Quadra™ Quickstart Guide V5.6

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. NETINT Overview

NETINT is a supplier of high-performance, low-latency, real-time video processing units (VPUs) for x86 and Arm servers. We can provide multiple stream transcoding functions and services, directly to video content providers. We can also provide Transcoding as a Service (TaaS), which can be integrated into video streaming systems and services. Our functions and services can be used for highly efficient Video-on-Demand file transcoding, as well as real-time, live video streaming applications.

2.1 Purpose of this document

This quick start guide provides an overview on how to quickly setup and start using the NETINT video transcoding unit.

It describes some of the more important transcoding parameters, and how they can be used and configured to integrate with other NETINT services and solutions.

This Quick Start guide uses FFmpeg and the **Linux operating system**.

More details on how to setup other environments and configurations, including other OSes like Android and Windows, and also for using Docker/Containers and VMs, can be found in the more detailed **InstallationGuideQuadra_V*.pdf** included with your release package.

2.2 Installation and Compatibility

All Hardware, Firmware and Software installation must follow the directions given in this document. The Quadra Video Processing Unit must be tested using the configuration details found within this document. Also, all component parts must conform to the details given here. This includes the versions of open source software, as well as the hardware detailed in this section.

2.3 Compatibility

This quick start guide is compatible with the following components.

Software

This guide is for NETINT Quadra Video Transcoder software release 5.6 and onwards

Hardware

This guide supports the following NETINT Quadra Video Transcoder hardware

- T1S (SODIMM)
- T1M (M.2)
- T1U (U.2)
- T1A (AIC)
- T2A (AIC)

2.4 Hardware Installation

Quadra Video Transcoders require only minimal CPU resources when running video transcoding tasks.

Recommended System
Processor : x64 Intel/AMD 6 Core (12 Threads) **System Memory T1A/T1U**: 16GiB - 2x 8GiB 2133Mhz DDR4 RAM **System Memory T2A** : 32GiB - 2x 16GiB 2133Mhz DDR4 RAM **Motherboard** : PCIe Gen 4 (PCIe Gen 5 Configure BIOS for PCIe Gen 4) **T1U** requires U.2 x4 slot **T1A** requires x8 slot or x16 slot (Linkup at x4) **T2A** requires x8 slot (bifurcate x4/x4) / x16 slot (x4/x4/x4/x4) or (x4/x4/x8). Connected 8 lanes need to be set for x4/x4 linkup. Remaining 8 lanes don't matter Notes 1. PCIe Gen 3 can be used but is not recommended due to lower performance 2. GPU optimized servers are recommended (to improve air flow cooling), e.g. Dell PowerEdge C4140 3. Only U.2 Cards (T1U) support Hot Plug, AIC Cards (T1A/T2A) do not support Hot Plug 4. **All T2A PCIe Slots require Bifurcation to be enabled in the host BIOS**

Quadra leverages Non-Volatile Memory express (NVMe) server technology. NVMe is a host controller interface and storage protocol for high speed data transfer over a server's high-speed Peripheral Component Interconnect express (PCIe) bus. Quadra is designed to plug directly into host servers.

Quadra modules are available in both U.2 and Add-In-Card (AIC) form factors. AIC cards do not support Hot Plug and require a cold boot of the server after insertion before they are recognized in the host system. Quadra U.2 cards do support Hot Plug.

2.5 Operating System Compatibility

A host server with any of the following operating systems is recommended:

- OS Ubuntu 16.04; Kernel 4.4.0
- OS Ubuntu 16.04; Kernel 4.10.0
- OS Ubuntu 18.04; Kernel 4.15.0
- OS Ubuntu 20.04; Kernel 5.4.0
- OS Ubuntu 22.04; Kernel 5.15.0
- OS Ubuntu 22.10; Kernel 5.15.0
- OS Ubuntu 22.04; Kernel 6.8.0
- OS Ubuntu 24.04; Kernel 6.8.0
- OS CentOS 7.9; Kernal 3.10.0
- OS CentOS Stream release 8; 4.18.0
- OS CentOS 9; Kernel 5.14.0
- OS Debian 12; Kernel 6.1.0
- OS Rocky Linux 9.5; Kernel 5.14.0

The following Windows operating system versions are also supported. Please refer to Section 9 for more details.

- Windows 10 Pro
- Windows 11 Pro
- Windows Server 2019

We also support the following MacOS operating systems

- Mac OS 12.6
- Mac OS 13.0.1

For complete support on all operating systems (including Android), please see the **InstallationGuideQuadra_V*.pdf**

2.6 Scripted FW/SW Installation

The **quadra_quick_installer.sh** script is provided for the installation of the Firmware to the Quadra device, and also to install a default configuration of the Software on the host system.

NOTE : The quick installer script is supported on Ubuntu, CentOS, and MacOS, **not Windows or Android**.

The quick install script can be found at the **top level** of the NETINT release package, along with the compressed firmware and software release files in the **.zip** format.

1. Copy the following file from the release package to the host system. Note that the compressed filename syntax will be as follows :

   ```
   Quadra_V5.4.0.zip
   ```

2. Uncompress the main Quadra_VN.N.N release package zip file

   ```
   $ unzip Quadra_V5.4.0.zip
   ```

3. Now change directory to the new Quadra release folder

   ```
   $ cd Quadra_V5.4.0
   ```

4. Check you can see the uncompressed files in the new folder using the **ls** command, note that the file names with versions will change, depending on the release version you are installing.

   Note : Ensure the **Quadra_FW_VN.N.N_RCN** and **Quadra_SW_VN.N.N_RCN** folders are contained in the new folder, as well as the **quadra_quick_installer.sh** script.

5. Execute the quick installer BASH script

```
$ bash quadra_quick_installer.sh
```

```
The install script scans the host system seeking both
installed Quadra devices, as well as any release upgrade
packages. It then prompts the user to confirm which packages
to install.
```

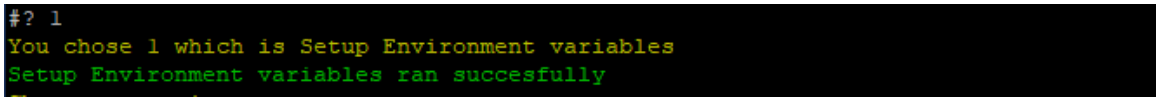
Check the correct packages have been chosen for installation and then press the '**y**' key

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

6. The following menu options are displayed and can be used to set up the system and install the listed frameworks

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 FFmpeg-n4.3.1
7) Install FFmpeg-n5.1.2
8) Install FFmpeg-n6.1
9) Install FFmpeg-n7.1
10) Install FFmpeg-n8.0.1
11) Install gstreamer-1.22.2
12) Install gstreamer-1.24.4
13) Install gstreamer-1.24.9
14) Install gstreamer-1.26.2
15) Firmware Update
16) Quit

7. Type in '**1**' and press enter to setup the environment variables. Make sure the action was successful



8. Type in '**3**' and press enter to install all the prerequisite software libraries for your OS (Ubuntu/CentOS/MacOS). Again, check the action was successful without any warnings or errors.

9. Type in '**4**' and press enter to install the **nvme-cli** tool

10. Type '**5**' to install the Libxcoder

11. Depending on version of FFmpeg desired, use an option from '**6**' to '**10**' to install FFmpeg. You can use the prompts to compile the libav shared libraries, add Gstreamer support or interface with the libavcodec.

12. If you also wish to install Gstreamer with Netint support through gst-libav, use an option from '**11'** to **'14'** and press enter. The pre-requisites are that an FFmpeg version >= n4.3.1 is installed with shared libraries and Gstreamer support.
    Note: Gstreamer support in libavcodec changes some of the timestamp handling to use the Gstreamer style over the FFmpeg style.

13. Type in '**15**' and press **Enter** to start the guided **Firmware** upgrade process.

14. Type in '**16**' and press **Enter** to quit the script

If any individual steps fail during the **quadra_quick_installer.sh** execution then it is recommended the output errors are inspected and to take the necessary action to resolve all issues. Once all issues are resolved repeat that installation step. If any errors persist, please contact NETINT support.

2.7 Manual FW/SW Installation

NOTE – This document is written with reference to a Linux Host running with Ubuntu 20.04 OS or newer. If using a host installed with a different OS please also refer to Section 3 of the InstallationGuideQuadra_V1 (or newer). This will detail any packages or environment configuration steps that may be different or specific to the installed OS.

Proceed with installing the pre-requisite packages and updating the environment settings.

1. If the host server is running Ubuntu run the following command to install all the prerequisite packages:

   ```
   $ sudo apt-get install -y nasm pkg-config git gcc make
   ```

2. Add the following lines in the file **/etc/bashrc**

   ```
   export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig/
   export LD_LIBRARY_PATH=/usr/local/lib/
   ```

3. Then source the file by running the command:

   ```
   $ source /etc/bashrc
   ```

4. Add the following line in the file, **/etc/ld.so.conf**

   ```
   /usr/local/lib
   ```

5. Then run the command:

   ```
   $ sudo ldconfig
   ```

For any Linux operating system check that the **/etc/sudoers** file is configured correctly to run the installed programs:

1. In the file **/etc/sudoers** find the line:

```
Defaults    secure_path =
```

2. Note its entries are separated by a colon ':'. If /usr/local/sbin and /usr/local/bin are not in the secure_path add them by appending the following to the secure_path line:

```
:usr/local/sbin:/usr/local/bin
```

3. Add the following line in the file **/etc/sudoers**

```
Defaults    env_keep += "PKG_CONFIG_PATH"
```

2.7.1. Verify the Quadra Hardware Installation

lspci is a utility for displaying information about PCIe buses in the system, and any devices that are connected to them.

To verify the installation through PCIe, run the following command:

```
$ lspci -d 1d82: | wc -l
```

The returned number indicates how many Quadra devices that are installed and detected on the system, through PCIe.

2.7.2. NVMe CLI Download and Install

If NVMe CLI is not installed by default on Linux server, it's recommended to install specific NVMe CLI version through choosing option 4 when running **quadra_quick_installer.sh**.

Following command might be used to install nvme CLI.

```
$ sudo apt-get install -y nvme-cli
```

However, installed version through above command might not be the versions which are tested and recommended by NETINT (1.6, 1.16) although that different version might work.

2.7.3. Quadra Device detection

Confirm that the Quadra device is detected by the host by running the following command:

```
$ sudo nvme list
```

2.7.4. Quadra Firmware Upgrade

Before upgrading the firmware the following requirements must be met:

1. Quadra devices are detected by the command

   ```
   $ sudo nvme list
   ```

2. The Quadra firmware update package is available (i.e. Quadra_FW_V4.1.0_RC1.tar.gz)

Once these requirements are verified, follow these steps to upgrade the firmware on the device.

NOTE: The User must stop all transcoding processes **before** performing a firmware upgrade, otherwise it may lead to a device malfunction.

1. Identify the Quadra Firmware Release package <version_number> from its file name. Eg. if the file name is Quadra_FW_V4.1.0_RC1.tar.gz, the <version_number> is '4.1.0_RC1'.
2. Untar the Quadra Firmware Release package:

   ```
   $ tar -zxf Quadra_FW_V<version_number>.tar.gz
   ```

3. Execute the quadra_auto_uprade.sh script within it:

   ```
   $ cd Quadra_FW_V<version_number>; ./quadra_auto_upgrade.sh
   ```

   Temp upgrade is a firmware upgrade method that does not write the new Firmware to the nor flash, and will therefore be lost on a device reset. It can be triggered by

   ```
   $ cd Quadra_FW_V<version_number>;./quadra_auto_upgrade.sh -t
   ```

4. Follow instructions in the guided script. It will display information about FW on card and to be upgraded to. There will be a confirmation prompt before FW upgrade begins.

2.7.5. Important Notes on Firmware Upgrade

Only PCIe ***physical*** devices may operate the FW upgrade. PCIe **virtual** devices will be excluded by the upgrade script.

The User must **stop all transcoding processes** before performing a firmware upgrade, **otherwise it may lead to a device malfunction**.

Each device will check the file integrity of the Firmware binary before committing to use it.

Before any temp upgrade, the device will check the Firmware binary version. The temp upgrade feature requires both the firmware version on the nor flash **as well as** the firmware binary the device is updating to, to be v5.1.5 or higher.

For temp upgrade, the device will also check the firmware **flavor**. Both the proposed new temp upgrade firmware binary and the binary currently loaded on to the nor flash must match. If the firmware flavors do not match the temp update will be rejected.

The User must NOT interrupt the upgrade process once it has started, doing so may lead to a device malfunction. If the cold upgrade fails, a system reboot may resolve the issue, the upgrade process should then be repeated.

Additional logs on the upgrade process can be found in **./upgrade_log.txt** which is generated by the upgrade script.

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

The temp upgrade is reverted when the device is reset. Temp firmware upgrade is a **temporary state only** and the firmware in the nor flash will be loaded and executed on reboot. If the temp upgrade fails the drive is unaffected, and will continue to execute the firmware in the nor flash.

2.7.6. FFmpeg Supported Versions

Interfacing with a Quadra device can be via *FFmpeg*.

Quadra supports the following FFmpeg versions:

- 4.3.1
- 5.1.2
- 6.1
- 7.1
- 8.0.1

NOTE: Only FFmpeg 4.3.1 is fully supported in Windows and Android.

2.7.7. FFmpeg Download and Installation

The FFmpeg stock versions are stored in the FFmpeg repository, e.g.:

- 4.3.1 https://github.com/FFmpeg/FFmpeg/tree/n4.3.1

To clone the GitHub FFmpeg repository from the command line, run the following command based on the `<version>` (example 2.3.1) that you want from the list of supported FFmpeg versions from Section 2.4.6:

```
$ git clone -b n<version> --depth=1
https://github.com/FFmpeg/FFmpeg.git FFmpeg
```

E.g.

```
$ git clone -b n4.3.1 --depth=1
https://github.com/FFmpeg/FFmpeg.git FFmpeg
```

NOTE: Download NETINT supported FFmpeg versions only, as per Section 2.4.6.

2.7.8. Apply FFmpeg Patch

The following instructions need to be done in sequence to prepare FFmpeg for use with the Quadra Video Processing Unit.

4. Untar the Quadra Software Release package:

   ```
   $ tar -zxf Quadra_SW_V*.tar.gz
   ```

5. Copy the ***libxcoder*** folder from the untarred SW release folder to the current parent folder of the FFmpeg repository (i.e. in the same folder as the FFmpeg folder):

   ```
   $ cp -r Quadra_SW_V*/libxcoder ./
   ```

6. Copy the FFmpeg patch file from the untarred SW release package to the *FFmpeg/* folder depending on which official `<version>` (example 4.3.1) of FFmpeg was downloaded per Section 2.4.6:

   ```
   $ cp Quadra_SW_V*/FFmpeg-n<version>_netint_v*.diff FFmpeg/
   ```

   E.g.

   ```
   $ cp Quadra_SW_V*/FFmpeg-n4.3.1_netint_v*.diff FFmpeg/
   ```

7. Change to the *FFmpeg/* folder:

   ```
   $ cd FFmpeg/
   ```

8. Apply the relevant NETINT patch to the `FFMpeg <version>` example 4.3.1:

   ```
   $ patch -t -p 1 < FFmpeg-n<version>_netint_v*.diff
   ```

   E.g.

   ```
   $ patch -t -p 1 < FFmpeg-n4.3.1_netint_v*.diff
   ```

The following instructions also need to be performed in sequence in order to prepare FFmpeg for use with the Quadra Video Processing Unit. These instructions are applied to the software package delivered as part of the NETINT release package.

NOTE: Run the correct patch for your FFmpeg version.

2.7.9. Build FFmpeg with NETINT Codec Library

The following instructions must be executed in the order specified to build FFmpeg with the NETINT Codec Library.

1. From ***FFmpeg/*** change directory to the ***libxcoder/*** directory with the command:

   ```
   $ cd ../libxcoder
   ```

2. Build and install libxcoder with the command:

   ```
   $ bash build.sh
   ```

3. Load the libxcoder shared library into ldconfig with the command:

   ```
   $ sudo ldconfig
   ```

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

   ```
   $ cd ../FFmpeg
   ```

5. Run the `build_ffmpeg.sh` script with the following command lines (these will build with Quadra by default):

   ```
   $ sudo make clean
   $ bash build_ffmpeg.sh
   ```

   NOTE: The **sudo make clean** command line above may return with an error if there are no previous build files to clean up. If you see a similar error message to the one below and you in the correct directory, if there are no existing build files then this is expected.

   ```
   $ sudo make clean
   Makefile:167: /tests/Makefile: No such file or directory
   make: *** No rule to make target '/tests/Makefile'. Stop.
   ```

6. Now install FFmpeg with Quadra to the system, with the following command:

   ```
   $ sudo make install
   ```

3. Setup Verification

After completing the installation steps above for both hardware and software, follow the steps below to activate and verify your Quadra setup.

1. Check the NVMe device presence with the following command:

   $ sudo nvme list

   The list of NVMe devices on the system will be displayed.

   Look for devices with the **Quadra** in the Model name. Also, after a reboot is performed (see below), check that the **FW Rev** is the new firmware version that was installed as part of this release. For example **40064rcD** would be the firmware version with release package **4.0.0.**

   See the example below:

Node	SN	Model	Namespace	Usage	Format	FW Rev
/dev/nvme0n1	Q1A10BA11FC060-0023	QuadraT1A	1	8.59 TB / 8.59 TB	4 KiB+0 B	40064rcD

2. To run FFmpeg without root privilege, the user needs to be added to the user ‘disk’ group:

   $ sudo usermod -a -G disk $USER

   * If you want to run FFmpeg as root, you can skip this step and use “sudo init_rsrc” to initialize all Quadra devices.

3. Reboot the server (ensuring the system is ready for a reboot):

   $ sudo reboot now

   Alternatively, perform a warm/temp upgrade without system reboot. To do that, run:

   $ ni_rsrc_update -D

4. Initialize Quadra devices with the following command:

   $ init_rsrc

4. Monitoring Load

To monitor the processing load of Quadra devices, use the NETINT resource monitor utility:

```
$ ni_rsrc_mon
```

```
[fpga@CLI335 FFmpegXcoder]$ ni_rsrc_mon
NI resource init'd already ..
*************************************************
1 devices retrieved from current pool at start up
Tue Nov 15 18:19:31 2022 up 00:00:00 v---6ADEV
Num decoders: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE         NAMESPACE
0     19   73         1    3    34        0       /dev/nvme0     /dev/nvme0n1
Num encoders: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE         NAMESPACE
0     35   31         1    13   34        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    34        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    34        0       /dev/nvme0     /dev/nvme0n1
*************************************************
```

```
$ ni_rsrc_mon -h
-------- ni_rsrc_mon v---6iDEV --------
The ni_rsrc_mon program provides a real-time view of NETINT Quadra resources
running on the system.

Usage: ni_rsrc_mon [OPTIONS]
-n  Specify reporting interval in one second interval. If 0 or no selection,
    report only once.
    Default: 0
-R  Specify if refresh devices on host in each monitor interval.
    If 0, only refresh devices at the start.
    Default: 1
-o  Output format. [text, simple, full, json, json1]
    Default: text
-r  Initialize Quadra device regardless firmware release version to
    libxcoder version compatibility.
    Default: only initialize devices with compatible firmware version.
-t  Set timeout time in seconds for device polling. Program will exit with
    failure if timeout is reached without finding at least one device. If 0 or
    no selection, poll indefinitely until a Quadra device is found.
    Default: 0
-S  Skip init_rsrc.
-d  print detail infomation. It only supports getting encoder/encoder info. Its output only has text and json formats.
-l  Set loglevel of libxcoder API.
    [none, fatal, error, info, debug, trace]
    Default: info
-h  Open this help message.
-v  Print version info.

Simple output shows the maximum firmware load amongst the subsystems on the
Quadra device.

Reporting columns for text output format
INDEX         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
MEM           usage of memory by the subsystem
SHARE_MEM     usage of memory shared across subsystems on the same device
P2P_MEM       usage of memory by P2P
DEVICE        path to NVMe device file handle
NAMESPACE     path to NVMe namespace file handle
```

Help on how to use the program can be accessed with the command:

There are 4 subsystems to each Quadra device. These subsystems are listed by the resource monitor. They are the

- Encoder
- Decoder
- Scaler
- AI

Below is an example of the resource monitor output in the default format.

```
[fpga@CLI335 FFmpegXcoder]$ ni_rsrc_mon
NI resource init'd already ..
*************************************************
1 devices retrieved from current pool at start up
Tue Nov 15 18:19:31 2022 up 00:00:00 v---6ADEV
Num decoders: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE         NAMESPACE
0     19   73         1    3    34        0       /dev/nvme0     /dev/nvme0n1
Num encoders: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE         NAMESPACE
0     35   31         1    13   34        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    34        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    34        0       /dev/nvme0     /dev/nvme0n1
*************************************************
```

You can see encoder/decoder detail information by using ni_rsrc_mon –d command

```
$ ni_rsrc_mon -d
NI resource init'd already ..
************************************************
1 devices retrieved from current pool at start up
Thu Sep 28 21:30:44 2023 up 00:00:00 v---6kDEV
Num decoders: 1
INDEX AvgCost FrameRate IDR InFrame OutFrame DEVICE         NAMESPACE
0     66      30        2   51      51       /dev/nvme0n1   /dev/nvme0n1
1     66      30        2   51      51       /dev/nvme0n1   /dev/nvme0n1
Num encoders: 1
INDEX AvgCost FrameRate IDR InFrame OutFrame BR         AvgBR      DEVICE         NAMESPACE
0     100     30        1   48      43       55526400   31013570   /dev/nvme0n1   /dev/nvme0n1
1     100     30        1   47      42       6014880    30429931   /dev/nvme0n1   /dev/nvme0n1
```

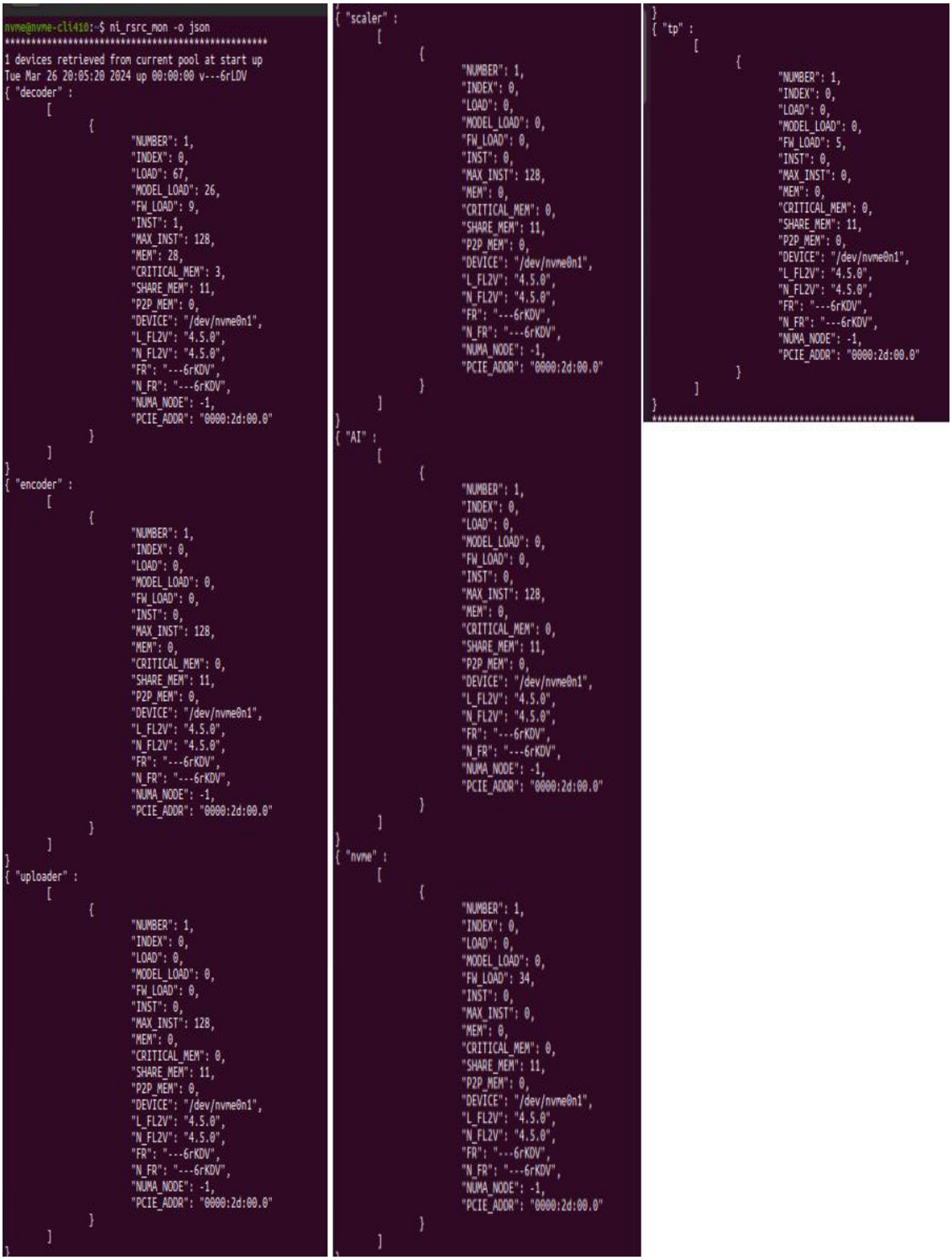
The following is an example of the resource monitor’s JSON output. There are two different output formats, “-o json” and “-o json1”.

In the output, there are two parameters added specifically for Linux. On other platforms, these two parameters are not available.

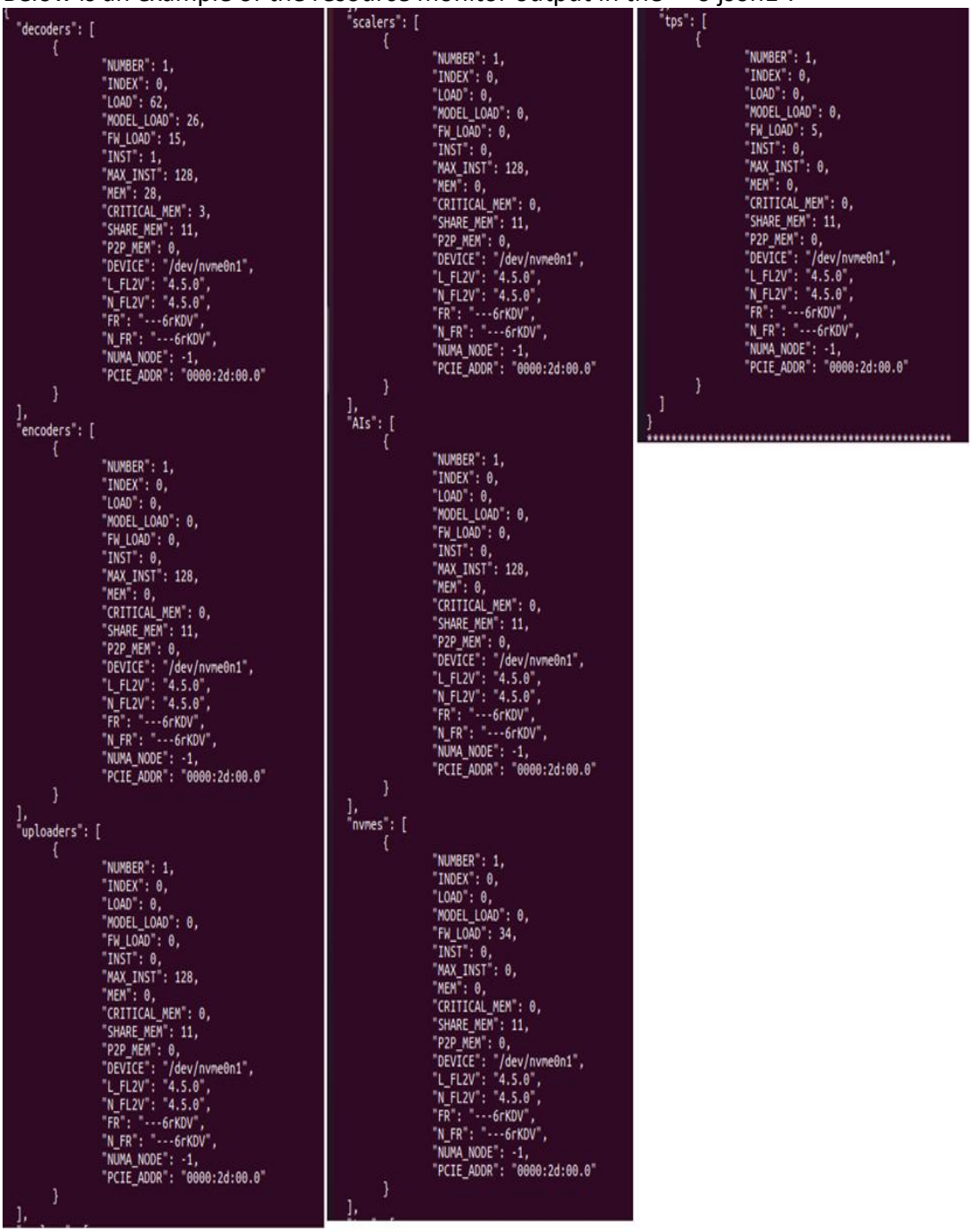
NUMA_ NODE: Numa node id

PCLE_ ADDR: PCIe address

Below is an example of the resource monitor output in the “- o json”



Below is an example of the resource monitor output in the “- o json1”.



```
{
 "decoders": [
        {
                "NUMBER": 1,
                "INDEX": 0,
                "LOAD": 62,
                "MODEL_LOAD": 26,
                "FW_LOAD": 15,
                "INST": 1,
                "MAX_INST": 128,
                "MEM": 28,
                "CRITICAL_MEM": 3,
                "SHARE_MEM": 11,
                "P2P_MEM": 0,
                "DEVICE": "/dev/nvme0n1",
                "L_FL2V": "4.5.0",
                "N_FL2V": "4.5.0",
                "FR": "---6rKDV",
                "N_FR": "---6rKDV",
                "NUMA_NODE": -1,
                "PCIE_ADDR": "0000:2d:00.0"
        }
 ],
 "encoders": [
        {
                "NUMBER": 1,
                "INDEX": 0,
                "LOAD": 0,
                "MODEL_LOAD": 0,
                "FW_LOAD": 0,
                "INST": 0,
                "MAX_INST": 128,
                "MEM": 0,
                "CRITICAL_MEM": 0,
                "SHARE_MEM": 11,
                "P2P_MEM": 0,
                "DEVICE": "/dev/nvme0n1",
                "L_FL2V": "4.5.0",
                "N_FL2V": "4.5.0",
                "FR": "---6rKDV",
                "N_FR": "---6rKDV",
                "NUMA_NODE": -1,
                "PCIE_ADDR": "0000:2d:00.0"
        }
 ],
 "uploaders": [
        {
                "NUMBER": 1,
                "INDEX": 0,
                "LOAD": 0,
                "MODEL_LOAD": 0,
                "FW_LOAD": 0,
                "INST": 0,
                "MAX_INST": 128,
                "MEM": 0,
                "CRITICAL_MEM": 0,
                "SHARE_MEM": 11,
                "P2P_MEM": 0,
                "DEVICE": "/dev/nvme0n1",
                "L_FL2V": "4.5.0",
                "N_FL2V": "4.5.0",
                "FR": "---6rKDV",
                "N_FR": "---6rKDV",
                "NUMA_NODE": -1,
                "PCIE_ADDR": "0000:2d:00.0"
        }
 ],
 "scalers": [
        {
                "NUMBER": 1,
                "INDEX": 0,
                "LOAD": 0,
                "MODEL_LOAD": 0,
                "FW_LOAD": 0,
                "INST": 0,
                "MAX_INST": 128,
                "MEM": 0,
                "CRITICAL_MEM": 0,
                "SHARE_MEM": 11,
                "P2P_MEM": 0,
                "DEVICE": "/dev/nvme0n1",
                "L_FL2V": "4.5.0",
                "N_FL2V": "4.5.0",
                "FR": "---6rKDV",
                "N_FR": "---6rKDV",
                "NUMA_NODE": -1,
                "PCIE_ADDR": "0000:2d:00.0"
        }
 ],
 "AIs": [
        {
                "NUMBER": 1,
                "INDEX": 0,
                "LOAD": 0,
                "MODEL_LOAD": 0,
                "FW_LOAD": 0,
                "INST": 0,
                "MAX_INST": 128,
                "MEM": 0,
                "CRITICAL_MEM": 0,
                "SHARE_MEM": 11,
                "P2P_MEM": 0,
                "DEVICE": "/dev/nvme0n1",
                "L_FL2V": "4.5.0",
                "N_FL2V": "4.5.0",
                "FR": "---6rKDV",
                "N_FR": "---6rKDV",
                "NUMA_NODE": -1,
                "PCIE_ADDR": "0000:2d:00.0"
        }
 ],
 "nvmes": [
        {
                "NUMBER": 1,
                "INDEX": 0,
                "LOAD": 0,
                "MODEL_LOAD": 0,
                "FW_LOAD": 34,
                "INST": 0,
                "MAX_INST": 0,
                "MEM": 0,
                "CRITICAL_MEM": 0,
                "SHARE_MEM": 11,
                "P2P_MEM": 0,
                "DEVICE": "/dev/nvme0n1",
                "L_FL2V": "4.5.0",
                "N_FL2V": "4.5.0",
                "FR": "---6rKDV",
                "N_FR": "---6rKDV",
                "NUMA_NODE": -1,
                "PCIE_ADDR": "0000:2d:00.0"
        }
 ],
 "tps": [
        {
                "NUMBER": 1,
                "INDEX": 0,
                "LOAD": 0,
                "MODEL_LOAD": 0,
                "FW_LOAD": 5,
                "INST": 0,
                "MAX_INST": 0,
                "MEM": 0,
                "CRITICAL_MEM": 0,
                "SHARE_MEM": 11,
                "P2P_MEM": 0,
                "DEVICE": "/dev/nvme0n1",
                "L_FL2V": "4.5.0",
                "N_FL2V": "4.5.0",
                "FR": "---6rKDV",
                "N_FR": "---6rKDV",
                "NUMA_NODE": -1,
                "PCIE_ADDR": "0000:2d:00.0"
        }
 ]
}
```

Below is an example of detail encoder/decoder output in the '–o json'

```
$ ni_rsrc_mon -d -o json
NI resource init'd already ..
**************************************************
1 devices retrieved from current pool at start up
Thu Sep 28 21:30:50 2023 up 00:00:00 v---6kDEV
{ "decoder" :
        [
                {
                        "NUMBER": 1,
                        "INDEX": 0,
                        "AvgCost": 66,
                        "FrameRate": 30,
                        "IDR": 3,
                        "InFrame": 101,
                        "OutFrame": 100,
                        "DEVICE": /dev/nvme0n1,
                },
                {
                        "NUMBER": 1,
                        "INDEX": 1,
                        "AvgCost": 66,
                        "FrameRate": 30,
                        "IDR": 3,
                        "InFrame": 104,
                        "OutFrame": 102,
                        "DEVICE": /dev/nvme0n1,
                },
        ]
}
```

```
{ "encoder" :
        [
                {
                        "NUMBER": 1,
                        "INDEX": 0,
                        "AvgCost": 100,
                        "FrameRate": 30,
                        "IDR": 1,
                        "InFrame": 98,
                        "OutFrame": 94,
                        "BR": 24714720,
                        "AvgBR": 25344089,
                        "DEVICE": /dev/nvme0n1,
                },
                {
                        "NUMBER": 1,
                        "INDEX": 1,
                        "AvgCost": 100,
                        "FrameRate": 30,
                        "IDR": 1,
                        "InFrame": 96,
                        "OutFrame": 92,
                        "BR": 17492160,
                        "AvgBR": 25596965,
                        "DEVICE": /dev/nvme0n1,
                },
        ]
}
```

Each Quadra subsystem is listed with its current status and usage details. Some example descriptions are given below.

INDEX 0 There is 1 Quadra device in the system

DEVICE path is **/dev/nvme0**

NAMESPACE is **/dev/nvme0n1**

LOAD (Decoder) is at 19%

LOAD (Encoder) is at 35%

INST (Decoder) shows 1 decoding instances in progress

INST (Encoder) shows 1 encoding instances in progress

In addition to the Encoder and Decoder load %, the CPU usage and memory usage on any Linux host can also be measured using the **htop** and **smem** applications respectively

5. FW Authentication and Bridging

NETINT cryptographically signs its firmware images to prevent the installation of corrupted or tampered firmware.

When NETINT compiles the Quadra firmware image, a digital signature is generated for the firmware image using NETINT's private key. This digital signature is then embedded in the firmware binary file (Example filename for 4.2.0 release would be **nor_flash_v4.2.0-Quadra.bin**).

Within NETINT's firmware is also a stored copy of NETINT's public key. When a firmware image is transferred from the host to Quadra for upgrade consideration, the executing firmware on Quadra will validate the digital signature of the firmware image under consideration. It does this by using the public key stored within the executing firmware on the device. If the firmware image under consideration is validated successfully and passes all other checks, then it will be accepted by the Quadra device as the next firmware image. A power-cycle for cold upgrade, or reset command for warm/temp-upgrade will then cause the new firmware image to be loaded and executed by the device.

The private key used to sign the new firmware image must be compatible with the public key stored in the existing firmware image. This protection prevents Quadra from being loaded with unofficial or corrupted firmware.

6. Encoding, Decoding and Transcoding testing

Once the previous sections for installation are complete, the host and Quadra device are now ready for use.

The user can now run the **run_ffmpeg_quadra.sh** script to test basic encoding, decoding and transcoding features.

To test the systems, change directory to the FFmpeg root folder and run the following command line.

```
$ bash run_ffmpeg_quadra.sh
```

run_ffmepg_quadra.sh example

After running script the test menu above is shown. Select tests to run (1 to 18)

1. **check pci device :** Will list all NETINT PCIe devices on the system.
2. **check nvme list:** Will call “sudo nvme list” listing all installed NVMe devices.
3. **rsrc_init:** This will call ../libxcoder/build/init_rsrc to initialize all Quadras.
4. **ni_rsrc_mon:** This runs the resource monitor.
5. **test 264 decoder:** Tests h264->yuv decoding function. After the decoding is finished it will display the output.yuv.
6. **test 265 decoder:** This item will test the h265->yuv decoding function. After the decoding is finished it will display the output.yuv file.
7. **test vp9 decoder:** This item will test the vp9->yuv decoding function. After the decoding is finished it will display the output.yuv.
8. **test 264 encoder:** This option will test the yuv->h264 encoding function. After the encoding is finished it will display the output.264 file.
9. **test 265 encoder:** This item will test the yuv->h265 encoding function. After the encoding is finished it will display the output.265 file.
10. **test av1 encoder:** This item will test the yuv->av1 encoding function. After the encoding is finished it will display the output.av1 file.
11. **test 264->265 transcoder:** This item will test the h264 to h265 transcoding function. After the transcoding is finished it will display the output.265 file.
12. **test 264->av1 transcoder:** This option will test the h264 to av1 transcoding function. After the transcoding is finished, it will display the output.av1 file.
13. **test 265->264 transcoder:** This item will test the h265 to h264 transcoding function. After the transcoding is finished it will display the output.264 file.
14. **test 265->av1 transcoder:** This item will test the h265 to av1 transcoding function. After the transcoding is finished it will display the output.av1 file.
15. **test vp9->264 transcoder:** This item will test the vp9 to h264 transcoding function. After the transcoding is finished it will display the output.264 file.
16. **test vp9->265 transcoder:** This item will test the vp9 to h265 transcoding function. After the transcoding is finished it will display the output.265 file.
17. **test vp9->av1 transcoder:** This item will test the vp9 to av1 transcoding function. After the transcoding is finished it will display the output.av1 file.
18. **Quit:** This will quit this menu and return the user to the command line.

All FFmpeg output messages from encoding, decoding or transcoding, will be output to the **ffmpeg.log** file contained in the same folder as the record.

NOTE: Between different releases, the encoded file size may be slightly different from the number recorded by the **run_ffmpeg_quadra.sh** script. This is normal and expected.

7. HW Frames vs SW Frames

Hardware transcoding is supported in Quadra, and this can substantially increase performance for certain workloads.

Note : Please refer to the application note **APPS548-Codensity Quadra YUVbypass** for more details.

To enable YUVbypass and use hardware transcoding, use the following parameter

-xcoder-params "out=hw"

Please refer to the example below.

Hwframes On (YUVbypass On):

If

Hwframes Off (Explicit, same as traditional transcoding):

ffmpeg -vsync 0 -c:v h265_ni_quadra_dec -dec 0 -xcoder-params "out=sw" -i input.h265 -c:v h264_ni_quadra_enc -enc 0 -xcoder-params "RcEnable=1:bitrate=7500000" output.h264 -y

8. NETINT FFmpeg Patch Overview

This section provides an overview of the included FFmpeg patch and its components.

8.1 NETINT Codec Library

The *NETINT Codec Library* is a software module that provides a codec API for users to control and operate NETINT's ASIC codecs. The low level details of the codecs are abstracted to form a simplified API. The codec library containing the API is a standalone library that can be easily integrated into any user application.

The source code for the library is located in the directory ***libxcoder/source/*.***

Please refer to the subsection in section 2 of this document entitled "**Build FFmpeg with NETINT Codec Library**" for compilation instructions.

8.2 NETINT Libavcodec

The *NETINT Codec Library* is supported in the FFmpeg application through the standard FFmpeg libavcodec interface layer. The library supports the NETINT decoder and encoder. This FFmpeg NETINT libavcodec interface layer allows complete integration of the NETINT codec into an FFmpeg application.

The NETINT libavcodec source code is located in the ***FFmpeg/libavcodec/*** directory and contains the following files:

File	Description
nicodec.c, nicodec.h	Common code for Netint decoders and encoders.
nidec.c, nidec.h	Common code for Netint decoders.
nienc.c, nienc.h	Common code for Netint encoders.
nidec_h264.c, nidec_hevc.c, nidec_jpeg.c, nidec_vp9.c	Libavcodec definitions for individual Netint decoder codecs.
nienc_h264.c, nienc_hevc.c, nienc_jpeg.c, nienc_av1.c	Libavcodec definitions for individual Netint encoder codecs.

Please refer to section 2, "**Build FFmpeg with NETINT Codec Library**" for full build instructions.

8.3 NETINT FFmpeg Changes

NETINT has added many bug fixes and updates to the base FFmpeg code. NETINT has also added support for the NETINT libavcodec, backported support for many features (e.g. HLG), and also added changes to improve overall performance. For a list of changes that NETINT applies to FFmpeg, please refer to the ***NETINT_FFMPEG_CHANGE_LIST.txt*** file contained in the software release package, ***Quadra_SW_V*.tar.gz***.

9. Windows Host Installation

Quadra is supported on Windows hosts.

9.1 Operating System

Windows operating systems are also supported. Please see the previous section **2.5 Operating System Compatibility** for a complete list of versions.

9.2 Installing and running FFmpeg using MSYS2

The **InstallationGuideQuadra** document included in the release package contains all the details for installing and configuring for the MSYS2 environment. The installation guide also explains how to compile and run libxcoder and FFmpeg on a Windows host.

9.3 Compiling libxcoder and FFmpeg using Visual Studio 2019

The **InstallationGuideQuadra** document included in the release package explains how to compile libxcoder and FFmpeg using Visual Studio 2019.

10. TROUBLESHOOTING

When attempting to decode, encode or transcode, if the following error message appears, then please remember to initialize all the Quadra resources. The initialization must be performed every time the system has is booted.

```
Error shm_open NI_SHM_CODERS ..: No such file or directory
```

To initialize all Quadra resources, run the following command

```
$ init_rsrc
```

If it is necessary to forcefully re-initialize Quadra resources (e.g. after the libxcoder is updated without a subsequent reboot) then please execute the following :

```
$ rm /dev/shm/NI_*
$ init_rsrc
```

11. NETINT AI Overview

This section provides an overview of the NETINT AI NPU (Neural Processing Unit).

NETINT provides a complete end-to-end pipeline to utilize Quadra's 18 TOPS AI inference capability.

11.1 NETINT AI Toolkit

The NETINT AI Toolkit provides an Artificial Intelligence development environment for the user to import, quantize, validate, optimize, and export, deep learning models.

To install the NETINT AI Toolkit, please download and **untar** the installation package called **Quadra_AI_V*.tar**

Follow the instruction under the folder **documents/**

There are also some examples located in the folder **examples/**

These examples demonstrate the complete workflow, starting with a model import from a third-party framework. The result of the workflow is an optimized network binary graph (NBG), which can be deployed on Quadra’s Neural Processing Units (NPUs).

For more detailed information, please refer to the complete **NETINT AI Toolkit User Guide** included in the release package.

11.2 NETINT AI Performance Evaluation Tool

The NETINT AI toolkit also includes a performance evaluation tool called **aiperf**. The user can use this tool to test the inference performance of Quadra. The tool can also be used to evaluate the accuracy of the network binary graph on Quadra's hardware.

Note : The **aiperf** tool uses the API from the libxcoder, which should be installed on the system in advance.

The **aiperf** tool is contained in the folder **aiperf/**

For more detailed information, please refer to the complete **NETINT AI Toolkit User Guide** included in the release package.

11.3 NETINT AI Python Inference API

For easy integration and fast prototyping, the NETINT AI toolkit provides Python APIs for inferencing with Quarda's NPU. The Python Inference API follows TensorFlow Lite's APIs (TFLite).

There is some example Python code located under folder **network_wrapper/**

For more detailed information about the Python Inference API, please refer to the **Quadra_AI_Python_Inference_API_Quick_Start_V*.pdf**

12. Release Package Files

Netint FW/SW release packages contains with the following files:

File	Description
clamscan.log	Clam AV scan log
InstallationGuideQuadra_V*.pdf	Netint SW installation guide for various systems
IntegrationProgrammingGuideQuadra_V*.pdf	Netint FW/SW primary usage guide including list of xcoder-params
libxcoder_API_Integration_guideQuadra_V*.pdf	Libxcoder API integration guide
md5sum	MD5 checksum of files
NETINT_AI_Toolkit_Quick_Start_V*.pdf	AI toolkit installation and introduction guide
NETINT_AI_Toolkit_User_Guide_V*.pdf	AI toolkit integration guide
Performance_Test_Report_V*.pdf	FW/SW performance test results
Quadra_AI_Python_Inference_API_Quick_Start_V*.pdf	AI toolkit inference feature introduction guide
Quadra_AI_V*.*.tar.gz	AI toolkit release tarball
Quadra_FW_V*.*.*.tar.gz	FW release tarball
Quadra_FW_V*.*.*_release_notes.txt	FW release notes
quadra_quick_installer.sh	FW/SW guided installation script for linux
Quadra_SW_V*.*.*.tar.gz	SW release tarball
Quadra_SW_V*.*.*_release_notes.txt	SW release notes
Quadra_Video_Quality_Report_V*.pdf	FW/SW encoding quality test results
QuickStartGuideQuadra_V*.pdf	FW/SW installation and introduction guide
README.md	Information about release package contents
sentinelscan.log	SentinelOne AV scan log

13. Version Numbers Schema

The Netint Quadra 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 maintains backward compatibility between FW and libxcoder in all releases.

13.4 Libxcoder API Version Number

The *libxcoder API version number* is a semantic major and minor version number pair that is used to track the libxcoder public API compatibility with all 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 always recommended to recompile applications linked to the libxcoder when updating to any new software *release version*.

14. Useful Documents and References

Please contact NETINT support for access to documents and guides, that support and explain specific types of applications. NETINT has a wide collection of Application Notes and other documents available on request.

Some examples Application Notes are

1. Using low latency mode
2. Configuring SRIOV, etc. etc.

15. Appendix A - Upgrading the Firmware manually on a Linux host

Before upgrading the firmware, please ensure the following 2 requirements are satisfied :

1. All Quadra devices are detected by the following command:

   `$ sudo nvme list`

2. The Quadra firmware release package is available (i.e. **Quadra_FW_V2.4.0_RC1.tar.gz**)

Once the above requirements are verified, please use the following steps to upgrade the firmware:

1. Use the GNU Tar tool to extract the update script and binaries from the firmware package tarball (i.e. **Quadra_FW_V2.4.0_RC1.tar.gz**)

   $ tar -xvf <path_to_FW_release_tarball>

2. From a command line terminal, enter the extracted folder and execute the Bash upgrade script as follows:

   $ cd <path_to_FW_release_folder>./quadra_auto_upgrade.sh

3. The upgrade script scans the host system to look for Quadra devices, it then proceeds to prompt the user for an upgrade confirmation for each one:

   Do you want to upgrade ALL of the above Quadra Transcoder(s)? Press [y/n]:

   NOTE: Only PCIe physical devices may operate FW upgrade. PCIe virtual devices will be excluded by the upgrade script.

   NOTE: The User must **stop all transcoding processes** before performing a firmware upgrade, **otherwise it may lead to a device malfunction**.

   NOTE: Each Quadra device will check the file integrity of the Firmware binary before committing to use it.

16. Appendix B - Upgrading Firmware manually on a Windows host

The user can perform a cold upgrade on the Quadra device from a **Windows** host command line.

Before upgrading the firmware, please acquire the Quadra firmware release package (i.e. **Quadra_FW_V4.1.0_RC1.tar.gz**).

Once the FW upgrade package is acquired, use the following steps to upgrade the firmware on each device:

1. Extract the contents of the firmware release package tarball (i.e. **Quadra_FW_V4.1.0_RC1.tar.gz**). Use a tool compatible with GNUzip and Tarball formats such as “7-zip”

2. From a CMD terminal with administrator privileges, determine some key device information using the following command:

```
$ wmic diskdrive get Name,Model,FirmwareVersion,SerialNumber
```

```
C:\msys64\home\nvme>wmic diskdrive get Name,Model,FirmwareRevision,SerialNumber
FirmwareRevision  Model              Name              SerialNumber
30051rc1          QuadraT1A          \\.\PHYSICALDRIVE1  Q1A10BA11FC060-0010_00000001.
CC61              ST1000DM003-1ER162 \\.\PHYSICALDRIVE0  Z4Y5F07G
```

Note the “`Name`” of the device you wish to upgrade (eg. \\.\PHYSICALDRIVE1)

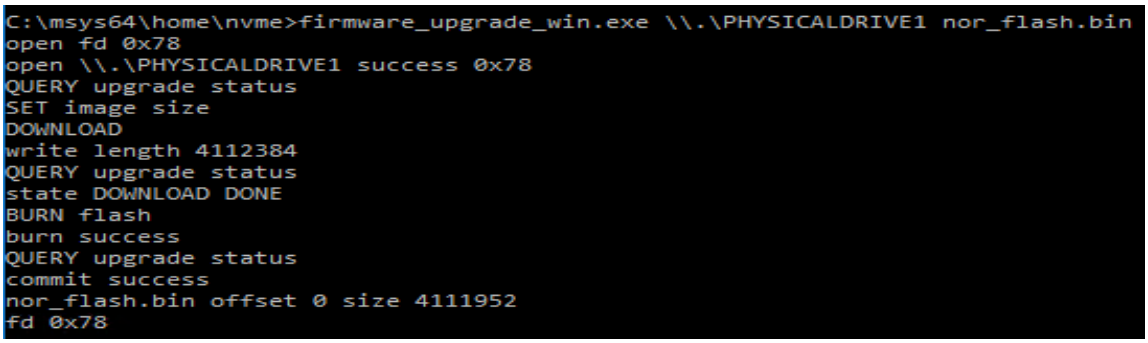
3. Run **firmware_upgrade_win.exe** to initiate the upgrade. It requires two arguments:

   i. The Name of the device (e.g. `\\.\PHYSICALDRIVE1`)
   ii. The Location of the **nor_flash.bin** (e.g. `.\FL_BIN\nor_flash.bin`)

   To upgrade, use the following command:

   $.\firmware_upgrade_win.exe \\.\PHYSICALDRIVE1 .\FL_BIN\nor_flash.bin

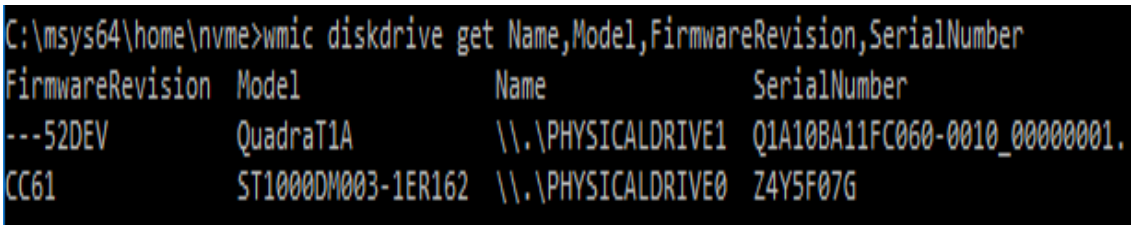
   Wait for about 2 minutes for the command to complete.



4. After the command execution completes, please reboot the host.

5. Confirm that the firmware version is upgraded by running the command.

   $ wmic diskdrive get Name,Model,FirmwareVersion,SerialNumber



CAUTION - Do not interrupt the upgrade process or send any commands to the card while the upgrade is in progress.

17. APPENDIX C - UPGRADING THE FIRMWARE MANUALLY FROM A MACOS HOST

A user can perform a cold upgrade on the Quadra device from a MacOS host terminal.

Before upgrading the firmware, please acquire the Quadra firmware release package (i.e. **Quadra_FW_4.2.0_RC3.tar.gz**).

Once the FW upgrade package is acquired, use the following steps to upgrade the firmware:

1. Use the GNU Tar tool to extract the update script and binaries from the firmware package tarball (i.e. **Quadra_FW_V4.2.0_RC3.tar.gz**)

   $ tar -xvf <path_to_FW_release_tarball>

2. List all the Quadra devices detected by running **the ni_rsrc_mon** command:

   $ sudo ni_rsrc_mon

```
Fri Jan 13 08:34:02 2023 up 00:00:00 v---6FDEV
Num decoders: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE          NAMESPACE
0     0    0          0    0    0         0       /dev/rdisk4    /dev/rdisk4
Num encoders: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE          NAMESPACE
0     0    0          0    0    0         0       /dev/rdisk4    /dev/rdisk4
Num scalers: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE          NAMESPACE
0     0    0          0    0    0         0       /dev/rdisk4    /dev/rdisk4
Num AIs: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE          NAMESPACE
0     0    0          0    0    0         0       /dev/rdisk4    /dev/rdisk4
********************************************
```

   Note the "`DEVICE`" name of the device you wish to upgrade (eg. /dev/rdisk4)

3. From the command line terminal, enter the extracted folder:

   $ cd <path_to_FW_release_folder>

4. Using the same command line terminal, start the **firmware_upgrade_macos** executable. It requires two arguments to perform the firmware upgrade

   i. The Name of the device (e.g. /dev/rdisk4)

   ii. The Location of the **nor_flash.bin** (e.g. ./FL_BIN/nor_flash.bin)

   To upgrade, use the following command:

   ```
   $ ./firmware_upgrade_macos /dev/rdisk4 ./FL_BIN//nor_flash.bin
   ```

   ```
   nvme@CLI458 ~ % sudo ./firmware_upgrade_macos /dev/rdisk4 nor_flash.bin
   This program performs cold FW upgrade for Netint Quadra devices
   example usage: firmware_upgrade_macos /dev/rdisk4 nor_flash.bin
   open /dev/rdisk4 success 0x3
   QUERY upgrade status
   SET image size
   DOWNLOAD
   write length 4091904
   QUERY upgrade status
   state DOWNLOAD DONE
   BURN flash
   burn success
   Wait 100 s
   QUERY upgrade status
   commit success
   nor_flash.bin offset 0 size 4090620
   ```

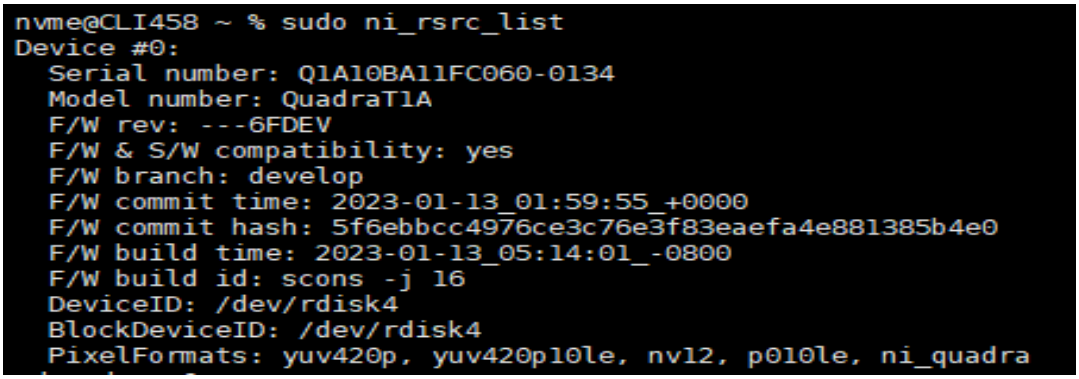
   Wait for about 2 minutes for the command to complete.

5. After the command completes its execution please reboot the host.

   sudo reboot now

6. Confirm that the firmware version has been successfully upgraded, by running the following command:

```
$ sudo ni_rsrc_list
```



The above output shows the F/W rev has been upgraded to ---6FDEV.

Perform these steps for each Netint device in the host.

CAUTION - Do not interrupt the upgrade process or send any commands to the card while the upgrade is progress.