.2MB/s, iops=3171, runt= 60243msec
    slat (usec): min=59, max=1312, avg=258.71, stdev=67.88
    clat (msec): min=8, max=498, avg=241.77, stdev=15.25
     lat (msec): min=8, max=499, avg=242.03, stdev=15.24
    clat percentiles (msec):
     |  1.00th=[  229],  5.00th=[  233], 10.00th=[  235], 20.00th=[  237],
     | 30.00th=[  239], 40.00th=[  239], 50.00th=[  241], 60.00th=[  243],
     | 70.00th=[  245], 80.00th=[  247], 90.00th=[  251], 95.00th=[  255],
     | 99.00th=[  262], 99.50th=[  277], 99.90th=[  420], 99.95th=[  449],
     | 99.99th=[  482]
    bw (KB  /s): min=248365, max=410826, per=8.34%, avg=270936.65,
stdev=6678.18
    lat (msec) : 10=0.01%, 20=0.03%, 50=0.11%, 100=0.11%, 250=86.87%
    lat (msec) : 500=12.88%
  cpu          : usr=3.53%, sys=3.87%, ctx=185996, majf=0, minf=218
  IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=0.2%,
>=64=99.6%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%,
>=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.1%,
>=64=0.0%
     issued    : total=r=0/w=191090/d=0, short=r=0/w=0/d=0, drop=r=0/w=0/
d=0
     latency   : target=0, window=0, percentile=100.00%, depth=64

Run status group 0 (all jobs):
  WRITE: io=191090MB, aggrb=3171.2MB/s, minb=3171.2MB/s, maxb=3171.2MB/s,
mint=60243msec, maxt=60243msec

Disk stats (read/write):
  nvme0n1: ios=44/1526654, merge=0/0, ticks=4/360903220,
in_queue=362539680, util=100.00%
```

- **Seq-Read command:**

```
fio --name=seq-read --ioengine=libaio --iodepth=64 --rw=read --bs=1024k --
direct=1 --size=100% --numjobs=12  --runtime=60 --filename=/dev/nvme0n1 --
group_reporting=1
seq-read: (g=0): rw=read, bs=1M-1M/1M-1M/1M-1M, ioengine=libaio,
iodepth=64
...
fio-2.2.10
Starting 12 processes
Jobs: 12 (f=12): [R(12)] [100.0% done] [1771MB/0KB/0KB /s] [1771/0/0
iops] [eta 00m:00s]
seq-read: (groupid=0, jobs=12): err= 0: pid=4123: Fri Mar 20 17:36:55 2020
  read : io=196587MB, bw=3253.1MB/s, iops=3253, runt= 60416msec
    slat (usec): min=42, max=873, avg=146.05, stdev=64.68
    clat (msec): min=29, max=835, avg=235.76, stdev=37.41
     lat (msec): min=29, max=836, avg=235.91, stdev=37.40
    clat percentiles (msec):
     |  1.00th=[  221],  5.00th=[  225], 10.00th=[  227], 20.00th=[  229],
     | 30.00th=[  229], 40.00th=[  231], 50.00th=[  231], 60.00th=[  233],
     | 70.00th=[  235], 80.00th=[  237], 90.00th=[  239], 95.00th=[  243],
     | 99.00th=[  437], 99.50th=[  437], 99.90th=[  725], 99.95th=[  775],
     | 99.99th=[  824]
    bw (KB  /s): min=44122, max=420207, per=8.32%, avg=277209.59,
stdev=30837.04
    lat (msec) : 50=0.15%, 100=0.10%, 250=96.99%, 500=2.45%, 750=0.23%
    lat (msec) : 1000=0.08%
  cpu          : usr=0.21%, sys=4.23%, ctx=188129, majf=0, minf=196789
```

```
  IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=0.2%,
>=64=99.6%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%,
>=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.1%,
>=64=0.0%
     issued    : total=r=196587/w=0/d=0, short=r=0/w=0/d=0, drop=r=0/w=0/
d=0
     latency   : target=0, window=0, percentile=100.00%, depth=64

Run status group 0 (all jobs):
   READ: io=196587MB, aggrb=3253.1MB/s, minb=3253.1MB/s, maxb=3253.1MB/s,
mint=60416msec, maxt=60416msec

Disk stats (read/write):
  nvme0n1: ios=1569192/0, merge=0/0, ticks=359879568/0,
in_queue=362147904, util=100.00%
```

Note: You can use the standard NVMe commands to check NVMe SSD device temperature when running FIO or any acceleration application. For example, `$ nvme smart-log /dev/nvme0n1` (Note that this command needs to be run as root and NVME CLI may need to be installed).

```
Smart Log for NVME device:nvme0n1 namespace-id:ffffffff
critical_warning                    : 0
temperature                         : 35 C
available_spare                     : 100%
available_spare_threshold           : 10%
percentage_used                     : 0%
data_units_read                     : 7,35,30,201
data_units_written                  : 44,37,21,492
host_read_commands                  : 1,68,55,85,846
host_write_commands                 : 21,16,45,54,518
controller_busy_time                : 1,403
power_cycles                        : 2,402
power_on_hours                      : 3,556
unsafe_shutdowns                    : 2,329
media_errors                        : 0
num_err_log_entries                 : 0
Warning Temperature Time            : 0
Critical Composite Temperature Time : 0
Temperature Sensor 1                : 35 C
Temperature Sensor 2                : 34 C
Temperature Sensor 3                : 32 C
Temperature Sensor 4                : 0 C
Temperature Sensor 5                : 0 C
Temperature Sensor 6                : 0 C
Temperature Sensor 7                : 0 C
Temperature Sensor 8                : 0 C
```

If `FIO` does not run and fail with an error, see Troubleshooting.

Running xbmgmt list

`xbmgmt list` confirms that the SmartSSD CSD module is available for use by the XRT for accelerating any application kernel.

```
$ xbmgmt list
INFO: Found total 1 card(s), 1 are usable
```

If `xbmgmt list` output does not match, see Troubleshooting.

Running xbmgmt flash --scan

Use the `xbmgmt flash --scan` command to view and validate the module's current firmware version, as well as display the details of the installed module, including module BDF, platform name, and timestamp.

1. Enter the following command:

   ```
   $ sudo /opt/xilinx/xrt/bin/xbmgmt flash --scan
   ```

   For each module in the server, an output similar to the following example is displayed.

   ```
   Card [0000:61:00.0]
       Card type:        u2
       Flash type:        SPI
       Flashable partition running on FPGA:
           xilinx_u2_gen3x4_xdma_base_1,[ID=0x413bee2e6154c502],[SC=UNKNOWN]
       Flashable partitions installed in system:
           xilinx_u2_gen3x4_xdma_base_1,[ID=0x413bee2e6154c502]
   ```

   In this example, the BDF is `0000:61:00.0`.

   Note: `SC=UNKNOWN` in the previous example can be ignored.

   The name of the platform and associated ID running on the FPGA are found under `Flashable partition running on FPGA` while the ones installed in the system are found under `Flashable partitions installed in system`.

   In the previous output example, the platform on the FPGA and system are identical; the deployment (or development) platform is named `xilinx_u2_gen3x4_xdma_base_1` and the ID/timestamp is `0x413bee2e6154c502`.

2. Verify that the deployment (or development) platform version installed on the FPGA is identical to that installed on the system. You can do this by making sure the lines under `Flashable partition running on FPGA` and `Flashable partitions installed in system` are identical.

   If these versions do not match, see Troubleshooting.

Running xbutil validate

The `xbutil validate` command validates the correct installation by performing the following set of tests:

1. Validates the device found.
2. Checks PCIe link status.
3. Runs a verify kernel on the module.

4. Performs the following data bandwidth tests:
   a. DMA test - Data transfer between host and FPGA DDR through PCIe.
   b. DDR test - Data transfer between kernels and FPGA DDR (device memory bandwidth test in the following `xbutil` command output log).

The `validate` command has the format:

```
xbutil validate -d <card_bdf>
```

where `-d` is optional but useful in an environment with multiple modules where the test needs to run on a specified module and `card_bdf` is the BDF of the module to be validated.

Run the following validate command:

```
$ /opt/xilinx/xrt/bin/xbutil validate
```

If the module was installed correctly, you will see a high-level summary of the tests performed similar to the following output. If the output displayed is not similar to the following, see Troubleshooting.

```
INFO: Found 1 cards

INFO: Validating card[0]: xilinx_u2_gen3x4_xdma_base_1
INFO: == Starting Kernel version check:
INFO: == Kernel version check PASSED
INFO: == Starting AUX power connector check:
AUX power connector not available. Skipping validation
INFO: == AUX power connector check SKIPPED
INFO: == Starting PCIE link check:
INFO: == PCIE link check PASSED
INFO: == Starting SC firmware version check:
INFO: == SC firmware version check PASSED
INFO: == Starting verify kernel test:
INFO: == verify kernel test PASSED
INFO: == Starting DMA test:
Host -> PCIe -> FPGA write bandwidth = 3034.528129 MB/s
Host <- PCIe <- FPGA read bandwidth = 2951.412657 MB/s
INFO: == DMA test PASSED
INFO: == Starting device memory bandwidth test:
.............Host buffer alignment 4096 bytes
Compiled kernel = /opt/xilinx/firmware/u2/gen3x4-xdma/base/test/
bandwidth.xclbin
Shell = b'xilinx_u2_gen3x4_xdma_base_1'
Index = 0
PCIe = GEN3 x 4
OCL Frequency = (0, 0) MHz
DDR Bank = 0
Device Temp = 0 C
MIG Calibration = True
Finished downloading bitstream /opt/xilinx/firmware/u2/gen3x4-xdma/base/
test/bandwidth.xclbin
CU[0] b'bandwidth1:bandwidth1_1' @0x1810000
CU[1] b'bandwidth2:bandwidth2_1' @0x1820000
[0] b'bank0' @0x4000000000
LOOP PIPELINE 16 beats
Test 0, Throughput: 5007 MB/s
```

```
LOOP PIPELINE 64 beats
Test 1, Throughput: 14139 MB/s
LOOP PIPELINE 256 beats
Test 2, Throughput: 15392 MB/s
LOOP PIPELINE 1024 beats
Test 3, Throughput: 10156 MB/s
TTTT: 5007
Maximum throughput: 15392 MB/s
INFO: == device memory bandwidth test PASSED
INFO: == Starting PCIE peer-to-peer test:
Performing P2P Test on bank0 ..............
INFO: == PCIE peer-to-peer test PASSED
INFO: == Starting memory-to-memory DMA test:
M2M is not available. Skipping validation
INFO: == memory-to-memory DMA test SKIPPED
INFO: Card[0] validated successfully.
INFO: All cards validated successfully.
```

Running Byte Copy Test Kernel

Byte copy test kernel is provided in the associated zip file to validate P2P transfers using the SmartSSD CSD module. This kernel exercises all the possible data paths through PCIe switch including P2P transfers by NVMe SSD.

This test exercises two sub-sequences:

1. P2P Read
2. P2P Write

During the P2P read sequence, the data flows from the SSD -> FPGA DDR -> Byte Copy Read (from FPGA DDR) -> Byte Copy Write (into FPGA DDR) -> Host DDR.

***Note*:** The flow of data from SSD to FPGA DDR is called P2P read. This is iteration 0 in the following log.

During the P2P write sequence, the data flows from Host DDR -> FPGA DDR -> Byte Copy Read from FPGA DDR -> Byte Copy Write into FPGA DDR -> SSD.

***Note*:** The flow of data from FPGA DDR to SSD is called P2P write. This is iteration 1 in the following log.

Run the following validate command.

```
./run_async_bytecopy.sh
iteration 0
INFO: Successfully opened NVME SSD /dev/nvme1n1
Detected 1 devices, using the 0th one
INFO: Importing ./bytecopy.xclbin
INFO: Loaded file
INFO: Created Binary
INFO: Built Program
INFO: Preparing 131072KB test data in 32 pipelines
INFO: Kick off test
SSD -> FPGA(p2p BO) -> FPGA(host BO) -> HOST
 overall           72708us        100.00%        1760.47MB/s
     p2p           39827us         54.78%        3213.90MB/s
INFO: Evaluating test result
INFO: Test passed
```

```
iteration 1
INFO: Successfully opened NVME SSD /dev/nvme1n1
Detected 1 devices, using the 0th one
INFO: Importing ./bytecopy.xclbin
INFO: Loaded file
INFO: Created Binary
INFO: Built Program
INFO: Preparing 131072KB test data in 32 pipelines
INFO: Kick off test
HOST -> FPGA(host BO) -> FPGA(p2p BO) -> SSD
 overall           97844us        100.00%        1308.20MB/s
     p2p           53183us         54.35%        2406.78MB/s
INFO: Evaluating test result
INFO: Test passed
iteration 2
INFO: Successfully opened NVME SSD /dev/nvme1n1
Detected 1 devices, using the 0th one
INFO: Importing ./bytecopy.xclbin
INFO: Loaded file
INFO: Created Binary
INFO: Built Program
INFO: Preparing 131072KB test data in 32 pipelines
INFO: Kick off test
SSD -> FPGA(p2p BO) -> FPGA(host BO) -> HOST
 overall           75113us        100.00%        1704.10MB/s
     p2p           40916us         54.47%        3128.36MB/s
INFO: Evaluating test result
INFO: Test passed
```

***Note*:** Byte copy application test is primarily a functional test and has not been optimized for performance. You may see variations in the performance numbers from iteration to iteration. Byte copy test sources are available at: Xilinx GitHub.

Next Steps

If you are an application developer who wants to develop and deliver accelerated applications, install the Vitis™ software platform. It allows you to develop, debug, and optimize accelerated applications for the SmartSSD CSD module. Installation instructions can be found in *Vitis Unified Software Platform Documentation: Application Acceleration Development* (UG1393).

For complete details on the development flow and getting started in Vitis, see *Vitis Unified Software Platform Documentation: Embedded Software Development* (UG1400). For an introduction to Vitis methodology, see *Vitis Unified Software Platform Documentation: Application Acceleration Development* (UG1393).

Troubleshooting

The following table lists potential issues, causes, and fixes related to module installation.

Table 2: **Module Troubleshooting**

Issue	Potential Cause	Fix
Card not found/ 0 card(s) found in `xbmgmt list`.	Module not correctly installed.	Reinstall the module following the installation instructions. Follow module bring up steps to check if module shows up.
	Module not compatible with server.	Check if the server is listed in the qualified servers.
	Kernel version is incompatible.	Run `uname -r` to check the kernel version. Ensure that the kernel version matches the version listed for your OS in Software Installation.

Card found but `xbmgmt list` shows 0 usable device.	XRT Driver has not loaded successfully, or the module is not flashed successfully.	Reload XRT drivers using the following commands:
		`sudo rmmod xocl`
		`sudo rmmod xclmgmt`
		`sudo modprobe xocl`
		`sudo modprobe xclmgmt`
		If `xbmgmt list` continues to have the same issue, then perform a cold-reboot.

Sudden unexpected reboot or loss of PCIe devices (lspci) or NVMe devices (lspci or lsblk).	Module may be overheating. ***Note***: The `xbutil query` command can be used to track the FPGA operating temperature and `nvme -smart log` command can be used to track the NVMe SSD operating temperature.	Ensure that operating ambient conditions do not exceed specifications and there is adequate cooling for module to function properly.
Read issued at some DDR location results in sudden reboot or some kind of PCIe or NVMe SSD I/O error. Running application suddenly crashes with some kind of PCIe error / machine reboot or some kind of NVMe error.	Uninitialized DDR location read can cause this behavior.	DDR controller's ECC function requires write to the memory location prior to performing read. Therefore, ensure that user application does not read the DDR location that is never written, that is, P2P or non-P2P buffers in DDR must be written or initialized prior to read.
`xbmgmt --flash` returns the error: `Specified DSA is not applicable`	Correct type of deployment platform package not installed.	Install the correct type of deployment platform package.
Flashing the module does not complete after 20 minutes.	The flash operation has failed.	Perform cold-reboot and then re-flash the module.
XRT installation incomplete or unsuccessful.	Missing dependent packages.	Contact your Linux administrator.
Deployment platform installation incomplete or unsuccessful.	Missing dependent packages.	Contact your Linux administrator.
Unable to install packages on RedHat and CentOS.	Incorrect permissions for download directory, for example, a `/home/` directory.	Download the packages to a directory where root has read access (for example, `/tmp`). Use the full path to the RPM package when installing. yum will fail with a relative path to RPM package.

Table 2: **Module Troubleshooting** *(cont'd)*

Issue	Potential Cause	Fix
Run time fails with following message: `Error: Failed to find Xilinx platform`	Failed to source the `setup.sh` script.	Source `/opt/xilinx/xrt/setup.sh`
XRT package fails to install on CentOS7.4, CentOS7.5, or CentOS7.6	Kernel development headers are missing. The XRT package is missing a dependency on `kernel-devel` and `kernel-headers`.	Manually install `kernel-devel` and `kernel-header` with yum install: `$ sudo yum install kernel-headers-` `uname -r` `$ sudo yum install kernel-devel-` `uname -r` ` ***Note***: Do not run `sudo yum upgrade`. This will update the kernel-headers to an incompatible version.
When installing the XRT, you see the following message: `N: Can't drop privileges for downloading as file '/root/xrt_201802.2.1.79_16.04.deb' couldn't be accessed by user '_apt'. - pkgAcquire::Run (13: Permission denied)`	This is caused by running `sudo apt install` as root.	The XRT will install correctly, despite the error. You can find more information about this error on AskUbuntu.

For help with additional issues, please contact Xilinx customer support.

Known Issues

The following table lists known issues. See Xilinx Answer Record 75177 for additional known issues.

Table 3: **Known Issues**

Area	Description	Comments/Recommendations
General	The module is not present when running `xbutil` or `lspci`. The module may not have been ready when the server enumerated PCI Express.	Potential Fix: Warm Reboot the server, disable Fast Boot.
General	The SmartSSD CSD module has not trained to the full expected PCI Express link width or link speed.	Ensure that the SmartSSD CSD module is plugged into a Gen 3x4 or higher capable slot. Then cold reboot and see if the module trains to the correct settings.

Table 3: **Known Issues** *(cont'd)*

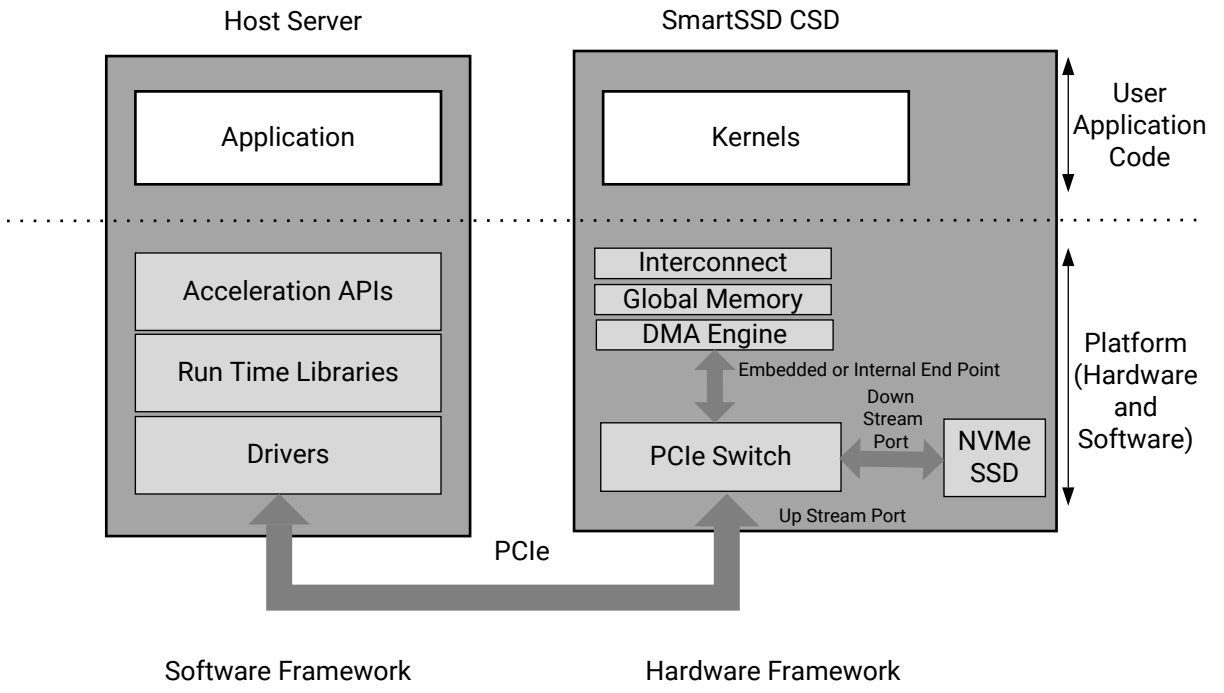
Area	Description	Comments/Recommendations
Power	The SmartSSD CSD module limits the power to 25W. Exceeding these power limits can cause system instability.	To manage power consumption, review design power usage and ensure that it is within the power limits. Actual application power consumption is provided by the `xbutil query`—which reports power consumption at the input to the power regulator.
Reset to Factory (Golden) Image	Rev 1.2 PoC version of the SmartSSD CSD modules does not come with write protected factory (Golden) image. This means that `xbmgmt flash -factory reset` is currently not supported.	Support for this feature has been added in the latest revision of the card (Rev FS and future revs). Refer to Xilinx Answer Record 75177 for details on the latest revision of the card and corresponding shell that needs to be flashed on this card.
General	The SmartSSD CSD U.2 platform does not support PLRAM memory for accelerator (Vitis accelerated kernel) uses.	N/A

Platform Notes

Overview

The Vitis™ core development kit provides verified platforms defining all the required hardware and software interfaces (shown in gray in the following figure), allowing you to design custom acceleration applications (shown in white) that are easily integrated into the Vitis programming model.

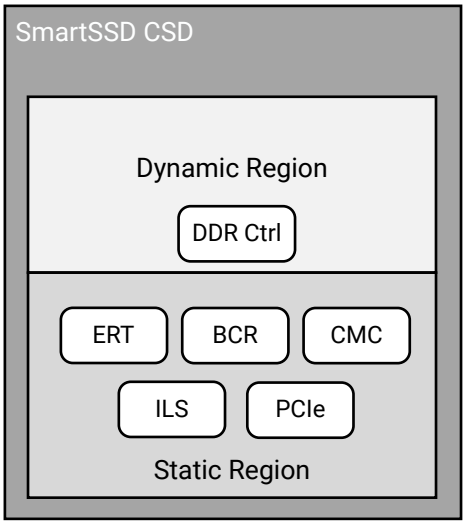
Figure 1: **Platform Overview**



The SmartSSD® CSD target platform (shell) consists of a static region and dynamic (user programmable) region. The static region of the platform provides the basic infrastructure for the module to communicate with the host and hardware support for the kernel. It includes the following main features.

- **Three port PCIe Switch (PCIe):** PCIe switch up stream port connects to the host PCIe link and PCIe switch down stream port connects to Samsung NVMe SSD Controller PCIe link (embedded on-board) to enable NVMe SSD access to the host in a transparent manner with minimal add-on latency without affecting the SSD I/O performance. PCIe switch embedded endpoint port connects to the XDMA IP internally and enables the acceleration function through the FPGA.
- **Base Clocking and Reset (BCR):** Basic clocking and reset for module bring-up and operation.
- **Isolation logic structure (ILS):** Reset and partial reconfiguration isolation structure which are required for isolating static region logic from dynamic region logic during the partial bitstream download.
- **Card Management Controller (CMC):** Responsible for module power and FPGA temperature monitoring as well as power and temperature driven kernel clock throttling.
- **Embedded Run Time Scheduler (ERT):** Schedules and monitors compute units during kernel execution.

Figure 2: **Target Platform Shell Dynamic and Static Regions**



Acceleration kernels go into the dynamic region. The features and resources available for accelerated kernels are described in U.2 Platform.

Development Target Shell

Xilinx provides a high-performance development target shell which can be used to create custom acceleration applications for the SmartSSD CSD. The U.2 platform (shell) provides:

- Host access to Samsung NVMe SSD in a transparent manner with minimal add-on latency and without affecting SSD I/O throughput and performance
- Direct PCIe peer-to-peer (P2P) transfers from NVMe SSD to FPGA-DDR used in storage application acceleration
  - P2P reduces the data transfer latencies (by avoiding hop to the Host-DDR) and improves the application performance
- Memory-mapped DMA transfers
- Kernel support for memory-mapped AXI4

The following table provides PCIe and DDR memory link widths and expected (theoretical) max performances.

Table 4: **Platform (Shell) Link Widths and Throughputs**

Feature	Properties	Note
Host Interface	PCIe Gen3 x4 with 128-bit data path, and 250 MHz clock	On the EEP side, this is brought out at the XDMA M_AXI interface (128-bit 250 MHz clock). Over this interface, the DMA transactions occur between the host and FPGA DDR memory.
SSD Interface	PCIe Gen3 x4 with 128-bit data path and 250 MHz clock	On the EEP side, this is brought out at the XDMA M_AXI_BYPASS interface (128-bit 250 MHz clock) which is used to make P2P transactions
PCIe Switch Internal Data Path	128-bit 250 MHz clocks	USP and DSP PCIe clocks are treated as asynchronously by PCIe Switch design
DDR Memory	Single channel 64-bit 1200 MHz DDR memory, Size = 4 GB	19.2 GB/s theoretical max throughput
DDR Memory Controller Interface	512-bit 300 MHz clock	19.2 GB/s theoretical max throughput

Platform Naming and Life Cycle

Platform Naming Convention for the 2020.2 Release

Starting with the 2020.2 release, the SmartSSD CSD platform is delivered through three types of Linux installation packages outlined in the following table.

Table 5: **Platform Installation Package Types**

Package	Description
Partition	Contains a device bitstream that implements part of the deployment platform in the SmartSSD CSD module.
Validate	Contains code to validate a platform installation and SmartSSD CSD module setup.

Table 5: **Platform Installation Package Types** *(cont'd)*

Package	Description
Firmware	Not applicable.

Notes:

1. CMC firmware is embedded in the U.2 platform bitfile and is loaded at power ON.

Partition and Validate Package Naming

This section describes the package naming convention for partition and validate types. The partition and validate installation package names are generated by concatenating the following elements:

```
<name>_<version>_<release>_<architecture>[-<OS version>].<extension>
```

Each element consists of one or more sub-elements and are further described in the following table.

Table 6: **Partition and Validate Package Naming Elements**

Element	Sub-Element	Description	Examples
Name	Company	Vendor name	Xilinx
	Card	Card name	u2
	Chassis	Connectivity to the server	gen3x4-xdma (PCIe Switch Up Stream Port)
	Partition	Partition name distinguishes the partition type and can be one of base, shell, or validate	base shell validate
Version	Iteration (s)	Version of chassis. Dot separated list of one or more integers. Increments when the corresponding chassis interface changes	1 1.1
Release	Release	Integer release number	Integer release number
Architecture	Architecture	Indicates the architecture the package is built for. noarch – No Architecture all	noarch all
OS Version	OS Version	Only present for Ubuntu packages. Indicates supported OS version	16.04 18.04
Extension	Extension	Package file extension	RPM DEB

An example of a deployment installation package is as follows.

```
xilinx-u2-gen3x4-xdma-base-1-*.noarch.rpm
<xilinx-u2-gen3x4-xdma-base-1*16.04.deb>
<xilinx-u2-gen3x4-xdma-base-1*18.04.deb>
```

After a deployment partition package is installed, you can use XRT commands to display the partition installed on the card (see xbmgmt utility in the Application Acceleration Development flow of the *Vitis Unified Software Platform Documentation* (UG1416). Because the version number indicates compatibility with other partitions, the release number is not displayed. The displayed partition name for the example package is as follows.

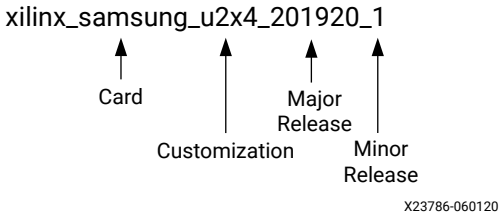
```
xilinx-u2-gen3x4-xdma-base-1
```

Platform Naming Convention Prior to the 2020.2 Release

Platforms are delivered via standard RPM and DEB Linux installation packages. The package name adheres to the nomenclature described in the following figure and provides the target module name, any customized configurations, the major release version, and the minor release versions.

For example, a platform for the SmartSSD CSD module that was built using the 2019.2 release of the Vitis tools might have a package name of `xilinx_samsung_u2x4_201920_1`.

Figure 3: **Example of Package Name**



Package Naming Conventions

Package names include the following information:
`<company>_<card>_<customization>_<major_release>_<minor_release>`

- **Company:** Xilinx
- **Card:** Series name of the module.
- **Customization:** Significant platform feature set. In this case, three port PCIe switch connecting to Gen3x4 NVMe SSD through Down Stream Port (DSP) is the available customization.
- **Major Release:** Includes new features or capabilities. The six-digit number represents the Vitis tool release (version) used to build the platform. The platform can work across multiple major XRT and Vitis releases.
- **Minor Release:** Includes bug fixes and minor updates. Appends the major release with another number.

RECOMMENDED: *Whenever possible, update to the latest release of a platform.*

Life-cycle of the U.2 Platform

Platforms have at least one year of backward compatibility with XRT, but not more than two. If IP used in the dynamic part of the platform is auto-upgraded for the same time frame, then generally:

- A platform generated from a release that has major revision of tools/run time such as 2019.1 is backward compatible until the last release of 2020 (2020.2).
- A platform generated from a release that has minor revision of tools/run time such as 2019.2 is also backward compatible until the last release of 2020 (2020.2).

***Note*:** Xilinx reserves the right to make a backward *incompatible* change once a year with a major revision of XRT, a platform, or the Vitis core development kit. Major revision changes are usually done in the first release of a calendar year.

U.2 Platform

- **Platform name:** `xilinx_u2_gen3x4_xdma_1_202020_1`

  IMPORTANT! *The `xilinx_u2_gen3x4_xdma_1_202020_1` platform is compatible to Rev 1.2 PoC version of the card. Latest revision of cards (RevFS or future versions), support Golden base image and the corresponding compatible shell is `xilinx_u2_gen3x4_gc_xdma_1_202020_1`. For various details about card revisions as well as general do's and dont's, see Xilinx Answer Record 75177.*

  CAUTION! *Rev 1.2 PoC version of the card MUST NOT be flashed with `xilinx_u2_gen3x4_gc_xdma_1_202020_1` to avoid bricking the card.*

- **Supported By:**
  - Vitis tools 2020.2.

    ***Note*:** Vitis tools 2020.2 is the recommended version for generating accelerator/kernel xclbins.

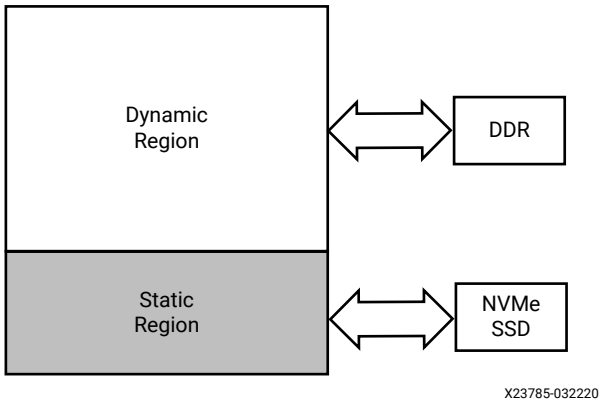
- **Platform (Shell) Version:** v2.13

  ***Note*:** Platform (shell) version can be identified using the version register as mentioned in Appendix E: Miscellaneous Information and Settings.

- **Timestamp:** `0x3ea8991214e76612`
- **Release Date:** November 2020
- **Created With:** 2020.2 tools
- **Supported XRT Versions:** 2020.2
- **Host Link Speed:** PCIe Gen3 x4
- **NVMe SSD Link Speed:** PCIe Gen3 x4
- **Target Module:** SmartSSD CSD module
- **Release Notes:** Release notes for the SmartSSD CSD module are available in Xilinx Answer Record 75176.
- **Known Issues:** For known issues related to the SmartSSD CSD module, see Xilinx Answer Record 75177.
- **Qualified Servers:** A list of qualified servers available for the SmartSSD CSD module can be found in Xilinx Answer Record 75178.

The platform implements the device floorplan shown in the following figure and uses resources from the static region Pblock of the device. The dynamic region Pblock instantiates the DDR memory controller that is connected to the host through the XDMA integrated PCIe switch in the static region.

Figure 4: **Floorplan**



Card Thermal and Electrical Protections

The SmartSSD CSD module provides FPGA clock throttling and SSD performance throttling to ensure production modules operate within electrical and thermal limits while running acceleration kernels.

FPGA clock throttling protection reduces the kernel clock frequencies when module power consumption or FPGA temperature reaches or exceeds their respective clock throttling threshold as listed in the following table. It is a dynamic process that lowers the clock frequencies while power exceeds associated threshold. By lowering the clock frequencies, clock throttling reduces the required power and subsequently generated heat. Only when both module power and FPGA temperature fall below their respective clock throttling threshold values will the application clocks be restored to full performance.

SSD performance throttling increases SSD I/O delay to reduce the SSD performance when module power consumption or SSD temperature reaches or exceeds their respective throttling threshold as listed in the following table. It is a dynamic process that lowers SSD performance while power exceeds associated threshold. By lowering the SSD performance, overall power consumption is reduced and subsequently SSD temperature is reduced. Only when both module power and SSD temperature fall below their respective threshold values will the SSD performance be restored to full performance.

Table 7: **Thermal and Electrical Protection Thresholds**

Parameter	Threshold
FPGA Temperature	93°C
Card Power Consumption	25W
SSD Temperature	74°C

Notes:

1. FPGA temperature and power thresholds used by the clock throttling protection feature are hard limits and cannot be increased.

IMPORTANT! *If needed, you can lower temperature or power thresholds for the FPGA clock throttling function (after every cold or warm reboot) as per the example provided in Changing (Lowering) FPGA Temperature and Power Thresholds for Clock Throttling.*

Card and Clock Shutdown Protection

The SmartSSD CSD platform implements card shutdown and accelerator (Vitis kernel) clock shutdown features (along with the performance throttling feature described previously) to ensure that production card operates within the specified electrical and temperature limits.

Card Shutdown Protection

Card shutdown feature is implemented using the SYSMON primitives thermal management option. This feature is provided to protect the card from permanent damage in certain failure scenarios, such as, blocking or disruption of airflow over the card due to fan failure which can cause excessive heating of the card. When card shutdown temperature threshold is reached, FPGA is disabled which results in loss of the host PCIe connection. Card shutdown feature is controlled by SYSMON hard primitive with default threshold value as 125°C. The platform (shell) design sets the FPGA shutdown temperature threshold as 100°C.

***Note*:** A cold reboot of the server is required to recover after the card shuts down.

Clock Shutdown Protection

Clock shutdown protection shuts down the accelerator (Vitis kernel) clocks when FPGA temperature exceeds clock shutdown threshold (97°C). This will cause an AXI firewall trip that can crash the application on the host. Because the card ends up in an unknown state the XRT driver will issue a command to reset the card. It typically takes a couple minutes until the card is usable again.

***Note*:** Review the Linux dmesg command output to determine if a protection was activated. A message containing following text may appear in the log indicating clock shutdown event:

```
...Critical temperature or power event. Kernel clocks have been stopped...
```

Clocking

The platform is designed to provide a 300 MHz default clock to run the accelerator. When the Vitis accelerator generation flow is run, platform (shell) dynamic region gets re-implemented. Based on the final placement/routing actual frequency is known after implementation.

Available Resources After Platform Installation

The following table lists the available resources in the dynamic region partition block. These resources are available for the Vitis accelerated kernel and associated interconnect needed to connect to the FPGA DDR memory controller (that is already present in the dynamic region).

Table 8: **SmartSSD CSD Platform Resource Availability**

Resource	Total Available Value
CLB LUT	330K (21K)
CLB Register	660K (25K)
Block RAM Tile	628 (26)
UltraRAM	80 (0)
DSP	1252 (3)

Notes:

1. Values provided in parenthesis are FPGA DDR memory controller resources consumed in the dynamic region.

Deployment Platform Installation

To run applications with this platform, download the deployment installation packages corresponding to your OS. Then, use the installation procedures described in Card Installation Procedures.

Accelerated applications have software dependencies. Work with your accelerated application provider to determine which XRT version to install.

Development Platform Installation

For developing applications for use with the SmartSSD CSD module you must install and use the Vitis software platform. To set up an accelerator module for use in the development environment, follow the installation steps in:

- Installing Data Center Platforms in the *Vitis Unified Software Platform Documentation* (UG1416)
- Installing Xilinx Runtime in the *Vitis Unified Software Platform Documentation* (UG1416)

To generate your own kernel binaries and run applications with this platform, download the development installation packages corresponding to your OS. Then, use the installation procedures described in Card Installation Procedures.

XILINX

Appendix A

Changing XRT and Target Platform Versions

The SmartSSD CSD target platform revisions can change significantly between releases. To ensure a successful upgrade (or downgrade) of the SmartSSD CSD module XRT and platform, carefully follow the instructions in the following sections. Failure to adhere to these procedures can result in an unstable installation or other issues.

RedHat and CentOS

During upgrading, downgrading, or uninstalling, it can be useful to list the currently installed SmartSSD CSD packages. To list the currently installed deployment platform package, run the following command in a Linux terminal:

```
$ yum list installed | grep -i xilinx
```

To list the currently installed XRT package, run the following command:

```
$ yum list installed | grep -i xrt
```

Upgrading Packages

You can upgrade the XRT and deployment platform on your SmartSSD CSD module by following these steps. Currently, both packages must be upgraded concurrently.

Download the desired XRT and deployment platform packages and follow installation steps 5 through 10 in XRT and Deployment/Development Platform Installation Procedures on RedHat and CentOS.

Downgrading Packages

No downgrade package is available. While beta packages are available for the SmartSSD CSD, it is not recommended to downgrade to a beta version.

`xilinx_samsung_u2x4_201920_3`, `xilinx_samsung_u2x4_202010_1`, and `xilinx_u2_gen3x4_xdma_1_202020_1` are the available production packages. It is recommended to use the latest production version i.e., `xilinx_u2_gen3x4_xdma_1_202020_1`.

Uninstalling Packages

To completely uninstall the SmartSSD CSD XRT and deployment platform packages, run the following command in a Linux terminal. Uninstalling XRT also uninstalls the deployment platform.

```
$ sudo yum remove ./<xrt_package_name>
```

***Note*:** Make sure that all of the platform packages are displayed in the output terminal after running the command. If not, manually list the packages using the `list` command as explained in the RedHat and CentOSsection, then delete the remaining packages using the `remove` command.

Ubuntu

During upgrading, downgrading, or uninstalling, it can be useful to list the currently installed SmartSSD CSD packages. To list the currently installed deployment platform package, run the following command in a Linux terminal:

```
$ apt list --installed | grep -i xilinx
```

To list the currently installed XRT package, run the following command:

```
$ apt list --installed | grep -i xrt
```

Upgrading Packages

You can upgrade the XRT and deployment platform on your SmartSSD CSD module by following these steps. Currently, both packages must be upgraded concurrently.

Download the desired XRT and deployment platform packages. Follow installation steps 5 through 10 in XRT and Deployment/Development Platform Installation Procedures on Ubuntu.

Downgrading Packages

No downgrade package is available. While beta packages are available for the SmartSSD CSD, it is not recommended to downgrade to a beta version.

`xilinx_samsung_u2x4_201920_3`, `xilinx_samsung_u2x4_202010_1`, and `xilinx_u2_gen3x4_xdma_1_202020_1` are the available production packages. It is recommended to use the latest production version i.e., `xilinx_u2_gen3x4_xdma_1_202020_1`.

Uninstalling Packages

To completely uninstall the SmartSSD CSD XRT and deployment platform packages, run the following command in a Linux terminal. Uninstalling XRT also uninstalls the deployment platform.

```
$ sudo apt remove ./<xrt_package_name>
```

***Note*:** Make sure that all of the target platform packages are displayed in the output terminal after running the command. If not, manually list the packages using the `list` command as explained in the Ubuntu section, then delete the remaining packages using the `remove` command.

IMPORTANT! *When upgrading from `xilinx_samsung_u2x4_202010_1` platform to `xilinx_u2_gen3x4_xdma_1_202020_1` platform, use the following command:*

```
xbmgmt flash --shell --card 68:00.0 --primary /opt/xilinx/firmware/u2/
gen3x4-xdma/base/partition.xsabin --secondary /opt/xilinx/firmware/u2/
gen3x4-xdma/base/partition.xsabin --flash_type spi
```

Here the `--card` option takes the `xclmgmt` BDF (Bus device function which can be found out by executing `lspci | grep -i xilinx`).

If you have multiple modules installed on the server, you must run the `xbmgmt` flash command separately for each module.

Creating a Vault Repository for CentOS

On CentOS, `yum install kernel-headers` always installs the latest version of the headers, but might not match your kernel version. This causes the installation of XRT to skip compilation of the driver modules and will silently fail. To correctly install XRT, you must create a vault repository file that points to versions matching the kernel.

The following is an example repository for CentOS 7.4 created in the following file:

`/etc/yum.repos.d/centos74.repo`

```
# CentOS-Base-7.4.repo
#
# This repo is locked to 7.4.1708 version
#
# C7.4.1708
[C7.4.1708-base]
name=CentOS-7.4.1708 - Base
baseurl=http://vault.centos.org/7.4.1708/os/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
enabled=1
[C7.4.1708-updates]
name=CentOS-7.4.1708 - Updates
baseurl=http://vault.centos.org/7.4.1708/updates/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
enabled=1
[C7.4.1708-extras]
name=CentOS-7.4.1708 - Extras
baseurl=http://vault.centos.org/7.4.1708/extras/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
enabled=1
[C7.4.1708-centosplus]
name=CentOS-7.4.1708 - CentOSPlus
baseurl=http://vault.centos.org/7.4.1708/centosplus/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
enabled=1
[C7.4.1708-fasttrack]
name=CentOS-7.4.1708 - CentOSPlus
baseurl=http://vault.centos.org/7.4.1708/fasttrack/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
enabled=1
```

***Note*:** For CentOS 7.5, create the repo file `/etc/yum.repos.d/centos75.repo` and add the above content, replacing "7.4.1708" with "7.5.1804". Similarly, for CentOS 7.6, create the repo file `/etc/yum.repos.d/centos76.repo` and add the above content, replacing "7.4.1708" with "7.6.1810".

XILINX

Appendix C

Accessing the SmartSSD CSD through a Virtual Machine

The SmartSSD CSD module has the following physical functions (PFs) that can be accessed in a virtualized setup.

- Xilinx Mgmt PF (PCIe ID `0x6987`)
- Xilinx User PF (PCIe ID `0x6988`)
- Samsung NVMe Controller PF (PCIe ID `0xa824`)

These three physical functions can be accessed by the virtual machine (VM) using the KVM-Linux packages part of the Linux kernel. The following VM configurations are supported on the SmartSSD CSD module.

- **Passthrough Xilinx User PF and Samsung NVMe Controller PF:** In this setup, Xilinx Mgmt PF will remain in the host and cannot be accessed by the VM.
- **Passthrough of All Three PFs:** In this setup, all three PFs can be accessed by the VM.

IMPORTANT! *Before attempting VM (passthrough) access, ensure that* `xbmgmt list` *displays one usable device. If not, check the Troubleshooting section.*

Achieving P2P Baremetal Performance in the VM

In a virtualized environment, for any transaction between the NVMe and Xilinx User PF, the request has to be routed all the way to the Input/Output Memory Management Unit (IOMMU) at the host for address translation and then re-routed to the Xilinx PF. This leads to degradation of the peer-to-peer (P2P) throughput in the VM.

To avoid this degradation, SmartSSD CSD platform has implemented a "Passthrough Virtualization Using Range Translation Feature". This feature is available to you at no extra cost and is enabled by default. With this feature, P2P performance in the virtualized setup is expected to be the same as baremetal setup.

XILINX

Appendix D

Hot Plug Support on the SmartSSD CSD

Managed Hot Add and Removal for the SmartSSD CSD

This section describes hot plug support for the SmartSSD CSD module. The xbmgmt tool has been enhanced to support hot plug operation for Xilinx devices. Python based xilinx-hotplug tool is provided to support hot plug for both, Samsung SmartSSD CSD and Xilinx devices.

Python scripts associated with this section can be found in the zip file that is available for download along with this user guide.

xbmgmt Tool

The xbmgmt tool has been enhanced to support the hot plug subcommand. This is a hidden command with two sub-commands, `--online | --offline`.

- **xbmgmt scan <card_bdf>:** Display the Xilinx device.

```
xilinx_samsung_u2x4_201920_2(ts=0x5e0ddca4) mgmt(inst=15873)
```

- **xbmgmt hotplug:** Display the hot plug command menu.

```
Experts only sub-command, use at your own risk. --offline bdf | --online
```

- **xbmgmt hotplug --offline <card_bdf>:** Remove a Xilinx device.

```
CAUTION: Performing hotplug command. This command is going to impact both
user pf and mgmt pf.
Please make sure no application is currently running. Are you sure you
wish to proceed? [y/n]: y
```

- **xbmgmt hotplug --online:** Rescan all Xilinx devices.

```
CAUTION: Performing hotplug command. This command is going to impact both
user pf and mgmt pf.
Please make sure no application is currently running. Are you sure you
wish to proceed? [y/n]: y
```

***Note*:** For Xilinx devices, only the mgmt PF will be listed out. Removing a mgmt PF will also remove the corresponding user PF. Multiple mgmt PF support is currently not available.

xilinx-hotplug Tool

This is a python based tool that supports hot plug functionality for both Xilinx and Samsung SmartSSD CSD. Xilinx and SSD can both be scanned and hot plug related commands can be triggered using this tool. This script uses xbmgmt internally to manage Xilinx devices. The xbmgmt path needs to be mentioned in the following location.

```
xilinx-hotplug-tool/utils/common.py
XILINX_MGMT_TOOL = “/opt/xilinx/xrt/bin/xbmgmt”
```

- **python xilinx-hotplug.py --help:** Display the tool's help menu.

```
xilinx-hotplug.py [-h] [--scan] [--remove REMOVE] [--rescan [RESCAN]]
optional arguments:
  -h, --help                Show this help message and exit
  --scan                    Scan the PCIe devices for hot plug support
  --remove REMOVE         Remove <B:D.F> a PCIe device given as an argument
  --rescan [RESCAN]       Rescan [None|<B:D.F>] the PCIe sub-tree under
this device. If no BDF if provided then rescan will be done from Root Port
```

- **python xilinx-hotplug.py --scan:** Display the PCIe devices for Xilinx and Samsung SSD.

```
 [RP] 5d:00.0
     --> [US] 5e:00.0
         --> [DS] 5f:00.0
             --> [ssd-EP] 60:00.0
         --> [DS] 5f:01.0
             --> [mgt-EP] 61:00.0
```

- **python xilinx-hotplug.py --remove 61:00.0:** Remove a PCIe device from the previous list.

```
# python xilinx-hotplug.py --scan
 [RP] 5d:00.0
     --> [US] 5e:00.0
         --> [DS] 5f:00.0
             --> [ssd-EP] 60:00.0
         --> [DS] 5f:01.0
```

- **python xilinx-hotplug.py –rescan:** Rescan Xilinx and Samsung SSD devices.

```
python xilinx-hotplug.py --scan
 [RP] 5d:00.0
     --> [US] 5e:00.0
         --> [DS] 5f:00.0
             --> [ssd-EP] 60:00.0
         --> [DS] 5f:01.0
             --> [mgt-EP] 61:00.0
```

***Note*:** This tool internally uses the xbmgmt (XRT) tool to remove and rescan a Xilinx device. Rescan can be done from a particular port (US/DS/RP) by providing the BDF of the port `python xilinx-hotplug.py --rescan 3b:00.0`. If no argument is provided for the rescan, then the rescan will be done from the top of the PCIe subsystem.

Limitations

- Virtualization is not supported. An admin must shut down the VM before performing hot plug.
- Removing the root port and rescanning from the `/sys/bus/pci` is mandatory for the first time after the machine boots up. This is a limitation from the Linux kernel.

Surprise Add Support on the SmartSSD CSD

This section describes surprise add support for the SmartSSD CSD module. To support surprise add, the BIOS/OS Linux kernel must support the reservation of extra buses (4) and memory mapped I/O (MMIO), memory used by SmartSSD CSD. This can be achieved by using the following parameters to configure the operating system kernel.

```
pci=hpbussize // Reserve the extra busses under hotplug hotplugs supported
bridge
pci=hpmemsize // Reserve extra memory under hotplug hotplug supported bridge
```

The required settings for the SmartSSD CSD module are as follows.

```
pci=assign-busses,
hpbussize=4,
realloc=on,
hpmemsize=8G
```

For example, on an Ubuntu machine, the following steps are required:

1. Open `/etc/default/grub`.
2. Append the following in the `GRUB_CMDLINE_LINUX` option:

   ```
   pci=assign-busses,
   hpbussize=4,
   realloc=on,
   hpmemsize=8G
   ```

3. Save and exit from `/etc/default/grub`.
4. Build the grub using `update-grub`.
5. Reboot the machine.

Known Issues and Resolution

If a SmartSSD CSD module is booted up with the previous options in the grub, the NVMe controller BAR of an already plugged in SmartSSD CSD module comes up as unallocated/ unassigned. To bring the device in to the correct state every time the machine boots up with the SmartSSD CSD, the PCIe root-port (to which SmartSSD CSD module is connected) needs to be removed and re-scanned. For example:

```
|           \-17.0
+-[0000:3a]-+-00.0-[3b-4c]----00.0-[3c-4c]--+-04.0-[3d-40]--
|           |                               +-05.0-[41-44]--
|           |                               +-06.0-[45-48]--
|           |                               \-07.0-[49-4c]----00.0-[4a-4c]--+-00.0-[4b]----00.0
|           |                                                               \-01.0-[4c]--+-00.0
|           |                                                                            \-00.1
|           +-05.0
```

In the above topology (via `lspci -t`), Bus 49 to 4c belongs to the SmartSSD CSD module. The PCIe root-port to which the SmartSSD CSD is connected has the `BDF=3a:00.0`. To bring the device in to the correct state after a reboot (cold/warm), the following operation must be performed.

```
echo 1 > /sys/bus/pci/devices/0000:3a:00.0/remove
echo 1 > /sys/bus/pci/rescan
```

To run this operation automatically, these commands can be added in the `/etc/rc.local` file so that they will be executed every time the machine boots up.

CAUTION!

- Before automating the operation, make sure the boot-drive is not connected to the same PCIe-root port bridge to which the PCIe module is connected or it could leave the machine in a bad state making it difficult to recover.
- The reservation of the PCI bus and memory resource depends on the usage of the PCI bus and memory resources in the system. The success of the previously listed steps is subject to the system configuration and kernel version used. These commands must be used with caution.
- PCIe remove/rescan from the root port bridge will not only remove/rescan the SmartSSD CSD module but also any other device connected to the bridge.

***Note*:** Surprise remove functionality is not supported with the SmartSSD CSD module.

Miscellaneous Information and Settings

Platform (Shell) Version Identification

The SmartSSD CSD platform (shell) has a version identification register at `0x330000` offset. This register can be read from the Xilinx PCIe mgmt. PF (Mgmt. BAR).

For example:

`[11:08]` - Major Version
`[07:00]` - Minor Version
`0x00000090` indicates -- Version 0.90
`0x00000100` indicates -- Version 1.00

Use the `./rwmem` utility provided in the zip file, available for download along with this user guide, to find the platform (shell) version.

Bit field	Access	Default value
[31:12]	Read-only	N/A
[11:0]	Read-only	Version number

Changing (Lowering) FPGA Temperature and Power Thresholds for Clock Throttling

Use the following command for Xilinx mgmt. PF identification.

```
sensors | grep -i xilinx
xilinx_u2_gen3x4_xdma_base_1_user-pci-dd01
xilinx_u2_gen3x4_xdma_base_1_mgmt-pci-dd00
```

Use the following command to know the default FPGA temperature and power thresholds used for clock throttling (FPGA temperature = 100.0°C; Power threshold = 29W). Note that default values are hard upper limits and cannot be increased.

```
sensors xilinx_u2_gen3x4_xdma_base_1_mgmt-pci-dd00 | grep -i CS_TARGET
CS_TARGET_TEMP:   +100.0°C  (high =  +0.0°C)
CS_TARGET_POWER:   29.00 W  (max =   0.00 W)
```

FPGA temperature and power thresholds can be lowered if needed. In this example, FPGA temperature threshold is lowered to 97.0°C and power threshold is lowered to 25W.

```
chip "xilinx_u2_gen3x4_xdma_base_1_mgmt-pci-dd00"
label temp15 CS_TARGET_TEMP
set temp15_input 97/1000

chip "xilinx_u2_gen3x4_xdma_base_1_mgmt-pci-dd00"
label power2 CS_TARGET_POWER
set power2_input 25
```

Save the settings using, `sensors -s`.

Confirm the change by running the following command again.

```
sensors xilinx_u2_gen3x4_xdma_base_1_mgmt-pci-dd00 | grep -i CS_TARGET
CS_TARGET_TEMP:     +97.0°C  (high =  +0.0°C)
CS_TARGET_POWER:    25.00 W  (max =   0.00 W)
```

FPGA-SSD DNA Pairing

The SmartSSD CSD module is implemented with the FPGA and NVMe SSD physically co-located on the same module/PCB. In a multiple card setup, it becomes essential for the application running on the host machines to be aware about this FPGA-SSD co-location for offloading of acceleration jobs to the right FPGA. This is important for using the efficacy of Peer-to-Peer (P2P) transfers. In the absence of co-location awareness, the application can end up choosing a random FPGA for acceleration and see inefficient data movements (higher latencies and higher power consumption).

Each Xilinx manufactured FPGA contains a unique physical identifier. This is known as the FPGA DNA which is a single unique (96-bit) non-volatile device identifier. For additional information about FPGA DNA, refer to the *UltraScale Architecture Configuration User Guide* (UG570).

Similarly, Samsung NVMe SSD provides a unique non-volatile physical identifier for each manufactured SSD. This is called SSD Serial Number (SN).

An FPGA-SSD pair can be identified using the FPGA DNA and SSD Serial Number as follows:

1. NVMe SSD's Identify controller command reports both the SSD Serial Number and 96-bit FPGA DNA value (represented in bold in the following log).

```
sudo nvme id-ctrl /dev/nvme0n1 -v

NVME Identify Controller:
vid     : 0x144d
ssvid   : 0x144d
sn      : 160401P00RC01

mn      : MZWLJ3T2HBLS-000D7-101
fr      : EPK9SB5E
rab     : 8
ieee    : 002538
cmic    : 0x3
mdts    : 5
cntlid  : 41
ver     : 10300
rtd3r   : e4e1c0
rtd3e   : 989680
oaes    : 0x300
oacs    : 0xdf
acl     : 127
aerl    : 15
frmw    : 0x17
lpa     : 0xe
elpe    : 255
npss    : 0
avscc   : 0x1
```

```
apsta    : 0
wctemp   : 353
cctemp   : 360
mtfa     : 130
hmpre    : 0
hmmin    : 0
tnvmcap  : 3840755982336
unvmcap  : 0
rpmbs    : 0
sqes     : 0x66
cqes     : 0x44
nn       : 32
oncs     : 0x7f
fuses    : 0
fna      : 0x4
vwc      : 0
awun     : 65535
awupf    : 0
nvscc    : 1
acwu     : 0
sgls     : f0002
ps    0 : mp:25.00W operational enlat:100 exlat:100 rrt:0 rrl:0
          rwt:0 rwl:0 idle_power:- active_power:-
vs[]:
       0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
0000: 41 ed 01 00 00 00 00 00 d0 95 7d 8a 00 00 00 00 "A.........}....."
0010: 00 00 00 00 00 00 00 00 01 07 20 20 00 12 11 00 "................"
0020: 00 00 00 00 00 00 00 00 00 00 00 00 08 00 00 00 "................"
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
0040: 17 00 00 00 00 01 01 02 00 00 00 00 05 c2 b0 34 "...............4"
0050: 29 1b 3a 01 00 00 02 40 99 00 00 02 00 00 00 00 ").:....@........"
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
0070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
```

2. Confirm the 96-bit FPGA DNA value from SmartSSD CSD User PF BAR through the register read.

```
PCIe User PF BAR + 0x330020 = 0x34b0c205
PCIe User PF BAR + 0x330024 = 0x013a1b29
PCIe User PF BAR + 0x330028 = 0x40020000
```

Both these steps can be used to identify the NVMe SSD and FPGA that are located on the same SmartSSD CSD.

Accessing FPGA DNA Primitive from Dynamic Region

The SmartSSD CSD platform instantiates the DNA primitive in the static region and uses it to obtain the 96-bit FPGA DNA value at power up. After latching the FPGA DNA value, SmartSSD CSD platform design releases the DNA primitive and makes it accessible for any Vitis™ kernel/ user logic from dynamic region. The following figure highlights the connections that need to be made during v++ kernel generation flow using IP integrator connect commands.

Figure 5: **FPGA DNA Access for User Kernels**

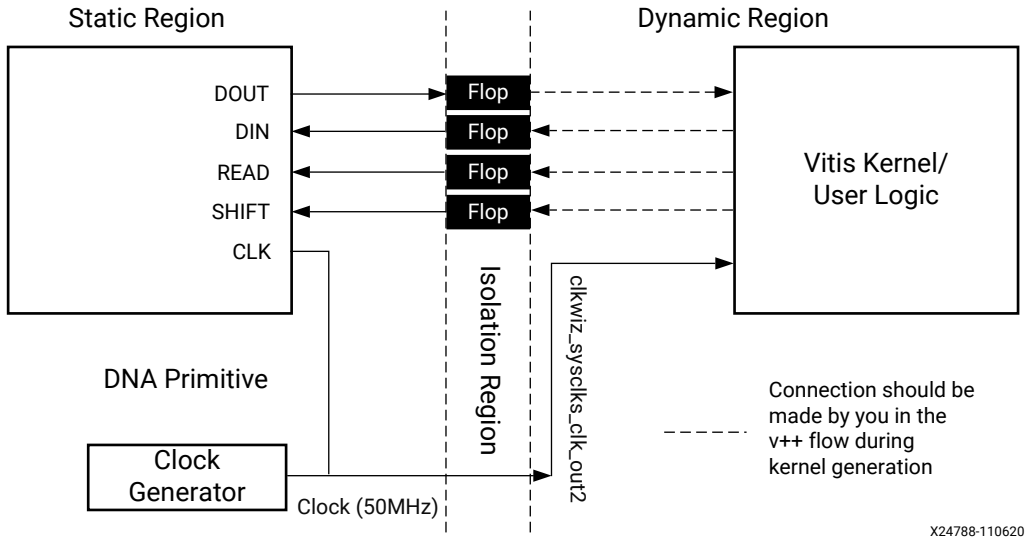


Table 9: **Interface Port Names**

Interface Port Name	Description
clkwiz_sysclks_clk_out2	DNA primitive clock port. This is 50 MHz clock.
ulp_m_data_dout_dna_00	Connects to the DOUT port of DNA primitive.
ulp_s_data_dna_from_ulp_00[0]	Connects to the DIN port of DNA primitive.
ulp_s_data_dna_from_ulp_00[1]	Connects to the READ port of DNA primitive
ulp_s_data_dna_from_ulp_00[2]	Connects to the SHIFT port of DNA primitive

User logic can connect to the DNA ports in the dynamic region using following IPI commands. As shown in the previous figure, user logic must use respective clock (clkwiz_sysclks_clk_out2 – 50 MHz) to drive DNA primitive ports and latch DOUT.

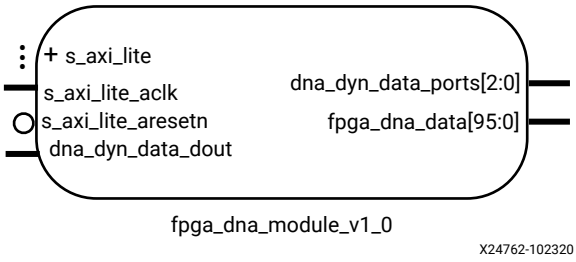
```
connect_bd_net [get_bd_pins ii_level0_wire/ulp_m_data_dout_dna_00] [get_bd_pins user_logic/
DOUT]
connect_bd_net [get_bd_pins user_logic/DIN] [get_bd_pins ii_level0_wire/
ulp_s_data_dna_from_ulp_00[0]]
connect_bd_net [get_bd_pins user_logic/READ] [get_bd_pins ii_level0_wire/
ulp_s_data_dna_from_ulp_00[1]]
connect_bd_net [get_bd_pins user_logic/SHIFT] [get_bd_pins ii_level0_wire/
ulp_s_data_dna_from_ulp_00[2]]
```

IMPORTANT! *There is one cycle of latency in inputs (DIN, READ, SHIFT) and one cycle of latency in output (DOUT). User logic must consider the isolation interface flop latency when latching the DOUT value.*

Example Module for Testing DNA Connectivity

SmartSSD CSD platform design has an example module (fpga_dna_module_0) present in the dynamic region which has been used to test the dynamic region DNA port connectivity. This example module runs at clkwiz_sysclks_clk_out2 (50 MHz clock) and is accessible through AXI4-Lite Mgmt. PF + 0x0110_0000 offset.

Figure 6: **Example FPGA DNA Module**



This module is connected to the DNA ports using following commands:

```
connect_bd_net [get_bd_pins ii_level0_wire/ulp_m_data_dout_dna_00]
[get_bd_pins fpga_dna_module_0/dna_dyn_data_dout]
connect_bd_net [get_bd_pins fpga_dna_module_0/dna_dyn_data_ports]
[get_bd_pins ii_level0_wire/ulp_s_data_dna_from_ulp_00]
```

After loading the implementation bitfile, this test module can be triggered by writing following register:

```
./rwmem AXI Lite Mgmt. PF + 0x01100010  0x1
```

Captured FPGA DNA are reported in the following registers:

```
./rwmem AXI Lite Mgmt. PF + 0x01100000
0x2460e205
./rwmem AXI Lite Mgmt. PF + 0x01100004
0x013a1b40
./rwmem AXI Lite Mgmt. PF + 0x01100008
0x40020000
```

Example module also drives the DNA value captured in the above registers on fpga_dna_data [95:0] output port.

Note that the example fpga_dna_module considers isolation interface flop latency and latches DOUT coming from DNA primitive after two cycles of READ assertion. Also, this module drives the latched DOUT value back to the DIN port towards DNA PRIMITIVE.

For more details about DNA PRIMITIVE, refer to the DNA_PORTE2 section in the *UltraScale Architecture Configuration User Guide* (UG570).

XILINX

Appendix H

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado® IDE, select **Help → Documentation and Tutorials**.
- On Windows, select **Start → All Programs → Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this guide:

Vitis Documents

1. *Vitis Unified Software Platform Documentation: Application Acceleration Development* (UG1393)
2. *Vitis Unified Software Platform Documentation: Embedded Software Development* (UG1400)
3. Vitis 2020.1 Software Platform Release Notes in the *Vitis Unified Software Platform Documentation* (UG1416)
4. *Vitis Application Acceleration Development Flow Tutorials* (GitHub)

Additional Xilinx Resources

1. Xilinx licensing website: https://www.xilinx.com/getproduct
2. Vitis Developer Zone: https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html
3. Xilinx Community Forums: https://forums.xilinx.com
4. *Xilinx Third-Party End User License Agreement*
5. *End-User License Agreement*

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright