クアドラ クイックスタートガイド V5.6

※以下の内容は、英語原文（QuickStartGuideQuadra_V5.6.pdf）より抄訳されたものです。抄訳が原文に相違する場合はすべて原文が優先します。

目次

1. 法的通知

この文書に記載されている情報は、NETINT（ネットイント）製品に関連して提供されています。本書によって、禁反言その他の法理に基づく明示的または黙示的な知的財産権のライセンスは一切付与されません。NETINT製品の販売に関するNETINTの販売条件に定められている場合を除き、NETINTは一切の責任を負わず、NETINT製品の販売および使用に関して、特定目的への適合性、商品性、または特許、著作権その他の知的財産権の侵害に関する明示的または黙示的な保証を否認します。

「ミッションクリティカルアプリケーション」とは、NETINT製品の故障が直接的または間接的に人身傷害または死亡につながる可能性のあるアプリケーションを指します。NETINT製品をこのようなミッションクリティカルアプリケーションに使用または購入する場合、購入者は、NETINTおよびその子会社、下請業者、関連会社、ならびにそれぞれの取締役、役員、従業員に対して、製品責任、人身傷害、死亡に関する一切の請求、費用、損害、経費、合理的な弁護士費用から直接的または間接的に生じる損害について補償し、免責するものとします。これには、NETINTまたはその下請業者がNETINT製品またはその部品の設計、製造、警告において過失があった場合も含まれます。

NETINTは、仕様、技術文書、製品説明を予告なく変更することがあります。本書の情報は予告なく変更される可能性があります。この情報をもとに設計を確定しないでください。本書に記載されている製品には、公開された仕様と異なる動作を引き起こす可能性のある設計上の欠陥やエラー（エラッタ）が含まれている場合があります。

NETINT™、Codensity™、Quadra™（クアドラ）およびNETINTロゴは、NETINT Technologies Inc.の商標です。その他の商標または登録商標は、それぞれの所有者に帰属します。

2. NETINTの概要

NETINTは、x86およびArmサーバー向けの高性能・低遅延・リアルタイムのビデオ処理ユニット（VPU）を提供するサプライヤーです。複数ストリームのトランスコーディング機能およびサービスを、動画コンテンツプロバイダーに直接提供することが可能です。また、動画配信システムやサービスに統合可能なトランスコーディング・アズ・ア・サービス（TaaS）も提供しています。当社の機能およびサービスは、高効率なビデオ・オン・デマンド（VOD）ファイルのトランスコーディングだけでなく、リアルタイムのライブ動画配信アプリケーションにも活用できます。

2.1　このドキュメントの目的

このクイックスタートガイドは、NETINTのビデオトランスコーディングユニットを迅速にセットアップし、使用を開始する方法の概要を提供します。

本ガイドでは、重要なトランスコーディングパラメータのいくつかと、それらをどのように設定・使用して、他のNETINTサービスやソリューションと統合するかについて説明します。

このクイックスタートガイドでは、FFmpegと**Linuxオペレーティングシステム**を使用しています。

AndroidやWindowsなどの他のOS、Docker/コンテナ、仮想マシン（VM）を使用する環境や構成の詳細については、リリースパッケージに含まれているより詳細な **InstallationGuideQuadra_V*.pdf** を参照してください。

2.2　インストールと互換性

すべてのハードウェア、ファームウェア、およびソフトウェアのインストールは、本書の指示に従って行う必要があります。Quadraビデオ処理ユニットは、本書に記載された構成詳細に基づいてテストされなければなりません。また、すべての構成部品は、本書に記載された内容に準拠している必要があります。これには、オープンソースソフトウェアのバージョンや、本セクションで詳述されているハードウェアも含まれます。

2.3 互換性

このクイックスタートガイドは、以下のコンポーネントに対応しています。

ソフトウェア

本ガイドは、NETINT Quadra Video Transcoder ソフトウェアリリース 5.6 以降に対応しています。

ハードウェア

本ガイドは、以下の NETINT Quadra Video Transcoder ハードウェアに対応しています。

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

2.4 ハードウェアのインストール

Quadraビデオトランスコーダは、ビデオトランスコーディングタスクの実行時に最小限のCPUリソースしか必要としません。

推奨システム構成
プロセッサ：x64 Intel/AMD 6 Core（12 スレッド） **システムメモリ**： **T1A/T1U**：16GiB（8GiB x 2）2133Mhz DDR4 RAM **T2A**：32GiB（16GiB x 2）2133Mhz DDR4 RAM **マザーボード**：PCIe Gen 4（PCIe Gen 5を使用する場合はBIOSでPCIe Gen 4に設定） **T1U**：U.2 x4スロットが必要 **T1A**：x8 または x16スロット（x4リンクアップ） **T2A**：x8スロット（Bifurcation x4/x4）または x16スロット（x4/x4/x4/x4 または x4/x4/x8）。接続された8レーンは、x4/x4リンクアップに設定する必要があります。残りの8レーンは影響しません。 <u>注意</u> 1. PCIe Gen 3も使用可能ですが、性能が劣るため推奨されません。 2. GPU最適化サーバーの使用を推奨します（エアフロー冷却の向上のため）。例：Dell PowerEdge C4140 3. U.2カード（T1U）のみホットプラグに対応しています。AICカード（T1A/T2A）はホットプラグ非対応です。 4. **T2AのすべてのPCIeスロットでは、ホストBIOSでの Bifurcation 設定が必要です。**

Quadraは、Non-Volatile Memory Express（NVMe）サーバーテクノロジーを活用しています。NVMeは、サーバーの高速なPCI Express（PCIe）バスを介して、高速なデータ転送を可能にするホストコントローラインターフェースおよびストレージプロトコルです。Quadraは、ホストサーバーに直接接続できるように設計されています。

Quadraモジュールは、U.2およびAdd-In-Card（AIC）のフォームファクターで提供されています。AICカードはホットプラグに対応しておらず、挿入後にホストシステムで認識されるためには、サーバーのコールドブート（電源を切って再起動）が必要です。一方、QuadraのU.2カードはホットプラグに対応しています。

2.5 オペレーティングシステムの互換性

ホストサーバーには、以下のいずれかのオペレーティングシステムの使用が推奨されます：

- Ubuntu 16.04；カーネル 4.4.0
- Ubuntu 16.04；カーネル 4.10.0
- Ubuntu 18.04；カーネル 4.15.0
- Ubuntu 20.04；カーネル 5.4.0
- Ubuntu 22.04；カーネル 5.15.0
- Ubuntu 22.10；カーネル 5.15.0
- Ubuntu 22.04；カーネル 6.8.0
- Ubuntu 24.04；カーネル 6.8.0
- CentOS 7.9；カーネル 3.10.0
- CentOS Stream リリース 8；カーネル 4.18.0
- OS CentOS 9; カーネル 5.14.0
- OS Debian 12; カーネル 6.1.0
- OS Rocky Linux 9.5; カーネル 5.14.0

以下の Windows オペレーティングシステムバージョンにも対応しています。詳細はセクション9をご参照ください。

対応しているWindowsオペレーティングシステム:

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

以下のmacOSオペレーティングシステムにも対応しています:

- macOS 12.6
- macOS 13.0.1

すべてのオペレーティングシステム(Androidを含む)に関する完全なサポート情報については、**InstallationGuideQuadra_V.pdf*** をご参照ください。

2.6 スクリプトによるファームウェア／ソフトウェアのインストール

quadra_quick_installer.sh スクリプトは、Quadraデバイスへのファームウェアのインストール、およびホストシステムへのソフトウェアのデフォルト構成のインストールを行うために提供されています。

注意:このクイックインストールスクリプトは、Ubuntu、CentOS、macOSでサポートされています。WindowsやAndroidではサポートされていません。

クイックインストールスクリプトは、NETINTのリリースパッケージの**最上位ディレクトリ**にあり、圧縮されたファームウェアおよびソフトウェアのリリースファイル(**.zip形式**)と一緒に含まれています。

1. リリースパッケージから以下のファイルをホストシステムにコピーしてください。圧縮ファイル名の構文は以下のようになります:

```
Quadra_V5.5.0.zip
```

2. メインのリリースパッケージ zip ファイル(例:Quadra_VN.N.N)を解凍します:

```
$ unzip Quadra_V5.5.0.zip
```

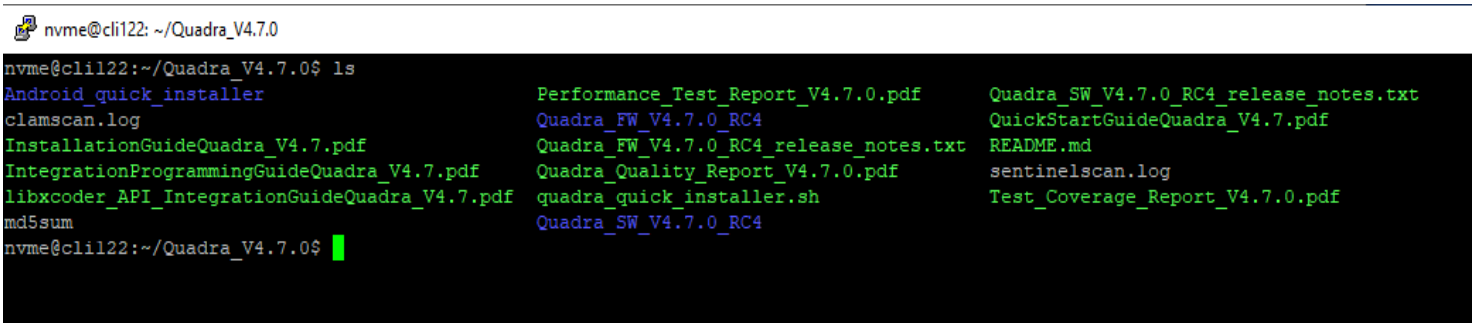
3. 新しく作成された Quadra リリースフォルダーに移動します:

```
$ cd Quadra_V5.5.0
```

4. **ls** コマンドを使って、新しいフォルダー内に解凍されたファイルが表示されることを確認してください。ファイル名はインストールするリリースバージョンによって異なります。

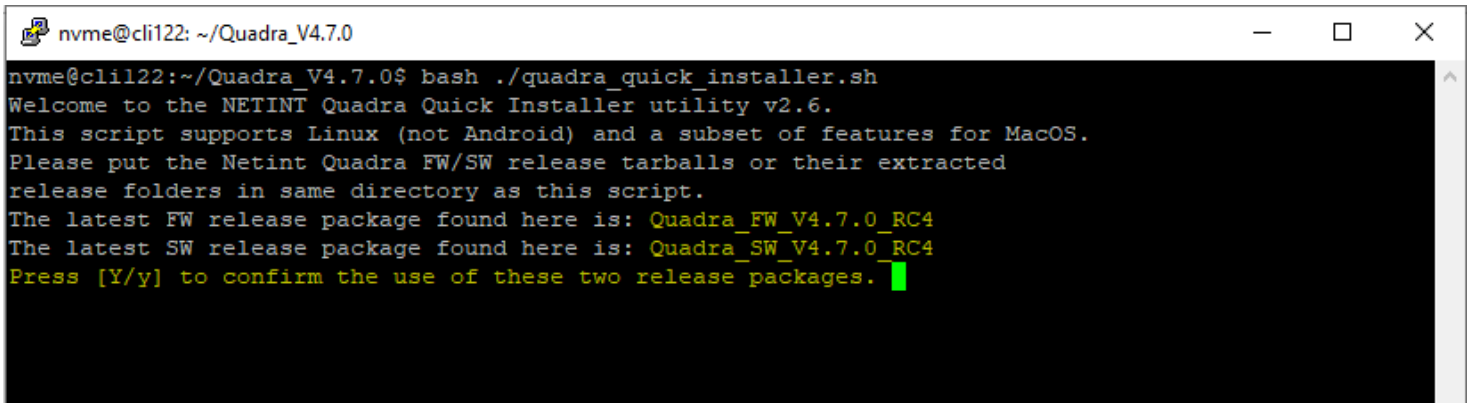
注意:新しいフォルダー内に以下の項目が含まれていることを確認してください:

- **Quadra_FW_VN.N.N_RCN** フォルダー
- **Quadra_SW_VN.N.N_RCN** フォルダー
- **quadra_quick_installer.sh** スクリプト



5. クイックインストーラーのBASHスクリプトを実行します:

```
$ bash quadra_quick_installer.sh
```



インストールスクリプトは、ホストシステムをスキャンして、インストール済みのQuadraデバイスおよびリリースアップグレードパッケージを検出します。その後、インストールするパッケージの確認をユーザーに求めます。

インストール対象として正しいパッケージが選択されていることを確認し、**y**キーを押してください：

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

6. さまざまなフレームワークのセットアップおよびインストールのために、以下のメニューオプションが表示されます。

```
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. **1** を入力して Enter キーを押し、環境変数を設定します。処理が正常に完了したことを確認してください。

```
#? 1
You chose 1 which is Setup Environment variables
Setup Environment variables ran succesfully
```

8. **3** を入力して Enter キーを押し、使用しているOS（Ubuntu／CentOS／macOS）に必要なすべてのソフトウェアライブラリをインストールします。こちらも、警告やエラーが出ていないことを確認してください。

9. **4** を入力して Enter キーを押し、nvme-cli ツールをインストールします。

10. **5** を入力して Enter キーを押し、Libxcoder をインストールします。

11. 使用したい FFmpeg のバージョンに応じて、**6** から **10** のいずれかのオプションを選択して FFmpeg をインストールします。インストール中に、libav 共有ライブラリや Gstreamer サポート、libavcodec インターフェースをコンパイルするかどうかの確認が表示されます。

12. 使用したい Gstreamer のバージョンに応じて、**11** から **14** のいずれかのオプションを選択して Gstreamer をインストールします。前提条件として、FFmpeg バージョンが n4.3.1以上6.1以下で、共有ライブラリおよび Gstreamer サポートが有効になっている必要があります。

    注意：libavcodec における Gstreamer サポートは、FFmpegとは異なるタイムスタンプ処理の挙動に変更される場合があります。

13. **15** を入力して Enter キーを押し、ガイド付きのファームウェアアップグレードプロセスを実行します。

14.16 を入力して Enter キーを押し、スクリプトを終了します。

quadra_quick_installer.sh の実行中に、いずれかのステップで失敗した場合は、出力されたエラーメッセージを確認し、必要な対処を行ってください。問題が解決したら、再度実行してください。エラーが解消されない場合は、NETINTサポートへお問い合わせください。

2.7 手動によるファームウェア／ソフトウェアのインストール

注意：このドキュメントは、Ubuntu 20.04 以降の Linux ホストを前提として記述されています。異なるOSを使用している場合は、InstallationGuideQuadra_V1（またはそれ以降）のセクション3も参照してください。インストールされているOSに特有のパッケージや環境設定手順が記載されています。

以下の手順に従って、必要なパッケージのインストールと環境設定を行ってください。

1. ホストサーバーが Ubuntu を使用している場合、以下のコマンドで必要なパッケージをすべてインストールします：

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

2. /etc/bashrc ファイルに以下の行を追加します：

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

3. 次に、以下のコマンドを実行してファイルを読み込みます：

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

4. /etc/ld.so.conf ファイルに以下の行を追加します：

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

5. その後、以下のコマンドを実行します：

```
$ sudo ldconfig
```

Linux オペレーティングシステムを使用している場合は、インストールされたプログラムを正しく実行できるように、/etc/sudoers ファイルが正しく設定されていることを確認してください：

1. /etc/sudoers ファイル内で、以下の行を探します：

```
Defaults    secure_path =
```

2. この行の値はコロン(:)で区切られています。/usr/local/sbin および /usr/local/bin が secure_path に含まれていない場合は、以下を追加してください：

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

3. /etc/sudoers ファイルに、以下の行も追加します：

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

2.7.1. Quadra ハードウェアのインストール確認

lspci は、システム内の PCIe バスおよびそれに接続されているデバイスの情報を表示するためのユーティリティです。

PCIe 経由でインストールを確認するには、以下のコマンドを実行してください：

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

このコマンドの返り値は、PCIe 経由でシステムにインストールされ、認識されている Quadr

a デバイスの数を示します。

2.7.2. NVMe CLI のダウンロードとインストール

LinuxサーバーにNVMe CLIがデフォルトでインストールされていない場合は、quadra_quick_installer.shを実行する際に4を選択して、特定のNVMe CLIバージョンをインストールすることが推奨されます。

NVMe CLIをインストールするには、以下のコマンドが使用できます。

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

ただし、上記のコマンドでインストールされるバージョンは、NETINTがテストおよび推奨しているバージョン(1.6、1.16)ではない可能性があります。異なるバージョンでも動作する場合がありますが、推奨バージョンとは異なる点に注意してください。

2.7.3. Quadraデバイスの検出

ホストが Quadra デバイスを認識していることを確認するには、以下のコマンドを実行してください:

```
$ sudo nvme list
```

2.7.4. Quadraファームウェアのアップグレード

ファームウェアをアップグレードする前に、以下の要件を満たしている必要があります：

1. Quadra デバイスが以下のコマンドで検出されていること：

```
$ sudo nvme list
```

2. Quadra ファームウェアアップデートパッケージが利用可能であること（例：**Quadra_FW_V4.1.0_RC1.tar.gz**）

これらの要件を確認したら、以下の手順に従ってデバイスのファームウェアをアップグレードしてください。

注意：ファームウェアのアップグレードを行う**前**に、すべてのトランスコーディングプロセスを停止してください。停止せずにアップグレードを行うと、デバイスの故障につながる可能性があります。

1. ファイル名から Quadra ファームウェアリリースパッケージの <version_number> を特定します。

   例：ファイル名が Quadra_FW_V4.1.0_RC1.tar.gz の場合、<version_number> は 4.1.0_RC1 です。

2. Quadra ファームウェアリリースパッケージを展開します：

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

3. 展開されたフォルダー内の quadra_auto_upgrade.sh スクリプトを実行します：

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

一時アップグレードは、ファームウェアを NOR フラッシュに書き込まず、一時的にアップグレードする方法です。デバイスをリセットすると元に戻ります。以下のコマンドで実行できます：

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

4. スクリプトの案内に従ってください。カード上の現在のファームウェア情報と、アップグレード対象の情報が表示されます。アップグレード開始前に確認プロンプトが表示されます。

2.7.5. ファームウェアのアップグレードに関する重要な注意事項

ファームウェアのアップグレードは、PCIe**物理**デバイスのみで実行可能です。PCIe**仮想**デバイスはアップグレードスクリプトによって除外されます。

ファームウェアのアップグレードを行う前に、**すべてのトランスコーディングプロセスを停止**してください。**停止しない場合、デバイスの故障につながる可能性**があります。

各デバイスは、ファームウェアバイナリの使用を開始する前に、ファイルの整合性を確認します。

一時アップグレードを行う前には、デバイスがファームウェアバイナリのバージョンを確認します。この機能を使用するには、NORフラッシュ上のファームウェアバージョンと、アップグレード対象のファームウェアバイナリの**両方**が v5.1.5以上 である必要があります。

一時アップグレードでは、ファームウェアの**フレーバー**も確認されます。新しい一時アップグレード用ファームウェアバイナリと、現在NORフラッシュにロードされているバイナリのフレーバーが一致している必要があります。一致しない場合、一時アップグレードは拒否されます。

アップグレードが開始されたら、絶対に途中で中断しないでください。中断すると、デバイスの故障につながる可能性があります。コールドアップグレード（通常のアップグレード）が失敗した場合は、システムの再起動で問題が解決することがあります。その後、アップグレードを再試行してください。

アップグレードプロセスの詳細ログは、スクリプトによって生成される **./upgrade_log.txt** に記録されます。

ウォームアップグレード（**再起動不要**のアップグレード）は、コールドアップグレードと同様の動作をします。ウォームアップグレードが失敗した場合は、システムの再起動で問題が解決し、そ

の後再試行できます。ただし、ウォームアップグレードが繰り返し失敗する場合は、コールドアップグレードと再起動を行ってください。

一時アップグレードは、デバイスがリセットされると元に戻ります。一時アップグレードはあくまで**一時的な状態**であり、再起動時には NOR フラッシュに保存されているファームウェアがロードされて実行されます。一時アップグレードが失敗しても、ドライブには影響がなく、NORフラッシュのファームウェアが引き続き実行されます。

2.7.6. FFmpeg 対応バージョン

Quadra デバイスとのインターフェースには FFmpeg を使用できます。Quadra が対応している FFmpeg のバージョンは以下の通りです：

- 4.3.1
- 5.1.2
- 6.1
- 7.1
- 8.0.1

注意：Windows および Android では、FFmpeg 4.3.1 のみがサポートされています。

2.7.7. FFmpegのダウンロードとインストール

FFmpeg の公式バージョンは、以下のように FFmpeg のリポジトリに保存されています：

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

コマンドラインから GitHub の FFmpeg リポジトリをクローンするには、セクション 2.4.6 に記載されている対応バージョンの中から希望する <version>（例：2.3.1）を指定して、以下のコマンドを実行してください：

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

例：

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

注意：セクション 2.4.6 に記載されている NETINT がサポートする FFmpeg バージョンのみ をダウンロードしてください。

2.7.8. FFmpegパッチの適用

Quadra VPU で FFmpeg を使用するためには、以下の手順を順番に実行する必要があります。

1. Quadra ソフトウェアリリースパッケージを展開します：

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

2. 展開された SW リリースフォルダーから libxcoder フォルダーを、FFmpeg リポジトリの親フォルダー（FFmpeg フォルダーと同じ階層）にコピーします：

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

3. 展開された SW リリースパッケージから、FFmpeg の公式 <version>（例：4.3.1）に対応するパッチファイルを FFmpeg/ フォルダーにコピーします：

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

例：

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

4. FFmpeg/ フォルダーに移動します：

```
$ cd FFmpeg/
```

5. 該当する NETINT パッチを FFmpeg <version>（例：4.3.1）に適用します：

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

例:

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

これらの手順は、NETINT のリリースパッケージに含まれるソフトウェアに対して適用されるものであり、Quadra Video Processing Unit で FFmpeg を使用するために必要です。

注意:FFmpeg のバージョンに対応した正しいパッチを必ず使用してください。

2.7.9. NETINT コーデック ライブラリを使用した FFmpeg のビルド

NETINT コーデックライブラリを使用して FFmpeg をビルドするには、以下の手順を順番に実行する必要があります。

1. **FFmpeg/** フォルダーから **libxcoder/** フォルダーに移動します：

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

2. 以下のコマンドで libxcoder をビルド・インストールします：

```
$ bash build.sh
```

3. 以下のコマンドで libxcoder 共有ライブラリを ldconfig に読み込ませます：

```
$ sudo ldconfig
```

4. **FFmpeg/** フォルダーに戻ります：

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

5. 以下のコマンドで **build_ffmpeg.sh** スクリプトを実行します（デフォルトで Quadra 対応のビルドになります）：

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

注意：上記の **sudo make clean** コマンドは、以前のビルドファイルが存在しない場合にエラーを返すことがあります。以下のようなエラーメッセージが表示されても、正しいディレクトリにいて、既存のビルドファイルがない場合は問題ありません。

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

6. 以下のコマンドで、Quadra 対応の FFmpeg をシステムにインストールします：

```
$ sudo make install
```

3. セットアップの確認

ハードウェアおよびソフトウェアのインストール手順が完了したら、以下の手順に従って Quadra のセットアップを有効化し、確認してください。

1. 以下のコマンドで NVMe デバイスの存在を確認します：

```
$ sudo nvme list
```

このコマンドにより、システム上の NVMe デバイスの一覧が表示されます。

Model 名に Quadra が含まれているデバイスを探してください。また、再起動後には、ファームウェアのバージョン（FW Rev）が今回のリリースでインストールされた新しいバージョンになっていることを確認してください。

たとえば、40064rcD はリリースパッケージ 4.0.0 に対応するファームウェアバージョンです。

以下に例を示します：

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. FFmpegをroot権限なしで実行するには、ユーザーを disk グループに追加する必要があります：

```
$ sudo usermod -a -G disk $USER
```

※FFmpegをrootとして実行したい場合は、この手順を省略し、代わりに sudo init_rsrc を使用してすべてのQuadraデバイスを初期化できます。

3. サーバーを再起動します(システムが再起動可能な状態であることを確認してください):

```
$ sudo reboot now
```

または、システムの再起動なしでウォーム／一時アップグレードを実行することも可能です。その場合は以下を実行してください:

```
$ ni_rsrc_update -D
```

4. 以下のコマンドでQuadraデバイスを初期化します:

```
$ init_rsrc
```

4. 負荷の監視

Quadraデバイスの処理負荷を監視するには、NETINTのリソースモニタユーティリティを使用してください:

```
$ 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
```

プログラムの使用方法に関するヘルプは、以下のコマンドで確認できます：

```
$ ni_rsrc_mon --help
```

各Quadraデバイスには4つのサブシステムがあります。これらのサブシステムはリソースモニタによって一覧表示されます。

- エンコーダ（encoder）
- デコーダ（decoder）
- スケーラー（scaler）
- AI

以下は、リソースモニタの出力例（デフォルト形式）です。

```
[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
*************************************************
```

エンコーダ／デコーダの詳細情報は、以下のコマンドを使用することで確認できます：

```
$ ni_rsrc_mon -d
```

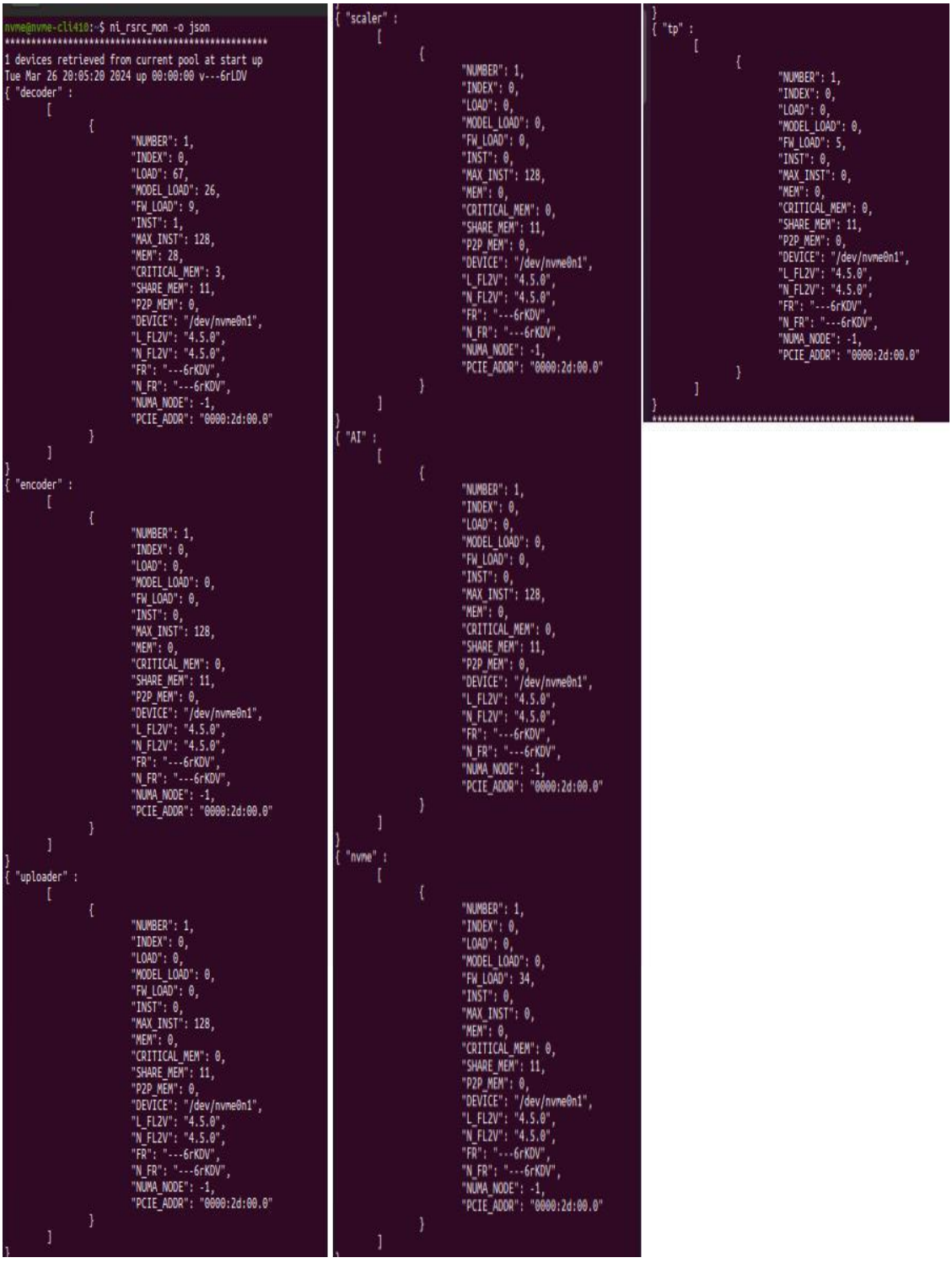
```
$ ni_rsrc_mon -d
NI resource init'd already ..
************************************************
1 devices retrieved from current pool at start up
Thu Sep 28 21:30:44 2023 up 00:00:00 v---6kDEV
Num decoders: 1
INDEX AvgCost FrameRate IDR InFrame OutFrame DEVICE         NAMESPACE
0     66      30        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
```

以下は、リソースモニタのJSON出力例です。出力形式には -o json と -o json1 の2種類があります。

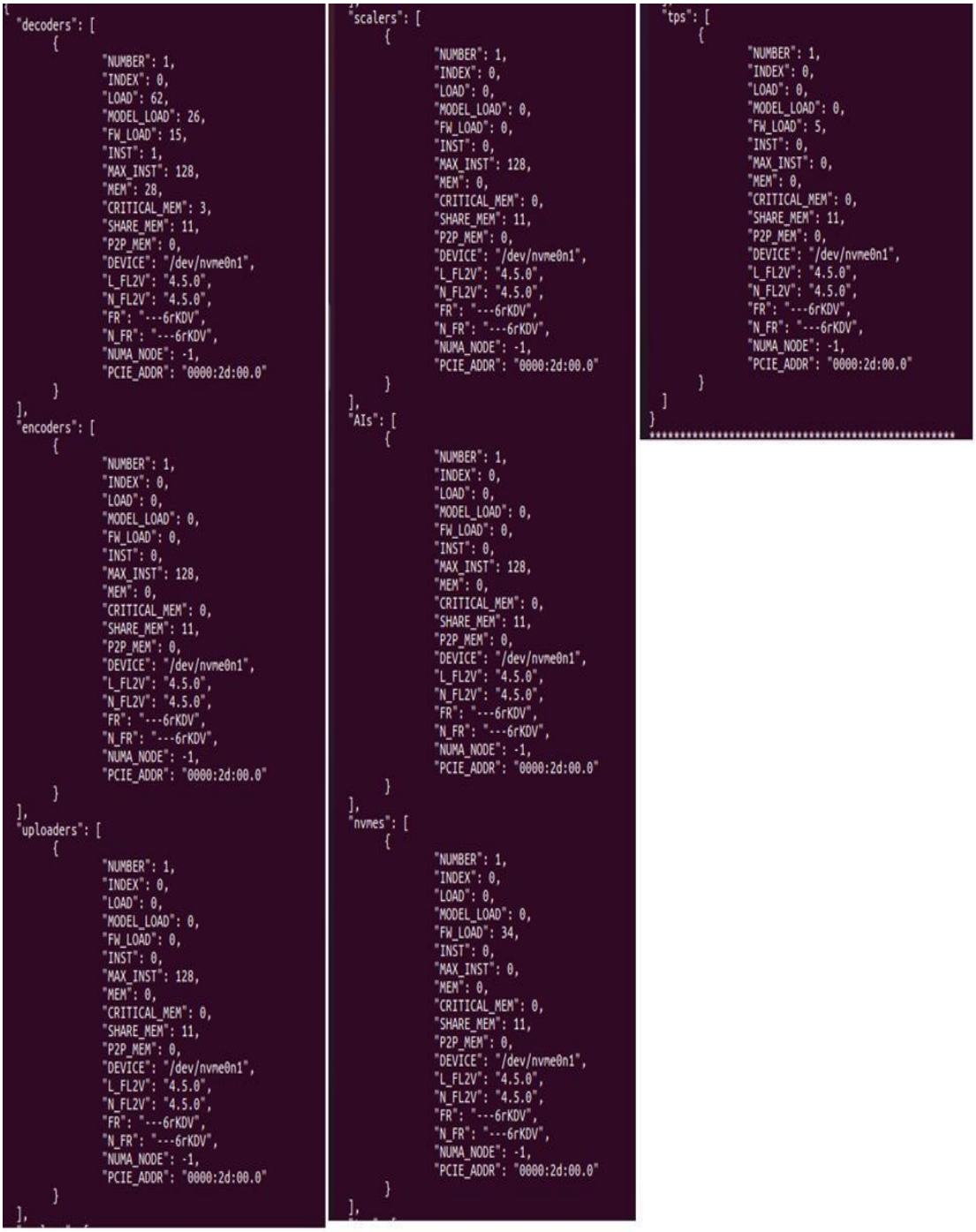
この出力には、Linux専用に追加された2つのパラメータが含まれています。他のプラットフォームでは、これらのパラメータは利用できません。

NUMA_NODE：NUMAノードID
PCIE_ADDR：PCIeアドレス

以下は、-o json オプションを使用したリソースモニタの出力例です。



以下は、-o json1 オプションを使用したリソースモニタの出力例です。



以下は、–o json オプションを使用した、エンコーダ／デコーダの詳細出力例です。

```
$ 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,
                },
        ]
}
```

各Quadraサブシステムは、現在のステータスと使用状況の詳細とともに一覧表示されます。以下にいくつかの説明例を示します。

INDEX 0:システム内に1つのQuadraデバイスがあります

DEVICEパス:/dev/nvme0

NAMESPACE:/dev/nvme0n1

LOAD(Decoder):19%

LOAD(Encoder):35%

INST(Decoder):1つのデコード処理が進行中

INST(Encoder):1つのエンコード処理が進行中

エンコーダおよびデコーダの負荷(%)に加えて、Linuxホスト上のCPU使用率やメモリ使用率も、それぞれ **htop** および **smem** アプリケーションを使用して測定できます。

5. ファームウェアの認証とブリッジング

NETINTは、破損または改ざんされたファームウェアのインストールを防ぐために、ファームウェアイメージに暗号署名を行っています。

NETINTがQuadraファームウェアイメージをコンパイルする際、NETINTの秘密鍵を使用してデジタル署名が生成されます。この署名は、ファームウェアのバイナリファイルに埋め込まれます（例：バージョン4.2.0のファイル名は **nor_flash_v4.2.0-Quadra.bin** など）。

NETINTのファームウェアには、NETINTの公開鍵のコピーも保存されています。ホストからQuadraにファームウェアイメージが転送されると、Quadra上で実行中のファームウェアが、そのイメージのデジタル署名を検証します。検証には、デバイス上の実行中ファームウェアに保存されている公開鍵が使用されます。

ファームウェアイメージが正常に検証され、その他のチェックもすべて通過した場合、そのイメージはQuadraデバイスによって次のファームウェアイメージとして受け入れられます。

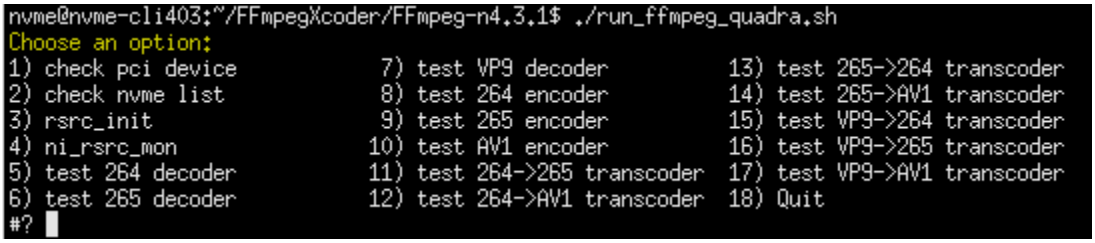
その後、コールドアップグレードには電源の再投入、ウォーム／一時アップグレードにはリセットコマンドを使用することで、新しいファームウェアイメージがロードされ、デバイスによって実行されます。

新しいファームウェアイメージに署名するために使用される秘密鍵は、既存のファームウェアイメージに保存されている公開鍵と互換性がある必要があります。この保護機能により、非公式または破損したファームウェアがQuadraにロードされることを防ぎます。

6. エンコード、デコード、トランスコーディングのテスト

前のセクションでインストールが完了すると、ホストとQuadraデバイスは使用可能な状態になります。
ユーザーは**run_ffmpeg_quadra.sh**スクリプトを実行することで、基本的なエンコード、デコード、およびトランスコード機能をテストできます。
システムをテストするには、FFmpegのルートフォルダーにディレクトリを移動し、以下のコマンドラインを実行してください。

```
$ bash run_ffmpeg_quadra.sh
```



スクリプトを実行すると、上記のテストメニューが表示されます。実行するテストを選択してください(1～18)。

1. **check pci device**:システム上のすべての NETINT PCIe デバイスを一覧表示します。

2. **check nvme list**:sudo nvme list を呼び出し、インストールされている NVMe デバイスを一覧表示します。

3. **rsrc_init**:../libxcoder/build/init_rsrc を呼び出して、すべての Quadra を初期化します。

4. **ni_rsrc_mon**:リソースモニタを実行します。

5. **test 264 decoder**:h264→yuv のデコード機能をテストします。デコード完了後、output.yuv を表示します。

6. **test 245 decoder**:h265→yuv のデコード機能をテストします。デコード完了後、output.yuv ファイルを表示します。

7. **test vp9 decoder**:vp9→yuv のデコード機能をテストします。デコード完了後、output.yuv を表示します。

8. **test 264 encoder**:yuv→h264 のエンコード機能をテストします。エンコード完了後、output.h264 ファイルを表示します。

9. **test 265 encoder**:yuv→h265 のエンコード機能をテストします。エンコード完了後、output.h265 ファイルを表示します。

10. **test av1 encoder**:yuv→av1 のエンコード機能をテストします。エンコード完了後、output.ivf ファイルを表示します。

11. **test 264→265 transcoder**：h264 から h265 へのトランスコード機能をテストします。完了後、output.h265 ファイルを表示します。

12. **test 264→av1 transcoder**：h264 から av1 へのトランスコード機能をテストします。完了後、output.ivf ファイルを表示します。

13. **test 265→264 transcoder**：h265 から h264 へのトランスコード機能をテストします。完了後、output.h264 ファイルを表示します。

14. **test 265→av1 transcoder**：h265 から av1 へのトランスコード機能をテストします。完了後、output.ivf ファイルを表示します。

15. **test vp9→264 transcoder**：vp9 から h264 へのトランスコード機能をテストします。完了後、output.h264 ファイルを表示します。

16. **test vp9→265 transcoder**：vp9 から h265 へのトランスコード機能をテストします。完了後、output.h265 ファイルを表示します。

17. **test vp9→av1 transcoder**：vp9 から av1 へのトランスコード機能をテストします。完了後、output.ivf ファイルを表示します。

18. **test split filter**: split filter 機能をテストします。完了後、output.h264、output.h265、output.ivf ファイルを表示します。

19. **test scaling filter**: scaling filter 機能をテストします。完了後、output_720p.h265、output_540p.h265、output_360p.h265 ファイルを表示します。

20. **test overlay filter**: overlay filter 機能をテストします。完了後、output.h264ファイルを表示します。

21. **Quit**：このメニューを終了し、コマンドラインに戻ります。

エンコード、デコード、またはトランスコードのすべての FFmpeg 出力メッセージは、同じフォルダー内の **output*.log** ファイルに出力されます。

注意：異なるリリース間では、エンコードされたファイルサイズが **run_ffmpeg_quadra.sh** スクリプトによって記録された数値とわずかに異なる場合があります。これは正常であり、想定された挙動です。

7. ハードウェアフレーム vs ソフトウエアフレーム

Quadra ではハードウェアトランスコードがサポートされており、特定のワークロードにおいてパフォーマンスを大幅に向上させることができます。

注意:詳細については、アプリケーションノート **APPS548-Codensity Quadra YUVbypass** を参照してください。

YUVbypass を有効にしてハードウェアトランスコードを使用するには、以下のパラメータを使用してください:

```
-xcoder-params "out=hw"
```

以下の例を参照してください。

Hwframes オン(YUVbypass オン) の場合:

```
ffmpeg -vsync 0 -c:v h265_ni_quadra_dec -dec 0 -xco
der-params "out=hw" -i input.h265 -c:v h264_ni_quad
ra_enc -enc 0 -xcoder-params "RcEnable=1:bitrate=75
00000" output.h264 -y
```

Hwframes オフ(明示的、従来のトランスコードと同様) の場合:

```
ffmpeg -vsync 0 -c:v h265_ni_quadra_dec -dec 0 -xco
der-params "out=sw" -i input.h265 -c:v h264_ni_quad
ra_enc -enc 0 -xcoder-params "RcEnable=1:bitrate=75
00000" output.h264 -y
```

8. NETINT FFmpeg パッチの概要

このセクションでは、同梱されている FFmpeg パッチとその構成要素の概要を説明します。

8.1 NETINT コーデックライブラリ

NETINT コーデックライブラリは、NETINT の ASIC コーデックをユーザーが制御・操作するためのコーデック API を提供するソフトウェアモジュールです。コーデックの低レベルの詳細は抽象化されており、簡素化された API として提供されます。この API を含むコーデックライブラリは、スタンドアロンのライブラリであり、任意のユーザーアプリケーションに容易にインテグレーションできます。

ライブラリのソースコードは、**libxcoder/source/*.*** ディレクトリにあります。

コンパイル手順については、**「2.7.9. NETINT コーデック ライブラリを使用した FFmpeg のビルド」**のサブセクションを参照してください。

8.2 NETINT Libavcodec

NETINT コーデックライブラリは、標準の FFmpeg libavcodec インターフェースレイヤーを通じて FFmpeg アプリケーションでサポートされています。このライブラリは、NETINT のデコーダおよびエンコーダをサポートしています。FFmpeg の NETINT libavcodec インターフェースレイヤーにより、NETINT コーデックを FFmpeg アプリケーションに完全にインテグレーションすることが可能になります。

NETINT libavcodec のソースコードは **FFmpeg/libavcodec/** ディレクトリにあり、以下のファイルが含まれています：

ファイル名	説明
nicodec.c, nicodec.h	NETINTのデコーダおよびエンコーダ用の共通コード。
nidec.c, nidec.h	NETINTデコーダの共通コード。
nienc.c、nienc.h	NETINTエンコーダの共通コード。
nidec_h264.c、nidec_hevc.c、nidec_jpeg.c、nidec_vp9.c	各NETINTのデコーダコーデックの Libavcodec 定義。

FFmpeg のビルド手順については、**「2.7.9. NETINT コーデック ライブラリを使用した FFmpeg のビルド」**を参照してください。

8.3 NETINT FFmpegの変更点

NETINT は、ベースとなる FFmpeg コードに多数のバグ修正とアップデートを加えています。また、NETINT は libavcodec のサポートを追加し、多くの機能（例：HLG）のサポートをバックポートし、全体的なパフォーマンス向上のための変更も加えています。NETINT が FFmpeg に適用した変更の一覧については、ソフトウェアリリースパッケージ **Quadra_SW_V*.tar.gz** に含まれている **NETINT_FFMPEG_CHANGE_LIST.txt** ファイルを参照してください。

9. WINDOWS ホストへのインストール

Quadra は Windows ホストでもサポートされています。

9.1 オペレーティングシステム

Windows オペレーティングシステムもサポートされています。対応バージョンの一覧については、前述のセクション 2.5 **「オペレーティングシステムの互換性」**を参照してください。

9.2 MSYS2 を使用した FFmpeg のインストールと実行

リリースパッケージに含まれている **InstallationGuideQuadra** ドキュメントには、MSYS2 環境でのインストールと設定に関する詳細が記載されています。このインストールガイドでは、Windows ホスト上で libxcoder および FFmpeg をコンパイル・実行する方法についても説明されています。

9.3 Visual Studio 2019 を使用した libxcoder と FFmpeg のコンパイル

リリースパッケージに含まれている **InstallationGuideQuadra** ドキュメントには、Visual Studio 2019 を使用して libxcoder および FFmpeg をコンパイルする方法が記載されています。

10. トラブルシューティング

デコード、エンコード、またはトランスコードを試みた際に、以下のエラーメッセージが表示された場合は、すべての Quadra リソースを初期化する必要があることを忘れないでください。この初期化は、システムが起動されるたびに実行する必要があります。

```
Error shm_open NI_SHM_CODERS ..: No such file or di
rectory
```

すべての Quadra リソースを初期化するには、以下のコマンドを実行してください：

```
$ init_rsrc
```

libxcoder を更新した後に再起動せずに使用するなど、強制的に Quadra リソースを再初期化する必要がある場合は、以下のコマンドを実行してください：

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

11. NETINT AI の概要

このセクションでは、NETINT の AI NPU（ニューラル・プロセッシング・ユニット）の概要を説明します。NETINT は、Quadra の 18 TOPS の AI 推論能力を活用するための、完全なエンドツーエンドのパイプラインを提供しています。

11.1 NETINT AIツールキット

NETINT AI ツールキットは、ユーザーがディープラーニングモデルをインポート、量子化、検証、最適化、エクスポートするための AI 開発環境を提供します。

NETINT AI ツールキットをインストールするには、**Quadra_AI_V*.tar** というインストールパッケージをダウンロードして展開してください。

インストール手順は、**documents/** フォルダー内の説明に従ってください。

また、**examples/** フォルダーにはいくつかのサンプルも含まれています。

これらのサンプルでは、サードパーティのフレームワークからモデルをインポートするところから始まる、完全なワークフローを示しています。このワークフローの結果として得られるのが、Quadra の NPU（ニューラル・プロセッシング・ユニット）上で展開可能な最適化されたネットワークバイナリグラフ（NBG）です。

詳細については、リリースパッケージに含まれている **NETINT AI Toolkit User Guide** を参照してください。

11.2 NETINT AI パフォーマンス評価ツール

NETINT AI ツールキットには、**aiperf** と呼ばれるパフォーマンス評価ツールも含まれています。ユーザーはこのツールを使用して、Quadra の推論性能をテストすることができます。また、Quadra のハードウェア上でネットワークバイナリグラフ（NBG）の精度を評価するためにも使用できます。

注意：aiperf ツールは libxcoder の API を使用しており、事前にシステムにインストールされている必要があります。

aiperf ツールは **aiperf**/ フォルダーに含まれています。

詳細については、リリースパッケージに含まれている **NETINT AI Toolkit User Guide** を参照してください。

11.3 NETINT AI Python 推論 API

簡単な統合と迅速なプロトタイピングのために、NETINT AI ツールキットは Quadra の NPU を使った推論のための Python API を提供しています。この Python 推論 API は、TensorFlow Lite（TFLite）の API に準拠しています。

サンプルの Python コードは **network_wrapper/** フォルダー内にあります。

Python 推論 API の詳細については、リリースパッケージに含まれている **Quadra_AI_Python_Inference_API_Quick_Start_V*.pdf** を参照してください。

12. リリースパッケージファイル

NETINT のファームウェア／ソフトウェアリリースパッケージには、以下のファイルが含まれています：

ファイル名	説明
clamscan.log	Clam AV によるウイルススキャンログ
InstallationGuideQuadra_V*.pdf	各種システム向けの NETINT ソフトウェアインストールガイド
libxcoder_API_Integration_guideQuadra_V*.pdf	NETINT のファームウェア／ソフトウェアの基本的な使用方法と xcoder-params の一覧を含む統合プログラミングガイド
md5sum	Libxcoder API 統合ガイド
NETINT_AI_Toolkit_Quick_Start_V*.pdf	ファイルの MD5 チェックサム
NETINT_AI_Toolkit_Quick_Start_V*.pdf	AI ツールキットのインストールと概要ガイド
NETINT_AI_Toolkit_User_Guide_V*.pdf	AI ツールキット統合ガイド
Performance_Test_Report_V*.pdf	ファームウェア／ソフトウェアのパフォーマンステスト結果
Quadra_AI_Python_Inference_API_Quick_Start_V*.pdf	AI ツールキットの推論機能紹介ガイド
Quadra_AI_V*.*.tar.gz	AI ツールキットのリリース用 tarball
Quadra_FW_V*.*.*.tar.gz	ファームウェアのリリース用 tarball

Quadra_FW_V*.*.*_release_notes.txt	ファームウェアのリリースノート
quadra_quick_installer.sh	Linux 用のファームウェア／ソフトウェアのインストール支援スクリプト
Quadra_SW_V*.*.*.tar.gz	ソフトウェアのリリース用 tarball
Quadra_SW_V*.*.*_release_notes.txt	ソフトウェアのリリースノート
Quadra_Video_Quality_Report_V*.pdf	ファームウェア／ソフトウェアのエンコード品質テスト結果
QuickStartGuideQuadra_V*.pdf	ファームウェア／ソフトウェアのインストールと概要ガイド
README.md	リリースパッケージの内容に関する情報
sentinelscan.log	SentinelOne によるウイルススキャンログ

13. バージョン番号スキーマ

NETINT Quadra のリリースパッケージには、複数のコンポーネントが含まれており、それぞれに独自のリリースバージョン番号と、セマンティック互換性を示す番号があります。

13.1 リリースバージョン番号

NETINT Quadra のリリースバージョン番号（例：4.7.0）は、3つの文字で構成されています：

Major.Minor.Micro（メジャー・マイナー・マイクロ）

メジャーリリース（重要な節目のリリース）の際には、メジャー番号が増加します。開発ブランチからの定期的なリリースでは、マイナー番号が増加します。その他のリリース（例：ホットフィックス）では、マイクロ番号が増加します。

リリースバージョン番号の文字は、0～9および A～Z の範囲で使用されます。

より大きなリリースバージョン番号のリリースは、より小さなバージョン番号のリリースを上書きするものであり、新しいリリースは古いリリースをベースにしています。最新の機能や修正にアクセスするためには、利用可能な中で最も大きなリリースバージョン番号のリリースを使用することが推奨されます。

NETINT のリリースパッケージにはリリースバージョン番号が付けられていますが、ファームウェア（FW）およびソフトウェア（SW）にもそれぞれ独自の3文字のリリースバージョン番号があります。1つのリリースパッケージ内に含まれる FW と SW のリリースバージョン番号が異なる場合、リリースパッケージのバージョン番号は、FW と SW のうち大きい方のバージョン番号になります。

13.2 フルバージョン番号

ファームウェアのリビジョンとソフトウェアアプリケーションの間には、8文字のフルバージョン番号（例：4706r3r4）があります。

ファームウェアのフルバージョン番号は、以下のコマンドで確認できます：

```
sudo nvme list
```

または

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

ソフトウェアのフルバージョン番号は、任意の libxcoder アプリケーションで以下のように引数を指定して確認できます：

```
ni_rsrc_mon -v
```

また、libxcoder/source/ni_defs.h ファイル内のコードにも定義されています。

フルバージョン番号の最初の3文字はリリースバージョン番号です。4～6文字目は FW API バージョン番号です（詳細は下記参照）。最後の2文字は、NETINT がリリース開発を追跡するために使用します。

13.3 FW API バージョン番号

FW および SW アプリケーションのフルバージョン番号（例：4706r3r4）のうち、4～6文字目は FW API バージョン番号を表します。これは、1つのセマンティックメジャー（主）バージョンと2つのマイナー（副）バージョンで構成されており、デバイス上のファームウェアと libxcoder バージョン間の API 互換性を追跡します。基本的な相互運用性を確保するためには、FWとlibxcoderの間でFW APIのメジャーバージョン番号が一致している必要があります。また、マイナーバージョン番号も一致していることで、FW/libxcoderのすべての新機能や拡張機能にアクセス可能になります。

NETINTは、すべてのリリースにおいてFWとlibxcoder間の後方互換性を維持しています。

13.4 Libxcoder API バージョン番号

libxcoder API バージョン番号は、セマンティックなメジャー（主）およびマイナー（副）バージョン番号のペアで構成されており、libxcoder の公開 API が他のリンクされた API（例：libavcodec）やアプリケーション（例：xcoder-util）と互換性を保っているかを追跡するために使用されます。このバージョン番号は、**libxcoder/source/ni_defs.h** から確認できます。

libxcoder API バージョン番号の文字は、厳密に数値のみで構成されます。メジャーおよびマイナーの各部分は、1桁以上の数字になる場合があります。

メジャー **libxcoder API バージョン番号**は、後方互換性のない変更が加えられた場合に増加します。マイナー **libxcoder API バージョン番号**は、新機能が追加され、アプリケーションコードの更新が必要となる場合に増加します。

libxcoder API バージョン番号に変更があった場合でも、新しいソフトウェアリリースバージョンに更新する際には、libxcoder にリンクされたアプリケーションを再コンパイルすることが常に推奨されます。

14. 有用なドキュメントと参考資料

特定の種類のアプリケーションをサポート・説明するドキュメントやガイドへのアクセスについては、NETINT サポートまでお問い合わせください。NETINT では、アプリケーションノートやその他のドキュメントを多数取り揃えており、リクエストに応じて提供可能です。

アプリケーションノートの一例：

1.低遅延モードの使用方法

2. SR-IOVの設定方法など

15. 付録A - LINUXホストでのファームウェアの手動アップグレード

ファームウェアをアップグレードする前に、以下の2つの要件が満たされていることを確認してください：

1. 以下のコマンドで、すべての Quadra デバイスが検出されていること：

```
$ sudo nvme list
```

2. Quadra ファームウェアリリースパッケージが入手可能であること（例：**Quadra_FW_V 2.4.0_RC1.tar.gz**）

上記の要件を確認したら、以下の手順でファームウェアをアップグレードしてください：

1. GNU Tar ツールを使用して、ファームウェアパッケージ tarball（例：**Quadra_FW_V 2.4.0_RC1.tar.gz**）からアップデートスクリプトとバイナリを展開します：

```
$ tar -xvf <path_to_FW_release_tarball>
```

2. ターミナルで展開されたフォルダーに移動し、以下のように Bash アップグレードスクリプトを実行します：

```
$ cd <path_to_FW_release_folder>
$ ./quadra_auto_upgrade.sh
```

3. アップグレードスクリプトはホストシステムをスキャンして Quadra デバイスを検出し、それぞれのデバイスに対してアップグレードの確認を求めます：

```
Do you want to upgrade ALL of the above Quadra Tran
```

```
scoder(s)? Press [y/n]:
```

注意: ファームウェアのアップグレードを実行できるのは、PCIe の物理デバイスのみです。PCIe の仮想デバイスはアップグレードスクリプトによって除外されます。

注意: ファームウェアのアップグレードを行う前に、**すべてのトランスコード処理を停止**する必要があります。**そうしないと、デバイスの不具合につながる可能性があります。**

注意: 各 Quadra デバイスは、ファームウェアバイナリの整合性を確認してから使用を確定します。

16. 付録B - WINDOWS ホストでのファームウェア手動アップグレード

ユーザーは、**Windows** ホストのコマンドラインから Quadra デバイスに対してコールドアップグレードを実行できます。

ファームウェアをアップグレードする前に、Quadra ファームウェアリリースパッケージ（例：**Quadra_FW_V4.1.0_RC1.tar.gz**）を入手してください。

ファームウェアアップグレードパッケージを入手したら、以下の手順で各デバイスのファームウェアをアップグレードします：

1. ファームウェアリリースパッケージ（例：**Quadra_FW_V4.1.0_RC1.tar.gz**）の内容を展開します。GNUzip および Tarball 形式に対応したツール（例：7-Zip）を使用してください。

2. 管理者権限で開いた CMD ターミナルから、以下のコマンドを使用してデバイスの基本情報を確認します：

```
$ 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
```

   アップグレードしたいデバイスの Name（例：￥￥.￥PHYSICALDRIVE1）をメモしてください。

3. **firmware_upgrade_win.exe** を実行してアップグレードを開始します。このツールに

は以下の2つの引数が必要です：

- アップグレード対象のデバイス名（例：¥¥.¥PHYSICALDRIVE1）
- nor_flash.bin の場所（例：.¥FL_BIN¥nor_flash.bin）

アップグレードを実行するには、以下のコマンドを使用します：

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

コマンドの実行が完了するまで、約2分間お待ちください。

```
C:\msys64\home\nvme>firmware_upgrade_win.exe \\.\PHYSICALDRIVE1 nor_flash.bin
open fd 0x78
open \\.\PHYSICALDRIVE1 success 0x78
QUERY upgrade status
SET image size
DOWNLOAD
write length 4112384
QUERY upgrade status
state DOWNLOAD DONE
BURN flash
burn success
QUERY upgrade status
commit success
nor_flash.bin offset 0 size 4111952
fd 0x78
```

4. コマンドの実行が完了したら、ホストを再起動してください。

5. 以下のコマンドを実行して、ファームウェアのバージョンが更新されたことを確認してください。

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

```
C:\msys64\home\nvme>wmic diskdrive get Name,Model,FirmwareRevision,SerialNumber
FirmwareRevision  Model             Name              SerialNumber
---52DEV         QuadraT1A          \\.\PHYSICALDRIVE1  Q1A10BA11FC060-0010_00000001.
CC61             ST1000DM003-1ER162  \\.\PHYSICALDRIVE0  Z4Y5F07G
```

注意:アップグレード処理中は、プロセスを中断したり、カードにコマンドを送信したりしないでください。

17. 付録 C - MacOS ホストでのファームウェア手動アップグレード

ユーザーは、MacOSホストのターミナルからQuadraデバイスに対してコールドアップグレードを実行できます。

ファームウェアをアップグレードする前に、Quadraファームウェアのリリースパッケージ（例：**Quadra_FW_4.2.0_RC3.tar.gz**）を取得してください。

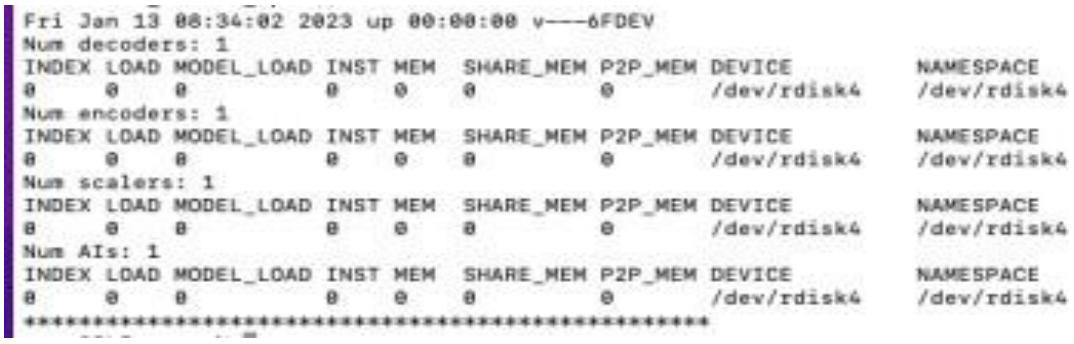
ファームウェアアップグレードパッケージを取得したら、以下の手順でアップグレードを行います：

1. GNU Tarツールを使用して、ファームウェアパッケージ（例：**Quadra_FW_V4.2.0_RC3.tar.gz**）からアップデートスクリプトとバイナリを展開します。

```
$ tar -xvf <FWリリースパッケージのパス>
```

2. **ni_rsrc_mon** コマンドを実行して、検出されたすべてのQuadraデバイスを一覧表示します：

```
$ sudo ni_rsrc_mon
```



アップグレードしたいデバイスの DEVICE名（例：/dev/rdisk4）をメモしてください。

3. コマンドラインターミナルで、展開したフォルダーに移動します：

```
$ cd <FWリリースフォルダーのパス>
```

4. 同じコマンドラインターミナルで、**firmware_upgrade_macos** 実行ファイルを起動します。
   ファームウェアのアップグレードを行うには、以下の2つの引数が必要です：
   - デバイス名（例：/dev/rdisk4）
   - **nor_flash.bin** の場所（例：./FL_BIN/nor_flash.bin）

   アップグレードを行うには、以下のコマンドを使用します：

```
$ ./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
```

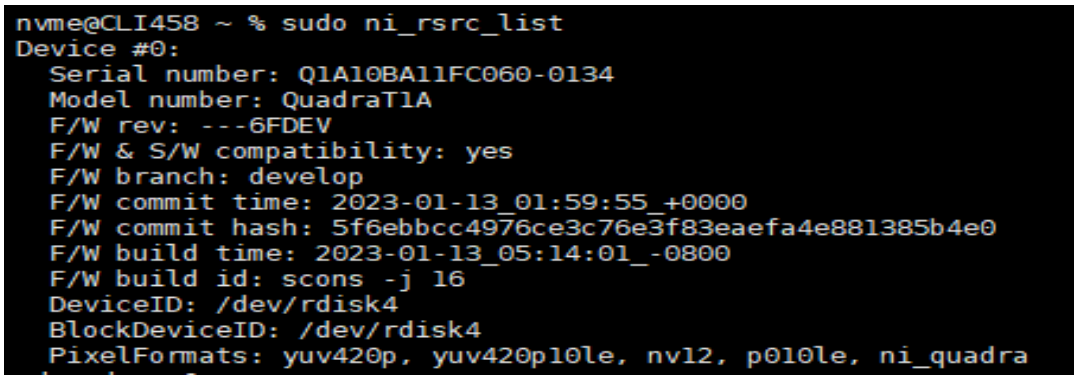
   約2分ほど待って、コマンドの実行が完了するのを待ちます。

5. コマンドの実行が完了したら、ホストを再起動してください：

```
$ sudo reboot now
```

6. 以下のコマンドを実行して、ファームウェアのバージョンが正常にアップグレードされたことを確認します：

```
$ sudo ni_rsrc_list
```



上記の出力により、ファームウェアのバージョンが ---6FDEV にアップグレードされたことが確認できます。

ホスト内の各NETINTデバイスに対して、同様の手順を実行してください。

注意:アップグレード処理中は、プロセスを中断したり、カードにコマンドを送信したりしないでください。