Quadra™ Installation Guide V5.2

Table of Contents

1 Legal Notice

Information in this document is provided in connection with NETINT products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in NETINT's terms and conditions of sale for such products, NETINT assumes no liability whatsoever and NETINT disclaims any express or implied warranty, relating to sale and/or use of NETINT products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right.

A "Mission Critical Application" is any application in which failure of the NETINT Product could result, directly or indirectly, in personal injury or death. Should you purchase or use NETINT's products for any such mission critical application, you shall indemnify and hold NETINT and its subsidiaries, subcontractors and affiliates, and the directors, officers, and employees of each, harmless against all claims costs, damages, and expenses and reasonable attorney's fees arising out of, directly or indirectly, any claim of product liability, personal injury, or death arising in any way out of such mission critical application, whether or not NETINT or its subcontractor was negligent in the design, manufacture, or warning of the NETINT product or any of its parts.

NETINT may make changes to specifications, technical documentation, and product descriptions at any time, without notice. The information here is subject to change without notice. Do not finalize a design with this information. The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications.

NETINT™, Codensity™, Quadra™ and the NETINT Logo are trademarks of NETINT Technologies Inc. All other trademarks or registered trademarks are the property of their respective owners.

2 Scope

NETINT is a supplier of high-performance, low-latency, real-time video processing units (VPUs) for x86 and Arm servers.

This installation guide provides information on setup and configuration of the host in which the NETINT video transcoder device will be installed and operated.

Various environments are supported by Quadra including

- Linux
- Windows
- Android
- MacOS
- Virtual Machines
- Docker containers

Quadra also supports various tools and toolchains like

- Kubernetes
- Windows with Visual Studio
- Windows with MSYS

Aside from this installation document, the **Quadra Quick Start Guide** (**QSG**) is a starting reference for any user trying to quickly setup and configure a NETINT Quadra video transcoder device. This installation document covers all the supported environment installations, whereas the installation steps in the **QSG** document will only explain how to install Quadra into Ubuntu 20.04, or a newer Linux host. The **QSG** is also included in Quadra's release package.

3 Release Package Files Guide

The NETINT Firmware and Software release packages contain many files. This installation guide will primarily use the **Quadra_SW_V*.*.*** folder, which is highlighted below in the table of release package contents:

File	Description
Android_quick_installer/	FW/SW guided installation script for Android
clamscan.log	Clam Anti Virus scan log
InstallationGuideQuadra_V*.pdf	NETINT SW installation guide for various systems
IntegrationProgrammingGuideQuadra_V*.pdf	NETINT FW/SW primary user guide that includes the full list of xcoder-params
libxcoder_API_Integration_guideQuadra_V*.pdf	The Libxcoder API Integration Guide
md5sum	MD5 checksum of files
Performance_Test_Report_V*.pdf	Quadra performance test report
Quadra_FW_V*.*.*/	FW release package folder
Quadra_FW_V*.*.*_*_release_notes.txt	FW release notes
Quadra_Quality_Report_V*.*.*.pdf	Visual quality test report
quadra_quick_installer.sh	FW/SW guided installation script for Linux
Quadra_SW_V*.*.*/	SW release package folder
Quadra_SW_V*.*.*_*_release_notes.txt	SW release notes
QuickStartGuideQuadra_V*.pdf	FW/SW Quick start installation and introduction guide
README.md	Information about release package contents
sentinelscan.log	SentinelOne Anti Virus scan log
Test_Coverage_Report_V*.*.*.pdf	Quadra test coverage report

4 Linux Host

The NETINT video transcoder device can operate in various Linux host environments including Ubuntu and CentOS. Various CPU architectures are also supported, including x86 (Intel, AMD) and ARM.

This section covers the NETINT software package installation on a typical Linux host (non-VM, non-container).

4.1 Scripted FW/SW Installation

The **quadra_quick_installer.sh** script is provided to handle automated installation of both FW and SW in its default configuration. The installer script supports Ubuntu, CentOS, and MacOS. It can be found at the top level of the release folder after unzipping the release package.

1. Copy the release package to the host. The filename syntax should be as follows, for example:

   ```
   Quadra_V5.1.0.zip
   ```

2. From a command line, unzip the release package

   ```
   $ unzip Quadra_V5.1.0.zip
   ```

3. Enter the release folder and start the upgrade script

   ```
   $ bash quadra_quick_installer.sh
   ```

4. The upgrade script looks for Quadra FW/SW release packages at the path beside it. Confirm the upgrade script selected the correct release versions

   ```
   Press [Y/y] to confirm the use of these two release packages.
   ```

5. Review the release information, press '**y**' to accept, these menu options appear

```
Choose an option:
 1) Setup Environment variables
 2) Unlock CPU governor
 3) Install OS prerequisite packages
 4) Install NVMe CLI
 5) Install Libxcoder
 6) Install Libxcoder_FFmpeg3.1.1only (FFmpeg-n3.1.1)
 7) Install FFmpeg-n3.1.1 (must install
libxcoder_FFmpeg3.1.1only first)
 8) Install FFmpeg-n3.4.2
 9) Install FFmpeg-n4.1.3
10) Install FFmpeg-n4.2.1
11) Install FFmpeg-n4.3.1
12) Install FFmpeg-n4.3
13) Install FFmpeg-n4.4.2
14) Install FFmpeg-n4.4
15) Install FFmpeg-n5.0
16) Install FFmpeg-n5.1.2
17) Install FFmpeg-n6.1
18) Install FFmpeg-n7.0
19) Install gstreamer-1.22.2
20) Install gstreamer-1.24.4
21) Firmware Update
23) Quit
```

5. Type ‘**1**’ and press enter to setup any environment variables
6. Type ‘**3**’ and press enter to install the prerequisite software libraries for your OS (Ubuntu/CentOS/MacOS)
7. Type ‘**4**’ and press enter to install the **nvme-cli** tool
8. Depending on the version of FFmpeg required, use an option from ‘**5**’ to ‘**6**’ to install the Libxcoder
9. Depending on the required version of FFmpeg, use an option from ‘**7**’ to ‘**18**’ to install FFmpeg. You will be prompted to also compile the libav shared libraries, and add Gstreamer support or interface with the libavcodec.
10. If you also wish to install Gstreamer NETINT support through the **gst-libav**, type **‘19’** or **‘20’** and press enter. The pre-requisites are that FFmpeg version >=n4.3.1 is installed with shared libraries,
11. Type ‘**21**’ and press enter to go through the guided Firmware update process.
12. Type ‘**22**’ and press enter to exit the script menu

If any individual step fails during the execution of **quadra_quick_installer.sh** it is recommended to check any printed errors, then take the action necessary to resolve it, and then try again. If any error persists, please contact your NETINT support representative.

NOTE: The User **must NOT interrupt** the firmware update process once it has started. Any interruption may cause the device to malfunction. If a cold upgrade fails then a system reboot may resolve the issue. The firmware update process should then be repeated.

NOTE: Additional information captured during the update process can be found in the **./upgrade_log.txt** file generated by the update script.

NOTE: The behavior of the warm upgrade process is identical to that of the cold upgrade, **except a system reboot is not required for warm upgrade**. If the warm upgrade fails then a system reboot will resolve the issue, and the warm upgrade process can be repeated. However, if warm upgrade repeatedly fails then a cold upgrade and reboot should be used instead.

4.2 Manual SW Installation

4.2.1 Unpacking NETINT SW release

The NETINT SW release package contains the host side driver (libxcoder*), as well as the NETINT FFmpeg patch that modifies the base FFmpeg to add support for the NETINT hardware codecs, filters and other functionalities.

1. Unzip the NETINT release. Replace <version> below with the NETINT release version number (eg. 5.0.0):

   unzip Quadra_V<version>.zip

2. Acquire base FFmpeg code. Replace <ff_version> with FFmpeg release version number (eg. n4.3.1):

   git clone -b <ff_version> --depth=1 https://github.com/FFmpeg/FFmpeg.git FFmpeg

3. Apply NETINT Patch to base FFmpeg code. Replace <version> with NETINT release version number (eg. n4.3.1). Replace <ff_version> with NETINT release version number (eg. 4.0.0_RCA):

cp Quadra_V<version>/Quadra_SW_V*/FFmpeg-<ff_version>_netint_v<version>.diff FFmpeg/ &&

cd FFmpeg/ && patch -t -p 1 < FFmpeg-n<ff_version>_netint_v<version>.diff &&

cd ..

4.2.2 Install Prerequisite 3rd party SW

1. Install the **nvme-cli** utility depending on your distro of Linux:

    a. Ubuntu:

    ```
    sudo apt install nvme-cli pkg-config git gcc
    ```

    b. Centos:

    ```
    sudo yum --enablerepo=extras install -y epel-releas
    sudo yum install nvme-cli pkgconfig git redhat-lsb-
    core make gcc
    ```

    c. Fedora:

    ```
    sudo dnf install nvme-cli pkgconfig git make gcc
    ```

    d. MacOS:

    ```
    xcode-select –install
    curl -O https://pkg-
    config.freedesktop.org/releases/pkg-config-0.28.tar.gz
    && tar -zxvf pkg-config-0.28.tar.gz && rm pkg-config-
    0.28.tar.gz && cd pkg-config-0.28 && export
    CC=/usr/bin/cc 2> /dev/null || setenv CC /usr/bin/cc
    2> /dev/null && ./configure --prefix=/usr/local CC=$CC
    --with-internal-glib && make && sudo make install &&
    cd .. && rm -rf pkg-config-0.28
    ```

2. Install YASM assembly optimizations library:

    ```
    curl -O http://www.tortall.net/projects/yasm/releases/yasm-
    1.3.0.tar.gz &&
    tar -zxf yasm-1.3.0.tar.gz && rm yasm-1.3.0.tar.gz &&
    cd yasm-1.3.0/ && ./configure && make && sudo make install
    &&
    cd .. && rm -rf yasm-1.3.0
    ```

4.2.3 Setup Environment Variables

FFmpeg compilation relies on pkg-config for linking libxcoder to FFmpeg. Certain Linux distros (eg. Centos) may not configure pkg-config's environment variables in a way that works for FFmpeg+libxcoder compilation. Use the below commands to apply Ubuntu's default pkg-config environment variables to your distro if necessary.

```
export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig/ &&
export LD_LIBRARY_PATH=/usr/local/lib/ &&
grep -qxF '/usr/local/lib' /etc/ld.so.conf ||
sudo sh -c 'echo "/usr/local/lib" >> /etc/ld.so.conf'
```

4.2.4 Quadra and Logan Co-existence

There are two product families of NETINT video transcoders, these are Quadra and Logan (also known as T400 series).

Logan and Quadra can now co-exist in the same host. This means all software APIs and symbols are named uniquely, with no clashes. Applications can now be built with either system on the same host. It is possible to link an application against either Logans or Quadra's libraries.

Starting from Quadra release v4.0.0, and Logan release v3.0.0, the NETINT host software packages can be built with options to work specifically with one or both products. This means any Quadra release starting from Quadra v4.0.0 can co-exist with any Logan release from Logan v3.0.0 onwards. FFMpeg versions that are supported by both platforms are 4.2.1 and 6.1.

With co-existence of products, it is important to understand feature compatibility between both Quadra and Logan releases.

4.2.5 Feature Compatibility between Quadra and Logan

As new features are added to Logan and released, they are not automatically included in any ***existing*** Quadra releases. Only when a new Quadra release is published, which includes all the latest Logan files for the new Logan features, will Quadra's release package support the new Logan features. For example, Quadra release v4.0.0 supports all Logan features in Logan release v3.0.0. If the new Logan feature A is then added to the next Logan release v3.1.0, this new Logan feature A will not be available in Quadra v4.0.0. Only when Quadra v4.1.0 is released, and only if it includes Logan v3.1.0 release files will the new Logan feature A be supported in Quadra v4.1.0.

4.2.6 Building Logan and Quadra from the Quadra Release

The following sections explain the required steps for building application software with Logan and Quadra on the same host. The assumption is that all host software packages have been downloaded and NETINT patches have been applied, all from Quadra's release package only.

NOTE : All code required for Logan and Quadra co-existence must be obtained from the **Quadra** release package.

4.2.7 Build and install libxcoder for Quadra

If a Quadra card is installed on the host and the host software is to be built for it, use the commands below to compile the Quadra driver and low-level utilities. Replace <version> with NETINT release version number (e.g. 4.0.0_RCA):

```
cp -r Quadra_SW_V<version>/libxcoder libxcoder &&
cd libxcoder &&
bash build.sh &&
cd ..
```

Note: if installing libxcoder on a system that already has another version installed, make sure to reinitialize device resource list by running the following commands:

ni_rsrc_update –D

init_rsrc

Alternatively, reboot the system after installing all components.

4.2.8 Build and install libxcoder_logan for Logan

If a Logan card is installed on the host and the host software is to be built for it, follow instructions in QuickStartGuideT408_T432_*.pdf to unpack the libxcoder_logan folder in the Logan release. Note, the libxcoder_logan code must be from a T4XX release version >=3.0.0.

Then, use the below commands to compile the Logan driver and low-level utilities:

```
cp -r release/libxcoder_logan libxcoder_logan &&
cd libxcoder_logan &&
bash build.sh &&
cd ..
```

4.2.9 Build FFmpeg

FFmpeg with the NETINT patch can be built with support for Quadra, Logan, or both, if arguments in build_ffmpeg.sh allow it.

Please consult NETINT engineering support for details on which versions of FFmpeg support which NETINT products, and which NETINT FFmpeg patch should be used for combined Quadra and Logan support.

To build FFmpeg, use the steps below:

1. Go to the FFmpeg directory (directory name depends on the FFmpeg version) with the following command:

   ```
   cd FFmpeg
   ```

2. Ensure folder is clean for compilation with the following command:

   ```
   sudo make clean
   ```

3. The options to build FFmpeg for different NETINT product lines (Quadra, Logan, etc.) are dependent on the options supported by the build_ffmpeg.sh script in the NETINT FFmpeg patch applied to the FFmpeg folder.

   a. Check the build_ffmpeg.sh help text to see which options are available with the following command:

      ```
      bash build_ffmpeg.sh -h | grep "build for "
      ```

   b. Build FFmpeg for Quadra with the following command:

      ```
      bash build_ffmpeg.sh --quadra
      ```

   c. Build FFmpeg for Logan with the following command:

      ```
      bash build_ffmpeg.sh --logan
      ```

   d. Build FFmpeg for Quadra+Logan with the following command:

      ```
      bash build_ffmpeg.sh --quadra –logan
      ```

4. Install FFmpeg to host $PATH with the following command:

   ```
   sudo make install
   ```

4.2.10 Build FFmpeg with LGPL License

By default, when FFmpeg is built using the build_ffmpeg.sh script, it will be configured with the following flags:

--enable-gpl --enable-nonfree

To build FFmpeg so that it qualifies for the LGPL license these flags need to be disabled. To disable the flags FFmpeg needs to be configured manually i.e. not using the **build_ffmpeg.sh** script. To view the default configuration set by the **build_ffmpeg.sh** script, use the steps below:

1. Go to the FFmpeg directory (directory name depends on the FFmpeg version) with the following command:

   ```
   cd FFmpeg
   ```

2. Clean the folder for compilation with the following command:

   ```
   sudo make clean
   ```

3. Get the configuration:

   ```
   bash build_ffmpeg.sh -dry
   ```

   The response for the above command should look like this :

   ```
   ./configure --pkg-config-flags=--static --enable-gpl --
   enable-nonfree --extra-ldflags=-lm --extra-ldflags=-ldl --
   enable-ni_quadra --disable-ni_logan --disable-
   filter=drawtext_ni_quadra --enable-pic --enable-pthreads --
   extra-libs=-lpthread --enable-encoders --enable-decoders --
   enable-avfilter --enable-muxers --enable-demuxers --enable-
   parsers --disable-debug --disable-ffplay --disable-ffprobe
   --disable-libx264 --disable-libx265 --disable-libaom --
   disable-libvpx --disable-libxml2 --disable-libsrt --
   disable-cuda-nvcc --disable-cuda --disable-cuvid --disable-
   nvdec --disable-nvenc --disable-libvmaf --enable-static --
   disable-shared --extra-cflags=-UNI_DEC_GSTREAMER_SUPPORT --
   extra-cflags=-UNI_MEASURE_LATENCY
   ```

To build FFmpeg so it qualifies for the LGPL license change the following flags

--enable-gpl --enable-nonfree

to be

--disable-gpl --disable-nonfree

Now the configure command need to be run:

```
./configure --pkg-config-flags=--static --disable-gpl --
disable-nonfree --extra-ldflags=-lm --extra-ldflags=-ldl --
enable-ni_quadra --disable-ni_logan --disable-
filter=drawtext_ni_quadra --enable-pic --enable-pthreads --
extra-libs=-lpthread --enable-encoders --enable-decoders --
enable-avfilter --enable-muxers --enable-demuxers --enable-
parsers --disable-debug --disable-ffplay --disable-ffprobe
--disable-libx264 --disable-libx265 --disable-libaom --
disable-libvpx --disable-libxml2 --disable-libsrt --
disable-cuda-nvcc --disable-cuda --disable-cuvid --disable-
nvdec --disable-nvenc --disable-libvmaf --enable-static --
disable-shared --extra-cflags=-UNI_DEC_GSTREAMER_SUPPORT --
extra-cflags=-UNI_MEASURE_LATENCY
```

Once FFmpeg is configured it needs to be built and installed on the system using the following steps:

1. Build FFmpeg with the command:

   make

2. Install FFmpeg to the host **$PATH** with the following command:

   sudo make install

Note: The **build_ffmpeg.sh** script with the **--dry** flag can be used with other custom configurations as shown in Section 4.2.9 (Step 3). The **--dry** flag can also be added at the end of the **build_ffmpeg.sh** script to get the configuration, for example

```
bash build_ffmpeg.sh --quadra –logan –dry
```

The above command line will print out the FFmpeg configuration with the Quadra and Logan libraries enabled. The above step can then be followed with an install of FFmpeg with the LGPL license, with Quadra and Logan features enabled.

5 Docker Containers

This section details the configuration and usage of the NETINT video transcoder device in a Docker container. This can also be used for other NETINT video transcoder solutions.

5.1 Pre-requisites for the Quadra Environment on Host

The Linux Host has the following working environment.

1. Quadra libxcoder and FFmpeg source installed and compiled successfully. FFmpeg 4.2.1 version is referenced in this document. Follow the Quick Start Guide to install libxcoder and FFmpeg.

2. Running 'ffmpeg' gives a dump similar to sample below.

```
$ ffmpeg
ffmpeg version 4.3.1 Copyright (c) 2000-2020 the FFmpeg
developers
  built with gcc 9 (Ubuntu 9.3.0-17ubuntu1~20.04)
  configuration: --pkg-config-flags=--static --enable-gpl --
-enable-nonfree --extra-ldflags=-lm --extra-ldflags=-ldl --
enable-libxcoder --enable-ni --enable-pthreads --extra-
libs=-lpthread --enable-encoders --enable-decoders --
enable-avfilter --enable-muxers --enable-demuxers --enable-
parsers --enable-x86asm --disable-debug --disable-ffplay --
enable-ffprobe --enable-libx264 --enable-libx265 --enable-
libaom --disable-libvpx --disable-cuda-nvcc --disable-cuda
--disable-cuvid --disable-nvdec --disable-nvenc --disable-
libvmaf --enable-static --disable-shared --extra-cflags=-
UNIENC_MULTI_THREAD
  libavutil      56. 51.100 / 56. 51.100
  libavcodec     58. 91.100 / 58. 91.100
  libavformat    58. 45.100 / 58. 45.100
  libavdevice    58. 10.100 / 58. 10.100
  libavfilter     7. 85.100 /  7. 85.100
  libswscale      5.  7.100 /  5.  7.100
  libswresample   3.  7.100 /  3.  7.100
  libpostproc    55.  7.100 / 55.  7.100
Hyper fast Audio and Video encoder
usage: ffmpeg [options] [[infile options] -i infile]...
{[outfile options] outfile}...
```

```
Use -h to get full help or, even better, run 'man ffmpeg'
```

3. Executing '**sudo nvme list**' will display all the installed Quadra NVMe devices.

```
$ sudo nvme list
Node             SN                   Model
Namespace Usage                      Format           FW
Rev
---------------- -------------------- ----------------------
------------------ --------- ------------------------- --
-------------- --------
/dev/nvme0n1     Q1A10BA11FC060-0124  QuadraT1A
1           8.59  TB /   8.59  TB      4 KiB +  0 B   ---
00DEV
/dev/nvme1n1     Q1A10BA11FC060-0142  QuadraT1A-EP1
1           8.59  TB /   8.59  TB      4 KiB +  0 B   ---
00DEV
```

4. Executing '**ni_rsrc_mon**' or '**sudo ni_rsrc_mon**' will produce an output similar to the sample below.

```
$ ni_rsrc_mon
NI resource init'd already ..
************************************************
2 devices retrieved from current pool at start up
Mon Jan 31 13:35:40 2022 up 00:00:00 v---00DEV
Num decoders: 2
BEST INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM DEVICE
NAMESPACE
L    0     0    0          0    0    0         /dev/nvme0
/dev/nvme0n1
     1     0    0          0    0    0         /dev/nvme1
/dev/nvme1n1
Num encoders: 2
BEST INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM DEVICE
NAMESPACE
L    0     0    0          0    0    0         /dev/nvme0
/dev/nvme0n1
     1     0    0          0    0    0         /dev/nvme1
/dev/nvme1n1
Num scalers: 2
BEST INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM DEVICE
NAMESPACE
L    0     0    0          0    0    0         /dev/nvme0
/dev/nvme0n1
     1     0    0          0    0    0         /dev/nvme1
/dev/nvme1n1
Num ais: 2
BEST INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM DEVICE
NAMESPACE
L    0     0    0          0    0    0         /dev/nvme0
/dev/nvme0n1
     1     0    0          0    0    0         /dev/nvme1
/dev/nvme1n1
************************************************
```

5.2 Create a Docker Image

This section describes how to build a Docker image from a Dockerfile and the NETINT Quadra SW release.

1. Install the Docker module on your Linux Host. Refer to the links in the previous section 5.1 of this document, **Pre-requisites for the Quadra Environment on Host,** this describes the installation steps based on the OS type of your Linux Host.

2. Install Python Docker package:
   sudo pip install docker

3. Download the Dockerfile from NETINTs public repository on Github:
   https://github.com/NETINT-Technologies/quadra_dockerfile/blob/main/Dockerfile

4. Copy the NETINT release package, for example **Quadra_V5.0.0.zip,** to the same folder as the Dockerfile.

5. Generate a Docker image:
   sudo docker build --tag ni_quadra_sw .

   Note, there are two options that can be set with the `--build-arg` command:

   a. `NI_RELEASE_VERSION`
      Version number of NETINT Quadra SW release package (eg. `--build-arg NI_RELEASE_VERSION=5.0.0`)

   b. `FFMPEG_VERSION`
      Version number of FFmpeg to use (eg. `--build-arg FFMPEG_VERSION=n5.0`)

6. Confirm the Docker image is created:
   $ sudo docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ni_quadra_sw	**latest**	**fc6cf4086561**	**5 weeks ago**	**1.32GB**

5.2.1 Useful Links

1. https://linuxhint.com/install_configure_docker_ubuntu/
2. https://docs.docker.com/engine/install/centos/
3. https://download.docker.com/linux/centos/8/x86_64/stable/Packages/

5.3 Docker container – Quadra interworking

5.3.1 Scope

This section demonstrates the Quadra NVMe device transcoding session in a Docker container.

5.3.2 Launch Docker container with Quadra device

Run the following command to start a container using Docker image **ni_quadra_sw**

```
sudo docker run -it --privileged --device /dev/nvme0n1 --device /dev/nvme0
ni_quadra_sw /bin/sh
```

Note – The '**-v**' option can be updated to point to the host folder where FFmpeg and libxcoder are installed. For example: ~/FFmpeg, ~/FFmpegXcoder etc.

You should get a shell prompt to demonstrate a successful container has started.

```
Sh-4.2#
```

Running '**sudo nvme list**' should list all the Quadra devices.

```
sh-4.4# sudo nvme list
Node             SN                   Model                                    Namespace Usage
Format           FW Rev
---------------------- -------------------- ---------------------------------------- --------- -------------------------
-- ---------------- --------
/dev/nvme0n1         Q1A10BA11FC060-0124  QuadraT1A                                1         8.59
TB /   8.59  TB     4 KiB +  0 B   ---00DEV
/dev/nvme1n1         Q1A10BA11FC060-0142  QuadraT1A-EP1                              1
8.59  TB /   8.59  TB     4 KiB +  0 B   ---00DEV
sh-4.4#
```

5.3.3 Transcoding

Run a transcoding operation using FFmpeg. The sample below is for an AVC to HEVC transcode on a **1920x1080p_ParkScene.264** clip.

Note – There are some test clips in the **FFmpegXcoder/libxcoder/test** folder

```
sh-4.4# ffmpeg -y -nostdin -hide_banner -vsync 0 -c:v h264_ni_quadra_dec -dec 0 -i
1920x1080p_ParkScene.264 -vf scale=1920:1080 -xcoder-params
intraQP=27:intraPeriod=120 -c:v h265_ni_quadra_enc -enc 0 -xcoder-params
intraQP=27:intraPeriod=120 1920x1080p_ParkScene-0-0.265
Input #0, h264, from '1920x1080p_ParkScene.264':
  Duration: N/A, bitrate: N/A
    Stream #0:0: Video: h264 (High), yuv420p(progressive), 1920x1080, 24 fps, 24 tbr,
1200k tbn, 48 tbc
Stream mapping:
  Stream #0:0 -> #0:0 (h264 (h264_ni_quadra_dec) -> hevc (h265_ni_quadra_enc))
[h265_ni_quadra_enc @ 0x2705180] Session state: 0 allocate frame fifo.
[h265_ni_quadra_enc @ 0x2705180] pix_fmt is 0, sw_pix_fmt is -1
[h265_ni_quadra_enc @ 0x2705180] sw_pix_fmt assigned to pix_fmt was 0, is now -1
[h265_ni_quadra_enc @ 0x2705180] p_param->hwframes = 0
[h265_ni_quadra_enc @ 0x2705180] dts offset: 7, gop_offset_count: 0
Output #0, hevc, to '1920x1080p_ParkScene-0-0.265':
  Metadata:
    encoder         : Lavf58.29.100
    Stream #0:0: Video: hevc (h265_ni_quadra_enc), yuv420p, 1920x1080, q=2-31, 200
kb/s, 24 fps, 24 tbn, 24 tbc
    Metadata:
      encoder         : Lavc58.54.100 h265_ni_quadra_enc
frame=  240 fps=208 q=-0.0 Lsize=    2572kB time=00:00:09.62 bitrate=2188.7kbits/s
speed=8.34x
video:2572kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB muxing
overhead: 0.000000%
sh-4.4#
```

6 Linux KVM VM

This section details the installation of KVM on a Linux host, and provides details on installing and connecting a NETINT video transcoder device to a Linux Guest or Windows Guest Virtual Machine (VM).

There are two methods for using a Virtual Machine, firstly the user can either pass the **physical device** to the VM, or secondly, they can pass the **virtual device** to the VM. The use of a virtual device on a host requires **SR-IOV** supporting firmware on the NETINT device. Quadra firmware has supported **SR-IOV** since version 4.1.1, and so any subsequent version will also support **SR-IOV**.

6.1 Linux Guest VM with Passthrough Physical Device

6.1.1 Pre-requisite packages

This section was tested using Ubuntu 20.04, with the VM Guest OS also being Ubuntu 20.04.

First retrieve the Ubuntu 20.04 iso file:

wget https://mirror.it.ubc.ca/ubuntu-releases/20.04/

Then install the packages needed for KVM:

sudo apt -y install bridge-utils cpu-checker libvirt-clients libvirt-daemon qemu qemu-kvm

Use **kvm-ok** to ensure the environment actually supports kvm:

```
fpga@CLI309:~$ kvm-ok
INFO: /dev/kvm exists
KVM acceleration can be used
```

Authorize a user to be able to use **KVM** and also the **libvirt**:

sudo usermod -aG kvm $USER
sudo usermod -aG libvirt $USER
sudo systemctl status libvirtd – Check with

fpga@CLI309:~$ sudo systemctl status libvirtd

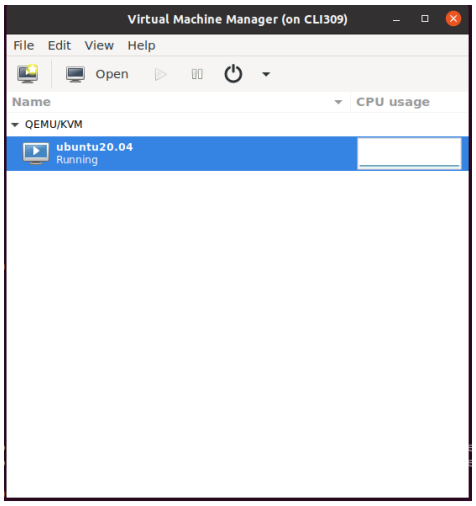
● libvirtd.service - Virtualization daemon
Loaded: loaded (/lib/systemd/system/libvirtd.service; enabled; vendor preset: enabled)
Active: active (running) since Thu 2022-04-14 12:16:25 PDT; 1h 49min ago
TriggeredBy: ● libvirtd.socket
● libvirtd-admin.socket
● libvirtd-ro.socket
Docs: man:libvirtd(8)
https://libvirt.org
Main PID: 1007 (libvirtd)
Tasks: 20 (limit: 32768)
Memory: 51.5M
CGroup: /system.slice/libvirtd.service
├─1007 /usr/sbin/libvirtd
├─1139 /usr/sbin/dnsmasq --conf-file=/var/lib/libvirt/dnsmasq/default.conf --leasefile-ro --dhcp-script=/usr/lib/libvirt/libvirt_leaseshelper
└─1140 /usr/sbin/dnsmasq --conf-file=/var/lib/libvirt/dnsmasq/default.conf --leasefile-ro --dhcp-script=/usr/lib/libvirt/libvirt_leaseshelper

If it is not enabled, then enable with

sudo systemctl enable --now libvirtd

Use the **virt-manager** for the VM Installation:

sudo apt install virt-manager
virt-manager



On the top left of the window click to add a new VM. Go through the menu, and select local install, choose your ISO, then set the storage, memory and number of cores. For the network selection, select the host device and use a bridge as the source mode.

Continue through the environment setup for the OS, such as setting timezone, etc.

6.1.2 Enabling IOMMU on Host

Enable the IOMMU feature via the grub configuration. For an AMD system run:

```
sudo nano /etc/default/grub
```

Edit the line that starts with **GRUB_CMDLINE_LINUX_DEFAULT** to match:

```
GRUB_CMDLINE_LINUX_DEFAULT="amd_iommu=on iommu=pt"
```

If you are using an Intel system, then the line should read:

```
GRUB_CMDLINE_LINUX_DEFAULT="intel_iommu=on"
```

Afterwards run this command

```
sudo update-grub
```

Then reboot the system, when these changes are complete.

Verify if the IOMMU is enabled, by running the following after a reboot:

```
dmesg |grep AMD-Vi
[    1.047041] pci 0000:00:00.2: AMD-Vi: IOMMU performance counters supported
[    1.048758] pci 0000:00:00.2: AMD-Vi: Found IOMMU cap 0x40
[    1.048760] pci 0000:00:00.2: AMD-Vi: Extended features (0x58f77ef22294ade):
[    1.048763] AMD-Vi: Interrupt remapping enabled
[    1.048764] AMD-Vi: Virtual APIC enabled
[    1.048765] AMD-Vi: X2APIC enabled
[    1.048867] AMD-Vi: Lazy IO/TLB flushing enabled
```

6.1.3 Add Quadra to VM

To add Quadra to a KVM, first find the identifier for Quadra's PCIe:

```
lcpci
2c:00.0 SATA controller: Advanced Micro Devices, Inc. [AMD] FCH SATA Controller
[AHCI mode] (rev 51)
2d:00.0 Non-Volatile memory controller: NETINT Technologies Inc. Device 0401
2e:00.0 Non-Essential Instrumentation [1300]: Advanced Micro Devices, Inc. [AMD]
Starship/Matisse PCIe Dummy Function
2f:00.0 Non-Essential Instrumentation [1300]: Advanced Micro Devices, Inc. [AMD]
Starship/Matisse Reserved SPP
2f:00.3 USB controller: Advanced Micro Devices, Inc. [AMD] Matisse USB 3.0 Host
Controller

virsh nodedev-list --cap pci
pci_0000_2a_00_1
pci_0000_2a_00_3
pci_0000_2b_00_0
pci_0000_2c_00_0
pci_0000_2d_00_0
pci_0000_2e_00_0
pci_0000_2f_00_0
pci_0000_2f_00_3
Use 'virsh nodedev-dumpxml pci_0000_2d_00_0' to dump Quadra's PCIe
info for the virtual machine.
fpga@CLI309:~$ virsh nodedev-dumpxml pci_0000_2d_00_0
<device>
 <name>pci_0000_2d_00_0</name>
 <path>/sys/devices/pci0000:00/0000:00:03.1/0000:2d:00.0</path>
 <parent>pci_0000_00_03_1</parent>
 <driver>
  <name>vfio-pci</name>
 </driver>
 <capability type='pci'>
  <class>0x010802</class>
  <domain>0</domain>
  <bus>45</bus>
  <slot>0</slot>
```

```
  <function>0</function>
  <product id='0x0401'/>
  <vendor id='0x1d82'>NETINT Technologies Inc.</vendor>
  <capability type='virt_functions' maxCount='7'/>
  <iommuGroup number='23'>
   <address domain='0x0000' bus='0x2d' slot='0x00' function='0x0'/>
  </iommuGroup>
  <numa node='0'/>
  <pci-express>
   <link validity='cap' port='0' speed='16' width='4'/>
   <link validity='sta' speed='16' width='4'/>
  </pci-express>
 </capability>
</device>
```

Use the information highlighted in red, to add to the virtual machine so that the card can be found.

Use **virsh** edit '***yourvmname***' and add the following info.

```
<hostdev mode='subsystem' type='pci' managed='yes'>
 <source>
   <address domain='0x0000' bus='0x2d' slot='0x00' function='0x0'/>
 </source>
</hostdev>
```

Once done, run 'virsh start *yourvmname*' and install required packages for quadra and build ffmpeg for quadra.

Verify that Quadra is **detected,** and work with the following commands and expected outputs.

```
nvme@logan:~$ sudo nvme list
Node          SN                   Model                                    Namespace Usage
Format        FW Rev
----------------- -------------------- ---------------------------------------- --------- -------------------------- --
---------- --------
/dev/nvme0n1     Q1U02CA13BC056-0009  QuadraT1U-EP1                            1         8.59
TB /   8.59  TB     4 KiB +  0 B   ---52DEV

nvme@logan:~$ ni_rsrc_mon
NI resource not init'd, continue ..
Reading device file: nvme0
Compatible FW API ver: 52
Block name /dev/nvme0n1
1. /dev/nvme0  num_hw: 4
Creating shm_name: SHM_CODERS  lck_name: /dev/shm/NI_LCK_CODERS
0. nvme0
decoder h/w id 0 create
ni_rsrc_fill_device_info type 0 fmt 0
Creating shm_name: shm_d0 , lck_name /dev/shm/NI_lck_d0
ni_rsrc_get_one_device_info written out.
encoder h/w id 1 create
ni_rsrc_fill_device_info type 1 fmt 0
Creating shm_name: shm_e0 , lck_name /dev/shm/NI_lck_e0
ni_rsrc_get_one_device_info written out.
scaler h/w id 2 create
ni_rsrc_fill_device_info type 2 fmt 0
Creating shm_name: shm_s0 , lck_name /dev/shm/NI_lck_s0
ni_rsrc_get_one_device_info written out.
AI h/w id 3 create
ni_rsrc_fill_device_info type 3 fmt 0
Creating shm_name: shm_a0 , lck_name /dev/shm/NI_lck_a0
ni_rsrc_get_one_device_info written out.
************************************************
1 devices retrieved from current pool at start up
Thu Apr 14 17:52:00 2022 up 00:00:00 v---52DEV
```

```
Num decoders: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE
NAMESPACE
0    0   0       0   0   0       0      /dev/nvme0    /dev/nvme0n1
Num encoders: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE
NAMESPACE
0    0   0       0   0   0       0      /dev/nvme0    /dev/nvme0n1
Num scalers: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE
NAMESPACE
0    0   0       0   0   0       0      /dev/nvme0    /dev/nvme0n1
Num AIs: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE
NAMESPACE
0    0   0       0   0   0       0      /dev/nvme0    /dev/nvme0n1
**************************************************
```

6.2 Linux Guest VM with Passthrough Virtual Device

This section requires that SR-IOV enabled firmware (v4.1.1+) is running on NETINT video transcoder device. Refer app note APPS525 – Codensity Quadra SR-IOV Configuration and Usage guide for details.

6.3 Windows Guest VM with Passthrough Virtual Device

Quadra does not support SR-IOV on Windows.

7 Kubernetes

The Kubernetes orchestration tool allows NETINT video transcoder devices to be managed as nodes. This section details the configuration and usage of NETINT video transcoder device with Kubernetes. The document refers to Quadra video transcoder device but can be used for other NETINT video transcoder solutions as well. This section assumes you have already setup Docker for Quadra. Please review section 4, if Docker has not yet been setup on your environment.

7.1 Pre-requisites for the Quadra Environment on Host

Ensure on the Linux Host the following pre-requisites

1. Quadra libxcoder and FFmpeg source code installed and compiled successfully. In this document FFmpeg 4.2.1 is used for illustration purposes. Please refer to the **Quick Start Guide** to install other versions of libxcoder and FFmpeg.

2. Running '**ffmpeg'** gives an ouput dump similar to the sample below

```
$ ffmpeg
ffmpeg version 4.3.1 Copyright (c)2000-2020 the FFmpeg developers
  built with gcc 9 (Ubuntu 9.3.0-17ubuntu1~20.04)
  configuration: --pkg-config-flags=--static --enable-gpl --enable-nonfree --extra-
ldflags=-lm --extra-ldflags=-ldl --enable-libxcoder --enable-ni --enable-pthreads --extra-
libs=-lpthread --enable-encoders --enable-decoders --enable-avfilter --enable-muxers --
enable-demuxers --enable-parsers --enable-x86asm --disable-debug --disable-ffplay --
enable-ffprobe --enable-libx264 --enable-libx265 --enable-libaom --disable-libvpx --
disable-cuda-nvcc --disable-cuda --disable-cuvid --disable-nvdec --disable-nvenc --
disable-libvmaf --enable-static --disable-shared --extra-cflags=-
UNIENC_MULTI_THREAD
  libavutil      56. 51.100 / 56. 51.100
  libavcodec     58. 91.100 / 58. 91.100
  libavformat    58. 45.100 / 58. 45.100
  libavdevice    58. 10.100 / 58. 10.100
  libavfilter     7. 85.100 /  7. 85.100
  libswscale      5.  7.100 /  5.  7.100
  libswresample   3.  7.100 /  3.  7.100
  libpostproc    55.  7.100 / 55.  7.100
Hyper fast Audio and Video encoder
usage: ffmpeg [options] [[infile options] -i infile]... {[outfile options] outfile}...
Use -h to get full help or, even better, run 'man ffmpeg'
```

3. Running '**sudo nvme list**' displays all installed Quadra NVMe devices.

```
$ sudo nvme list
Node          SN                   Model                                    Namespace Usage
Format        FW Rev
----------------- -------------------- ---------------------------------------- --------- -------------------------- --
----------
/dev/nvme0n1     Q1A10BA11FC060-0124  QuadraT1A                                1         8.59  TB
/   8.59  TB     4 KiB +  0 B   ---00DEV
/dev/nvme1n1     Q1A10BA11FC060-0142  QuadraT1A-EP1                              1         8.59
TB /   8.59  TB     4 KiB +  0 B   ---00DEV
```

4. Running '**ni_rsrc_mon**' or '**sudo ni_rsrc_mon**' provides an output dump similar to the sample below

```
$ ni_rsrc_mon
NI resource init'd already ..
**************************************************
2 devices retrieved from current pool at start up
Mon Jan 31 13:35:40 2022 up 00:00:00 v---00DEV
Num decoders: 2
BEST INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM DEVICE       NAMESPACE
L   0    0   0       0    0   0       /dev/nvme0    /dev/nvme0n1
    1    0   0       0    0   0       /dev/nvme1    /dev/nvme1n1
Num encoders: 2
BEST INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM DEVICE       NAMESPACE
L   0    0   0       0    0   0       /dev/nvme0    /dev/nvme0n1
    1    0   0       0    0   0       /dev/nvme1    /dev/nvme1n1
Num scalers: 2
BEST INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM DEVICE       NAMESPACE
L   0    0   0       0    0   0       /dev/nvme0    /dev/nvme0n1
    1    0   0       0    0   0       /dev/nvme1    /dev/nvme1n1
Num ais: 2
BEST INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM DEVICE       NAMESPACE
L   0    0   0       0    0   0       /dev/nvme0    /dev/nvme0n1
    1    0   0       0    0   0       /dev/nvme1    /dev/nvme1n1
**************************************************
```

7.2 Kubernetes Package Install

Some additional packages are required for Kubernetes to run on the host. Commands are for X86 host with Ubuntu 20.

1. Install the kubectl package for kubernetes

   sudo apt install apt-transport-https curl
   curl https://mirrors.aliyun.com/kubernetes/apt/doc/apt-key.gpg | sudo apt-key add -
   sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"
   sudo apt-get install -y kubelet kubeadm kubectl

2. Check that the kubelet service is working on the host.

   systemctl restart kubelet.service

   systemctl status kubelet.service

   ● kubelet.service - kubelet: The Kubernetes Node Agent
   Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: enabled)
   Drop-In: /etc/systemd/system/kubelet.service.d
   └─10-kubeadm.conf
   Active: active (running) since Tue 2022-04-12 21:02:02 PDT; 22h ago

Note: Kubernetes will not work if swap memory is enabled, make sure to disable.

3. Install minikube, a tool that creates a local cluster for kubernetes

   curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube_latest_amd64.deb

   sudo dpkg -i minikube_latest_amd64.deb

   sudo minikube start --vm-driver=none --image-repository=registry.cn-hangzhou.aliyuncs.com/google_containers --extra-config=kubeadm.pod-network-cidr='10.244.0.0/16'

4. After that, run the following to finish the installation.

   mkdir -p $HOME/.kube
   sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
   sudo chown $(id -u):$(id -g) $HOME/.kube/config

5. Your installation should have succeeded, you can now check available nodes with:

   kubectl get nodes
   kubectl get pods -n kube-system

NAME	READY	STATUS	RESTARTS	AGE
coredns-65c54cc984-g9qdr	1/1	Running	60 (22h ago)	34d
etcd-cli406	1/1	Running	68 (22h ago)	34d
kube-apiserver-cli406	1/1	Running	68 (22h ago)	34d
kube-controller-manager-cli406	1/1	Running	69 (22h ago)	34d
kube-proxy-6p4th	1/1	Running	60 (22h ago)	34d
kube-scheduler-cli406	1/1	Running	61 (22h ago)	34d
storage-provisioner	1/1	Running	110 (22h ago)	34d

7.3 Install Packages for Quadra's Kubernetes Plugin

Quadra has a specialized plugin for Kubernetes so it can be used as a pod. This section will install the required packages for the plugin to work.

6. Install the following packages:

   sudo apt install golang-go
   sudo apt install gccgo-go
   sudo apt-get install apt-transport-https --yes
   echo "deb https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list
   sudo apt-get update
   sudo apt-get install helm

7.4 Acquire NETINT Kubernetes plugin

7. Download the plugin compilation files from the NETINT's github public repo: https://github.com/NETINT-Technologies/k8_device_plugin

7.5 Activate NETINT plugin

Activate the plugin:

8. Enter the plugin folder and run the following commands:

   cd netint-device-plugin_release
   make buildImage
   make deploy

9. Check available nodes with the following:

   kubectl get pods -n kube-system

```
NAME                              READY   STATUS    RESTARTS
AGE
coredns-65c54cc984-g9qdr          1/1     Running   60 (22h ago)
34d
etcd-cli406                       1/1     Running   68 (22h ago)
34d
kube-apiserver-cli406             1/1     Running   68 (22h ago)
34d
kube-controller-manager-cli406    1/1     Running   69 (22h ago)
34d
kube-proxy-6p4th                  1/1     Running   60 (22h ago)
34d
kube-scheduler-cli406             1/1     Running   61 (22h ago)
34d
netint-qs7ck                      1/1     Running   33 (22h ago)
20d
storage-provisioner               1/1     Running   110 (22h ago)
34d
```

You should find the netint-#####, pod running with ready 1/1 and status should be running. When you access this pod, you can install all packages required for quadra within the pod through the quadra_setup script and build FFmpeg for quadra within the pod.

After copying quadra_setup over, run it in the pod, it will install all the packages required. Copy over the FFmpeg folder to the pod and build FFmpeg, you will be able to run commands within the pod now.

After that try running ‘sudo nvme list’ and ‘ni_rsrc_mon’, you should see something like the following:

```
[root@netint-qs7ck /]# sudo nvme list
Node             SN                   Model
Namespace Usage                      Format           FW Rev
---------------- -------------------- --------------------------
------------ --------- -------------------------- --------------
-- --------
/dev/nvme0n1     Q1A10BA11FC060-0124  QuadraT1A
1           8.59  TB /   8.59  TB      4 KiB +  0 B   ---51DEV
/dev/nvme1n1     Q1A10BA11FC060-0142  QuadraT1A-EP1
1           8.59  TB /   8.59  TB      4 KiB +  0 B   ---51DEV

[root@netint-qs7ck /]# ni_rsrc_mon
NI resource init'd already ..
************************************************
2 devices retrieved from current pool at start up
Thu Apr 14 02:54:11 2022 up 00:00:00 v---51DEV
Num decoders: 2
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE
NAMESPACE
0     0    0          0    0    0         0       /dev/nvme0
/dev/nvme0n1
1     0    0          0    0    0         0       /dev/nvme1
/dev/nvme1n1
Num encoders: 2
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE
NAMESPACE
0     0    0          0    0    0         0       /dev/nvme0
/dev/nvme0n1
1     0    0          0    0    0         0       /dev/nvme1
/dev/nvme1n1
Num scalers: 2
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE
NAMESPACE
0     0    0          0    0    0         0       /dev/nvme0
/dev/nvme0n1
1     0    0          0    0    0         0       /dev/nvme1
/dev/nvme1n1
Num AIs: 2
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE
NAMESPACE
```

```
0      0     0           0     0     0          0        /dev/nvme0
/dev/nvme0n1
1      0     0           0     0     0          0        /dev/nvme1
/dev/nvme1n1
```

8 Windows Host

This section explains how to install and use the NETINT video transcoder device on a Windows host.

8.1 Operating System Support

The following Windows operating system versions are supported:

- Windows 10
- Windows Server 2012
- Windows Server 2016
- Server 2019

8.2 Installing and running FFmpeg using MSYS2

This section describes the Windows MSYS2 environment setup and usage with NETINT video transcoding device.

8.2.1 Operating Systems

A host server with one of the following operating systems installed is recommended (minor versions can be inconsistent):

- OS: Windows 10 [Version 10.0.19042.1052]
- OS: Windows Server 2012 R2 [Version 6.3.9600]
- OS: Windows Server 2016 [Version 10.0.14393.693]
- OS: Windows Server 2019 [Version 10.0.17763.737]

8.2.2 Setup MSYS2 tools on Windows

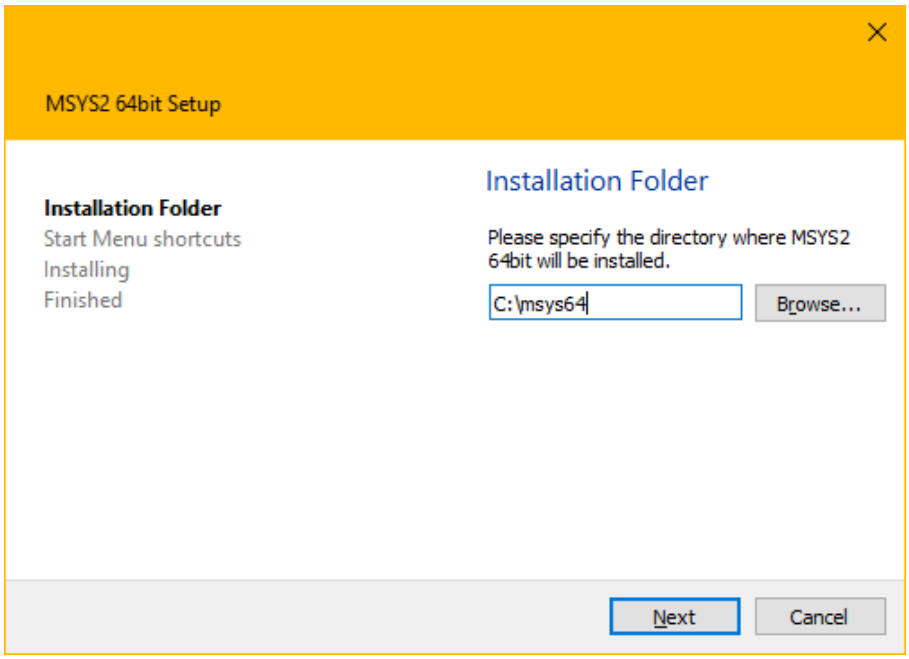
Provide two ways to install the MSYS2 tools package on Windows:

- Online installation
- Offline installation

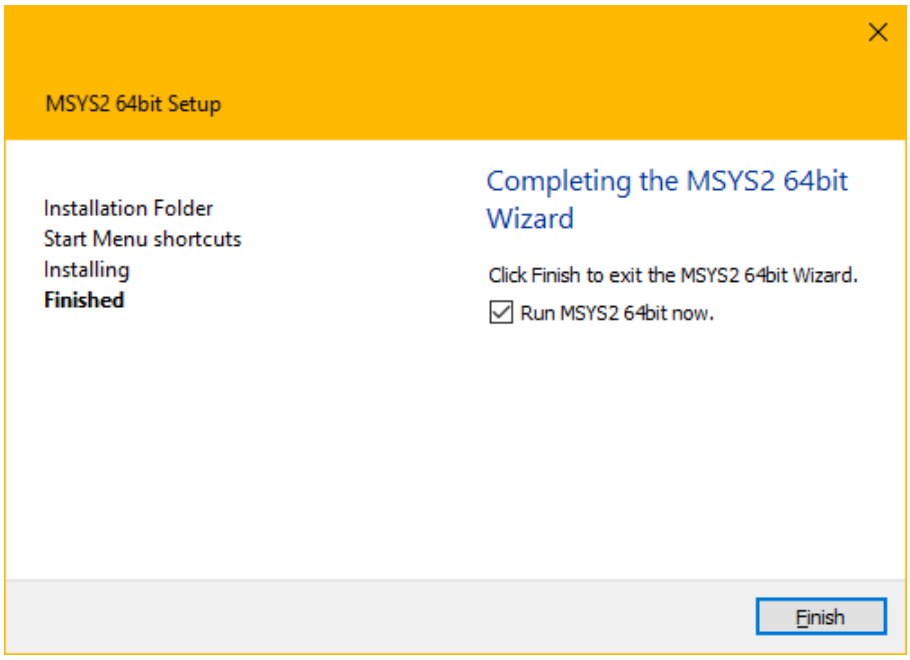
8.2.2.1 Install the MSYS2 - Online

This is a reference to MSYS2. A few additional notes have also been added.

1. Download the latest msys2 installer from https://www.msys2.org/
2. Run the installer. MSYS2 requires 64 bit Windows 7 or newer.
3. Enter your desired Installation Folder (short ASCII-only path on an NTFS volume, no accents, no spaces, no symlinks, no subst or network drives, no FAT).



4. When done, tick Run MSYS2 now



5. Update the package database and base packages. If you miss the step 4 and the Msys2 is not started, go to the installation folder “C:\msys64” and run “***msys2.exe***”. Unless your setup file is very recent, it will take two steps. First run ***pacman -Syu*** (*__Note__: Select **Y** at the end):

```
$ pacman -Syu
:: Synchronizing package databases...
 mingw32                 805.0 KiB
 mingw32.sig               438.0   B
 mingw64                 807.9 KiB
 mingw64.sig               438.0   B
 msys                    289.3 KiB
 msys.sig                  438.0   B
:: Starting core system upgrade...
warning: terminate other MSYS2 programs before proceeding
resolving dependencies...
looking for conflicting packages...

Packages (6) bash-5.1.004-1  filesystem-2021.01-1
             mintty-1~3.4.4-1  msys2-runtime-3.1.7-4
             pacman-5.2.2-9  pacman-mirrors-20201208-1

Total Download Size:   11.05 MiB
Total Installed Size:  53.92 MiB
Net Upgrade Size:      -1.24 MiB

:: Proceed with installation? [Y/n]
:: Retrieving packages...
 bash-5.1.004-1-x86_64          2.3 MiB
 filesystem-2021.01-1-any       33.2 KiB
 mintty-1~3.4.4-1-x86_64       767.2 KiB
 msys2-runtime-3.1.7-4-x86_64    2.6 MiB
 pacman-mirrors-20201208-1-any   3.8 KiB
 pacman-5.2.2-9-x86_64          5.4 MiB
```

(6/6) checking keys in keyring 100%
(6/6) checking package integrity 100%
(6/6) loading package files 100%
(6/6) checking for file conflicts 100%
(6/6) checking available disk space 100%
:: Processing package changes...
(1/6) upgrading bash 100%
(2/6) upgrading filesystem 100%
(3/6) upgrading mintty 100%
(4/6) upgrading msys2-runtime 100%
(5/6) upgrading pacman-mirrors 100%
(6/6) upgrading pacman 100%
:: To complete this update all MSYS2 processes including this terminal will be closed. Confirm to proceed [Y/n]

6. Go to the installation folder “C:\msys64” and run “***msys2.exe***”. Update the rest of the base packages with ***pacman -Su*** (*__Note__: Select **Y** at the end):

$ pacman -Su
:: Starting core system upgrade...
there is nothing to do
:: Starting full system upgrade...
resolving dependencies...
looking for conflicting packages...

Packages (20) base-2020.12-1 bsdtar-3.5.0-1
[... more packages listed ...]

Total Download Size: 12.82 MiB
Total Installed Size: 44.25 MiB
Net Upgrade Size: 3.01 MiB

:: Proceed with installation? [Y/n]
[... downloading and installation continues ...]

7. Now MSYS2 is ready. You need to install some essential tools to compile libxcoder and ffmpeg. Below are the commands to install the tools, select the default option **Y** if asked to confirm installation :

   $ pacman -S gcc

   $ pacman -S mingw-w64-x86_64-toolchain

   [NOTE: If multiple choices are given for the above, then select the default option **all]**

   $ pacman -S base-devel

   $ pacman -S yasm

   $ pacman -S mingw-w64-x86_64-SDL2

   $ pacman -S git

   $ pacman -S xz

8. To start building using the mingw-w64 GCC, go to the installation folder "C:\msys64" and run "***mingw64.exe***". Now the tool is ready to build ffmpeg for Windows.

Note – Refer Appendix-A to update Windows environment variables PATH to include MSYS2 environment. This will help access MSYS2 linux commands from Windows Command Terminal window as well.

8.2.2.2 Install the MSYS2 – Offline

Skip these steps if you have already run steps in Section 2.1.

1. Get the offline installation package “msys64.zip” from NETINT Support team.
2. Unzip the msys64.zip to the C: folder. The essential tools have already been installed in this package.
3. To start building using the mingw-w64 GCC, go to the installation folder “C:\msys64” and run “***mingw64.exe***”.

Note 1 – This offline zip package comes with home folder C:\msys64\home\Administrator.

Note 2 – This offline zip package comes with a ffmpeg tar file base_ffmpeg_n4.2.1.tar.gz. If following the offline Section 2.2, you will untar this file in the steps for Section 3.2. File can be found in C:\msys64\home\Administrator folder.

8.2.3 Compile Libxcoder and FFmpeg

8.2.3.1 Overview

In order to use NETINT Quadra encoding and decoding features using FFmpeg, the user needs the following minimum steps:

1. Clone an FFmpeg Repository
2. Get a Quadra Release Package with libxcoder and patch files
3. Patch the FFmpeg installation
4. Compile libxcoder
5. Compile FFmpeg
6. Locate and run init_rsrc.exe in a command window
7. Run the FFmpeg command in a second command window

Full instructions are given below.

8.2.3.2 Download the FFmpeg Repository

Start an MSYS2 Mingw64 terminal. There are two methods for installing FFmpeg. The first is to clone the repository from GitHub, and the second is to download and install a tarball from the FFmpeg website.

Sourcing FFmpeg from GitHub

To clone the FFmpeg repository from GitHub, you need to specify the correct FFmpeg version. Once example is FFmpeg version 4.3.1. Using the Mingw64 command line, run the following

```
cd C:
cd msys64/home/nvme
git clone -b n4.3.1 --depth=1 https://github.com/FFmpeg/FFmpeg.git FFmpeg
```

Note – the example below assumes the username '**nvme**', which means the home folder is C:/msys64/home/nvme. Replace '**nvme**' with the correct username on your PC. Check the correct subfolder name in the folder **C:/msys64/home**

Other valid options for '**git clone -b'** are other versions of FFMpeg for example n4.2.4, n4.3.2, and n4.4

Sourcing FFmpeg from the official FFmpeg Website

Alternatively, to install FFmpeg from the official website follow these steps.

Use the 'Download xz tarball' from the official FFmpeg website

https://ffmpeg.org/download.html#releases

Move the file to the C:/msys64/home/nvme directory, then execute the following steps.

```
cd C:
cd msys64/home/nvme
xz -d ffmpeg-4.3.1.tar.xz
tar -xvf ffmpeg-4.3.1.tar
mv ffmpeg-4.3.1 FFmpeg
```

Note that FFMpeg release 4.3.1 is used in the example above.

Sourcing FFmpeg from the offline pre-packaged ffmpeg tar file

If you followed the offline steps in Section 2.2 then you can also find a downloaded FFmpeg package "base_ffmpeg_n4.2.1.tar.gz" in the offline package folder

C:\msys64\home\Administrator

Run the following command to untar this tar file

```
tar -zxf base_ffmpeg_n4.3.1.tar.gz
```

8.2.3.3 Download Quadra Release Package

Contact the NETINT Support team to obtain the latest Quadra release package. The release package will contain a zip file entitled ‘Quadra_V*.*.*.zip’.

Where, *.*.* is the software release version=, for example 5.0.0 would be the file

Quadra_V5.0.0.zip

8.2.3.4 Apply Patch to FFmpeg Repository

The following instructions need to be followed in sequence to prepare FFmpeg for use with the NETINT Quadra Video Transcoder.

1. Unzip the Codensity Quadra Software Release package to the $HOME folder:

   cd C:
   cd msys64/home/nvme
   unzip Quadra_V5.0.0.zip

2. Copy the *libxcoder/* folder from the uncompressed release package to the parent folder of FFmpeg (i.e. same level as FFmpeg):

   cp -r Quadra_V5.0.0/libxcoder ./

3. Copy the FFmpeg patch file from the release package to the *FFmpeg* folder:
   Note – this example is for patching a FFmpeg 4.3.x release.

   cp Quadra_V5.0.0/Quadra_SW_V5.0.0_RC2/FFmpeg-n4.3.1_netint_v5.0.0_RC2.diff $HOME/FFmpeg

4. Change directories to the *FFmpeg/* folder:

   cd C:
   cd msys64/home/nvme
   cd FFmpeg

5. Apply the patch depending on the FFmpeg <version> (ex. 3.1.1, 3.4.2, 4.1.3, 4.2.1, 4.3.1):

   patch -t -p 1 < FFmpeg-n4.3.1_netint_v5.0.0_RC2.diff

It is critical to apply the correct NETINT patch for the FFmpeg version in use. Check with NETINT Support team for the patch to use. In general, patch file FFmpeg-**n4.2.x**_netint will work with all FFmpeg 4.2 versions, patch file FFmpeg-n4.3.x_netint will work with all FFmpeg 4.3 versions, and so on.

8.2.3.5 Build FFmpeg with NETINT Codec Library

The following instructions need to be done in sequence to build FFmpeg with the NETINT Codec Library.

1. From the *FFmpeg/* folder, go to the *libxcoder/* folder with the following command:

   cd C:
   cd msys64/home/nvme
   cd FFmpeg
   cd ../libxcoder

2. Build and install libxcoder with the following commands. Upon a successful compile the files are auto installed to /usr/local/bin folder.

   $ bash build.sh -w
   $ ls /usr/local/bin
   init_rsrc.exe ni_rsrc_list.exe ni_rsrc_mon.exe ni_rsrc_update.exe

3. Update pkg-config path:

   export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig:$PKG_CONFIG_PATH

4. Go to the *FFmpeg/* directory with the following command:

   cd ../FFmpeg

5. Run the build_FFmpeg.sh script with the following commands. When successfully built, the executables (.EXE) and libraries can be found in the ***/usr/local/bin*** and ***/usr/local/lib*** folders:

   make clean
   bash build_FFmpeg.sh -w
   make install

6. Some related libraries are compiled dynamically, so the following libraries “libiconv-2.dll, libwinpthread-1.dll, SDL2.dll, zlib1.dll” need to be copied from

   C:/msys64/mingw64/bin to C:/msys64/usr/local/bin

   In a MSYS2 or Mingw64 terminal window run the following commands to confirm the folder contents:

   $ ls /usr/local/bin
   SDL2.dll init_rsrc.exe libwinpthread-1.dll ni_rsrc_mon.exe zlib1.dll ffmpeg.exe
   libiconv-2.dll ni_rsrc_list.exe ni_rsrc_update.exe

   $ ls /usr/local/lib/*
   /usr/local/lib/libavcodec.a /usr/local/lib/libpostproc.a
   /usr/local/lib/libavdevice.a /usr/local/lib/libswresample.a
   /usr/local/lib/libavfilter.a /usr/local/lib/libswscale.a
   /usr/local/lib/libavformat.a /usr/local/lib/libxcoder.a
   /usr/local/lib/libavutil.a

   $ ls /usr/local/lib/pkgconfig
   libavcodec.pc libavfilter.pc libavutil.pc libswresample.pc xcoder.pc
   libavdevice.pc libavformat.pc libpostproc.pc libswscale.pc

This completes the installation steps in prep to start using Quadra card on your Windows host.

8.2.3.6 Build FFmpeg with Libxcoder on Quadra and Logan Co-existence Cards

The feature compatibility between Quadra and Logan can be referred to Chapter 4.4.1. The following instructions need to be done in sequence to build FFmpeg with the NETINT Codec Library.

1. From the *FFmpeg/* folder, go to the *libxcoder/* folder with the following command:

   cd C:
   cd msys64/home/nvme
   cd FFmpeg
   cd ../libxcoder

2. Build and install libxcoder with the following commands. Upon a successful compile the files are auto installed to /usr/local/bin folder.

   $ bash build.sh -w
   $ ls /usr/local/bin
   init_rsrc.exe ni_rsrc_list.exe ni_rsrc_mon.exe ni_rsrc_update.exe
   ni_rsrc_namespace.exe test_rsrc_api.exe

3. From the *FFmpeg/* folder, go to the *libxcoder_logan/* folder with the following command:

   cd C:
   cd msys64/home/nvme
   cd FFmpeg
   cd ../libxcoder_logan

4. Build and install libxcoder_logan with the following commands. Upon a successful compile the files are auto installed to /usr/local/bin folder.

   $ bash build.sh -w
   $ ls /usr/local/bin
   init_rsrc_logan.exe ni_logan_rsrc_list.exe ni_logan_rsrc_mon.exe
   ni_logan_rsrc_update.exe test_rsrc_api_logan.exe

5. Update pkg-config path:

   export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig:$PKG_CONFIG_PATH

6. Go to the *FFmpeg/* directory with the following command:

   cd ../FFmpeg

7. Run the build_FFmpeg.sh script with the following commands. When building successfully, the EXEs and libraries can be found in */usr/local/bin* and */usr/local/lib* folders:

   make clean
   bash build_FFmpeg.sh -w –-quadra --logan
   make install

8. Some related libraries are compiled dynamically, so the following libraries “libiconv-2.dll, libwinpthread-1.dll, SDL2.dll, zlib1.dll” need to be copied from C:/msys64/mingw64/bin to C:/msys64/usr/local/bin
   In a MSYS2 or Mingw64 terminal window run the following commands to confirm the folder contents:

   $ ls /usr/local/bin
   SDL2.dll init_rsrc.exe libwinpthread-1.dll ni_rsrc_mon.exe zlib1.dll ffmpeg.exe
   libiconv-2.dll ni_rsrc_list.exe ni_rsrc_update.exe init_rsrc_logan.exe
   ni_logan_rsrc_mon.exe ni_logan_rsrc_list.exe ni_logan_rsrc_update.exe
   test_rsrc_api_logan.exe test_rsrc_api.exe

   $ ls /usr/local/lib/*
   /usr/local/lib/libavcodec.a /usr/local/lib/libpostproc.a
   /usr/local/lib/libavdevice.a /usr/local/lib/libswresample.a
   /usr/local/lib/libavfilter.a /usr/local/lib/libswscale.a
   /usr/local/lib/libavformat.a /usr/local/lib/libxcoder.a
   /usr/local/lib/libavutil.a /usr/local/lib/libxcoder_logan.a
   $ ls /usr/local/lib/pkgconfig
   libavcodec.pc libavfilter.pc libavutil.pc libswresample.pc xcoder.pc
   xcoder_logan.pc
   libavdevice.pc libavformat.pc libpostproc.pc libswscale.pc

This completes the installation steps in prep to start using Quadra and Logan cards on your Windows host.

8.2.4 Run FFmpeg with Quadra Cards

After a successful compilation, we can test with Quadra cards.

1. Check Hardware.
   With administrator privileges, open a Windows command terminal. Check the Quadra card presence with the following command:
   $ wmic diskdrive get Name,SerialNumber,Model,Size

```
C:\>wmic diskdrive get Name,SerialNumber,Model,Size
Model                   Name                SerialNumber                 Size
QuadraT1A-EP1           \\.\PHYSICALDRIVE1  Q1A10BA11FC060-0048 _00000001.  4294912204800
WDC  WDS250G2B0A-00SM50 \\.\PHYSICALDRIVE0  212024806551                   250056737280
```

2. Initialize Quadra
   With administrator privileges, open a **Windows command terminal**. Go to C:/msys64/usr/local/bin folder. Initialize the Quadra with the following command (*Note: this process must keep running during transcoding).

   cd C:/msys64/usr/local/bin

   init_rsrc.exe

```
Microsoft Windows [Version 10.0.19042.1466]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>init_rsrc
NETINT resources not initialized, starting initialization ..
Searching for NETINT NVMe devices ...

Total Number of NETINT NVMe Transcoders indentified: 1

decoder h/w id 0 create
Creating shm_name: shm_d0
CreateFileMapping created a new mapFile for shm_d0, handle: 00000000000000e4  ..
encoder h/w id 1 create
Creating shm_name: shm_e0
CreateFileMapping created a new mapFile for shm_e0, handle: 00000000000000e8  ..
scaler h/w id 2 create
Creating shm_name: shm_s0
CreateFileMapping created a new mapFile for shm_s0, handle: 00000000000000ec  ..
ai h/w id 3 create
Creating shm_name: shm_a0
CreateFileMapping created a new mapFile for shm_a0, handle: 00000000000000f0  ..
NETINT Resources Intitialized Successfully
```

3. Run FFmpeg

With administrator privileges, open a **Windows command terminal**. Go to your *C:/msys64/usr/local/bin folder*. Run the following commands with ffmpeg to do the transcoding.
The following examples use clips folder *C:/msys64/home/nvme/libxcoder/test*. Update this path if you wish to run custom clips for decoding or encoding.

```
cd C:/msys64/usr/local/bin
ffmpeg.exe
```

- test h264 decoder:

  $ ffmpeg.exe -y -hide_banner -nostdin -vsync 0 -c:v h264_ni_quadra_dec -i C:/msys64/home/nvme/libxcoder/test/1280x720p_Basketball.264 -c:v rawvideo output_5.yuv

- test h265 decoder:

  $ ffmpeg.exe -y -hide_banner -nostdin -vsync 0 -c:v h265_ni_quadra_dec -i C:/msys64/home/nvme/libxcoder/test/akiyo_352x288p25.265 -c:v rawvideo akiyo_352x288p25.yuv

- test h264 encoder:

  $ ffmpeg.exe -y -hide_banner -nostdin -f rawvideo -pix_fmt yuv420p -s:v 352x288 -r 25 -i C:/msys64/home/nvme/libxcoder/test/akiyo_352x288p25.yuv -c:v h264_ni_quadra_enc output_7.h264

- test h265 encoder:

  $ ffmpeg.exe -y -hide_banner -nostdin -f rawvideo -pix_fmt yuv420p -s:v 352x288 -r 25 -i C:/msys64/home/nvme/libxcoder/test/akiyo_352x288p25.yuv -c:v h265_ni_quadra_enc output_8.h265

- test 264->265 transcoder:

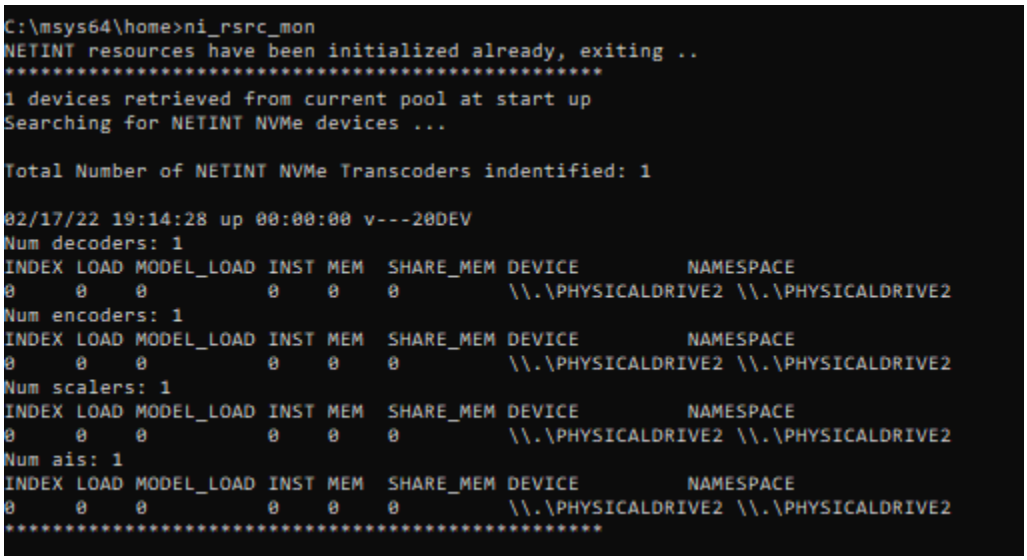
  $ ffmpeg.exe -y -hide_banner -nostdin -vsync 0 -c:v h264_ni_quadra_dec -i C:/msys64/home/nvme/libxcoder/test/1280x720p_Basketball.264 -c:v h265_ni_quadra_enc output_9.h265

Note: On Windows, when you add some params, you must use double quotation mark.
For example: -xcoder-params “out=hw”.

4. Monitoring Load

With administrator privileges, open a **Windows command terminal**. Go to your *C:/msys64/usr/local/bin* folder. Run the following command:

ni_rsrc_mon.exe



Reporting columns

INDEX	**number used by resource manager to identify the resource**
LOAD	**realtime load**
MODEL_LOAD	**estimated load based on framerate and resolution**
INST	**number of job instances**
DEVICE	**path to NVMe device file handle**
NAMESPACE	**path to NVMe namespace file handle**

8.2.5 Run FFmpeg with Quadra and Logan Co-existence Cards

After a successful compilation, we can test Quadra and Logan co-existence cards.

1. Check Hardware

   With administrator privileges, open a Windows command terminal. Check the Quadra and Logan card presence with the following command:

   $ wmic diskdrive get Name,SerialNumber,Model,Size

```
C:\Windows\System32\wbem>wmic diskdrive get Name,SerialNumber,Model,Size
Model                  Name                SerialNumber              Size
KINGSTON SA400S37240G  \\.\PHYSICALDRIVE0  50026B7380A743B8          240054796800
QuadraT1A-EP1          \\.\PHYSICALDRIVE2  Q1A10BA11FC060-0131       8589890211840
T408-U2                \\.\PHYSICALDRIVE1  TU05-08-02-C10-0972       536864025600
```

2. Initialize Quadra

   With administrator privileges, open a **Windows command terminal**. Go to C:/msys64/usr/local/bin folder. Initialize the Quadra with the following command (*Note: this process must keep running during transcoding).

   cd C:/msys64/usr/local/bin
   init_rsrc.exe

```
c:\build_mingw>init_rsrc.exe
NETINT resources not initialized, starting initialization ..
Searching for NETINT NVMe devices ...

Total Number of NETINT NVMe Transcoders identified: 1

Found NVME Controller at \\.\PHYSICALDRIVE2
WARNING - Query \\.\PHYSICALDRIVE2 FW version: ---6IDEV is below the minimum support version for this SW version. Some f
eatures may be missing.
decoder h/w id 0 create
ni_rsrc_fill_device_info type 0 fmt 0
Creating shm_name: NI_shm_d0
CreateFileMapping created a new mapFile for NI_shm_d0, handle: 00000000000000ec  ..
encoder h/w id 1 create
ni_rsrc_fill_device_info type 1 fmt 0
Creating shm_name: NI_shm_e0
CreateFileMapping created a new mapFile for NI_shm_e0, handle: 00000000000000f0  ..
scaler h/w id 2 create
ni_rsrc_fill_device_info type 2 fmt 0
Creating shm_name: NI_shm_s0
CreateFileMapping created a new mapFile for NI_shm_s0, handle: 00000000000000f4  ..
AI h/w id 3 create
ni_rsrc_fill_device_info type 3 fmt 0
Creating shm_name: NI_shm_a0
CreateFileMapping created a new mapFile for NI_shm_a0, handle: 00000000000000f8  ..
NETINT Resources Intitialized Successfully
```

3. Initialize Logan

   With administrator privileges, open a **Windows command terminal**. Go to C:/msys64/usr/local/bin folder. Initialize the Quadra with the following command (*Note: this process must keep running during transcoding).

   cd C:/msys64/usr/local/bin
   init_rsrc_logan.exe

```
c:\build_vs2019>init_rsrc_logan.exe
NETINT resources not initialized, starting initialization ..
Searching for NETINT NVMe devices ...

Identity information retrieved from the device at port \\.\PHYSICALDRIVE0
    VID:            0x7423
    SSVID:          0x2062
Device at port \\.\PHYSICALDRIVE0 is not a NETINT NVMe device
Identity information retrieved from the device at port \\.\PHYSICALDRIVE1
    VID:            0x1d82
    SSVID:          0x1d82
    Device Model:   T408-U2
    Firmware Rev:   310X2013
    Serial Number: TU05-08-02-C10-0972
NETINT T408-U2 NVMe video transcoder identified at port \\.\PHYSICALDRIVE1

Identity information retrieved from the device at port \\.\PHYSICALDRIVE2
    VID:            0x0
    SSVID:          0x0
Device at port \\.\PHYSICALDRIVE2 is not a NETINT NVMe device
Total Number of NETINT NVMe Transcoders indentified: 1

decoder h/w id 0 create
Creating shm_name: NI_LOGAN_shm_d0
CreateFileMapping created a new mapFile for NI_LOGAN_shm_d0, handle: 0000000000000108  ..
decoder h/w id 0 update
encoder h/w id 1 create
Creating shm_name: NI_LOGAN_shm_e0
CreateFileMapping created a new mapFile for NI_LOGAN_shm_e0, handle: 0000000000000110  ..
encoder h/w id 1 update
NETINT Logan Resources Intitialized Successfully
```

4. Run FFmpeg

   With administrator privileges, open a **Windows command terminal**. Go to your *C:/msys64/usr/local/bin folder*. Run the FFmpeg commands to do the transcoding just as mentioned in Chapter 8.2.4.

5. Monitoring Load

   With administrator privileges, open a **Windows command terminal**. Go to your *C:/msys64/usr/local/bin* folder. Run the following command:

   ni_rsrc_mon.exe or ni_logan_rsrc_mon.exe

```
c:\build_vs2019>ni_rsrc_mon.exe
NETINT resources have been initialized already, exiting ..
**************************************************
1 devices retrieved from current pool at start up
Searching for NETINT NVMe devices ...

Total Number of NETINT NVMe Transcoders identified: 1

Mon Mar 20 15:04:43 2023 up 00:00:00 v---6HDEV
Num decoders: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE         NAMESPACE
Found NVME Controller at \\.\PHYSICALDRIVE2
0     0    0          0    0    0         0       \\.\PHYSICALDRIVE2 \\.\PHYSICALDRIVE2
Num encoders: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE         NAMESPACE
Found NVME Controller at \\.\PHYSICALDRIVE2
0     0    0          0    0    0         0       \\.\PHYSICALDRIVE2 \\.\PHYSICALDRIVE2
Num scalers: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE         NAMESPACE
Found NVME Controller at \\.\PHYSICALDRIVE2
0     0    0          0    0    0         0       \\.\PHYSICALDRIVE2 \\.\PHYSICALDRIVE2
Num AIs: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE         NAMESPACE
Found NVME Controller at \\.\PHYSICALDRIVE2
0     0    0          0    0    0         0       \\.\PHYSICALDRIVE2 \\.\PHYSICALDRIVE2
**************************************************
```

```
c:\build_vs2019>ni_rsrc_mon_logan.exe
NETINT resources have been initialized already, exiting ..
**************************************************
1 devices retrieved from current pool at start up
Searching for NETINT NVMe devices ...

Identity information retrieved from the device at port \\.\PHYSICALDRIVE0
    VID:           0x7423
    SSVID:         0x2062
Device at port \\.\PHYSICALDRIVE0 is not a NETINT NVMe device
Identity information retrieved from the device at port \\.\PHYSICALDRIVE1
    VID:           0x1d82
    SSVID:         0x1d82
    Device Model:  T408-U2
    Firmware Rev:  310X2013
    Serial Number: TU05-08-02-C10-0972
NETINT T408-U2 NVMe video transcoder identified at port \\.\PHYSICALDRIVE1

Identity information retrieved from the device at port \\.\PHYSICALDRIVE2
    VID:           0x0
    SSVID:         0x0
Device at port \\.\PHYSICALDRIVE2 is not a NETINT NVMe device
Total Number of NETINT NVMe Transcoders indentified: 1

Mon Mar 27 10:46:54 2023 up 00:00:00 v320R2013
Num decoders: 1
BEST INDEX LOAD MODEL_LOAD MEM  INST DEVICE         NAMESPACE
L    0     0    0          0    0    \\.\PHYSICALDRIVE1 \\.\PHYSICALDRIVE1
Num encoders: 1
BEST INDEX LOAD MODEL_LOAD MEM  INST DEVICE         NAMESPACE
L    0     0    0          0    0    \\.\PHYSICALDRIVE1 \\.\PHYSICALDRIVE1
**************************************************
```

Reporting columns

INDEX **number used by resource manager to identify the resource**

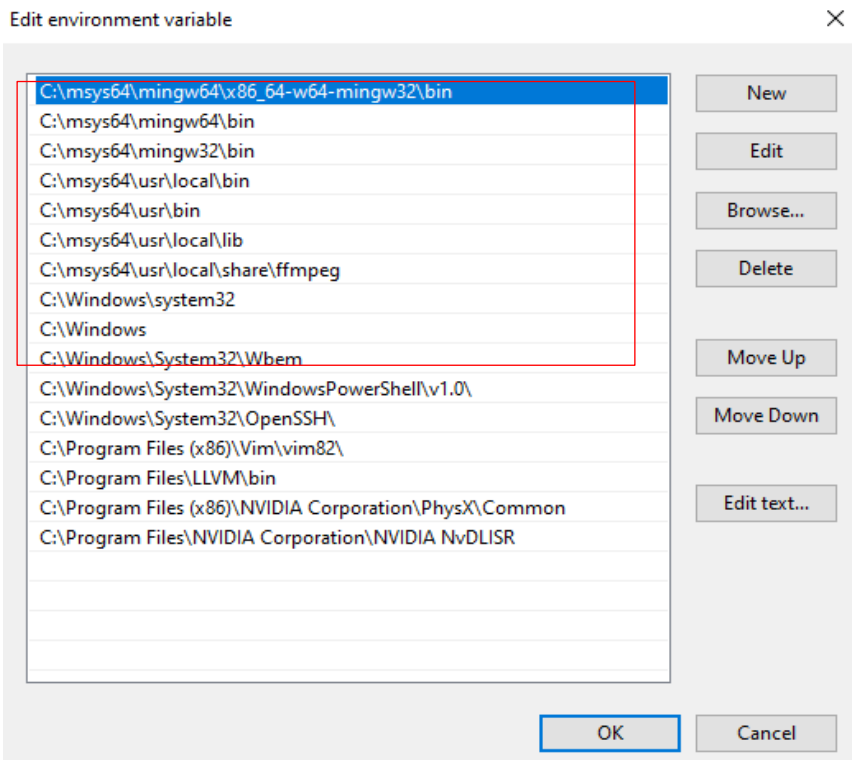
LOAD **realtime load**

MODEL_LOAD **estimated load based on framerate and resolution**
INST **number of job instances**
DEVICE **path to NVMe device file handle**
NAMESPACE **path to NVMe namespace file handle**

8.2.6 Sourcing MSYS2 environment from Windows Environment Variables

This section is optional. Proceed with the step below to add msys paths to the $PATH on windows.

In Windows ‘Control Panel → System → Advanced System Settings → System Properties → Environment Variables… → System Variables → Path → Edit’ you can add new environment variables for C:\msys64 as per sample picture below. Use the ‘Move Up’ button to prioritize the C:\msys64 paths over C:\Windows paths.



This will configure MSYS2 linux type commands from the Windows Command Terminal. Once the above is configured, open a new Windows Command Terminal and test the commands:

which bash
which sh
which ffmpeg.exe
which init_rsrc.exe

Compiling libxcoder and FFmpeg using Visual Studio 2019

This section describes the Windows VS2019 environment setup and steps to compile libxcoder and FFmpeg using VS2019.

8.2.7 Platform and Configuration

8.2.7.1 Configuration of VS2019 project

Configuration	**Debug**	**Release**	**DebugDLL**	**ReleaseDLL**
Compiled result	To generate LIB or EXE without optimization	To generate LIB or EXE with optimization	To generate DLL or EXE without optimization	To generate DLL or EXE with optimization

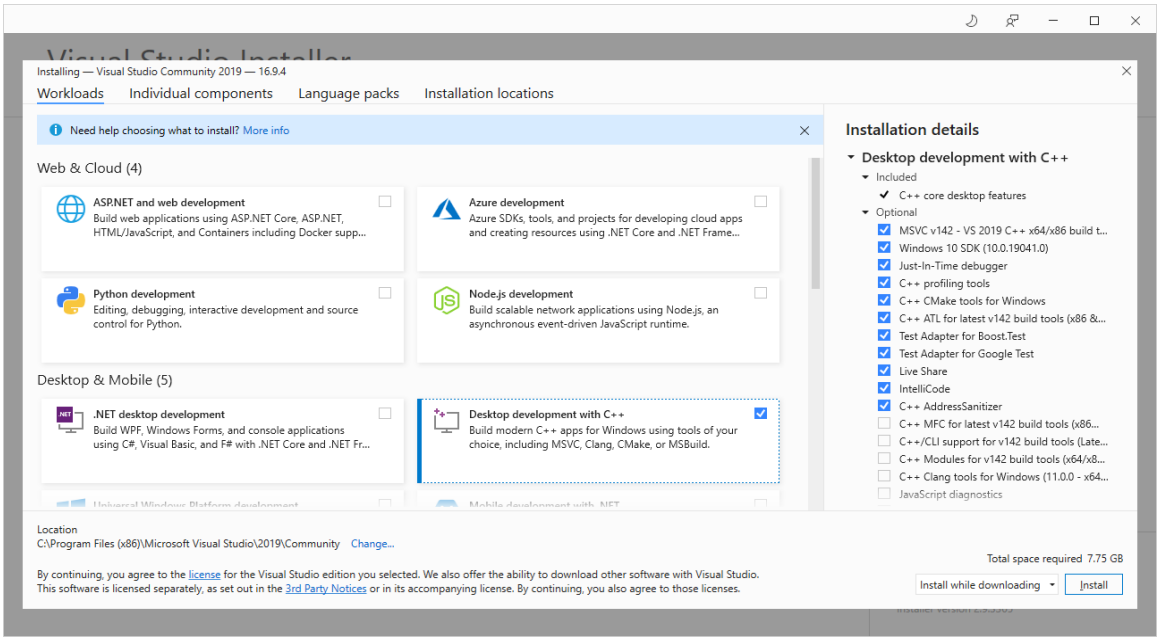
8.2.7.2 Supported Platform and Configuration

Object:	**Configuration**	**Debug**		**Release**		**DebugDLL**		**ReleaseDLL**	
	Platform	**X86**	**X64**	**X86**	**X64**	**X86**	**X64**	**X86**	**X64**
libxcoder		✓	✓	✓	✓	✓	✓	✓	✓
ffmpeg					✓				✓

8.2.8 Setup the VS2019 environment

8.2.8.1 Setup the VS2019 with on-line

1. Download the VS2019 from https://visualstudio.microsoft.com/
2. Install the VS2019. Select “Desktop development with C++” option when installing.
3. Install the “Microsoft Visual Studio Installer Projects” plug-in through “Visual Studio Marketplace”.



8.2.8.2 Setup the VS2019 with off-line

Some customers may have an off-line environment, so provide a method to build a compilation environment in an offline environment.

1. We provide an English and a Chinese VS2019 off-line package. Get the VS2019 off-line installation package from NETINT. To request this please let us know.
2. Unzip the VS2019 package, run the *vs_setup.exe*. Select the default option.
3. We also provide the installer projects plug-in package. Get the *InstallerProjects.vsix* file and run it.

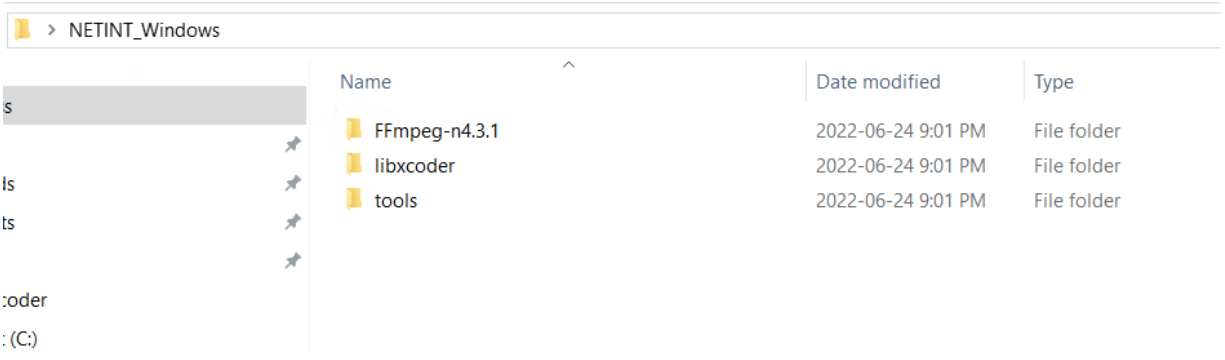
8.2.9 Compile the Libxcoder and FFmpeg for Quadra/Logan Project

8.2.9.1 Preparation

Since we need to build those two separate solutions at one time, a batch script referencing devenv.exe rather than IDE is more proper for this requirement. Of course, you can run the sln files with IDE and copy the intermediate and output files according to the content of batch script by hand though.

Let's take compilation of FFmpeg v4.3.1 for example. Before the job begins:

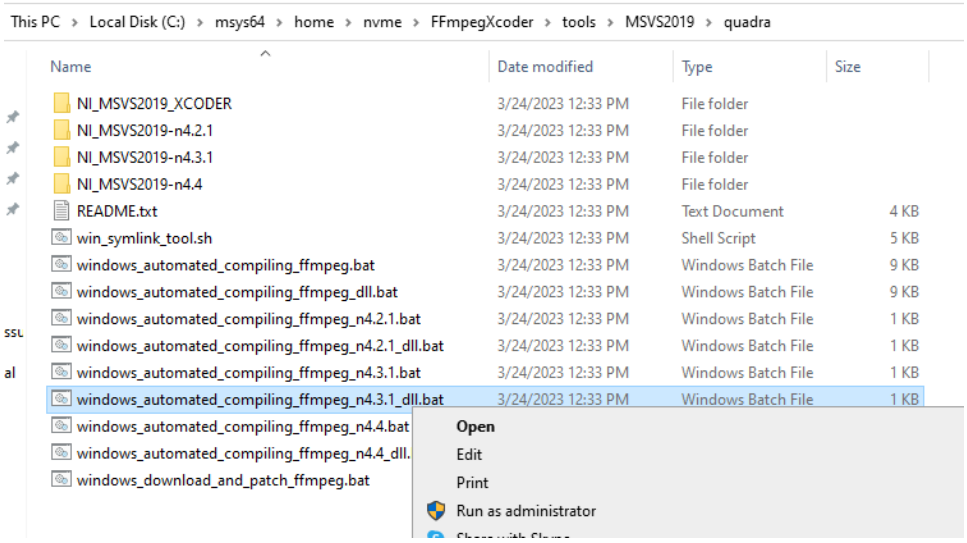
- copy these two directories: libxcoder and tools (from the release package folder: Quadra_SW_v3.0.0) into a folder (in this example, the folder is named NETINT_Windows).
- Clone FFmpeg-n4.3.1 from git using command for example**: git clone -b n4.3.1 --depth=1 https://github.com/FFmpeg/FFmpeg.git FFmpeg-n4.3.1**
- Copy the patch file: FFmpeg-n4.3.1_netint_v3.1.1_EN3.diff from Quadra_SW_V3.1.1_EN3 to the FFmpeg-n4.3.1 folder, which is created from the previous step
- Change directory into FFmpeg-n4.3.1 and apply the patch**: patch -p 1 -t -i FFmpeg-n4.3.1_netint_v3.1.1_EN3.diff**



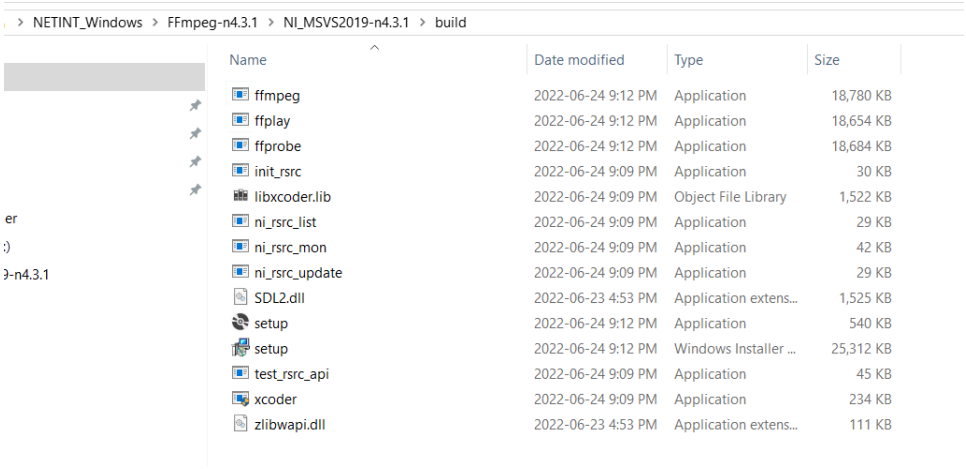
8.2.9.2 Compile libxcoder and FFmpeg at one time for Quadra Release or ReleaseDLL build

The complete process is quite simple that only three steps are needed to be taken.

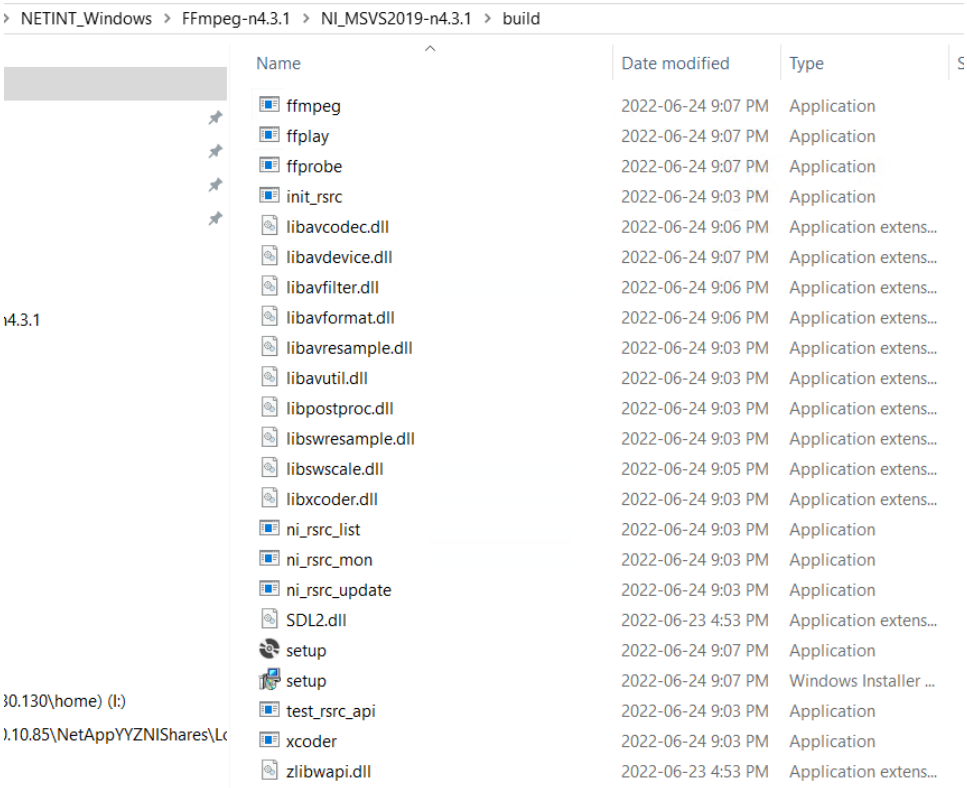
1. Change directory into tools\MSVS2019\quadra.
2. Run the batch file to perform the libxcoder and FFmpeg Release or ReleaseDLL build. windows_automated_compiling_ffmpeg_n4.3.1.bat as admin → for Release build
   windows_automated_compiling_ffmpeg_n4.3.1_dll.bat as admin → for ReleaseDLL build
3. **Note** – Ensure the VS2019 --> Extensions --> Manage Extensions --> Microsoft Visual Studio Installer Projects , extension is installed on host prior to running the batch files.



4. If the user runs the Release batch files, the output build executable and setup files can be found in the build directory.

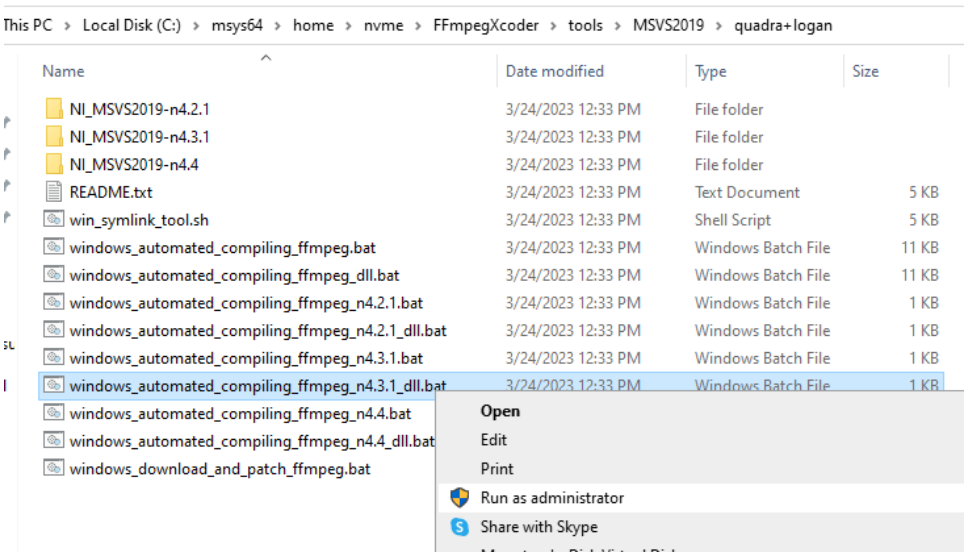


5. If the user runs the ReleaseDLL batch files, the contents of the build folder will also have additional DLL files, as in the example below.

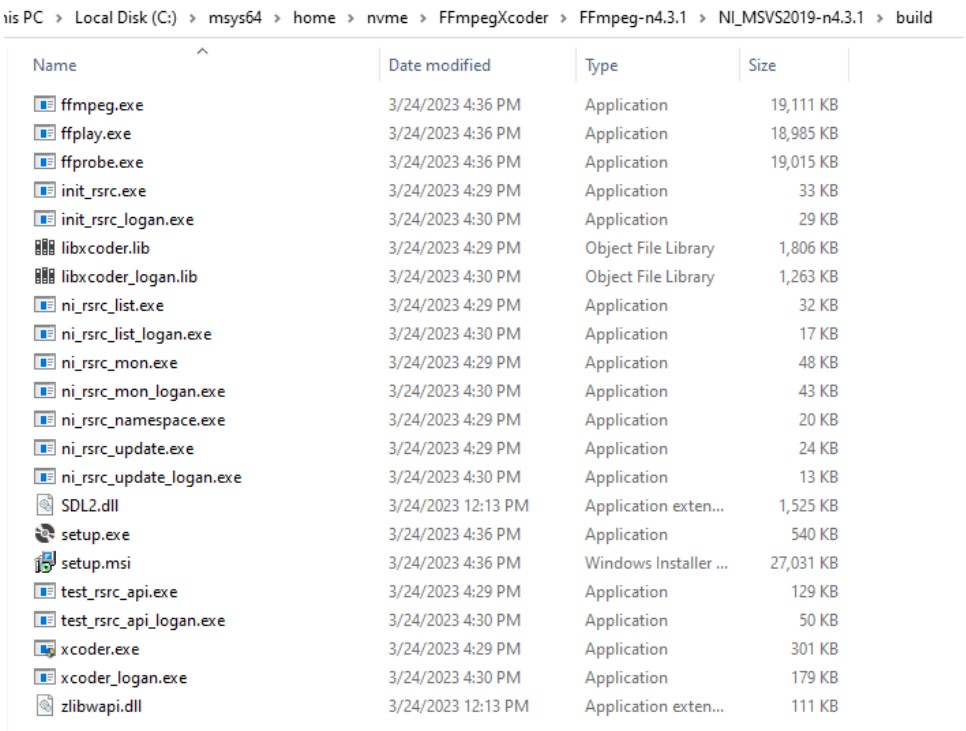


8.2.9.3 Compile Libxcoder and FFmpeg at one time for Quadra+Logan Release or ReleaseDLL build

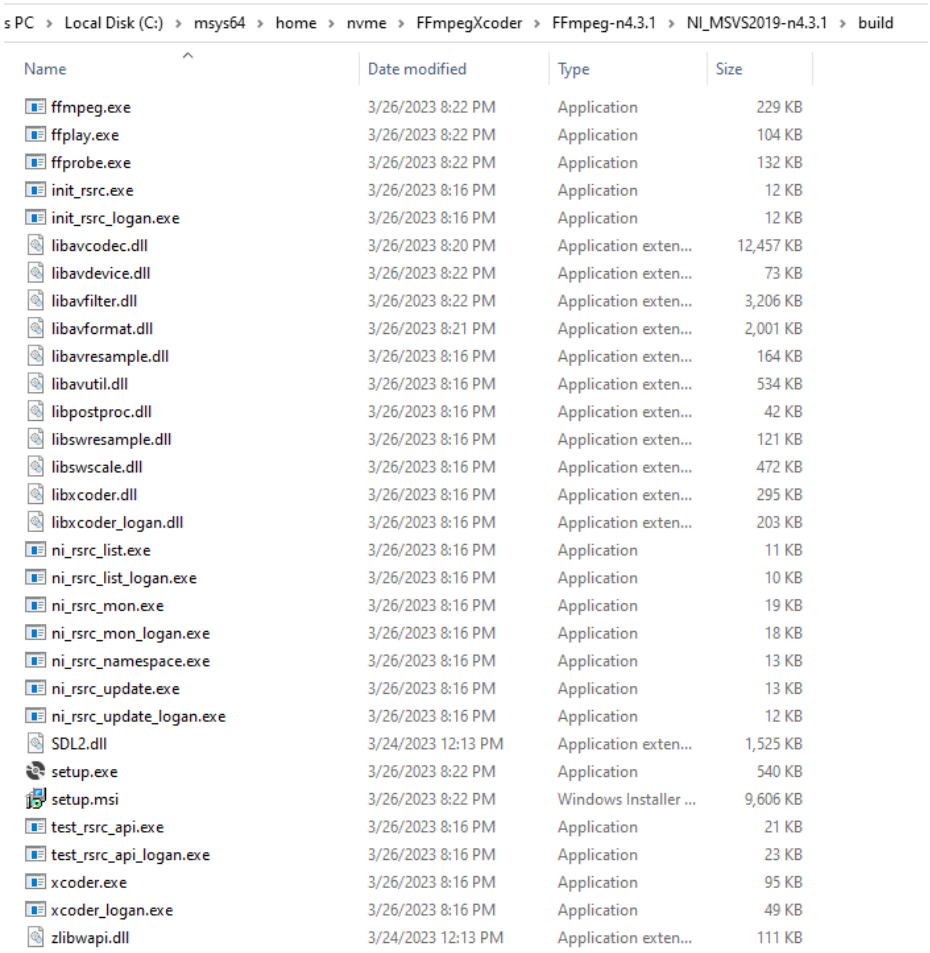
1. Change directory into tools\MSVS2019\quadra+logan\.
2. Run the batch file to perform the libxcoder, libxcoder_logan and FFmpeg Release or ReleaseDLL build. windows_automated_compiling_ffmpeg_n4.3.1.bat as admin → for Release build
   windows_automated_compiling_ffmpeg_n4.3.1_dll.bat as admin → for ReleaseDLL build
3. **Note** – Ensure the VS2019 --> Extensions --> Manage Extensions --> Microsoft Visual Studio Installer Projects, extension is installed on host prior to running the batch files.



4. If the user runs the Release batch files, the output build executable and setup files can be found in the build directory.



5. If the user runs the ReleaseDLL batch files, the contents of the build folder will also have additional DLL files, as in the example below.

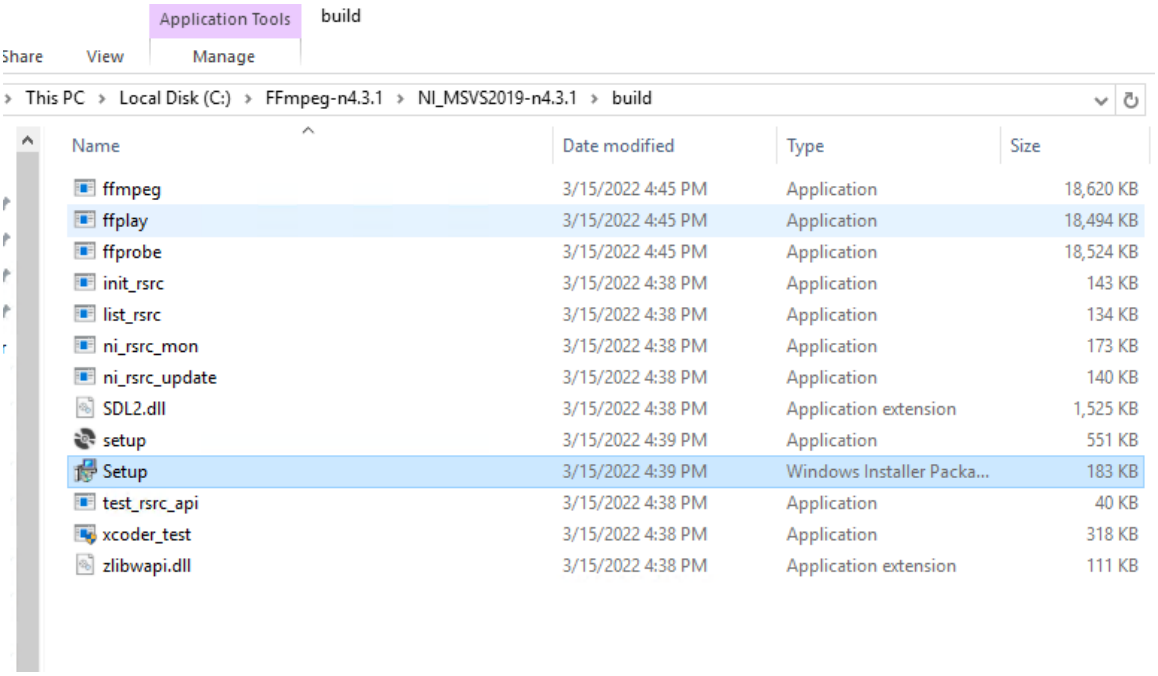


8.2.10 Install and Run FFmpeg

The installation of FFmpeg will NOT modify any environment variable or registry item of Win10 system but only need the authority as administrator.

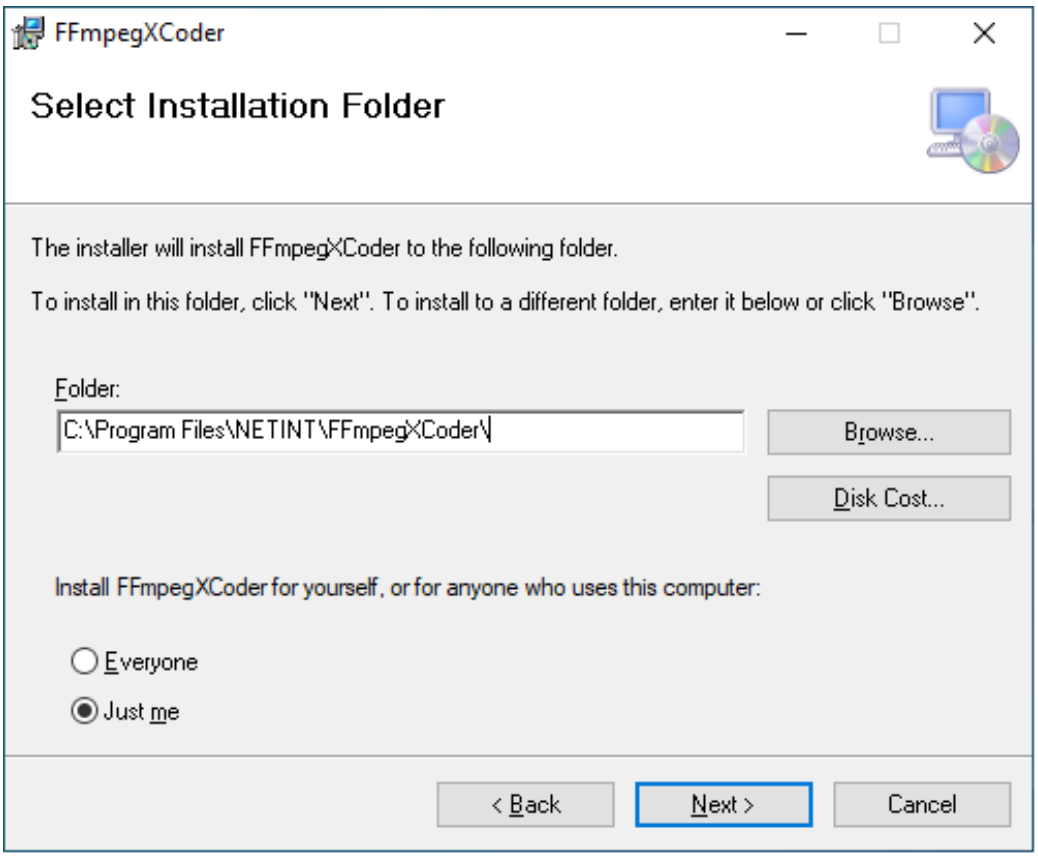
Let’s take FFmpeg-n4.3.1 for example. Assume that the build directory is C:\FFmpeg-n4.3.1\NI_MSVS2019-n4.3.1\build and we have got all the output files there.

8.2.10.1 Step 1: Click the setup.msi generated in build directory

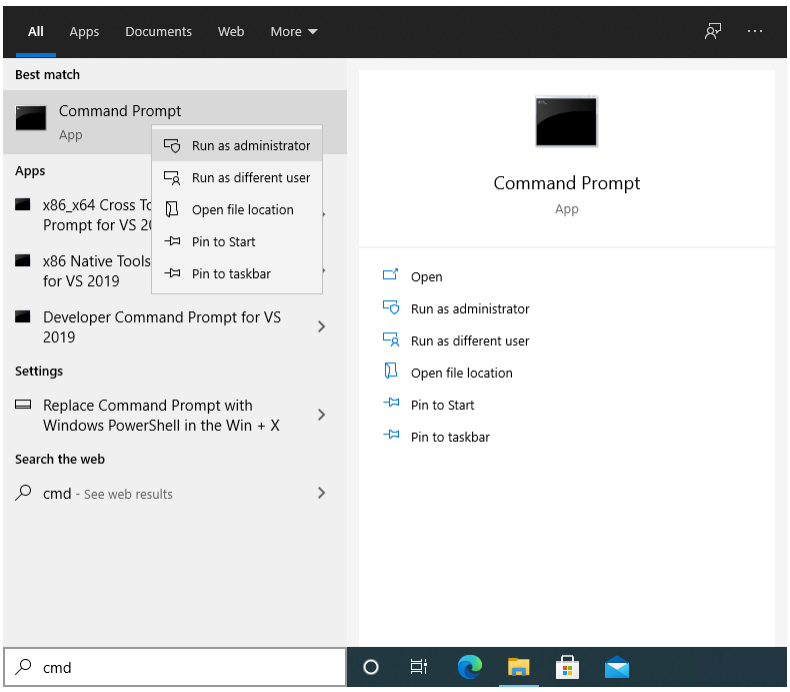


8.2.10.2 Step 2: Choose your own installation path as administrator

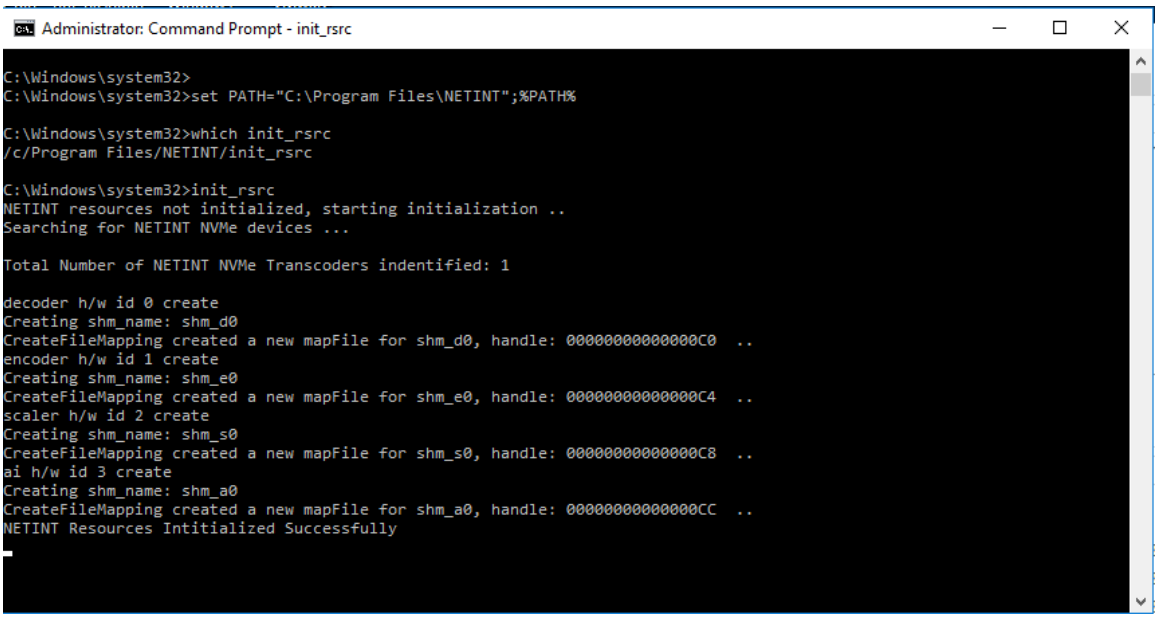
For example, C:\Program Files\NETINT



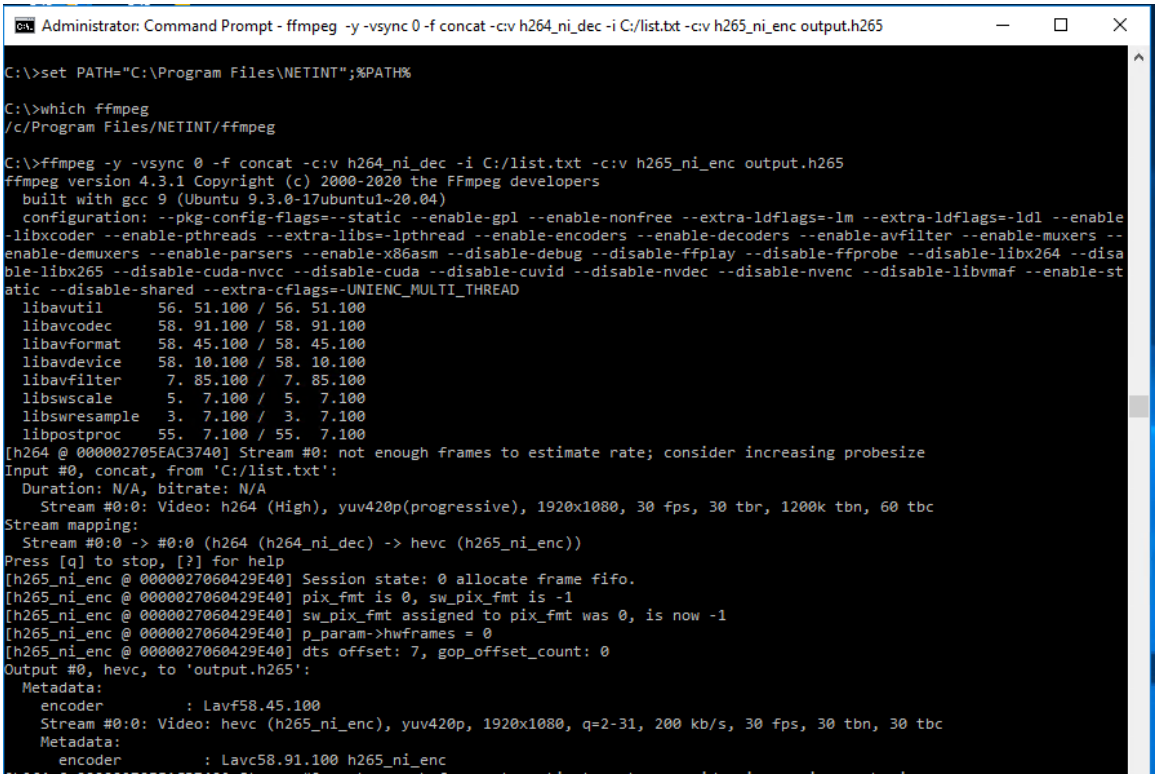
8.2.10.3 Step 3: Run one command prompt as administrator



8.2.10.4 Step 4: Add installation path into %PATH% variable and run init_rsrc command



8.2.10.5 Step 5: Start another command prompt and run your own ffmpeg command



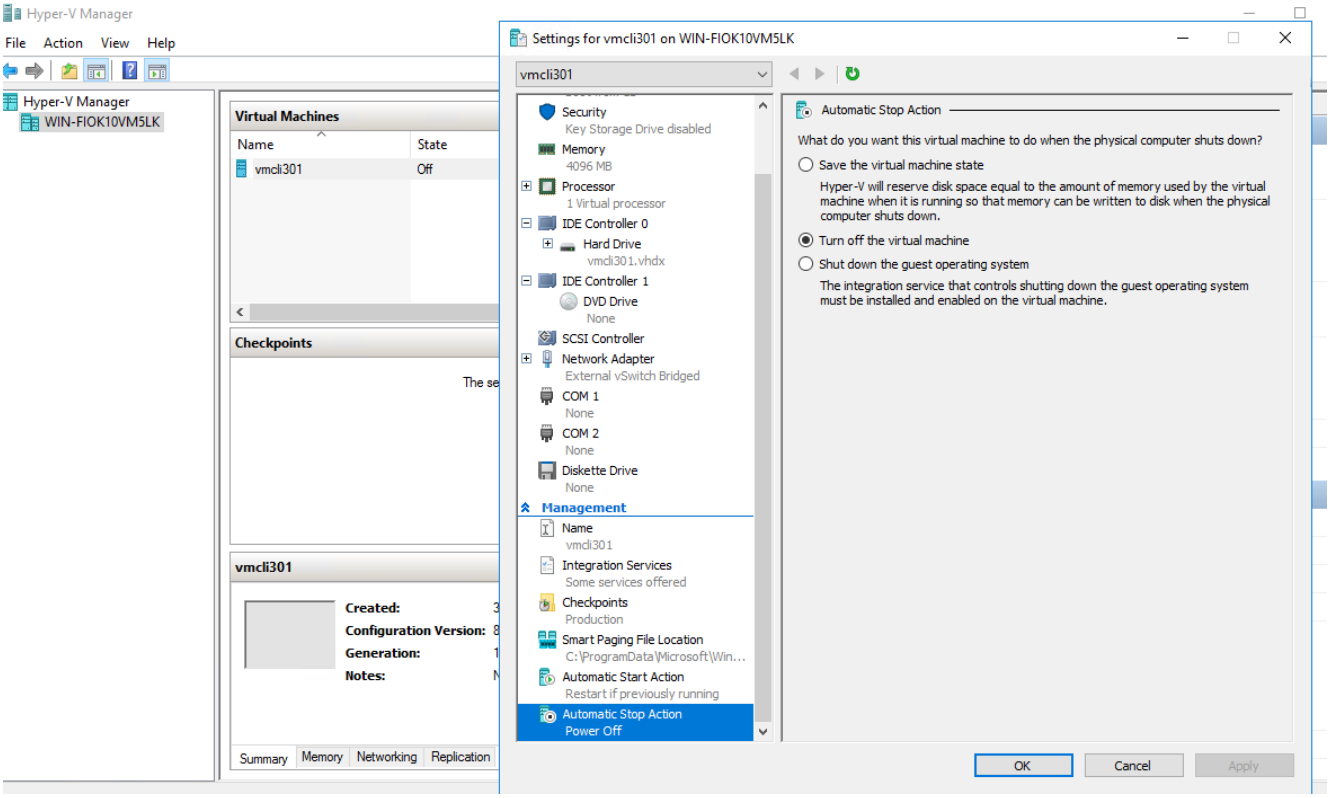
9 Windows Host Hyper-V VM

This section details the installation and usage of the NETINT video transcoder device in a Windows Hyper-V VM.

9.1 Hyper-V Linux Guest VM

On Windows host, search for 'Turn Windows Features On or Off' and ensure Hyper-V is set to on, and click OK. This will trigger an installation of Hyper-V. Once installation is complete, reboot the host.

Installation and configuration of a Windows hyper-v VM is not in the scope of this document. Follow the links in Section 8.1.7 for online material to install a hyper-V, and a VM of your choice. Ensure the VM can be started and shutdown and has a working network connection.

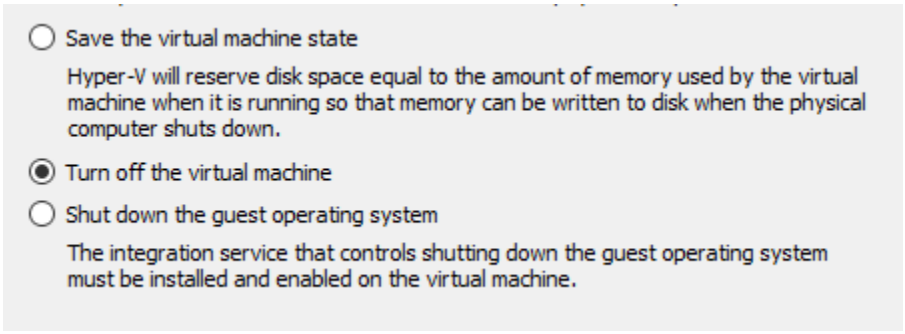


For scope of this document:

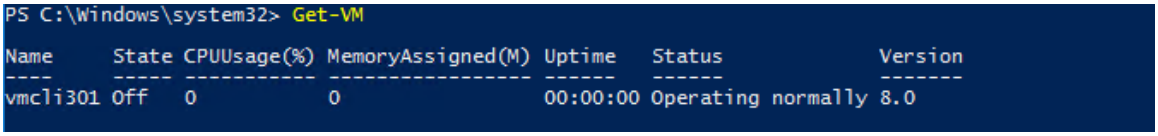
- VM is a Ubuntu 20.04 desktop version
- VM name is 'vmcli301'

9.1.1 Additional settings on VM

1. Under VM Settings, the ‘Automatic Stop Action’ must be set to ‘Turn off the Virtual Machine’.



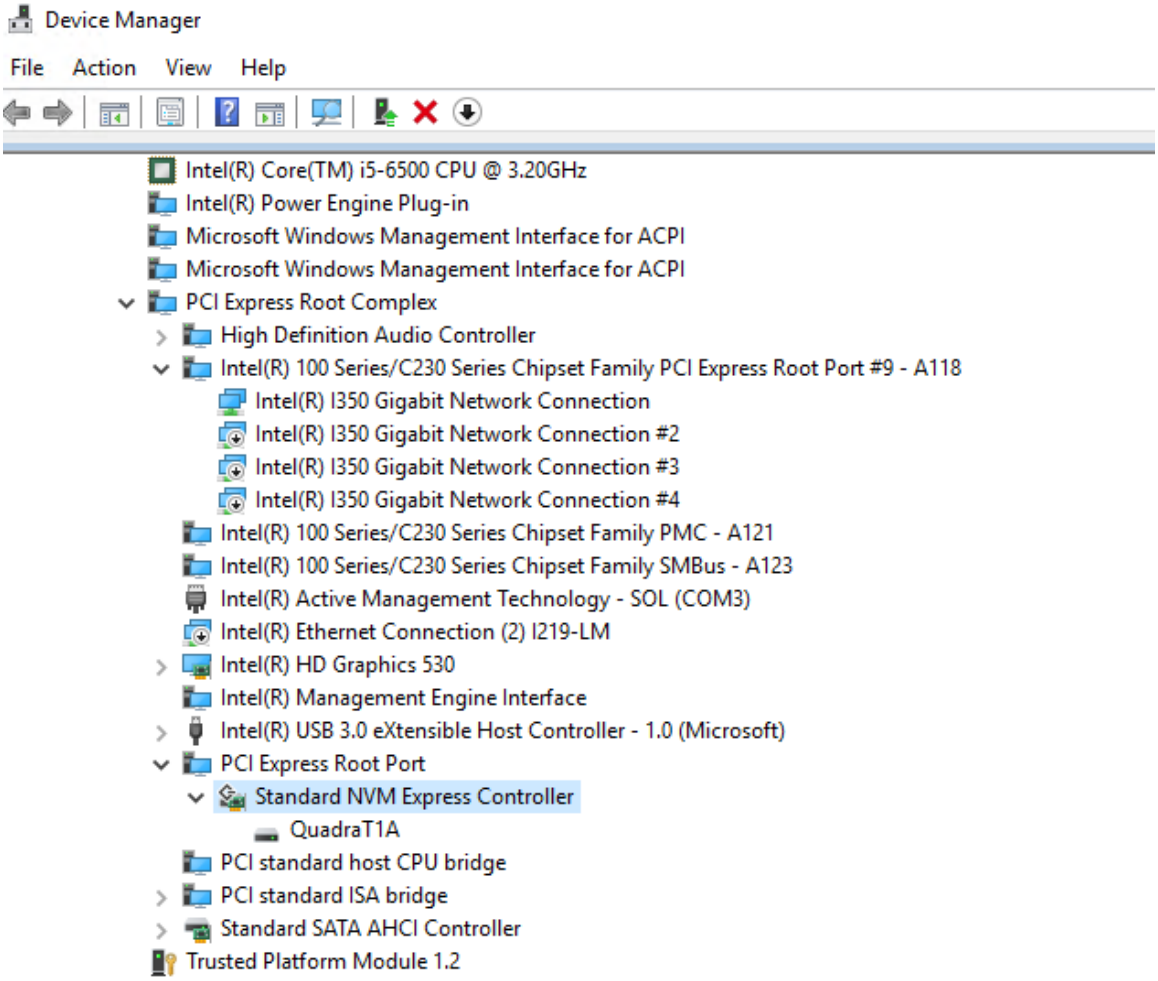
2. In VM settings , confirm the VM state is at ‘Off’.

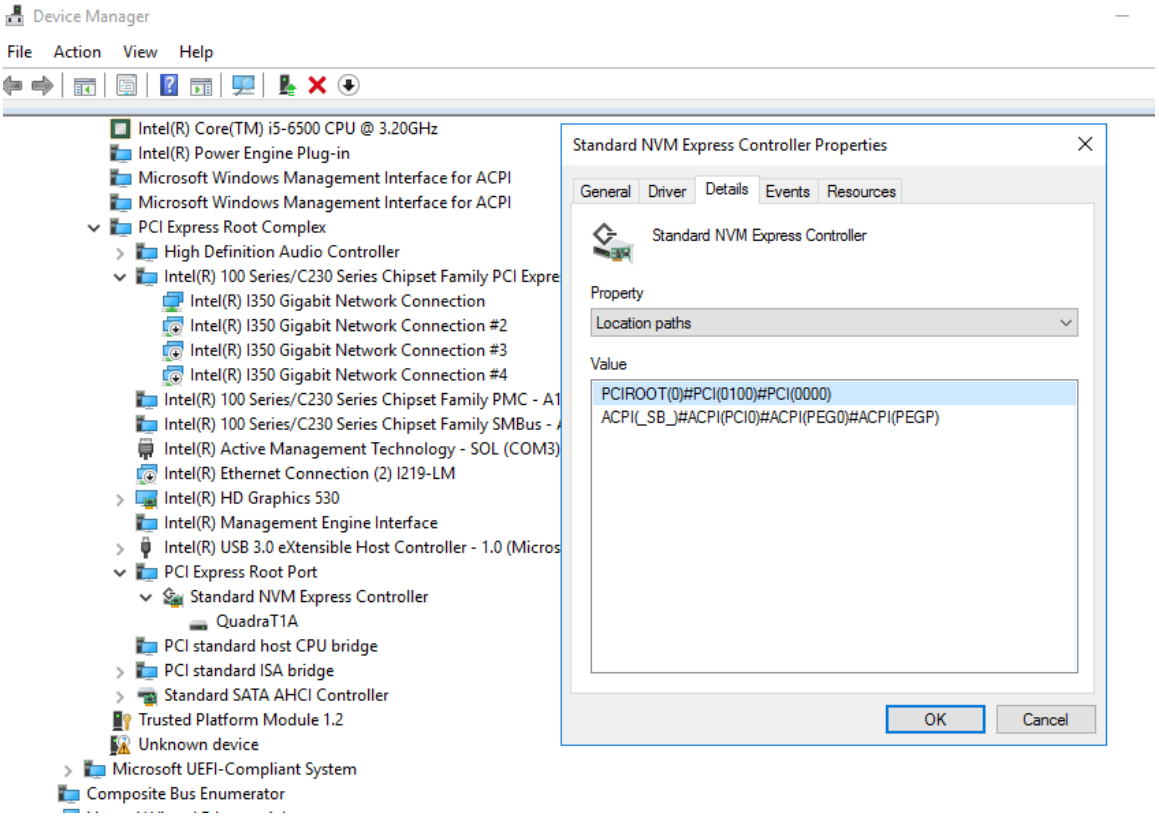


9.1.2 NVMe device Location Path

Each device on the host is assigned a ‘Location Path’ upon host boot up.

To detect the Location Path for the NVMe device, launch Device Manager and select View → Devices by connection.





In this case, the Location Path for the ‘Standard NVM Express Controller’ is `PCIROOT(0)#PCI(0100)#PCI(0000)`.

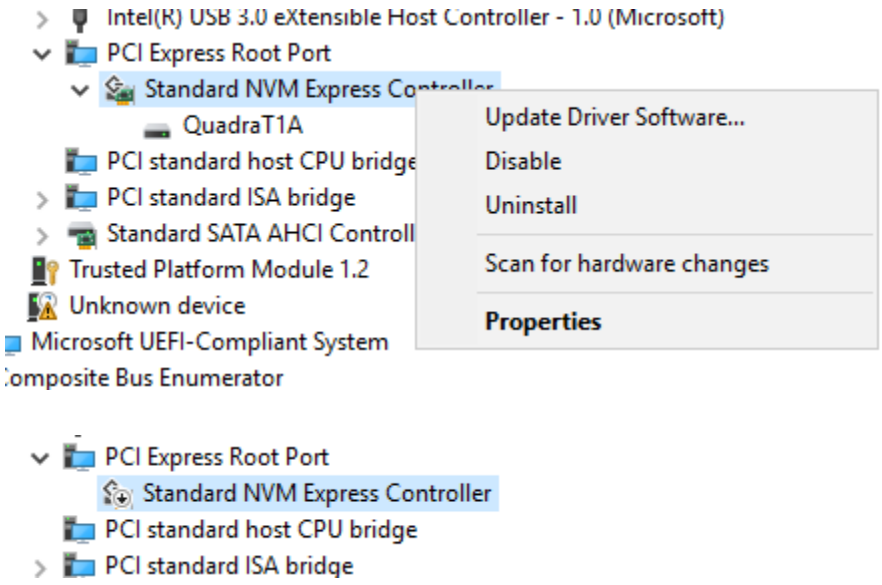
9.1.3 Disable the NVMe device

Check the Quadra NVMe device is installed on the Host by running the following command:

```
wmic diskdrive get Name,Model,SerialNumber,FirmwareRevision
```

```
C:\Windows\system32>wmic diskdrive get Name,Model,SerialNumber,FirmwareRevision
FirmwareRevision  Model               Name               SerialNumber
---41DEV          QuadraT1A           \\.\PHYSICALDRIVE1  Q1A10BA11FC060-0010_00000001.
CC61              ST1000DM003-1ER162  \\.\PHYSICALDRIVE0  Z4Y5F07G
```

In Windows ‘Device Manager’ click view a Devices by connection, disable the ‘Standard NVM Express Controller’. It may be found under ‘PCI Express Root Complex --> PCI Express Root Port’ --> ‘Standard NVM Express Controller’. Right click on ‘Standard NVM Express Controller’ to see the menu selection and select ‘Disable’.



The NVMe device will now not show up in wmic output.

```
C:\Windows\system32>wmic diskdrive get Name,Model,SerialNumber,FirmwareRevision
FirmwareRevision  Model               Name               SerialNumber
CC61              ST1000DM003-1ER162  \\.\PHYSICALDRIVE0  Z4Y5F07G
```

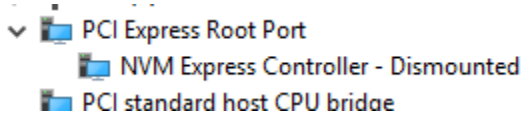
9.1.4 Passthrough physical device to VM

Launch Windows Powershell in administrator mode.

Run the following commands to dismount the NVMe device from the host and attaching the device to the VM. Ensure to use the 'Location Path' for your NVMe device as collected in Section 8.1.2.

```
Dismount-VMHostAssignableDevice -LocationPath
"PCIROOT(0)#PCI(0100)#PCI(0000)"
```

Confirm the Windows Device Manager shows the NVMe device as dismounted.



Assign MMIO space.

```
Set-VM -LowMemoryMappedIoSpace 1Gb -VMName vmcli301
Set-VM -HighMemoryMappedIoSpace 2Gb -VMName vmcli301
```

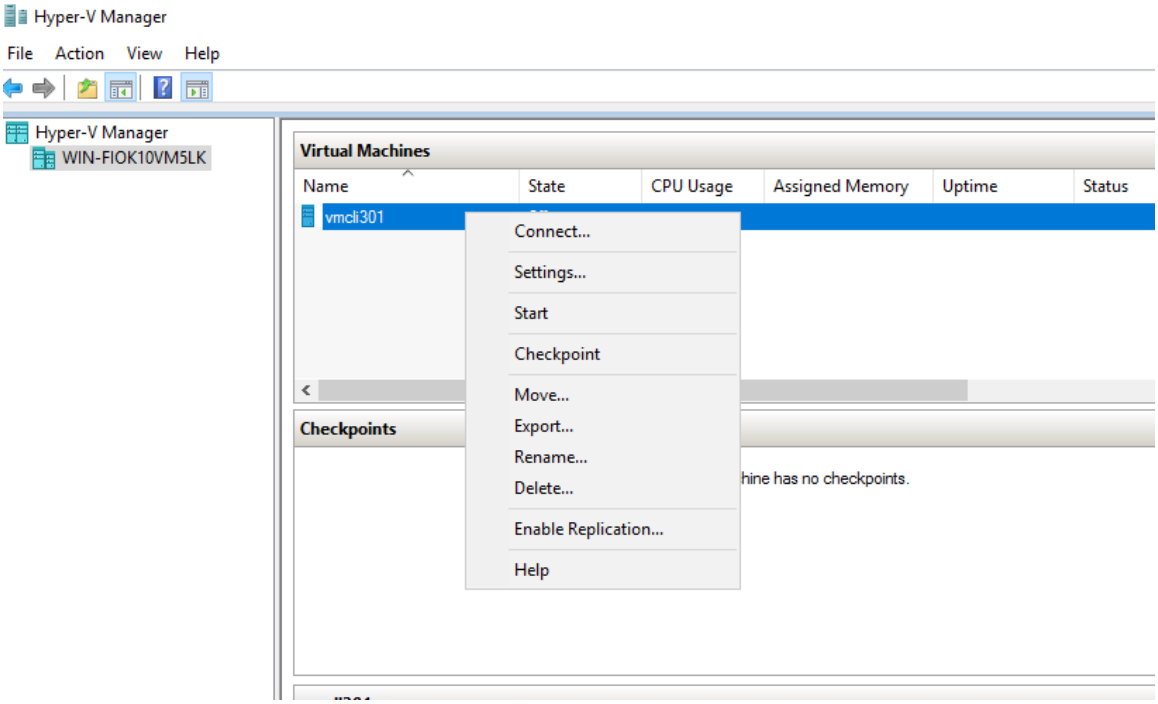
Attach NVMe device to VM.

```
Add-VMAssignableDevice -LocationPath
"PCIROOT(0)#PCI(0100)#PCI(0000)" -VMName vmcli301
Get-VMAssignableDevice
```

Example:

```
PS C:\Windows\system32> Get-VMAssignableDevice

cmdlet Get-VMAssignableDevice at command pipeline position 1
Supply values for the following parameters:
VMName[0]: vmcli301
VMName[1]:

InstanceID       :
PCIP\VEN_1D82&DEV_0401&SUBSYS_04011D82&REV_00\4&1F0B2711&0&0008
LocationPath     : PCIROOT(0)#PCI(0100)#PCI(0000)
ResourcePoolName : Primordial
Name             : Virtual PCI Express Port Settings
Id               : Microsoft:132F09C1-A7E3-44CD-B619-
A218F8524B6F\46FF7B1F-9A20-48E9-8679-8D202A8FC245
VMId             : 132f09c1-a7e3-44cd-b619-a218f8524b6f
VMName           : vmcli301
VMSnapshotId     : 00000000-0000-0000-0000-000000000000
VMSnapshotName   :
CimSession       : CimSession: .
ComputerName     : WIN-FIOK10VM5LK
IsDeleted        : False
VMCheckpointId   : 00000000-0000-0000-0000-000000000000
VMCheckpointName :
```

Start the VM, by clicking on Hyper-V Manager → Virtual Machines → vmcli301 → Start.



Run the following command to get status of VM.

```
get-VM
```

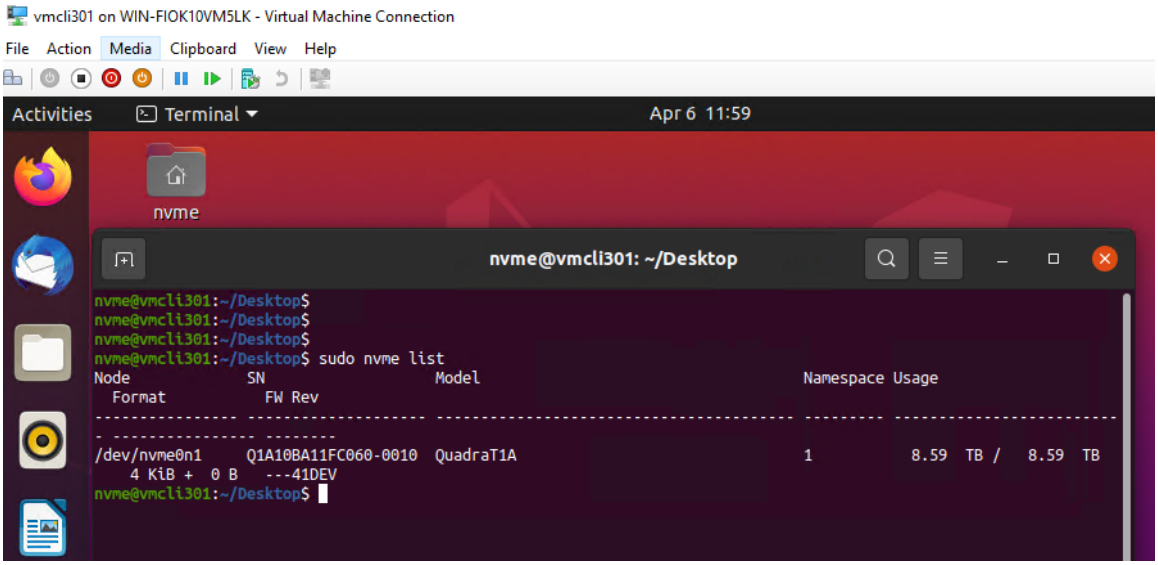
```
PS C:\Windows\system32> Get-VM

Name     State   CPUUsage(%) MemoryAssigned(M) Uptime           Status             Version
----     -----   ----------- ----------------- ------           ------             -------
vmcli301 Running 0           4096              00:32:45.4440000 Operating normally 8.0
```

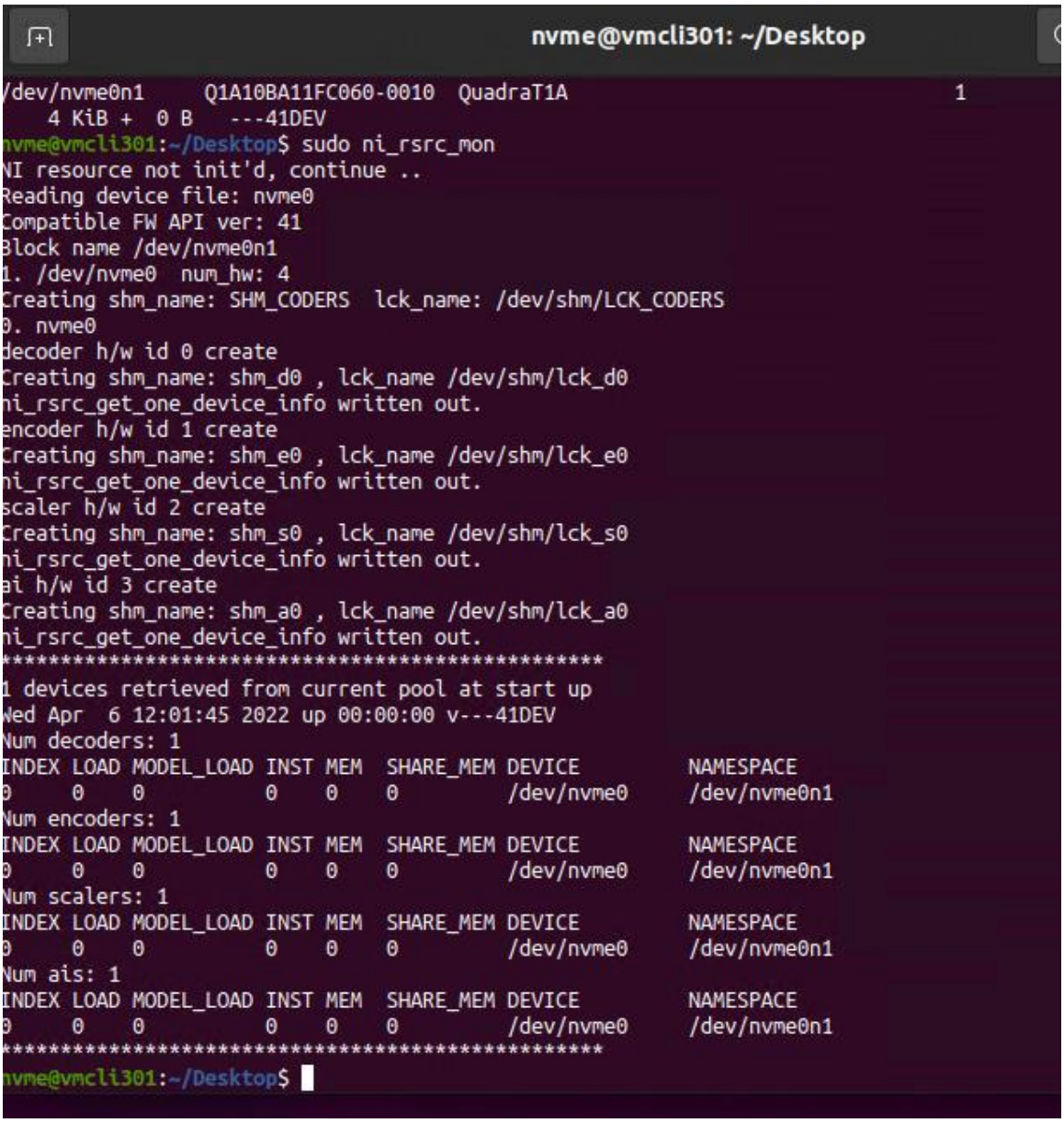
Connect to VM, to launch the VM desktop view.

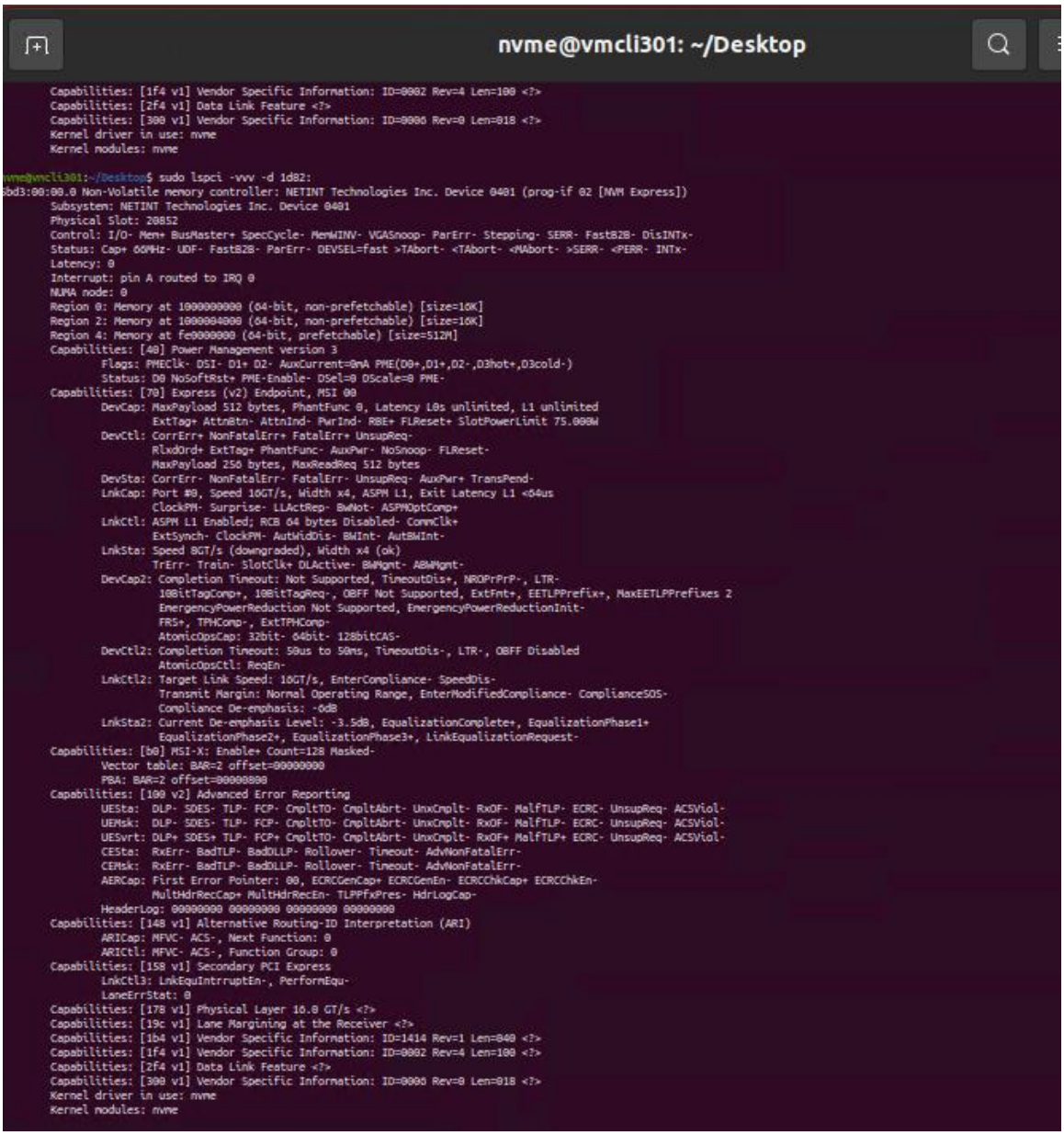
Follow the QuickStartGuideQuadra_V3 or newer, to install the Linux and FFmpeg environment to VM to help see the Quadra device and perform transcoding operations.

Confirm the NVMe device is mounted and accessible to the VM by running the following commands :

```
sudo nvme list
sudo ni_rsrc_mon
sudo lspci -vvv -d 1d82:
```




nvme@vmcli301: ~/Desktop
/dev/nvme0n1 Q1A10BA11FC060-0010 QuadraT1A 1
4 KiB + 0 B ---41DEV
nvme@vmcli301:~/Desktop$ sudo ni_rsrc_mon
NI resource not init'd, continue ..
Reading device file: nvme0
Compatible FW API ver: 41
Block name /dev/nvme0n1
1. /dev/nvme0 num_hw: 4
Creating shm_name: SHM_CODERS lck_name: /dev/shm/LCK_CODERS
0. nvme0
decoder h/w id 0 create
Creating shm_name: shm_d0 , lck_name /dev/shm/lck_d0
ni_rsrc_get_one_device_info written out.
encoder h/w id 1 create
Creating shm_name: shm_e0 , lck_name /dev/shm/lck_e0
ni_rsrc_get_one_device_info written out.
scaler h/w id 2 create
Creating shm_name: shm_s0 , lck_name /dev/shm/lck_s0
ni_rsrc_get_one_device_info written out.
ai h/w id 3 create
Creating shm_name: shm_a0 , lck_name /dev/shm/lck_a0
ni_rsrc_get_one_device_info written out.
****
1 devices retrieved from current pool at start up
Wed Apr 6 12:01:45 2022 up 00:00:00 v---41DEV
Num decoders: 1
INDEX LOAD MODEL_LOAD INST MEM SHARE_MEM DEVICE NAMESPACE
0 0 0 0 0 0 /dev/nvme0 /dev/nvme0n1
Num encoders: 1
INDEX LOAD MODEL_LOAD INST MEM SHARE_MEM DEVICE NAMESPACE
0 0 0 0 0 0 /dev/nvme0 /dev/nvme0n1
Num scalers: 1
INDEX LOAD MODEL_LOAD INST MEM SHARE_MEM DEVICE NAMESPACE
0 0 0 0 0 0 /dev/nvme0 /dev/nvme0n1
Num ais: 1
INDEX LOAD MODEL_LOAD INST MEM SHARE_MEM DEVICE NAMESPACE
0 0 0 0 0 0 /dev/nvme0 /dev/nvme0n1
****
nvme@vmcli301:~/Desktop$



The Quadra NVMe device is now ready for use in the hyper-v VM.

9.1.5 Passthrough virtual device to VM

This section requires firmware support for SR-IOV on the NETINT video transcoder device, this was introduced since release 4.1.1.

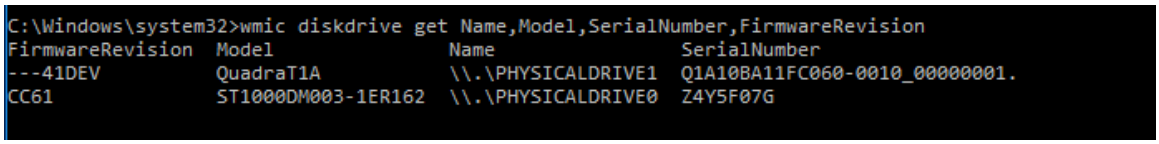
9.1.6 Restoring the NVMe device back to the host

1. Launch Windows Powershell in adminstrator mode. Run the following commands to remove the NVMe device from the hyper-v VM and mounting it back to the host. Replace Location Path and VM name values in the commands to values specific to your host and VM.

```
Remove-VMAssignableDevice -LocationPath
"PCIROOT(0)#PCI(0100)#PCI(0000)" -VMName vmcli301
Mount-VMHostAssignableDevice -LocationPath
"PCIROOT(0)#PCI(0100)#PCI(0000)"
```

2. In Windows ‘Device Manager’ à view à Devices by connection, ensable the ‘Standard NVM Express Controller’. It may be found under ‘PCI Express Root Complex à PCI Express Root Port’ à ‘Standard NVM Express Controller’. Right click on ‘Standard NVM Express Controller’ to see the menu selection and select ‘Enable’.
3. Check the Quadra NVMe device is available on the Host by running the following command.

```
wmic diskdrive get Name,Model,SerialNumber,FirmwareRevision
```



9.1.7 Useful links and References

1. https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/quick-start/quick-create-virtual-machine
2. https://ubuntu.com/download/desktop
3. https://docs.microsoft.com/en-us/windows-server/virtualization/hyper-v/deploy/deploying-graphics-devices-using-dda#:~:text=Starting%20with%20Windows%20Server%202016,leverage%20the%20devices%20native%20drivers
4. https://docs.microsoft.com/en-us/windows-server/virtualization/hyper-v/deploy/deploying-graphics-devices-using-dda#:~:text=Starting%20with%20Windows%20Server%202016,leverage%20the%20devices%20native%20drivers
5. https://www.techtarget.com/searchvirtualdesktop/tip/Running-GPU-passthrough-for-a-virtual-desktop-with-Hyper-V
6. https://www.zdnet.com/article/windows-10-tip-find-out-if-your-pc-can-run-hyper-v/

10 Android Emulator

This section details the installation and usage of NETINT video transcoder device on an Android emulator.

10.1 Scripted build of Android Emulator and Netint SW on Linux host

The Android quick installer script can be used to install the Android emulator environment, then compile Netint libxcoder & FFmpeg using the Android NDK to run them in an Android emulator environment.

10.1.1 Prepare Installation Files

The installation scripts is located in the Android_quick_installer folder at the top level of the Netint release package. Or, it can be found in the tools/Android_quick_installer folder of the Netint SW release package folder (eg. Quadra_SW_V*.*.*_RC*/).

1. Move the contents of the Android_quick_installer folder to where you want Android development folders installed. Note, the development folders can be moved after installation too.
2. Move the Quadra_SW_V*.*.*_RC*/ folder to be with the contents of the Android_quick_installer folder.
3. Optionally, move the Quadra_FW_V*.*.*_RC*/ folder to be with the contents of the Android_quick_installer folder if you wish to use the quadra_android_quick_installer.sh to upgrade FW.
4. Make sure that the virtualization option (vt-x, vt-d) is enabled in BIOS before building android emulator environment.

10.1.2 Script usage help

Make sure that all files/folders required for installation are in the same directory:

- quadra_android_quick_installer.sh
- adb_push.sh
- build_all_android.sh
- init_android_rsrc_mon.sh
- run_emulator.sh
- patch/
- Quadra_SW_V*.*.*_RC*/

The following command may be used to see help text for the script:
bash quadra_android_quick_installer.sh -h

```
nvme@cli433:Android_quick_installer$ ./quadra_android_quick_installer.sh -h
This script performs installation of Android emulator environment with Netint
Quadra SW on Ubuntu host. Before use, please go to the BIOS and enable
virtualization options (eg. vt-x, vt-d).

For further installation help see README file

Install environment sequence:
Step 1:  Select usage of '64 bit' or '32 bit' Android.
Step 2:  Select option to 'Setup Environment variables'.
Step 3:  Select option to 'Install prerequisite Linux packages (Ubuntu)'.
Step 4:  Select option to 'Install nvme-cli'.
Step 5:  Select option to 'Download Android NDK-r21d and SDK-r24.4.1'.
Step 6:  Select option to 'Install Libxcoder to Linux host for build test
         (Optional)' if desiring to test libxcoder can install on host directly.
Step 7:  Select option to 'Install FFmpeg-n4.3 to Linux host for build test
         (Optional)' if desiring to test FFmpeg can install on host directly.
Step 8:  Select option to perform 'Firmware Update (Optional)' if desiring to
         upgrade firmware.
Step 9:  Select option to 'Download and build Android'. Download requires 60GB
         disk space, compile requires another 160GB of disk space.
Step 10:  Select option to 'Setup VFIO'. Host needs to be rebooted after this
         step. Continue steps after reboot.
Step 11: Select option to 'Check VFIO' is setup successfully.
Step 12: Select option to 'Build libxcoder and FFmpeg-n4.3 on Android emulator'.
Step 13: If previous step succeeds, the environment has been successfully
         installed. Select option to 'Quit'.
nvme@cli433:Android_quick_installer$
```

10.1.3 Script execution instructions

Start the script with the following command:
bash android_auto_install.sh

Then, follow the guided process. Refer to the steps in the scripts help text pictured above (bash android_auto_install.sh -h).

The main menu of the script is used to run individual steps of the Android environment installation.

```
nvme@cli433:Android_quick_installer$ ./quadra_android_quick_installer.sh
This script performs installation of Android emulator environment with Netint
Quadra SW on Ubuntu host. Before use, please go to the BIOS and enable
virtualization options (eg. vt-x, vt-d).

For further installation help see README file
Please put the Netint FW and SW release package you wish to install in the same
directory as this script.
The latest FW release package found here is:     Quadra_FW_V4.3.0_RC3.tar.gz
The latest FFmpeg release package found here is: Quadra_SW_V4.3.0_RC3.tar.gz
Press [Y/y] to confirm the use of these two release packages. y
Select usage of 64bit or 32bit Android:
1) 64 bit
2) 32 bit
3) Quit
#? 1
Android environment x86_64 (64 bit) selected
Choose an option:
 1) Setup Environment variables
 2) Install prerequisite Linux packages (Ubuntu)
 3) Install nvme-cli
 4) Download Android NDK-r21d and SDK-r24.4.1
 5) Install Libxcoder to Linux host for build test (Optional)
 6) Install FFmpeg-n4.3 to Linux host for build test (Optional)
 7) Firmware Update (Optional)
 8) Download and build Android
 9) Setup VFIO
10) Check VFIO
11) Build libxcoder and FFmpeg-n4.3 on Android emulator
12) Quit
#?
```

Option 1: Setup Environment variables

Set terminal environment variables ($PKG_CONFIG_PATH, $LG_LIBRARY_PATH), sudo $PATH, and ldconfig paths.

Option 2: Install prerequisite Linux packages (Ubuntu)

Use apt-get to install pre-requisite software packages for Ubuntu Linux host: yasm, libssl-dev, m4, libncurses5, zip, git, build-essential, make, cmake, gcc, g++, patch, ncurses-dev, valgrind,pkg-config, python2.7, curl

Option 3: Install NVMe CLI

Install nvme-cli app to aid in NVMe device administration and FW upgrade. Nvme-cli app is not necessary for transcoding.

Option 4: Download Android NDK-r21d and SDK-r24.4.1

Download the necessary Android NDK and SDK packages.

Option 5: Install Libxcoder to Linux host for build test (Optional)

Check prerequisites are installed well on Linux host by building Libxcoder for Linux.

Option 6: Install FFmpeg-n4.3 to Linux host for build test (Optional)

Check prerequisites are installed well on Linux host by building FFmpeg-n4.3 for Linux.

Option 7: Firmware Update (Optional)

Update FW for Quadra cards on system. NVMe-cli required.

Option 8: Download and build Android

Download and build AOSP and Android kernel, then attempt to start Android emulator. Note, this step requires about 60GB of downloads and 220GB of free disk space. During this step there will be the option to select either global (Google) or chinese (Tsinghua university) download source.

Option 9: Setup VFIO

Install virt-manager to configure VFIO. When the VFIO installation is complete, a reboot is required to activate it. If installation is successful, the script will notify user to reboot the host for VFIO activation.

Use Option 12: Quit to leave the install script and sudo reboot now to reboot the host.

Option 10: Check VFIO

Check VFIO successfully activated.

Option 11: Build libxcoder and FFmpeg on Android emulator

This step uses the following scripts so make sure they are in the same directory as the main script:

- build_all_android.sh
  - unzip android-ndk and android-sdk, copy libxcoder+FFmpeg to android_work/ external folder and build them
- adb_push.sh
  - push the files generated by build_all_android.sh to the Android emulator environment
- init_android_rsrc_mon.sh
  - launch the Android emulator environment and verify that the Quadra device is detected
- run_emulator.sh
  - launch the Android emulator environment in the background

If and error occurs during build of libxcoder and FFmpeg on Android emulator, please check log, debug issue, restart the host, and perform this step again.

Option 12: Quit

Exit script.

At this point, the Android environment is installed. Below are the manual installation steps, if there is a problem with the automatic installation script, you can try manual installation.

10.2 Manual build on Linux host with 64 bit or 32 bit Android Emulator

10.2.1 Install the VFIO on the server

Run the following command to install the vfio

```
sudo apt-get install virt-manager python-spice-client-gtk
```

Reboot and go to the BIOS and enable virtualization option (vt-x, vt-d)

After boot up Modify the sudo vim /etc/default/grub

Add below:

```
GRUB_CMDLINE_LINUX_DEFAULT="intel_iommu=on"
GRUB_CMDLINE_LINUX="intel_iommu=on"
sudo update-grub
```

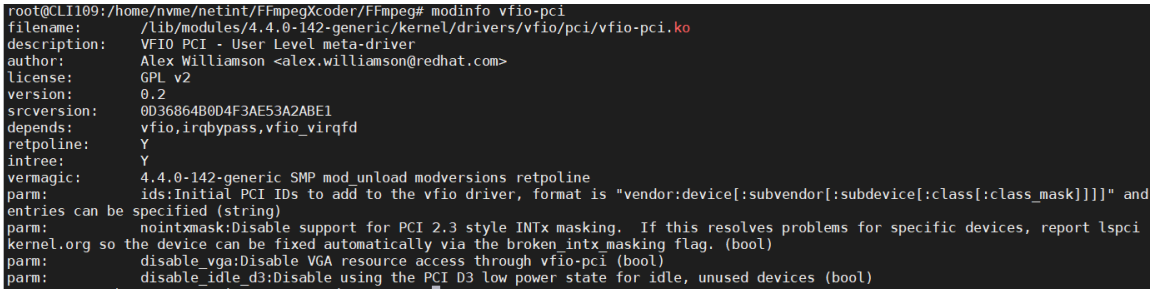
Then reboot again.

10.2.2 Check and insmod the VFIO module on your server

Run the commands below

```
modprobe vfio
modprobe vfio-pci
modinfo vfio-pci
```

If success it will print below info:



10.2.3 VFIO PCI Quadra to KVM

Running “lspci” to identify your Quadra pci address(Ex:0000:01:00.0) and sub-number: (1d82 0401) then run below cmd:

```
echo "0000:01:00.0" > /sys/bus/pci/drivers/nvme/unbind
echo 1d82 0401 > /sys/bus/pci/drivers/vfio-pci/new_id
chown fpga:fpga  /dev/vfio/1
```

Note: replace the “0000:01:00.0” with the actual PCI address that found in the “lspci”, replace fpga:fpga with the current user ID. Change the vfio device number based on the actual number that shows up under /dev/vfio. Different hosts may have different numbers.

Modify /etc/security/limits.conf to add

```
@fpga    hard    memlock 15928456
@fpga    soft    memlock 15928456
```

Note: some hosts require root access to modify limits.conf

Replace @fpga with your user name and replace 15442440 with your system memory size

Reference: https://www.kernel.org/doc/Documentation/vfio.txt

10.2.4 Start Android Emulator using VFIO PCI Quadra card

On the AOSP(android_work or android_work32) folder.

```
source build/envsetup.sh
lunch aosp_x86_64-eng  (for 32bit: lunch aosp_x86-eng)
emulator -memory 4096 -partition-size 4096 -writable-system -
netdelay none -netspeed full -gpu off -show-kernel -no-window -
qemu -enable-kvm -device vfio-pci,host=0000:01:00.0 -kernel
goldfish/arch/x86/boot/bzImage
```

Note: use your own PCI address, 0000:01:00.0 is just an example. Increase the virtual memory and partition size to 8g, “-memory 8192 -partition-size 8192” if you want to generate clips that are bigger than 1.5GB inside the emulator.

10.2.5 Check the card status when Android Emulator boot up

After the emulator boots up, open another terminal and run:

```
source build/envsetup.sh
```

lunch aosp_x86_64-eng .For 32bit lunch aosp_x86-eng

```
adb shell
ls /dev/
ls /dev/block/
```

If the command prints the /dev/nvme0 and dev/block/nvme0n1, it means the environment is ready.

10.2.6 Build /hardware/interfaces/nidec (Must build on Android source code)

In the Android AOSP build folder copy the nidec folder to aosp/hardware/interfaces/

(You can also get nidec from https://1drv.ms/u/s!As9sncgYSSJhh3-RxB6qeTzfrkT7?e=DCd176)

From inside the AOSP (android_work or android_work32) folder

```
source build/envsetup.sh
lunch aosp_x86_64-eng  (for 32bit: lunch aosp_x86-eng)
mmm aosp/hardware/interfaces/nidec/1.0
```

Copy the below generate files to android emulator:

```
adb root
adb remount
adb push aosp/out/target/product/generic_x86_64/system/lib/vndk-28/android.hardware.nidec\@1.0.so /system/lib/vndk-28/
adb push aosp/out/target/product/generic_x86_64/system/lib64/vndk-28/android.hardware.nidec\@1.0.so /system/lib64/vndk-28/
adb push aosp/out/target/product/generic_x86_64/system/lib64/android.hardware.nidec\@1.0.so /system/lib64/
adb push aosp/out/target/product/generic_x86_64/system/lib/android.hardware.nidec\@1.0.so /system/lib/
adb push aosp/out/target/product/generic_x86_64/system/lib64/android.hardware.nidec\@1.0.so /vendor/lib64/
adb push aosp/out/target/product/generic_x86_64/vendor/lib64/hw/android.hardware.nidec\@1.0-impl.so /vendor/lib64/hw/
adb push aosp/out/target/product/generic_x86_64/vendor/lib64/hw/android.hardware.nidec\@1.0-impl.so /vendor/lib64/
adb push aosp/out/target/product/generic_x86_64/vendor/bin/hw/android.hardware.nidec\@1.0-service /vendor/bin/hw/
adb push aosp/out/target/product/generic_x86_64/vendor/etc/init/android.hardware.nidec\@1.0-service.rc /vendor/etc/init/
```

10.2.7 Build libxcoder (Must build on android source code)

On the AOSP(android_work or android_work32) folder.

```
source build/envsetup.sh
```

lunch aosp_x86_64-eng (for 32bit: lunch aosp_x86-eng)

```
mmm ../FFmpegXcoder/libxcoder/source
```

Note: FFmpegXcoder is a reference folder. Folder name will depend on where the software package is extracted (Quadra_SW_v*).

Note: Try below command if the above build command is not working:

```
cp /home/$USER/FFmpegXcoder/libxcoder ./external/ -r
mmm ./external/libxcoder/source
```

Copy the below generate files to android emulator

```
adb root
adb remount
adb push
aosp/out/target/product/generic_x86_64/system/lib64/libxcoder.so
/system/lib64/
adb push aosp/out/target/product/generic_x86_64/system/lib/libxcoder.so
/system/lib/
adb push aosp/out/target/product/generic_x86_64/system/lib/vndk-
28/libxcoder.so /system/lib/vndk-28/
adb push aosp/out/target/product/generic_x86_64/system/lib64/vndk-
28/libxcoder.so /system/lib64/vndk-28/
adb push aosp/out/target/product/generic_x86_64/vendor/bin/ni_rsrc_mon
/system/bin/
```

Note: This should be the relative path to FFmpegXoder/libxcoder/source

An error will be reported if the path is absolute. An example of an absolute path

/home/fpga/FFmpegXcoder/libxcoder/source

10.2.8 Push security change to environment

(You can also get nidec from https://1drv.ms/u/s!As9sncgYSSJhh3-RxB6qeTzfrkT7?e=DCd176)

On the AOSP(android_work or android_work32) folder.

```
adb root
adb remount
adb push mediacodec.policy /system/etc/seccomp_policy/
adb push manifest.xml vendor/
```

10.2.9 Prepare ffmpeg4.3 environment

Download android-ndk-r21d from https://developer.android.com/ndk/downloads

You can also get it from https://1drv.ms/u/s!As9sncgYSSJhh3-RxB6qeTzfrkT7?e=DCd176

Download android-sdk_r24.4.1-linux from https://dl.google.com/android/android-sdk_r24.4.1-linux.tgz?utm_source=androiddevtools.cn&utm_medium=website

You can also get it from https://1drv.ms/u/s!As9sncgYSSJhh3-RxB6qeTzfrkT7?e=DCd176

Unpack it on your home dir

```
unzip android-ndk-r21d-linux-x86_64.zip
```

Now you should have the ndk folder

```
fpga@CLI206:~$ ls -laht |grep ndk
drwxr-xr-x   13 fpga fpga 4.0K Sep 28 19:02 android-ndk-r21d
```

10.2.10 Build ffmpeg 4.3 from the FFmpegXcoder

Build the libxcoder:

```
fpga@CLI206:~/FFmpegXcoder$ cd libxcoder
fpga@CLI206:~/FFmpegXcoder/libxcoder$ make clean; bash build.sh
```

Build the ffmpeg 4.3

```
fpga@CLI206:~/FFmpegXcoder$ cd FFmpeg-n4.3
fpga@CLI206:~/FFmpegXcoder/FFmpeg-n4.3$ make clean
fpga@CLI206:~/FFmpegXcoder/FFmpeg-n4.3$ bash build_ffmpeg.sh -a -
-shared
fpga@CLI206:~/FFmpegXcoder/FFmpeg-n4.3$ make install
fpga@CLI206:~/FFmpegXcoder/FFmpeg-n4.3$ sudo ldconfig
```

Then you will see the build out lib&bin on ./android/x86_64/lib and ./android/x86_64/bin

Note: if the build failed with unaccepted sdk license from build.log:

1. record the name of the license and
2. go to the android-sdk-linux/tools
3. ./android list sdk –all And check for the ID of the license you previously recorded
4. ./android update sdk –u -a –t ID1,ID2... or ./android update sdk –u -a (will take longer)

10.2.11 Push all the libs and bin to android

Note: The **adb** command requires an emulator running in another terminal

Start another terminal, under aosp folder:

```
source build/envsetup.sh
lunch aosp_x86_64-eng    #for 32bit: lunch aosp_x86-eng
adb root
adb remount
```

If you do not have android source code you can directly use our build libs and bin (for android x86_64 platform)

Push the aosp build output:

```
adb push
aosp/out/target/product/generic_x86_64/system/bin/ASharedBufferSe
rver   /system/bin/
adb push
aosp/out/target/product/generic_x86_64/system/bin/ni_rsrc_mon
/system/bin/
adb push
aosp/out/target/product/generic_x86_64/system/lib64/libxcoder.so
/system/lib64/
```

(note for 32bit) :

```
adb push
aosp/out/target/product/generic_x86/system/bin/ASharedBufferServe
r  /system/bin/
adb push
aosp/out/target/product/generic_x86/system/bin/ni_rsrc_mon
/system/bin/
adb push
aosp/out/target/product/generic_x86/system/lib/libxcoder.so
/system/lib/
```

Push the ffmpeg4.3 bin and lib build output:

```
adb push ./FFmpegXcoder/FFmpeg-n4.3/android/x86_64/bin/*
/system/bin/
adb push ./FFmpegXcoder/FFmpeg-n4.3/android/x86_64/lib/*
/system/lib64/
```

Make soft link on the android env for libraries:

```
source build/envsetup.sh
lunch aosp_x86_64-eng    #for 32bit: lunch aosp_x86-eng
adb root
adb remount
adb shell
cd  /system/lib64        #for 32bit: cd  /system/lib
ln -s libxcoder.so libxcoder.so.250R1F10
reboot the android system
```

Note: 250R1F10 is the version number, change according to your release version

10.2.12 Run the ffmpeg on android platfrom

(Only run once when android system boot up)

```
adb shell
./vendor/bin/hw/android.hardware.nidec@1.0-service &
 ni_rsrc_mon
```

run the ffmpeg:

```
ffmpeg -y -hide_banner -nostdin -vsync 0 -c:v h264_ni_quadra_dec
-i libxcoder/test/1280x720p_Basketball.264 -c:v rawvideo
output_5.yuv
```

11 Android Docker

For Android 11 and later, we use a Docker container rather than the Android emulator for running an Android system. In this document we will introduce how to setup the Android Docker environment and how to use Quadra under the Android Docker environment.

11.1 Download and start Android docker

ReDroid (https://github.com/remote-android/redroid-doc) is an open source project for an Android Docker container solution.

We can make a customized Android Docker image from ReDroid, or we could use the provided image by ReDroid. ReDroid provides Docker images for multiple versions of Android.

In this document we will use the Docker image for Android 11, provided by ReDroid.

1. Install the required kernel modules

```
apt install linux-modules-extra-`uname -r`
modprobe binder_linux devices="binder,hwbinder,vndbinder"
modprobe ashmem_linux
```

2. Download Android Docker image

```
docker pull redroid/redroid:11.0.0-latest
```

NOTE: Here we are using a Docker image for Android 11 provided by ReDroid, you can choose other images depending on your specific requirements.

3. Start the Android Docker container

```
docker run --name=android11 -itd --rm --privileged -v ~/data:/data
redroid/redroid:11.0.0-latest
```

NOTE: We should use the “--privileged” option when launching, as this argument is required in order to successfully launch the Android Docker container.

After the Android Docker container has been launched successfully you can enter into the Android container by executing the following command:

```
docker exec -it android11 /system/bin/sh
```

You will find some NVMe devices:

```
ls /dev/nvme*
ls /dev/block/nvme*
```

11.2 Download and Build AOSP

For the Android Docker environment, the NETINT software package consists of three main parts. Each needs to be built separately:

1) android.hardware.nidec service: Our customized Android service, this is built based on AOSP.
2) libxcoder: This is a required dynamic library for using NETINT's decoders/encoders and filters in FFmpeg. We need to build the library based on AOSP.
3) FFmpeg: These are NETINT's decoders/encoders/filters modules in FFmpeg, and we build FFmpeg based on the Android NDK. This requires an Android NDK environment.

Because AOSP is required when building the android.hardware.nidec service and the libxcoder, we need to setup the AOSP environment first.

1. Download the repo tool

```
mkdir ~/bin

curl http://commondatastorage.googleapis.com/git-repo-downloads/repo
> ~/bin/repo

chmod a+x ~/bin/repo

export PATH=~/bin:$PATH
```

2. Download AOSP

```
mkdir ~/android_work && cd android_work

repo init -u https://android.googlesource.com/platform/manifest -b
android-11.0.0_r48
```

```
repo sync -j16
```

NOTE: Here android-11.0.0_r48 is just an example. You should choose the specific branch depending on your system's requirements.

3. Build AOSP

```
source build/envsetup.sh
lunch aosp_x86_64-eng
make –j16 //Use -j4 with for CPUs with fewer cores
```

NOTE: aosp_x86_64-eng is also just an example. Choose the option based on the requirements of the system. If the environment is x86_64 architecture then aosp_x86_64-eng is needed. If the environment is arm64 architecture then aosp_arm64-eng is needed. In this document we are using aosp_x86_64-eng as an example.

11.3 Building the android.hardware.nidec service

The android.hardware.nidec is NETINTs customized Android service, it is referenced by the libxcoder.

1. Change the directory to the AOSP folder

```
cd ~/android_work
```

2. Copy the nidec folder to the path hardware/interfaces/ under the AOSP folder, you can get the nidec folder from FFmpegXcode

```
cp ~/ FFmpegXcoder/AOSP/Android11/nidec  hardware/interfaces/ -r
```

3. Build the nidec service

```
source build/envsetup.sh
lunch aosp_x86_64-eng
mmm ./hardware/interfaces/nidec/1.0/
```

If built successfully there will be the following files generated under the AOSP out directory

```
  out/target/product/generic_x86_64/system/lib64/android.hardware.nidec\
@1.0.so

  out/target/product/generic_x86_64/vendor/lib64/hw/android.hardware.nid
ec\@1.0-impl.so

  out/target/product/generic_x86_64/vendor/bin/hw/android.hardware.nidec
\@1.0-service

  out/target/product/generic_x86_64/vendor/etc/init/android.hardware.nid
ec\@1.0-service.rc
```

Push the generated files to the Android run environment, see section11.7 for more details.

NOTE: In this document we assume FFmpegXcoder is placed under path ~/

11.4 Build libxcoder

libxcoder is required for NETINT's decoders/encoders and filters in FFmpeg, and we need build it based on AOSP.

For Quadra and Logan co-existence case, we should build libxcoder and libxcoder_logan separately.

1. Change directory to the AOSP folder, and then copy the libxcoder folder to the path external/ under the AOSP folder.

```
cd ~/android_work
cp ~/FFmpegXcoder/libxcoder external/ -r
```

   For Logan we should copy libxcoder_logan:

```
cp ~/FFmpegXcoder/libxcoder_logan external/ -r
```

2. Build the libxcoder

```
source build/envsetup.sh
lunch aosp_x86_64-eng
mmm ./external/libxcoder/source/
```

   For Logan we should build libxcoder_logan:

```
mmm ./external/libxcoder_logan/source/
```

If the build is successful there will be the following generated files under the AOSP out directory

```
out/target/product/generic_x86_64/system/lib64/libxcoder.so
out/target/product/generic_x86_64/vendor/bin/ni_rsrc_mon
```

For Logan there are following generated files under AOSP out directory:

```
out/target/product/generic_x86_64/system/lib64/libxcoder_logan.so
out/target/product/generic_x86_64/vendor/bin/ni_rsrc_mon_logan
```

We need to push the generated files to Android run environment, please refer to chapter11.7 for details.

11.5 Build ffmpeg based on NDK

NETINT's has decoder, encoder and filter modules in FFmpeg. We build FFmpeg based on the Android NDK.

1. Create the pkg-config file for libxcoder

The libxcoder is referenced by FFmpeg, which is based on the pkg-config mechanism. So we need to generate the pkg-config file for libxcoder first.

```
cd ~/FFmpegXcoder/libxcoder/ &&  sh build.sh -a
```

The above step is for Quadra, for Logan (or for Quadra and Logan co-existence) also create the pkg-config file for libxcoder_logan

```
cd ~/FFmpegXcoder/libxcoder_logan/ &&  sh build.sh -a
```

Generate the pkg-config file for libxcoder and then copy the generated libxcoder dynamic library by AOSP to the directory specified in the libxcoder pkg-config file.

```
cp ~/android_work/out/target/product/generic_x86_64/system/lib64/libxcoder.so /usr/local/lib/
```

For Logan we should copy libxcoder_logan:

```
cp ~/android_work/out/target/product/generic_x86_64/system/lib64/libxcoder_logan.so /usr/local/lib/
```

2. Build FFmpeg

```
cd ~/FFmpegXcoder/FFmpeg-n4.3.1 && make clean
sh build_ffmpeg.sh -a –shared --android_arch=x86_64
make install
sudo ldconfig
```

For Quadra and Logan co-existence build FFmpeg with the following command:

```
sh build_ffmpeg.sh --quadra --logan -a -shared --android_arch=x86_64
```

NOTE: This uses FFmpeg-n4.3.1 as an example, but you can use other FFmpeg versions depending on system requirements.

android_arch can be a range of values [arm,arm64,x86,x86_64(default)], here x86_64 is used in this example.

If the build is successful, a folder named “android” will be created under the path ~/FFmpegXcoder/FFmpeg-n4.3.1

There will also be some FFmpeg libraries, and bin files generated under the folder.

Push the generated files to the Android run environment, please refer to section 11.7 for details.

11.6 Copying the build to the Android container

To use the NETINT transcoding card inside the Android Docker container copy the build output lib/bin files to the Android Docker container first.

1. Push the android.hardware.nidec service build

Change directory to AOSP first and then copy the following files to the Android Docker container

```
cd ~/android_work

docker cp out/target/product/generic_x86_64/system/lib64/android.hardware.nidec\@1.0.so /system/lib64/

docker cp out/target/product/generic_x86_64/system/lib64/android.hardware.nidec\@1.0.so / vendor/lib64/

docker cp out/target/product/generic_x86_64/vendor/lib64/hw/android.hardware.nidec\@1.0-impl.so /vendor/lib64/

docker cp out/target/product/generic_x86_64/vendor/bin/hw/android.hardware.nidec\@1.0-service /vendor/bin/hw/

docker cp out/target/product/generic_x86_64/vendor/etc/init/android.hardware.nidec\@1.0-service.rc /vendor/etc/init/
```

2. Push the libxcoder build

Change directory to AOSP and then copy the following file to the Android Docker container:

```
cd ~/android_work

docker cp out/target/product/generic_x86_64/system/lib64/libxcoder.so /system/lib64/

docker cp out/target/product/generic_x86_64/vendor/bin/ni_rsrc_mon /system/bin/
```

For Quadra and Logan co-existence case we should also copy the following file:

```
cd ~/android_work

docker cp out/target/product/generic_x86_64/system/lib64/libxcoder_logan.so /system/lib64/

docker cp out/target/product/generic_x86_64/vendor/bin/ni_rsrc_mon_logan /system/bin/
```

3. Copy the FFmpeg build

Change directory to ffmpeg first and then copy the ffmpeg libs/bin files to the Android container

```
cd ~/FFmpegXcoder/FFmpeg-n4.3.1

docker cp android/x86_64/lib/* /system/lib64/

docker cp android/x86_64/bin/* /system/bin/
```

11.7 Copy the customized manifest file to Android container

As described in section 3.4 android.hardware.nidec is the customized Android service. To start the service register it to the Android system using the manifest file.

The file android.hardware.nidec@1.0-service.xml is the manifest file for the customized Android service.

Copy the android.hardware.nidec@1.0-service.xml file to the Android run environment

```
cd ~/FFmpegXcoder/AOSP/Android11/

docker cp android.hardware.nidec@1.0-service.xml
/vendor/etc/vintf/manifest/
```

NOTE: Change the destination path to the manifest file being copied according to the desired running environment.

11.8 Run FFmpeg inside the Android container

After copying the build to the Android container restart the Android container

```
docker restart android11
```

Use Quadra by the following steps:

1. Initialization

```
ni_rsrc_mon

ni_rsrc_mon_logan //this step is need for Quadra and Logan co-existence
```

NOTE: ni_rsrc_mon is for Quadra, and for Logan use ni_rsrc_mon_logan. For Logan also exec the ni_rsrc_mon_logan

2. Execute the ffmpeg command

Test command for Quadra, for example:

```
ffmpeg -c:v h264_ni_quadra_dec -i 1280x720p_Basketball.h264 -c:v h265_ni_quadra_enc test.h265 -y
```

For Quadra and Logan co-existence case test command for Logan, for example:

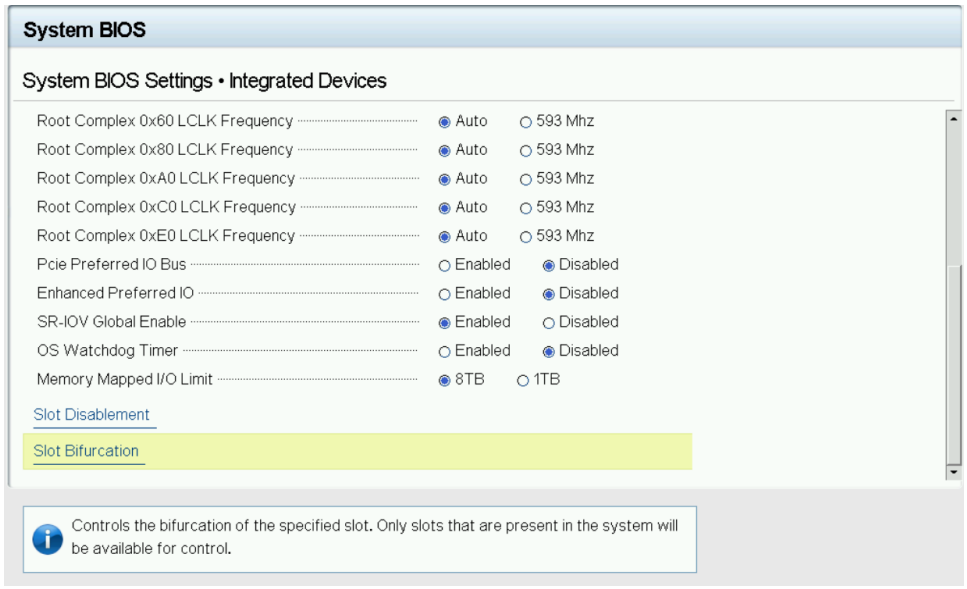
```
ffmpeg -c:v h264_ni_logan_dec -i 1280x720p_Basketball.264 -c:v h265_ni_logan_enc test.265 -y
```

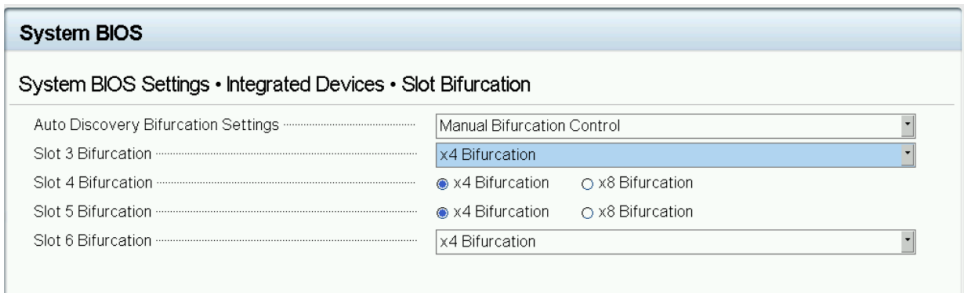
12 Enable PCIe bifurcation for Quadra T2A

Please note that the host has to support x4 PCIe bifurcation.

1. Boot into BIOS/CMOS
2. Set bifucation to use x4
   Find bifucation setting in BIOS and set it to use x4 or x4x4x4x4.
   Please note that bifucation settings might be named differently with different BIOS

   Demo bifurcation setting:





3. Confirm if the bifurcation setting is correct
   You should be able to see two units from T2 card

```
nvme@nvme-cli444:~$ sudo nvme list
Node             SN                   Model
---------------- -------------------- --------------
/dev/nvme0n1     Q2A224A11DC066-0014B QuadraT2A-EP2
/dev/nvme1n1     Q2A224A11DC066-0014A QuadraT2A-EP2
```

13 Versioning Number Schema

NETINT release package contains multiple components with their own release version numbers and —semantic like— compatibility numbers.

13.1 Release Version Number

The Netint Quadra *release version numbers* (eg. 4.7.0) consist of 3 single characters:

```
Major.Minor.Micro
```

Major milestone releases will increment major character. Periodic releases from development trunk will increment minor number. All other types of releases (eg. hot-fix) will increment micro number.

The *release version number* characters may be 0-9 and A-Z.

A release of greater *release version number* supersedes releases of lesser *release version number* as newer releases are based upon older releases. It is preferable to use release of greatest available release version number to have access to latest features and fixes.

NETINT release packages have a *release version number*, but FW and SW releases also have their own 3 character FW/SW *release version number*. FW and SW releases in a release package does not always share the same *release version number*. If FW and SW releases in a release package are of different *release version numbers*, the release package's *release version number* will be the larger *release version number* between FW and SW releases.

13.2 Full Version Number

Between the Firmware revision and Software applications, there is also an 8 character *full version number* (e.g. 4706r3r4).

The firmware *full version number* can be seen and read from the following command

```
sudo nvme list
```

or

```
./quadra_fw_info FL_BIN/*.bin
```

The Software *full version number* can be read from any libxcoder applications with the `-v` argument. For example:

```
ni_rsrc_mon -v
```

It is also defined in code of the **`libxcoder/source/ni_defs.h`** file.

The first 3 characters of the *full version number* is the *release version number*.

The 4th to 6th characters are the FW API version number (see below).

The last 2 characters of the *full version number* are for NETINT to track release development.

13.3 FW API Version Number

Within the *full version number* (eg. 4706r3r4) of FW and SW applications the 4th to 6th characters contain the *FW API version number*. This is one semantic major and two minor version numbers that tracks API compatibility between the firmware on the device and the libxcoder version. The major *FW API version number* must match between FW and libxcoder for basic interoperability. The minor *FW API version numbers* should match to access full/new feature set of FW/libxcoder.

NETINT endeavours to maintain backward compatibility between FW and libxcoder in all releases.

13.4 Libxcoder API Version Number

The *libxcoder API version number* is semantic major and minor version number pair that tracks libxcoder public API compatibility with linked APIs (eg. libavcodec) and applications (eg. xcoder-util). It can be read from code in `libxcoder/source/ni_defs.h`.

The *libxcoder API version number* characters will be strictly numeric. The individual major and minor portions may have more than one digit.

The major *libxcoder API version number* will be incremented when the API changes in a backward incompatible manner. The minor *libxcoder API version number* will be incremented when new features are added that require updating application code to access.

Regardless of any *libxcoder API version number* changes, it is recommended to recompile applications linked to libxcoder when updating to a new SW *release version*.

14 Useful documents and reference

There are several documents included in each Quadra release package. Please ask NETINT support if you require any helping finding them.

1. **Quick Start Guide Quadra**
   This guide is a quick and easy setup guide for Quadra. It walks through a typical setup for a user who requires FFmpeg only.

2. **Integration Programming Guide Quadra**
   This is a detailed guide that shows how to integrate Quadra into customer solutions. It covers how to integrate Quadra in your application using various SDKs, for example FFMpeg, GStreamer or our own NETINT proprietary libxcoder API.

3. Please contact NETINT support for a complete list of **Application Notes** for Quadra. These are some examples of Quadra's App Notes are

   - Encoder Quality
   - Low Latency
   - Sequence Change
   - Live Streaming

   - Android
   - Windows 2019/MSYS64
   - MacOS

   - SR-IOV in Linux
   - Docker

   - Libavcodec API
   - GStreamer