JetPack ????

主要介绍　　JetPack系统的基础功能和配置方法

??jtop

jtop 是英伟达专为 **Jetson** 系列边缘计算设备 开发的 交互式系统监控工具。

1 ??jtop

- 安装 jtop需要的依赖库

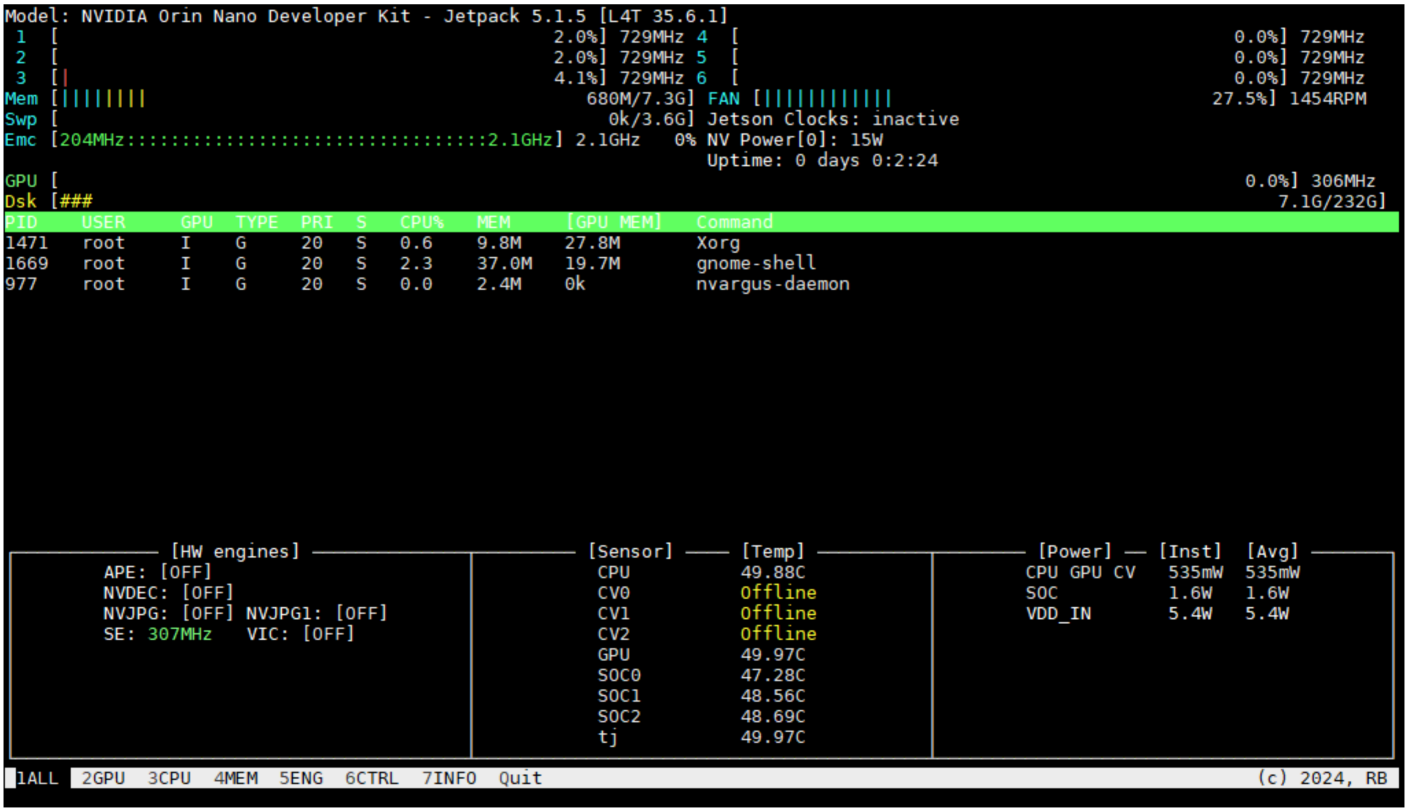
```
sudo apt update
sudo apt install python3
sudo apt install python3-pip
```

- 安装 jtop

```
sudo pip3 install -U pip -i https://pypi.tuna.tsinghua.edu.cn/simple
sudo pip3 install jetson-stats -i https://pypi.tuna.tsinghua.edu.cn/simple
sudo systemctl restart jtop.service
```

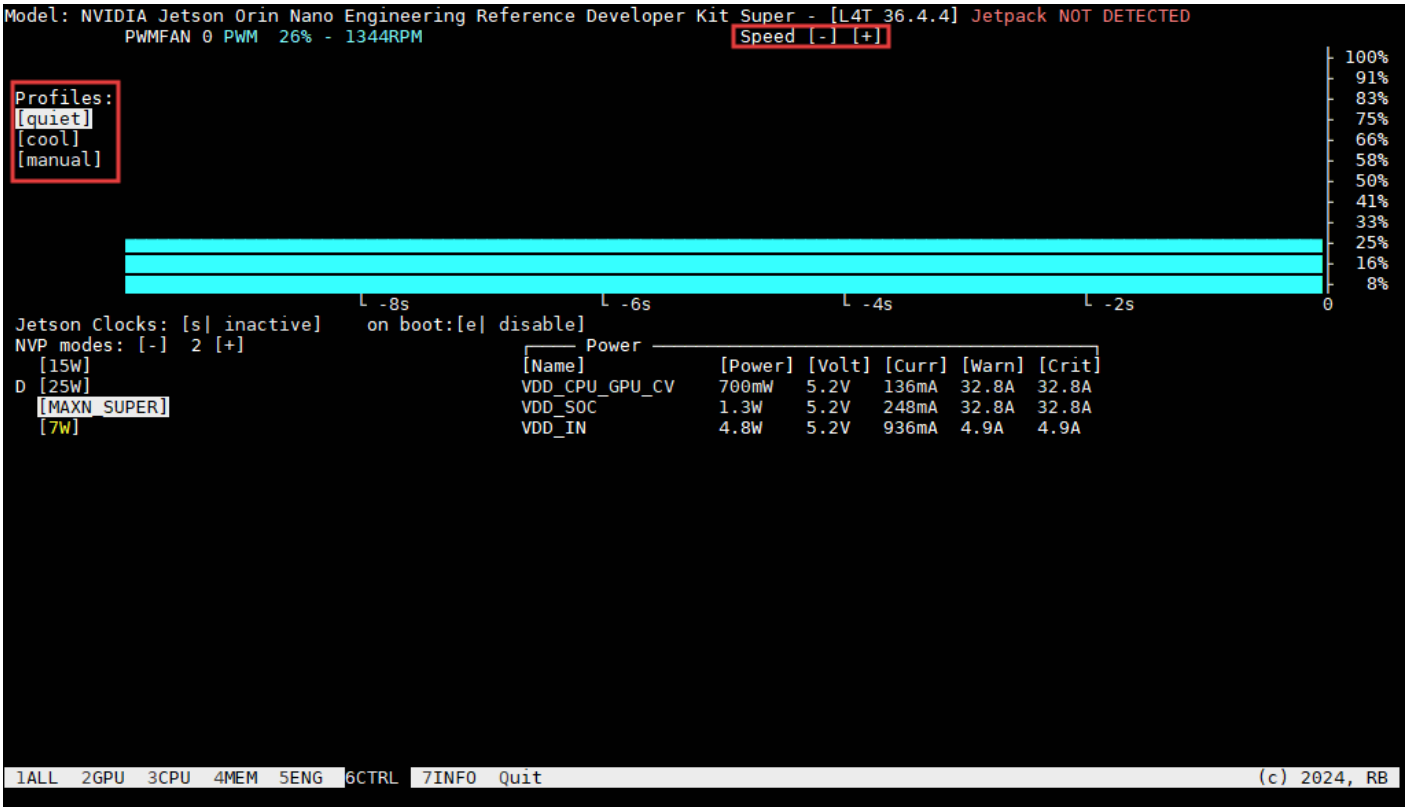
2 jtop???

- 安装完成后在控制台中输入 `jtop` 打开主界面

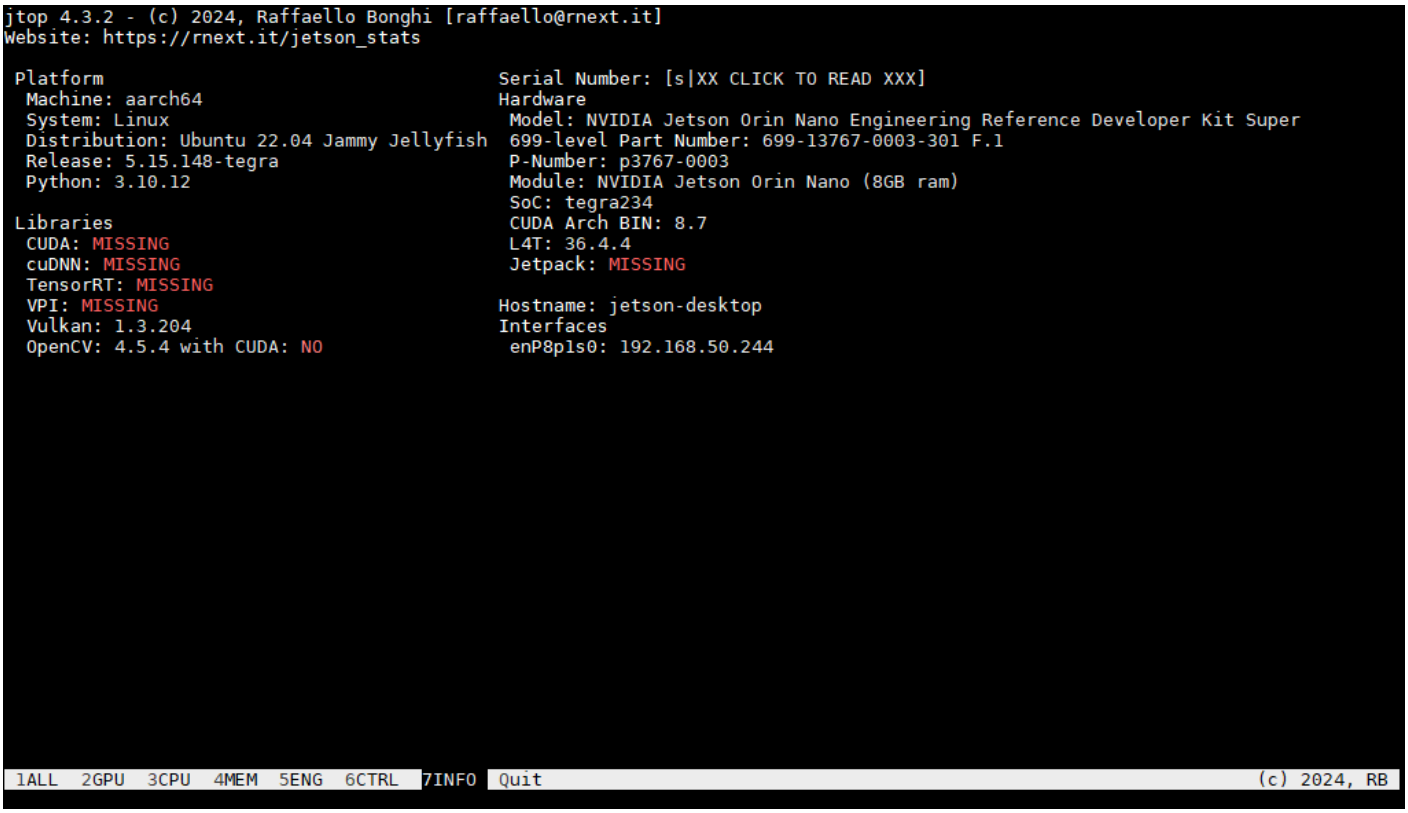


2.1 监视各个模块的工作信息

2.2 □□□□


Model: NVIDIA Jetson Orin Nano Engineering Reference Developer Kit Super - [L4T 36.4.4] Jetpack NOT DETECTED
PWMFAN 0 PWM 26% - 1344RPM
Speed [-] [+]
Profiles:
[quiet]
[cool]
[manual]
100%
91%
83%
75%
66%
58%
50%
41%
33%
25%
16%
8%
-8s
-6s
-4s
-2s
0
Jetson Clocks: [s| inactive] on boot:[e| disable]
NVP modes: [-] 2 [+]
[15W]
D [25W]
[MAXN_SUPER]
[7W]
Power
[Name] [Power] [Volt] [Curr] [Warn] [Crit]
VDD_CPU_GPU_CV 700mW 5.2V 136mA 32.8A 32.8A
VDD_SOC 1.3W 5.2V 248mA 32.8A 32.8A
VDD_IN 4.8W 5.2V 936mA 4.9A 4.9A
1ALL 2GPU 3CPU 4MEM 5ENG 6CTRL 7INFO Quit
(c) 2024, RB

2.3 □□□□□□□□

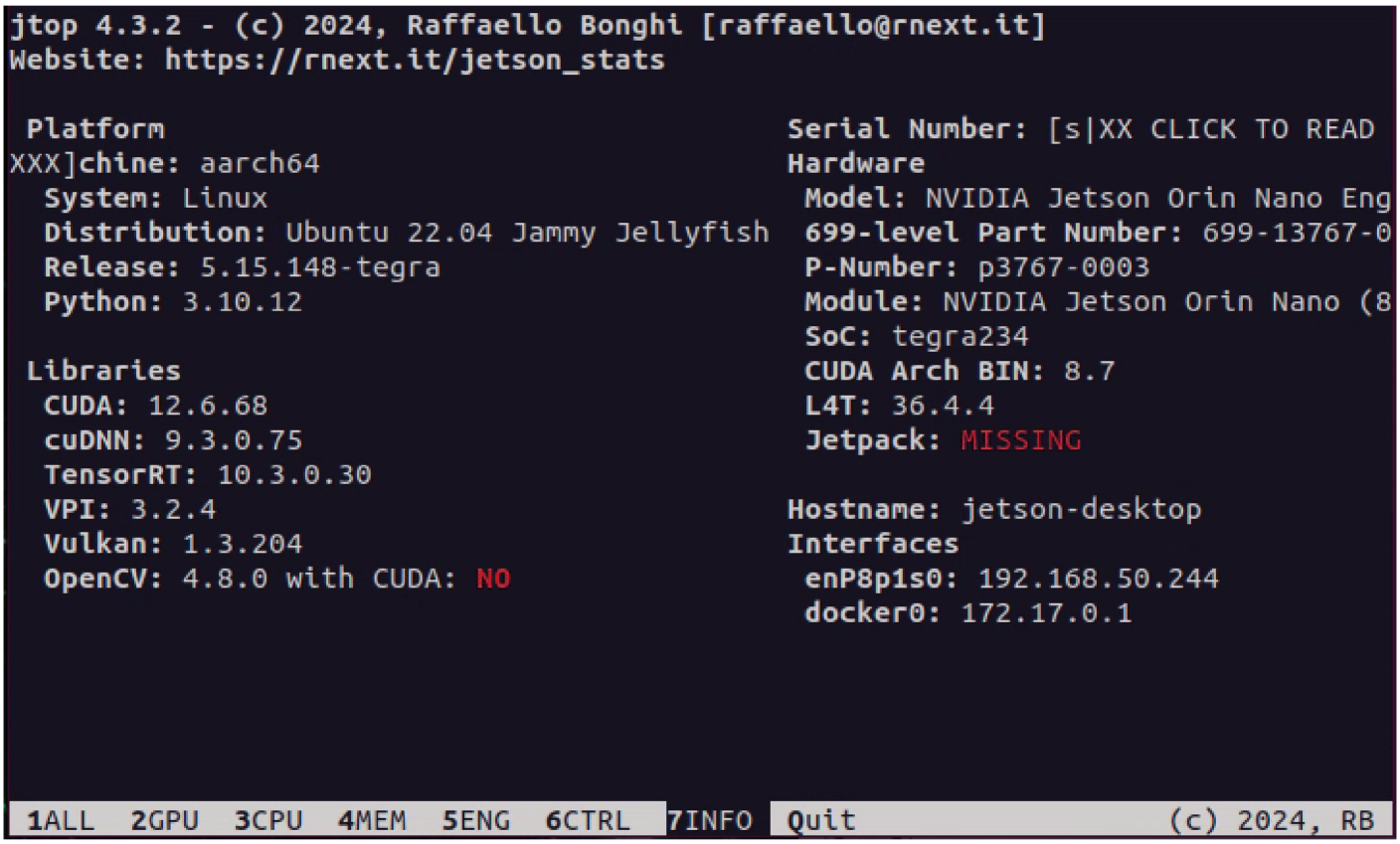

jtop 4.3.2 - (c) 2024, Raffaello Bonghi [raffaello@rnext.it]
Website: https://rnext.it/jetson_stats
Platform
Machine: aarch64
System: Linux
Distribution: Ubuntu 22.04 Jammy Jellyfish
Release: 5.15.148-tegra
Python: 3.10.12
Libraries
CUDA: MISSING
cuDNN: MISSING
TensorRT: MISSING
VPI: MISSING
Vulkan: 1.3.204
OpenCV: 4.5.4 with CUDA: NO
Serial Number: [s|XX CLICK TO READ XXX]
Hardware
Model: NVIDIA Jetson Orin Nano Engineering Reference Developer Kit Super
699-level Part Number: 699-13767-0003-301 F.1
P-Number: p3767-0003
Module: NVIDIA Jetson Orin Nano (8GB ram)
SoC: tegra234
CUDA Arch BIN: 8.7
L4T: 36.4.4
Jetpack: MISSING
Hostname: jetson-desktop
Interfaces
enP8p1s0: 192.168.50.244
1ALL 2GPU 3CPU 4MEM 5ENG 6CTRL 7INFO Quit
(c) 2024, RB

??CUDA

Jetson CUDA是 NVIDIA为边缘 AI设计的并行计算平台，基于 ARM的 Jetson系列硬件，支持 **GPU**加速深度学习与 实时推理 ，低 功耗高性能 。

??CUDA JETSON SDK

```
sudo apt update
sudo apt install nvidia-jetpack
```



??????????

> 以下测试仅在 **JetPack 6.1 (rev1)** 和 **JetPack 5.1.5** 版本中通过验证

调整功耗上限:

```
sudo nvpmodel -m 2 #nano 模式2为super
sudo jetson_clocks --fan
```

cpu压力测试:

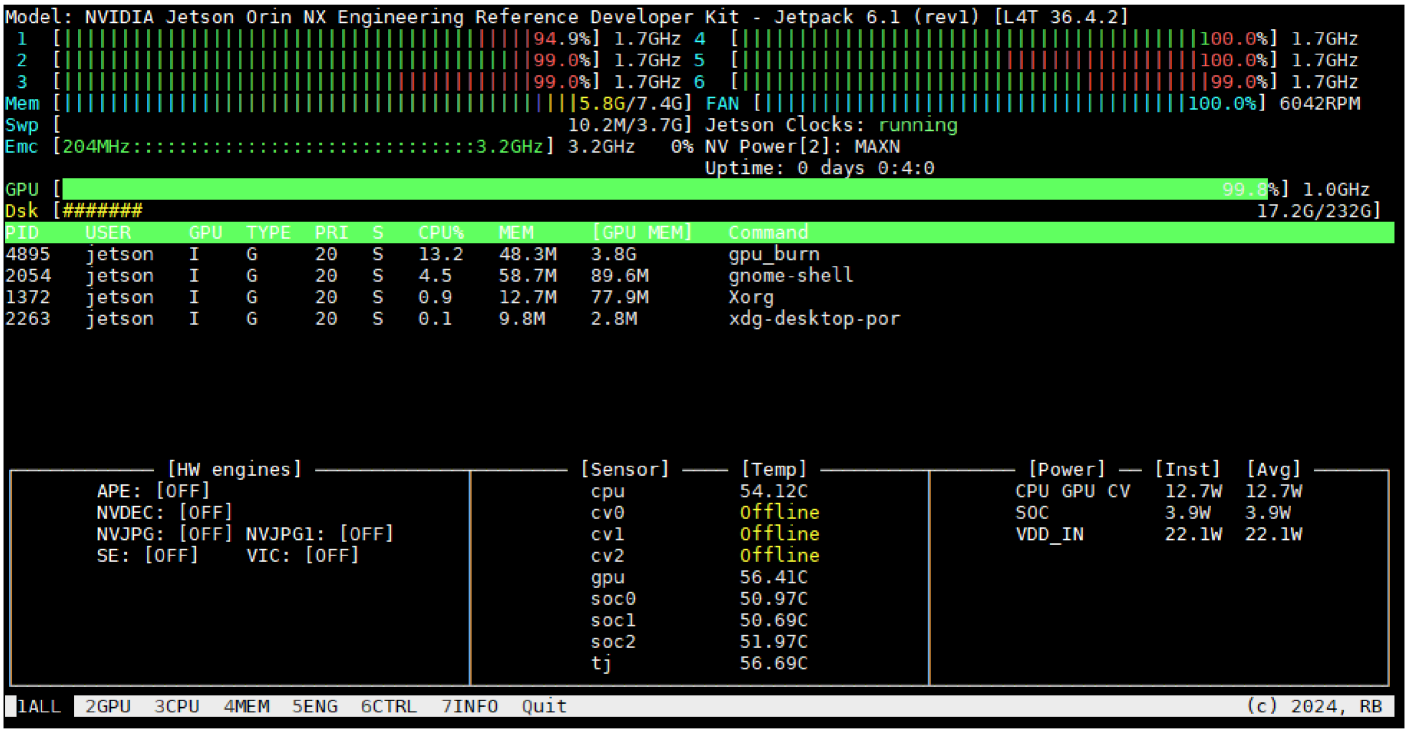
```
sudo apt install stress
stress --cpu 8 --io 4 --vm 2 --vm-bytes 128M --hdd 1 --hdd-bytes 1024M
```

gpu压力测试:

```
git clone https://github.com/anseeto/jetson-gpu-burn/
cd jetson-gpu-burn
make
./gpu_burn 100000
```

状态查询:

```
sudo jtop
```



> Orin Nano 整机最大功率为 **30W.**
>
> Orin NX 整机最大功率为 **40W**.

C1901/1902 USB????

??Jetson Orin Nano????

1. JetPack 5.1.5????

1. 1??**JetPack 5.1.5**???

参考 官方固件刷机流程

1.2 ????????????

设备正常开机后，打开终端命令行，将 修改后的设备树文件 下载到开发板。

```
git clone https://gitee.com/kongyuantech/document.git
cd document/AN002\ Orin\ Nano\ NX\ USB配置/5.1.4/
```

1.3 ???????????

选择执行以下命令替换 **DTB**设备树 文件并重启设备。

- **Jetson Orin Nano 4G** 请使用以下代码

```
sudo cp /boot/dtb/kernel_tegra234-p3767-0004-p3768-0000-a0.dtb /boot/dtb/kernel_tegra234-
p3767-0004-p3768-0000-a0.dtb.backup # 备份原设备树文件
sudo cp kernel_tegra234-p3767-0004-p3768-0000-a0.dtb /boot/dtb
sudo chown 0:0 /boot/dtb/kernel_tegra234-p3767-0004-p3768-0000-a0.dtb
sudo chmod 644 /boot/dtb/kernel_tegra234-p3767-0004-p3768-0000-a0.dtb
sudo reboot
```

- **Jetson Orin Nano 8G** 请使用以下代码

```
sudo cp /boot/dtb/kernel_tegra234-p3767-0003-p3768-0000-a0.dtb /boot/dtb/kernel_tegra234-
p3767-0003-p3768-0000-a0.dtb.backup # 备份原设备树文件
sudo cp kernel_tegra234-p3767-0003-p3768-0000-a0.dtb /boot/dtb
sudo chown 0:0 /boot/dtb/kernel_tegra234-p3767-0003-p3768-0000-a0.dtb
sudo chmod 644 /boot/dtb/kernel_tegra234-p3767-0003-p3768-0000-a0.dtb
sudo reboot
```

2. JetPack 6.2.1????

2.1 ??**JetPack 6.2.1**???

参考 官方固件刷机流程

2.2 ???????????

- 设备正常开机后，打开终端命令行，将 修改后的设备树文件 下载到开发板。

```
git clone https://gitee.com/kongyuantech/document.git
cd document/AN002\ Orin\ Nano\ NX\ USB配置/6.2/
```

1.3 ???????????

选择执行以下命令 增加 **DTB**设备树 文件并重启设备。

- **Jetson Orin Nano 4G** 请执行以下代码

```
sudo cp orin_nano_4g.dtb /boot/dtb
sudo sed -i 's#console=tty0#console=tty0\n      FDT /boot/dtb/orin_nano_4g.dtb#g'
/boot/extlinux/extlinux.conf
sudo reboot
```

- **Jetson Orin Nano 8G** 请执行以下代码

```
sudo cp orin_nano_8g.dtb /boot/dtb
sudo sed -i 's#console=tty0#console=tty0\n      FDT /boot/dtb/orin_nano_8g.dtb#g'
/boot/extlinux/extlinux.conf
sudo reboot
```

??Jetson Orin NX????

1. JetPack 5.1.5???

1. 1??**JetPack 5.1.5**???

参考 官方固件刷机流程

1.2 ???????????

- 设备正常开机后，打开终端命令行，将 修改后的设备树文件 下载到开发板。

```
git clone https://gitee.com/kongyuantech/document.git
cd document/AN002\ Orin\ Nano\ NX\ USB配置/5.1.4/
```

1.3 ???????????

选择执行以下命令替换 **DTB**设备树 文件并重启设备。

- **Jetson Orin NX 8G** 请执行以下代码

```
sudo cp /boot/dtb/kernel_tegra234-p3767-0001-p3768-0000-a0.dtb /boot/dtb/kernel_tegra234-p3767-0001-p3768-0000-a0.dtb.backup # 备份原设备树文件
sudo cp kernel_tegra234-p3767-0001-p3768-0000-a0.dtb /boot/dtb
sudo chown 0:0 /boot/dtb/kernel_tegra234-p3767-0001-p3768-0000-a0.dtb
sudo chmod 644 /boot/dtb/kernel_tegra234-p3767-0001-p3768-0000-a0.dtb
sudo reboot
```

- **Jetson Orin NX 16G** 请执行以下代码

```
sudo cp /boot/dtb/kernel_tegra234-p3767-0000-p3768-0000-a0.dtb /boot/dtb/kernel_tegra234-p3767-0000-p3768-0000-a0.dtb.backup # 备份原设备树文件
sudo cp kernel_tegra234-p3767-0000-p3768-0000-a0.dtb /boot/dtb
sudo chown 0:0 /boot/dtb/kernel_tegra234-p3767-0000-p3768-0000-a0.dtb
sudo chmod 644 /boot/dtb/kernel_tegra234-p3767-0000-p3768-0000-a0.dtb
sudo reboot
```

2. JetPack 6.2.1????

2.1 ??**JetPack 6.2.1**???

参考 官方固件刷机流程

2.2 ????????????

- 设备正常开机后，打开终端命令行，将 修改后的设备树文件 下载到开发板。

```
git clone https://gitee.com/kongyuantech/document.git
cd document/AN002\ Orin\ Nano\ NX\ USB配置/6.2/
```

2.3 ???????????

选择执行以下命令 增加 **DTB**设备树 文件并重启设备。

- **Jetson Orin NX 8G** 请使用以下代码

```
sudo cp orin_nx_8g.dtb /boot/dtb
sudo sed -i 's#console=tty0#console=tty0\n      FDT /boot/dtb/orin_nx_8g.dtb#g'
```

```
/boot/extlinux/extlinux.conf
sudo reboot
```

- **Jetson Orin NX 16G** 请使用以下代码

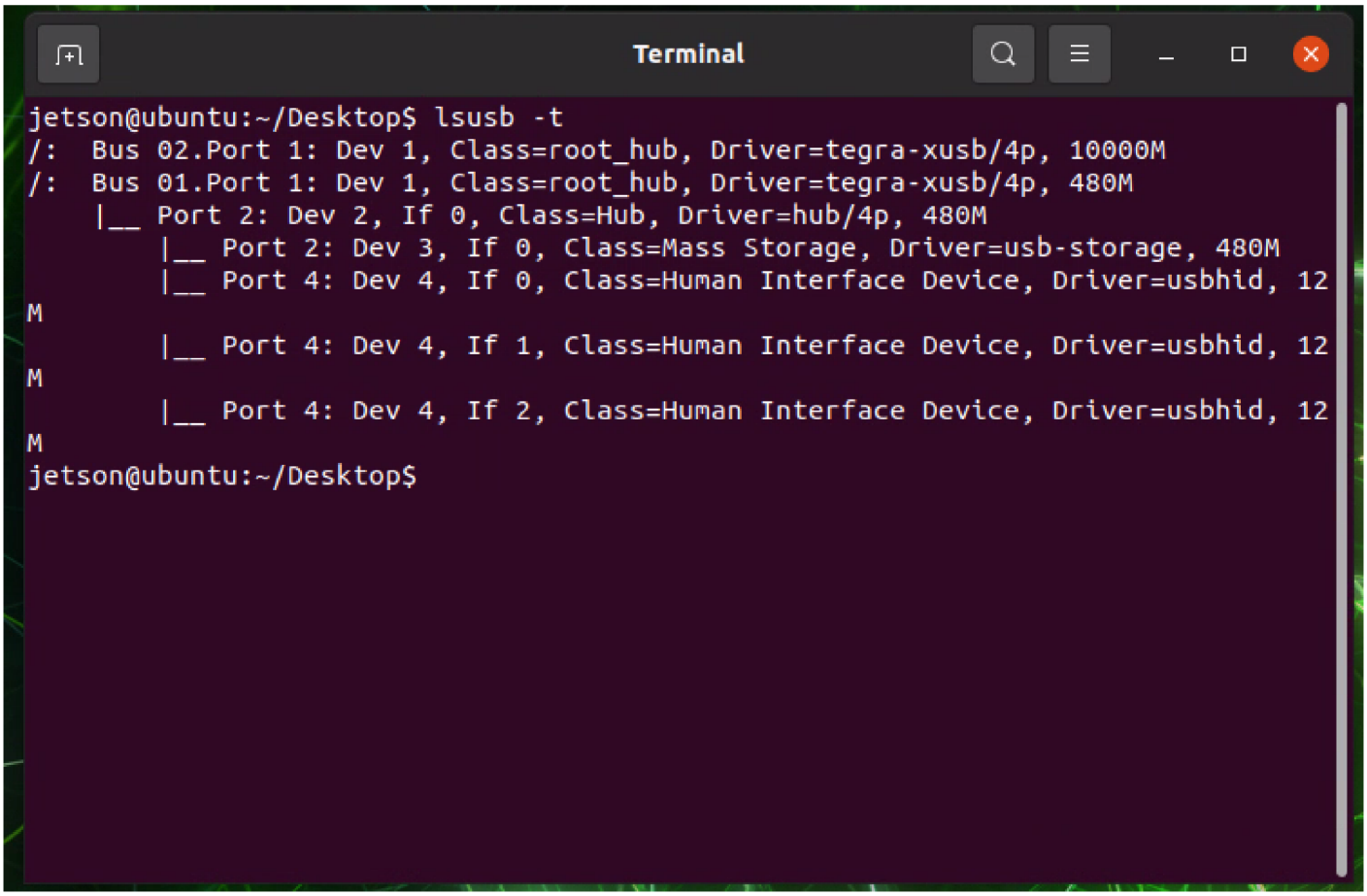
```
sudo cp orin_nx_16g.dtb /boot/dtb
sudo sed -i 's#console=tty0#console=tty0\n      FDT /boot/dtb/orin_nx_16g.dtb#g'
/boot/extlinux/extlinux.conf
sudo reboot
```

??????

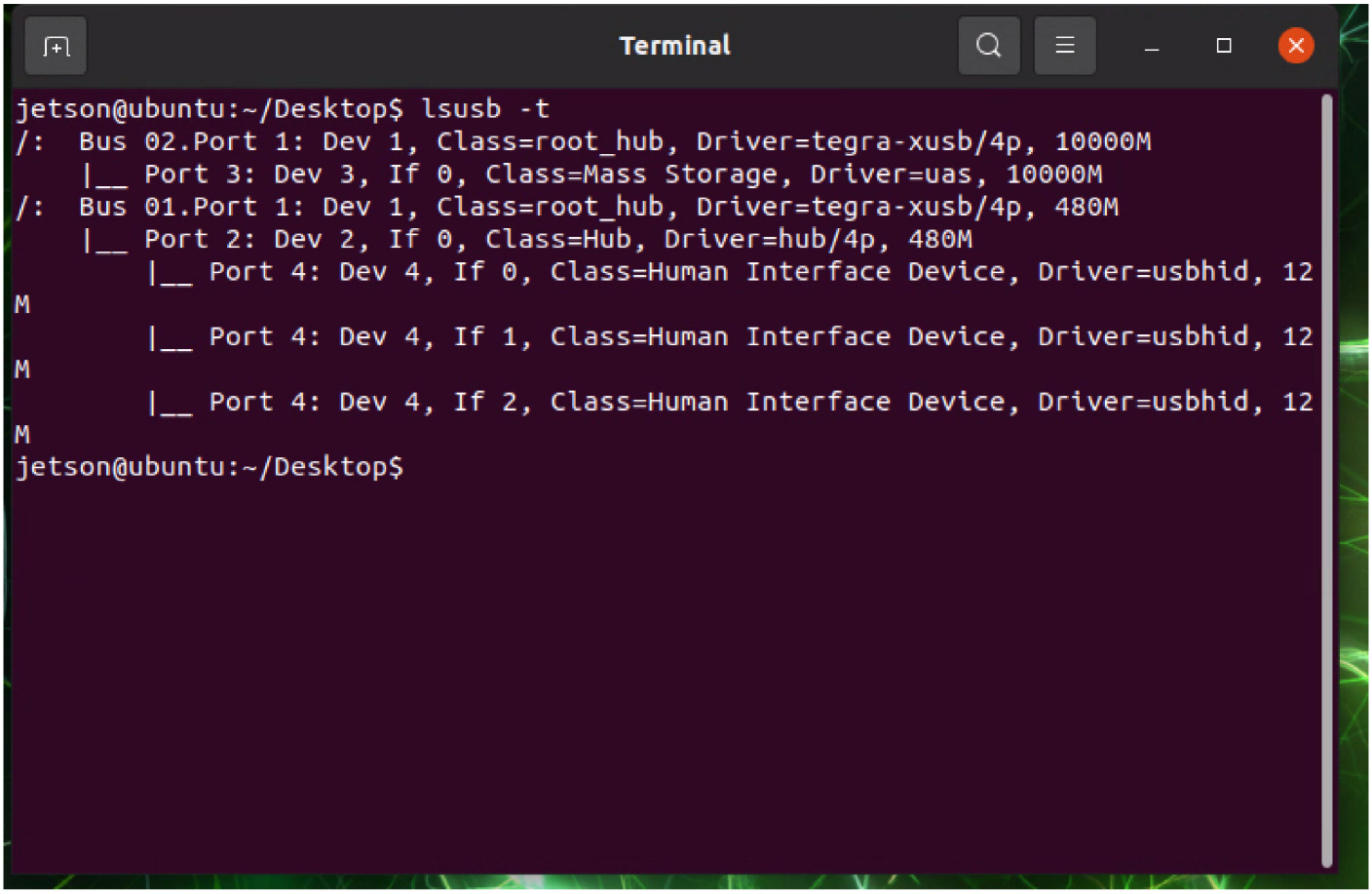
替换设备树后，4个 USB3.0端口（ C1901为 3个）可正常工作， Type-C端口系统默认处于关闭状态
可使用以下命令查看 USB端口工作情况。

```
lsusb -t
```

- 未替换设备树前:



- 替换设备树成功后:



您也可以自行使用 DISK工具或 dd指令测试读写速率是否达到 USB3.0的标准。

????Tpye-C????

- 配置 **Type-C端口 为 Device模式**

> 注意：此命令为临时启用，重启会失效。

```
sudo bash -c 'echo device > /sys/class/usb_role/usb2-0-role-switch/role'
```

设置后，使用 Type-C端口连接到 PC主机时，具有以下功能

1. COM口，设备的终端命令行
2. 虚拟网卡，设备 ip地址默认为: 192.168.55.1
3. NCM (Network Control Model)，设备可通过 PC主机联网，仅支持 Linux、Mac

- 开机默认配置 **Type-C端口 为 Device模式**（永久有效）

```
sudo sed -i 's#exit 0#echo device > /sys/class/usb_role/usb2-0-role-switch/role\nexit 0#g'
/opt/nvidia/l4t-usb-device-mode/nv-l4t-usb-device-mode-start.sh
```

?????LOGO

开机时，开发板会进入 UEFI引导系统启动，期间会显示 NVIDIA默认的 LOGO 背景，如果您需要自定义该图片，需要按照一下操作获取 UEFI源码替换图片内容重新编译 UEFI 并刷入到开发板中。

一、安装 docker环境

如果已安装 docker可以跳过此步骤，但要确保当前用户添加到 `docker` 用户组

```
sudo apt install docker.io
```

将当前用户添加到 `docker` 用户组后重启生效

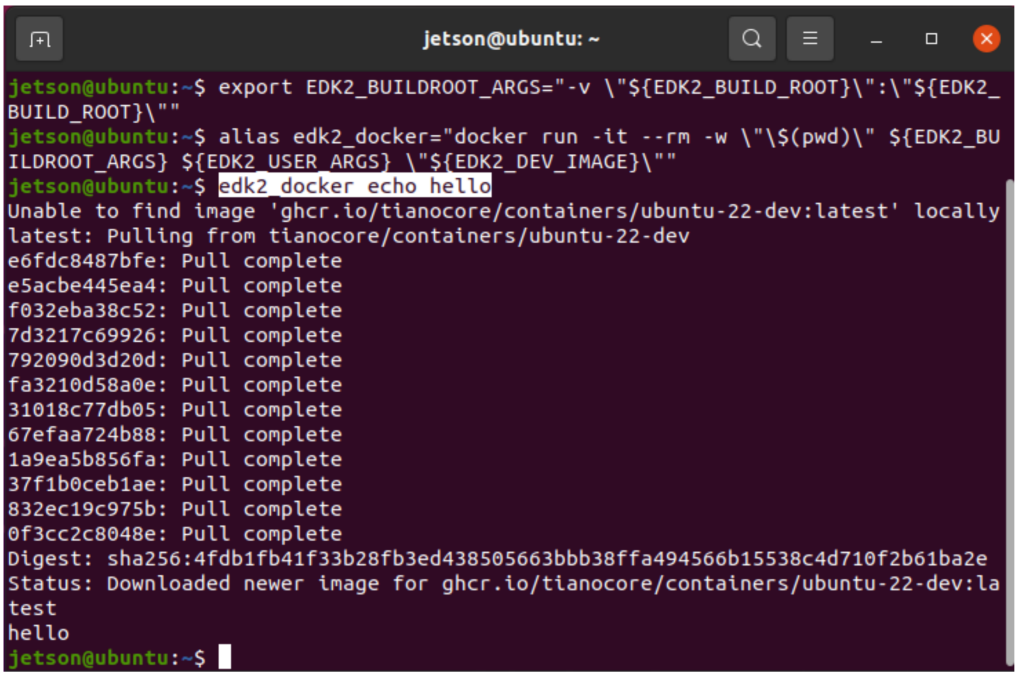
```
sudo usermod -a -G docker ${USER}
sudo reboot
```

设置变量

```
export EDK2_DEV_IMAGE="ghcr.io/tianocore/containers/ubuntu-22-dev:latest"
export EDK2_USER_ARGS="-v \"${HOME}\":\"${HOME}\" -e EDK2_DOCKER_USER_HOME=\"${HOME}\""
export EDK2_BUILD_ROOT="/build"
export EDK2_BUILDROOT_ARGS="-v \"${EDK2_BUILD_ROOT}\":\"${EDK2_BUILD_ROOT}\""
alias edk2_docker="docker run -it --rm -w \"\$(pwd)\" ${EDK2_BUILDROOT_ARGS} ${EDK2_USER_ARGS}
\"${EDK2_DEV_IMAGE}\""
```

拉取并验证环境

```
edk2_docker echo hello
```



二、拉取源码

初始化 edk2环境

```
edk2_docker init_edkrepo_conf
edk2_docker edkrepo manifest-repos add nvidia https://github.com/NVIDIA/edk2-edkrepo-manifest.git main nvidia
```

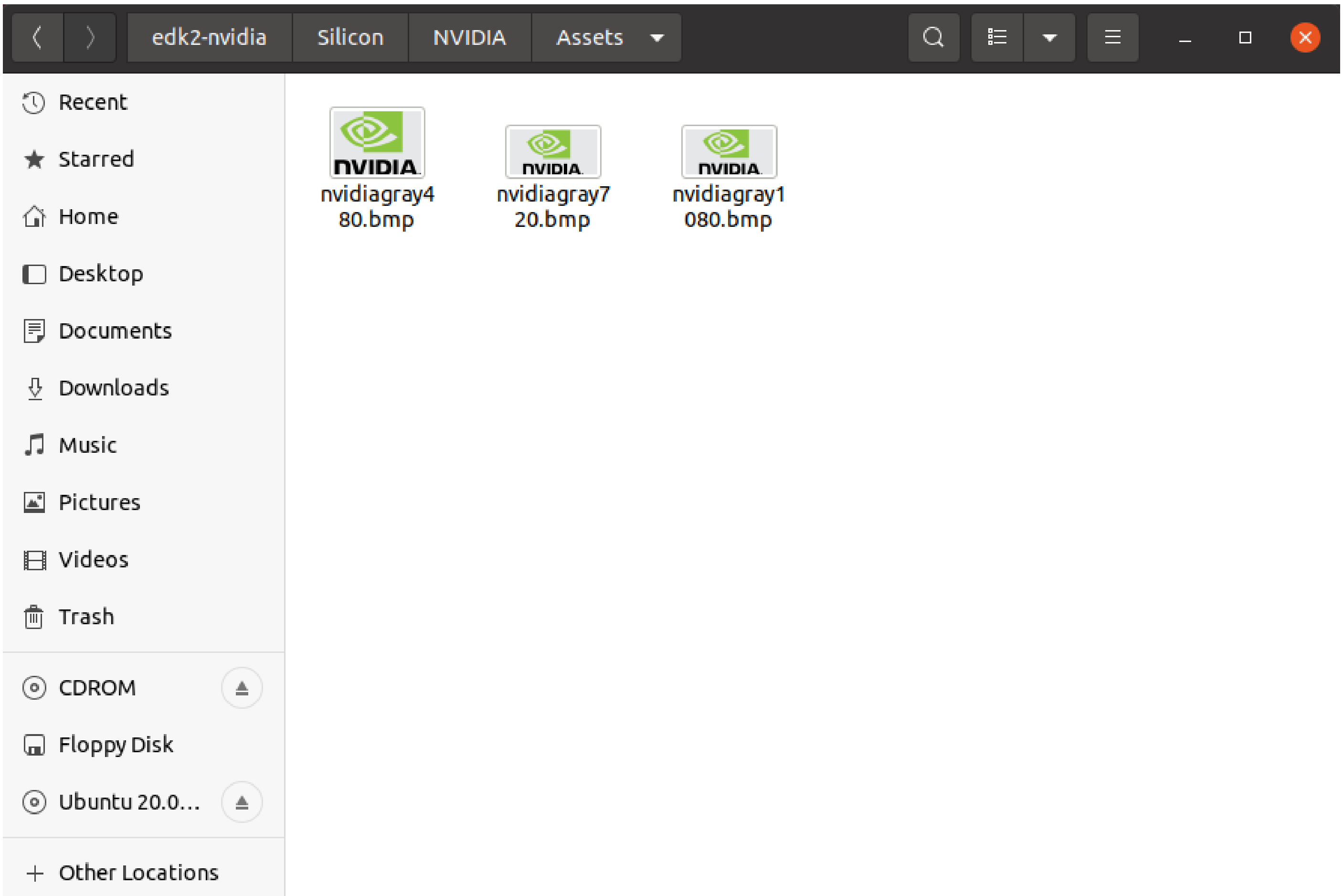
根据 JetPack版本拉取 uefi源码 （以 **JetPack6.2.1/r36.4.4**为例 ）

```
edk2_docker edkrepo clone nvidia-uefi-r36.4.4 NVIDIA-Platforms r36.4.4-updates
```

该过程会同步多个 git项目，可能耗时较长。

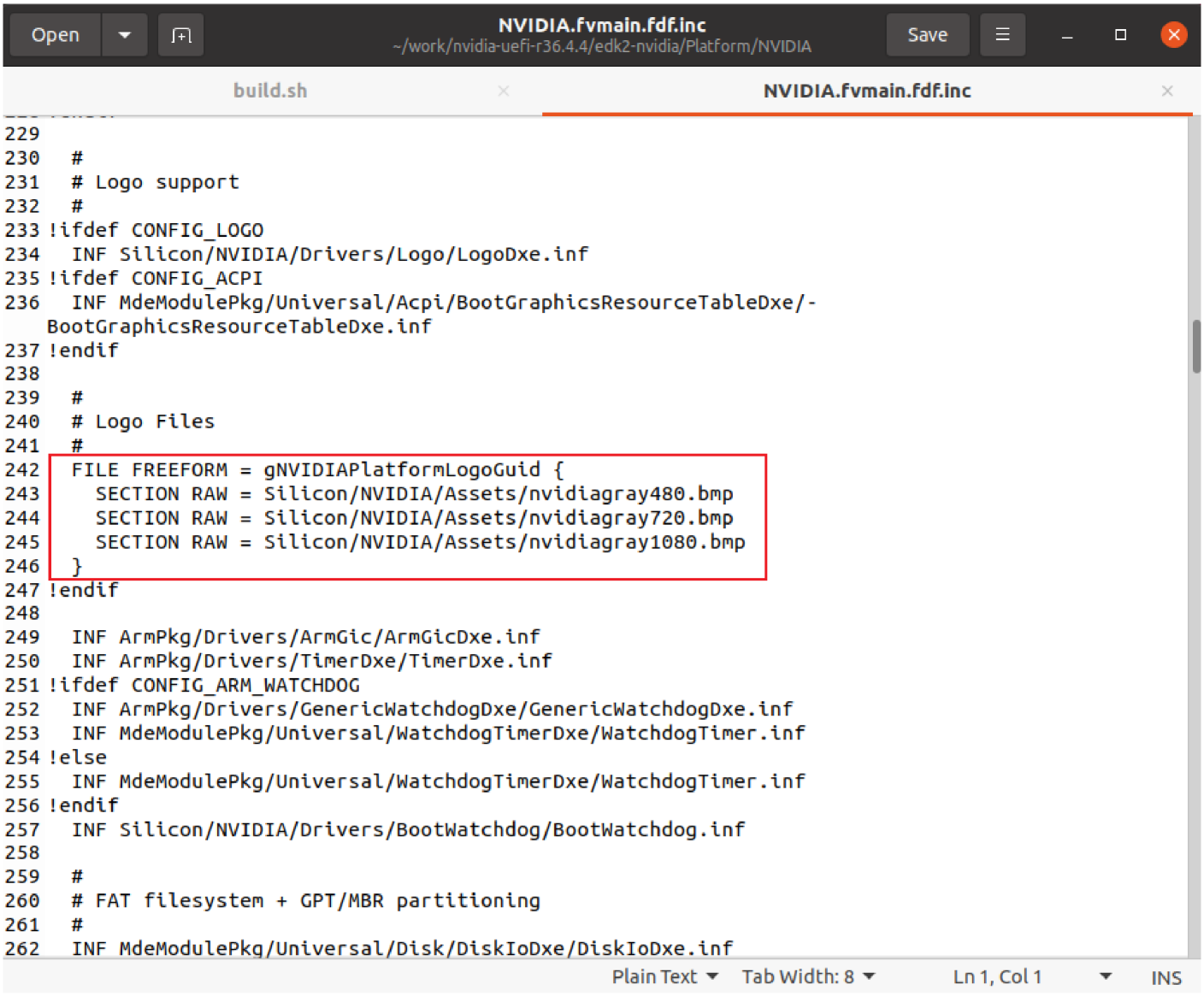
同步完成后可以在以下目录找到默认的 LOGO文件

```
cd nvidia-uefi-r36.4.4/edk2-nvidia/Silicon/NVIDIA/Assets/
```



[illegible] LOGO[illegible]

```
cd nvidia-uefi-r36.4.0/edk2-nvidia/Platform/NVIDIA/NVIDIA.fvmain.fdf.inc
```

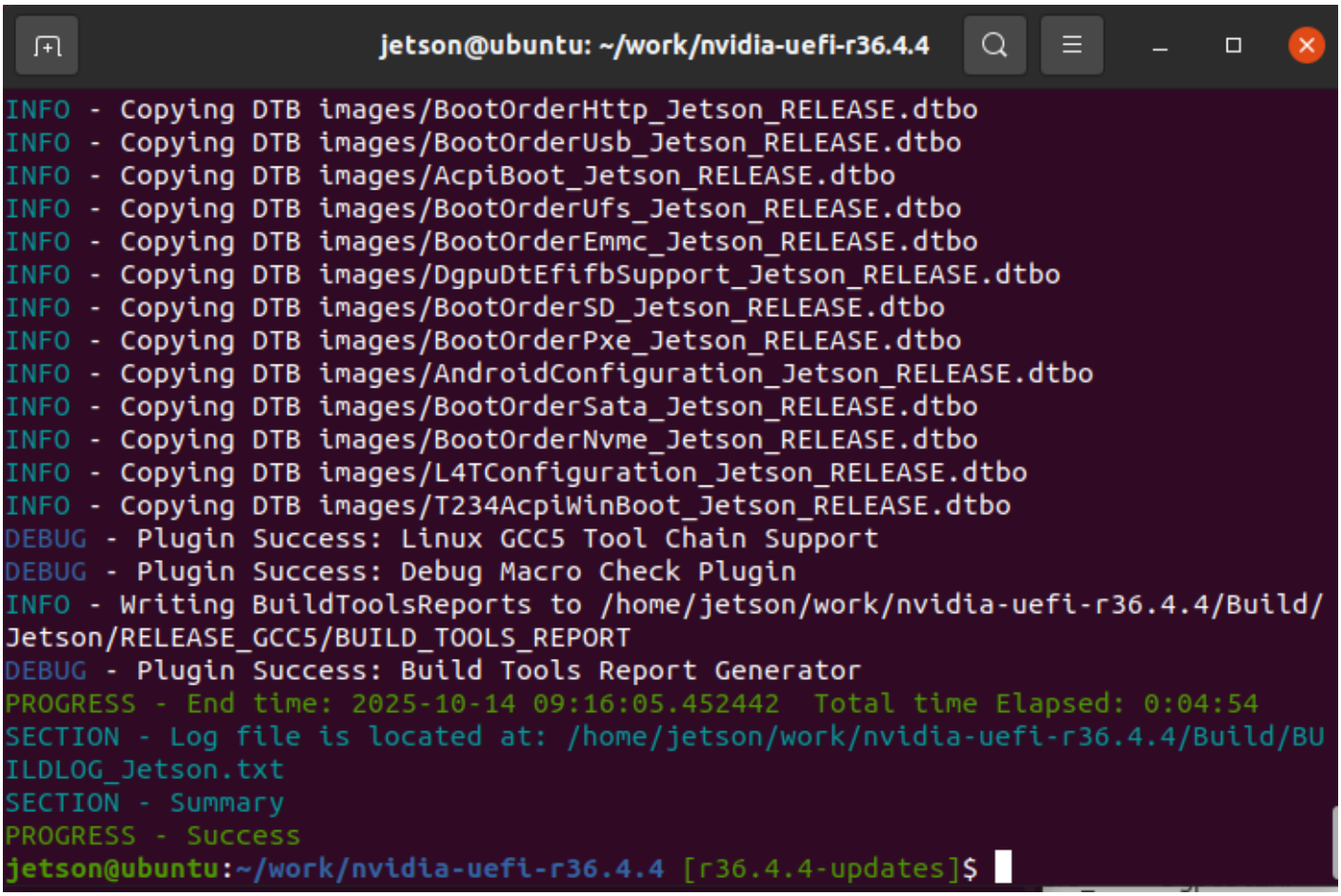


> 应该尽可能地控制文件大小，最后的编译出的 uefi_xxx.bin不得超过 **3.5MB**
> ，否则刷入后开发板将无法启动。

三、编译

替换完成后，执行以下命令编译 UEFI固件

```
cd nvidia-uefi-r36.4.4/
edk2_docker edk2-nvidia/Platform/NVIDIA/Jetson/build.sh
```


jetson@ubuntu: ~/work/nvidia-uefi-r36.4.4
INFO - Copying DTB images/BootOrderHttp_Jetson_RELEASE.dtbo
INFO - Copying DTB images/BootOrderUsb_Jetson_RELEASE.dtbo
INFO - Copying DTB images/AcpiBoot_Jetson_RELEASE.dtbo
INFO - Copying DTB images/BootOrderUfs_Jetson_RELEASE.dtbo
INFO - Copying DTB images/BootOrderEmmc_Jetson_RELEASE.dtbo
INFO - Copying DTB images/DgpuDtEfifbSupport_Jetson_RELEASE.dtbo
INFO - Copying DTB images/BootOrderSD_Jetson_RELEASE.dtbo
INFO - Copying DTB images/BootOrderPxe_Jetson_RELEASE.dtbo
INFO - Copying DTB images/AndroidConfiguration_Jetson_RELEASE.dtbo
INFO - Copying DTB images/BootOrderSata_Jetson_RELEASE.dtbo
INFO - Copying DTB images/BootOrderNvme_Jetson_RELEASE.dtbo
INFO - Copying DTB images/L4TConfiguration_Jetson_RELEASE.dtbo
INFO - Copying DTB images/T234AcpiWinBoot_Jetson_RELEASE.dtbo
DEBUG - Plugin Success: Linux GCC5 Tool Chain Support
DEBUG - Plugin Success: Debug Macro Check Plugin
INFO - Writing BuildToolsReports to /home/jetson/work/nvidia-uefi-r36.4.4/Build/Jetson/RELEASE_GCC5/BUILD_TOOLS_REPORT
DEBUG - Plugin Success: Build Tools Report Generator
PROGRESS - End time: 2025-10-14 09:16:05.452442 Total time Elapsed: 0:04:54
SECTION - Log file is located at: /home/jetson/work/nvidia-uefi-r36.4.4/Build/BUILDLOG_Jetson.txt
SECTION - Summary
PROGRESS - Success
jetson@ubuntu:~/work/nvidia-uefi-r36.4.4 [r36.4.4-updates]$

□□□□

□□□□

????

????

在 **JetPack6** 以后官方移除了自带的 **Intel 8625NGW** 驱动，同时为了能够使用性能更优的 **Intel AX200** 和 **AX210** 网卡我们需要自行手动安装 **Intel** 网卡驱动包。

1. 插入网卡，通电开机，并查看网卡状态

```
sudo lshw -C network
```

若显示 product: Wi-Fi 6 AX200，则硬件已经检测到，若看到 `UNCLAIMED` 字样则表明驱动未安装

2. 安装 **iwlwifi** 驱动

iwlwifi 是英特尔（Intel）为其无线网卡开发的开源驱动程序，专为 Linux 操作系统设计，用于支持英特尔全系列无线网络适配器（如 Centrino、Wi-Fi 6/6E/7 等芯片）在 Linux 环境下的高效运行。

```
sudo apt update
sudo apt install backport-iwlwifi-dkms
```

3. 重启开发板

```
sudo reboot
```

4. 再次查看网卡状态

```
jetson@jetson-desktop:~$ sudo lshw -C network
[sudo] password for jetson:
  *-network
       description: Wireless interface
       product: Wi-Fi 6 AX200
       vendor: Intel Corporation
       physical id: 0
       bus info: pci@0001:01:00.0
       logical name: wlan0
       version: 1a
       serial: ac:12:03:a0:4c:db
       width: 64 bits
       clock: 33MHz
```

```
        capabilities: pm msi pciexpress msix bus_master cap_list ethernet physical wireless
        configuration: broadcast=yes driver=iwlwifi driverversion=5.10.216-tegra
firmware=59.601f3a66.0 cc-a0-59.ucode latency=0 link=no multicast=yes wireless=IEEE 802.11
        resources: irq:55 memory:20a8000000-20a8003fff
  *-network
```

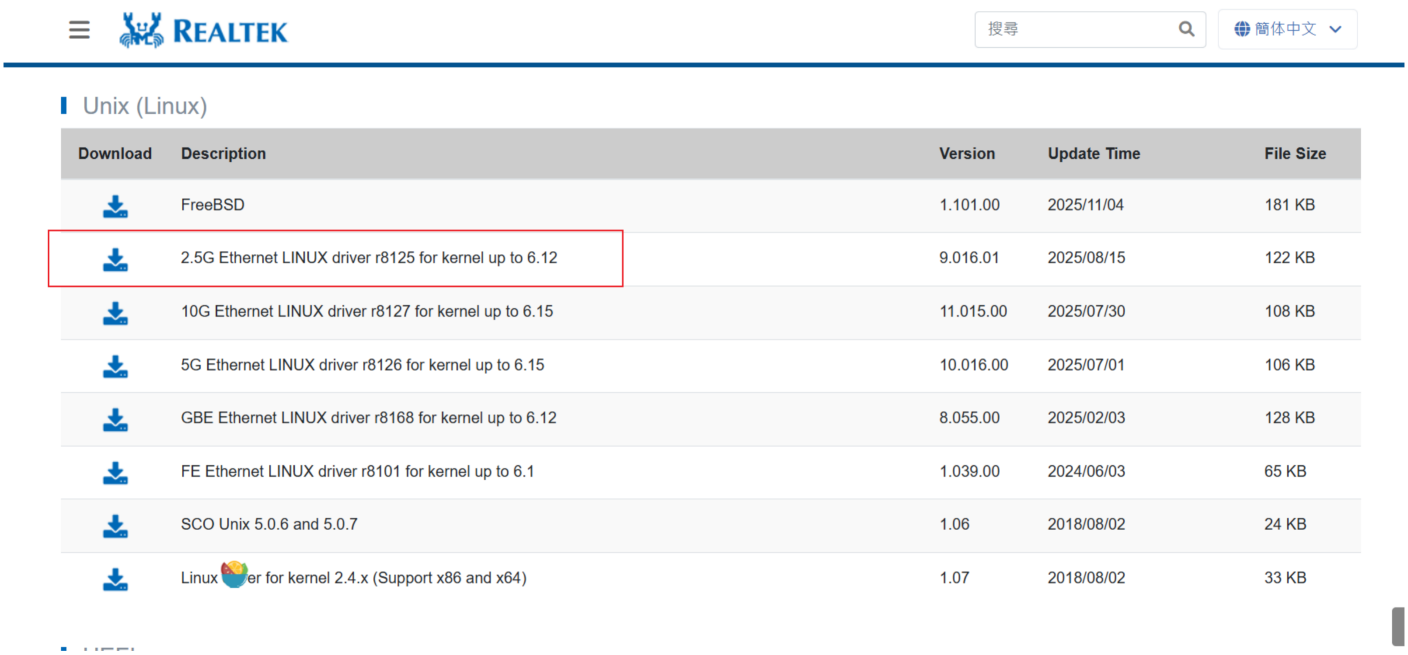
`description` 字段显示为 `Wireless interface` / `Ethernet interface` 便可正常使用无线网卡

????

JetPack自带了 RTL8111的驱动，对于 RTL8125(2.5G)或以上网卡需要下载官方驱动安装。

> 使用官方驱动时，网口指示灯可能工作不正常，如果需要修改的驱动，请与客服联系。

1、获取驱动， 点击此处_____转跳下载



2、将驱动上传到开发板并解压打开文件夹

```
cd r8125-9.016.01  #注意实际版本号
```

3、运行安装脚本

```
sudo bash autorun.sh
```

4、重启后运行命令查看网卡状态

```
jetson@jetson-desktop:~$ sudo lshw -C network
  *-network
       description: Ethernet interface
       product: RTL8125 2.5GbE Controller
       vendor: Realtek Semiconductor Co., Ltd.
       physical id: 0
       bus info: pci@0007:01:00.0
       logical name: eth1
       version: 05
       serial: 86:b1:ce:38:66:80
       size: 1Gbit/s
       capacity: 1Gbit/s
       width: 64 bits
       clock: 33MHz
       capabilities: pm msi pciexpress msix vpd bus_master cap_list ethernet
phy                                                                    sical tp 10bt
10bt-fd 100bt 100bt-fd 1000bt-fd autonegotiation
       configuration: autonegotiation=on broadcast=yes driver=r8125
driverversio
n=9.016.00-NAPI duplex=full ip=10.0.0.191 latency=0 link=yes multicast=yes
port=                                                                    twisted
pair speed=1Gbit/s
       resources: irq:59 ioport:200000(size=256) memory:3228000000-322800ffff
me
mory:3228010000-3228013fff
  *-network
```

4G??????

移远通信 LTE Standard EM05系列是一款专为 IoT/M2M应用而设计的 LTE Cat 4模块。采用节省空间的 M.2（NGFF）封装类型，超薄、紧凑的设计使其更易于嵌入到小尺寸产品中。

EM05系列支持最大下行速率 150 Mbps，最大上行速率 50 Mbps,包含三个型号: EM05-CN、EM05-E和 EM05-G。支持 LTE-FDD、LTE-TDD、DC-HSDPA、HSPA+、HSDPA、HSUPA、WCDMA和 CDMA 等多种网络制式。

要在 Jetson系列上使用 EM-05模块需要重新编译系统内核并刷入新编译的系统。

?????????

1.1 ????????

1.2 ?????????

1.3 ???????????

1.4 ?????????

???????????

2.1 ????????

- 安装编译依赖工具

```
sudo apt install build-essential bc git bison flex libssl-dev zip libncurses-dev make git
```

- 创建工具链放置目录

```
mkdir $HOME/l4t-gcc-toolchain
cd $HOME/l4t-gcc-toolchain
```

- 下载并解压交叉编译工具链

```
wget https://developer.nvidia.com/downloads/embedded/l4t/r36_release_v3.0/toolchain/aarch64--glibc--stable-2022.08-1.tar.bz2
tar xf aarch64--glibc--stable-2022.08-1.tar.bz2
```

- 进入到缓存好的刷机固件目录

```
cd
$HOME/nvidia/nvidia_sdk/JetPack_6.2.1_Linux_JETSON_ORIN_NANO_TARGETS/Linux_for_Tegra/source  #
根据实际目录进行修改
```

- 根据 release tag 拉取源码。 Jetson Linux Archive | NVIDIA Developer

```
 ./source_sync.sh -k -t <release-tag> #<release-tag>替换为你自己的发行版本如 jetson_36.4.3
```

2.2 ??????

> 本部分内容可参考 移远 Linux&Andriod驱动技术手册

Jetson Orin Nano配置文件路径: arch/arm64/configs/defconfig

- 配置 USB转串口驱动添加如下内容

```
CONFIG_USB_SERIAL=y
CONFIG_USB_SERIAL_WWAN=y
CONFIG_USB_SERIAL_OPTION=y
CONFIG_USB_NET_DRIVERS=y
CONFIG_USB_USBNET=y
CONFIG_USB_NET_QMI_WWAN=y
CONFIG_USB_WDM=y
```

- 修改 drivers/usb/serial/option.c, 向 USB转串口驱动添加 VID和 PID

```
static const struct usb_device_id option_ids[] = {
#if 1 // 2025-04-24 Added by Quectel
    { USB_DEVICE(0x2C7C, 0x0125) },
#endif
    ... ...
}
```

- 使用 USBNet驱动, 文件路径: drivers/usb/serial/option.c

```
static int option_probe(struct usb_serial *serial,
            const struct usb_device_id *id)
{
    ... ...

#if 1  // 2025-04-24 Added by Quectel
```

```
if (serial->dev->descriptor.idVendor == cpu_to_le16(0x2C7C)) {
        __u16 idProduct = le16_to_cpu(serial->dev->descriptor.idProduct);
        struct usb_interface_descriptor *intf = &serial->interface->cur_altsetting->desc;

        if (intf->bInterfaceClass != 0xFF || intf->bInterfaceSubClass == 0x42) {
                //ECM, RNDIS, NCM, MBIM, ACM, UAC, ADB
                return -ENODEV;
        }

        if ((idProduct&0xF000) == 0x0000) {
                //MDM interface 4 is QMI
                if (intf->bInterfaceNumber == 4 && intf->bNumEndpoints == 3
                        && intf->bInterfaceSubClass == 0xFF && intf->bInterfaceProtocol ==
0xFF)
                        return -ENODEV;
        }
}
#endif

    /* Store the device flags so we can use them during attach. */
    usb_set_serial_data(serial, (void *)device_flags);

    return 0;
}
```

- □□□□□□□□□□□□ drivers/usb/serial/usb_wwan.c

```
static struct urb *usb_wwan_setup_urb(struct usb_serial_port *port,
                      int endpoint,
                      int dir, void *ctx, char *buf, int len,
                      void (*callback) (struct urb *))
{
 ... ...
    usb_fill_bulk_urb(urb, serial->dev,
              usb_sndbulkpipe(serial->dev, endpoint) | dir,
              buf, len, callback, ctx);

#if 1   //2025-04-24 Added by Quectel for zero packet
    if (dir == USB_DIR_OUT) {
        struct usb_device_descriptor *desc = &serial->dev->descriptor;
```

```
        if (desc->idVendor == cpu_to_le16(0x2C7C))
        urb->transfer_flags |= URB_ZERO_PACKET;
    }
#endif
    return urb;
}
```

- 添加 Reset-resume机制（休眠唤醒）文件路径: drivers/usb/serial/option.c

```
static struct usb_serial_driver option_1port_device = {
    ... ...
#ifdef CONFIG_PM
    .suspend           = usb_wwan_suspend,
    .resume            = usb_wwan_resume,
#if 1  //2025-04-24 Added by Quectel
    .reset_resume   = usb_wwan_resume,
#endif
#endif
};
```

- 添加 QMI_WWAN驱动
  移远提供了 QMI_WWAN驱动源文件 qmi_wwan_q.c，将其复制到 drivers/net/usb/ 目录下。同时修改 drivers/net/usb/Makefile，使其能编译 qmi_wwan_q.c

```
#Makefile
# must insert qmi_wwan_q.o before qmi_wwan.o
obj-${CONFIG_USB_NET_QMI_WWAN} += qmi_wwan_q.o
obj-${CONFIG_USB_NET_QMI_WWAN} += qmi_wwan.o
```

2.3 ????

> 此步骤可能需要进行半个小时以上

- 创建输出目录

```
cd ../../
mkdir kernel_out
```

- 编译安装内核

```
export CROSS_COMPILE=$HOME/l4t-gcc-toolchain/aarch64--glibc--stable-2022.08-1/bin/aarch64-buildroot-linux-gnu-         #设置交叉编译工具
make -C kernel                          #构建 Jetson Linux 内核镜像
sudo -E make install -C kernel      #安装内核模块和树内模块
```

- 将内核镜像复制到刷机目录下

```
cp kernel/kernel-jammy-src/arch/arm64/boot/Image ../Linux_for_Tegra/kernel/Image
```

- 构建 NVIDIA树外模块（驱动程序）

```
export CROSS_COMPILE=$HOME/l4t-gcc-toolchain/aarch64--glibc--stable-2022.08-1/bin/aarch64-buildroot-linux-gnu-
export KERNEL_HEADERS=$PWD/kernel/kernel-jammy-src
make modules
```

- 安装到刷机目录下

```
export INSTALL_MOD_PATH=$HOME/nvidia/nvidia_sdk/JetPack_6.2.1_Linux_JETSON_ORIN_NANO_TARGETS/Linux_for_Tegra/rootfs/ #根据实际目录进行修改
sudo -E make modules_install
```

- 开始构建 DTB

```
export CROSS_COMPILE=$HOME/l4t-gcc-toolchain/aarch64--glibc--stable-2022.08-1/bin/aarch64-buildroot-linux-gnu-
export KERNEL_HEADERS=$PWD/kernel/kernel-jammy-src
make dtbs
```

- 将生成的 dtb文件拷贝到刷机目录

```
cp kernel-devicetree/generic-dts/dtbs/* ../kernel/dtb/
```

2.4 ????

- 进入刷机固件缓存目录

```
cd $HOME/nvidia/nvidia_sdk/JetPack_6.2.1_Linux_JETSON_ORIN_NANO_TARGETS/Linux_for_Tegra/source  #根据实际目录进行修改
```

使用命令行刷机：

- super□□

```
sudo ./tools/kernel_flash/l4t_initrd_flash.sh --external-device nvme0n1p1 -c
tools/kernel_flash/flash_l4t_t234_nvme.xml -p "-c bootloader/generic/cfg/flash_t234_qspi.xml"
--showlogs --network usb0 jetson-orin-nano-devkit-super internal
```

- □□□□

```
sudo ./flash.sh jetson-orin-nano-devkit-nvme internal
```

2.5 ????

- □ Quectel_Qconnectmanager□□□□□
- □□□□

```
unizp Quectel_QConnectManager_Linux_V1.6.5.zip
cd Quectel_QConnectManager_Linux_V1.6.5
make
```

□□□□□ 4G□□□

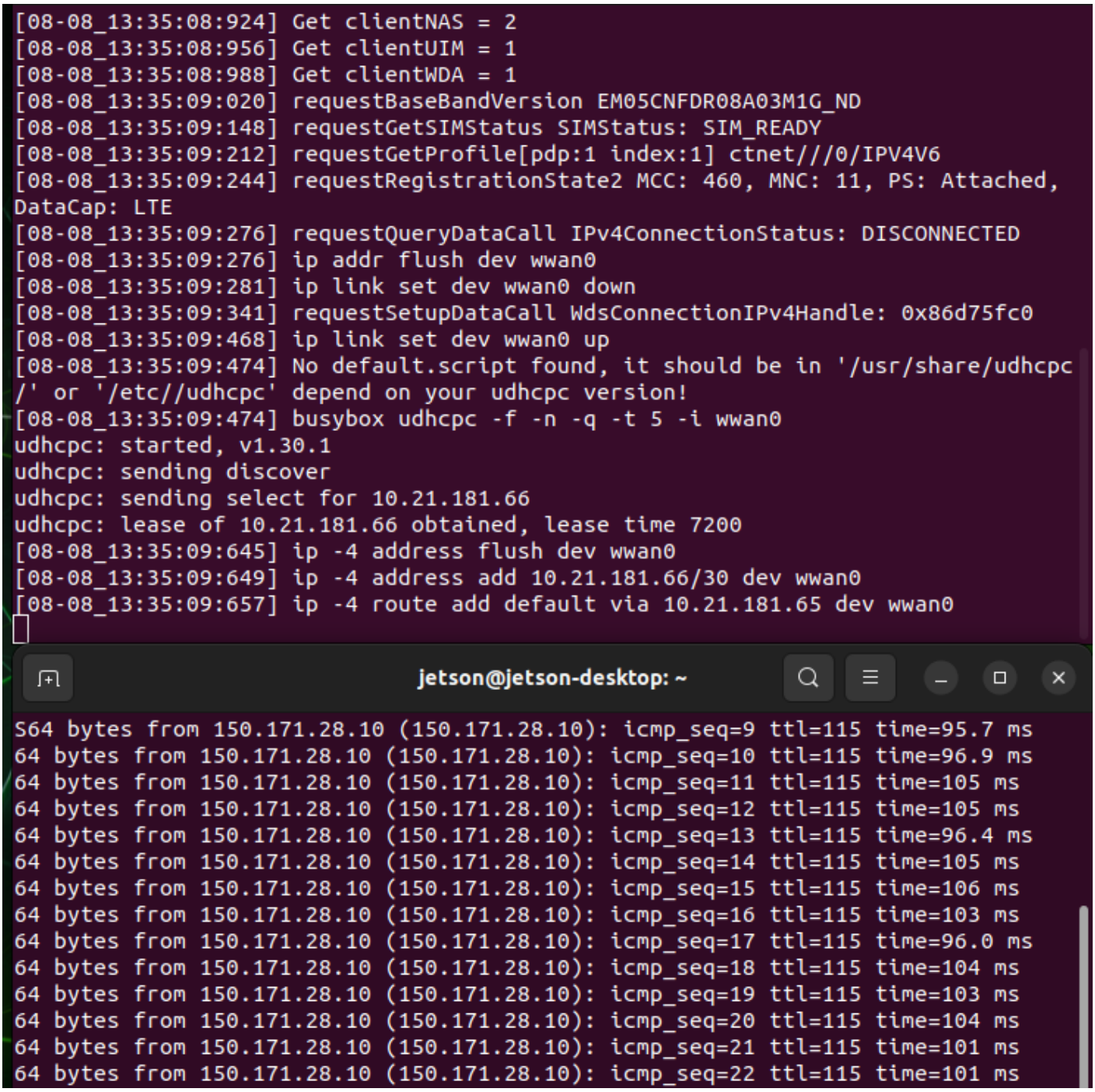
```
jetson@jetson-desktop:~/Downloads/Quectel_QConnectManager_Linux_V1.6.5$ sudo ./quectel-CM
[08-08_11:39:04:415] QConnectManager_Linux_V1.6.5
[08-08_11:39:04:416] Find /sys/bus/usb/devices/1-2.2 idVendor=0x2c7c idProduct=0x125,
bus=0x001, dev=0x007
[08-08_11:39:04:416] Auto find qmichannel = /dev/cdc-wdm0
[08-08_11:39:04:417] Auto find usbnet_adapter = wwan0
[08-08_11:39:04:417] netcard driver = qmi_wwan_q, driver version = V1.2.6
[08-08_11:39:04:417] Modem works in QMI mode
[08-08_11:39:04:447] cdc_wdm_fd = 7
[08-08_11:39:04:523] Get clientWDS = 5
[08-08_11:39:04:557] Get clientDMS = 1
[08-08_11:39:04:589] Get clientNAS = 2
[08-08_11:39:04:620] Get clientUIM = 1
[08-08_11:39:04:653] Get clientWDA = 1
[08-08_11:39:04:684] requestBaseBandVersion EM05CNFDR08A03M1G_ND
[08-08_11:39:04:812] requestGetSIMStatus SIMStatus: SIM_READY
[08-08_11:39:04:876] requestGetProfile[pdp:1 index:1] ctnet///0/IPV4V6
[08-08_11:39:04:908] requestRegistrationState2 MCC: 460, MNC: 11, PS: Attached, DataCap: LTE
[08-08_11:39:04:940] requestQueryDataCall IPv4ConnectionStatus: DISCONNECTED
[08-08_11:39:04:941] ip addr flush dev wwan0
[08-08_11:39:04:947] ip link set dev wwan0 down
```

```
[08-08_11:39:05:003] requestSetupDataCall WdsConnectionIPv4Handle: 0x8723e530
[08-08_11:39:05:132] ip link set dev wwan0 up
[08-08_11:39:05:141] No default.script found, it should be in '/usr/share/udhcpc/' or
'/etc//udhcpc' depend on your udhcpc version!
[08-08_11:39:05:142] busybox udhcpc -f -n -q -t 5 -i wwan0
udhcpc: started, v1.30.1
udhcpc: sending discover
udhcpc: sending select for 10.21.181.66
udhcpc: lease of 10.21.181.66 obtained, lease time 7200
[08-08_11:39:05:282] ip -4 address flush dev wwan0
[08-08_11:39:05:286] ip -4 address add 10.21.181.66/30 dev wwan0
[08-08_11:39:05:292] ip -4 route add default via 10.21.181.65 dev wwan0
```

□□□□□□

```
jetson@jetson-desktop:~$  ifconfig wwan0
wwan0: flags=193<UP,RUNNING,NOARP>  mtu 1500
        inet 10.21.181.66  netmask 255.255.255.252
        inet6 fe80::5804:41ff:feda:ce83  prefixlen 64  scopeid 0x20<link>
        ether 5a:04:41:da:ce:83  txqueuelen 1000  (Ethernet)
        RX packets 9  bytes 2304 (2.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 22  bytes 1854 (1.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

□□□□□


[08-08_13:35:08:924] Get clientNAS = 2
[08-08_13:35:08:956] Get clientUIM = 1
[08-08_13:35:08:988] Get clientWDA = 1
[08-08_13:35:09:020] requestBaseBandVersion EM05CNFDR08A03M1G_ND
[08-08_13:35:09:148] requestGetSIMStatus SIMStatus: SIM_READY
[08-08_13:35:09:212] requestGetProfile[pdp:1 index:1] ctnet///0/IPV4V6
[08-08_13:35:09:244] requestRegistrationState2 MCC: 460, MNC: 11, PS: Attached,
DataCap: LTE
[08-08_13:35:09:276] requestQueryDataCall IPv4ConnectionStatus: DISCONNECTED
[08-08_13:35:09:276] ip addr flush dev wwan0
[08-08_13:35:09:281] ip link set dev wwan0 down
[08-08_13:35:09:341] requestSetupDataCall WdsConnectionIPv4Handle: 0x86d75fc0
[08-08_13:35:09:468] ip link set dev wwan0 up
[08-08_13:35:09:474] No default.script found, it should be in '/usr/share/udhcpc
/' or '/etc//udhcpc' depend on your udhcpc version!
[08-08_13:35:09:474] busybox udhcpc -f -n -q -t 5 -i wwan0
udhcpc: started, v1.30.1
udhcpc: sending discover
udhcpc: sending select for 10.21.181.66
udhcpc: lease of 10.21.181.66 obtained, lease time 7200
[08-08_13:35:09:645] ip -4 address flush dev wwan0
[08-08_13:35:09:649] ip -4 address add 10.21.181.66/30 dev wwan0
[08-08_13:35:09:657] ip -4 route add default via 10.21.181.65 dev wwan0
jetson@jetson-desktop: ~
S64 bytes from 150.171.28.10 (150.171.28.10): icmp_seq=9 ttl=115 time=95.7 ms
64 bytes from 150.171.28.10 (150.171.28.10): icmp_seq=10 ttl=115 time=96.9 ms
64 bytes from 150.171.28.10 (150.171.28.10): icmp_seq=11 ttl=115 time=105 ms
64 bytes from 150.171.28.10 (150.171.28.10): icmp_seq=12 ttl=115 time=105 ms
64 bytes from 150.171.28.10 (150.171.28.10): icmp_seq=13 ttl=115 time=96.4 ms
64 bytes from 150.171.28.10 (150.171.28.10): icmp_seq=14 ttl=115 time=105 ms
64 bytes from 150.171.28.10 (150.171.28.10): icmp_seq=15 ttl=115 time=106 ms
64 bytes from 150.171.28.10 (150.171.28.10): icmp_seq=16 ttl=115 time=103 ms
64 bytes from 150.171.28.10 (150.171.28.10): icmp_seq=17 ttl=115 time=96.0 ms
64 bytes from 150.171.28.10 (150.171.28.10): icmp_seq=18 ttl=115 time=104 ms
64 bytes from 150.171.28.10 (150.171.28.10): icmp_seq=19 ttl=115 time=103 ms
64 bytes from 150.171.28.10 (150.171.28.10): icmp_seq=20 ttl=115 time=104 ms
64 bytes from 150.171.28.10 (150.171.28.10): icmp_seq=21 ttl=115 time=101 ms
64 bytes from 150.171.28.10 (150.171.28.10): icmp_seq=22 ttl=115 time=101 ms

??????

????

- 安装编译依赖工具

```
sudo apt install build-essential bc git bison flex libssl-dev zip libncurses-dev make git
```

创建工具链放置目录

```
mkdir $HOME/l4t-gcc-toolchain
cd $HOME/l4t-gcc-toolchain
```

- 下载并解压交叉编译工具链

```
wget https://developer.nvidia.com/downloads/embedded/l4t/r36_release_v3.0/toolchain/aarch64--glibc--stable-2022.08-1.tar.bz2
tar xf aarch64--glibc--stable-2022.08-1.tar.bz2
```

- 进入到缓存好的刷机固件目录

```
cd $HOME/nvidia/nvidia_sdk/JetPack_6.2.1_Linux_JETSON_ORIN_NANO_TARGETS/Linux_for_Tegra/source
 #根据实际目录进行修改
```

- 同步源码

```
 ./source_sync.sh -k -t <release-tag> #<release-tag>替换为你自己的发行版本如 jetson_36.4.3
```

??????????

Intel????

参考 网卡说明_____ 在开发板安装驱动后进入 `/usr/lib/modules/$(uname -r)/updates` 提取 ko驱动，

复制到固件缓存目录 `Linux_for_Tegra/rootfs/lib/modules/5.15.148-tegra/updates/` 文件夹

realtek 8125??

将在开发板上编译安装好的 ko驱动放入 `Linux_for_Tegra/rootfs/lib/modules/5.15.148-tegra/updates/dkms`

烧录完成进入系统后需要运行命令使能 **r8125.ko**

```
sudo depmod -a
sudo modprobe r8125
```

4G??

参考 4G模块使用说明 2.2

需要改动内核镜像

??HDMI 4K

修改 `tegra234-dcb-p3737-0000.dtsi`
替换 `nvidia,dcb-image` 字段，对应的 HDMI-dcb二进制串可在源码的 `source/hardware/nvidia/t23x/nv-public/overlay/tegra234-dcb-p3767-0000-hdmi.dts` 中找到。

USB???

修改 `tegra234-p3768-0000.dtsi` 的 `padctl@3520000` 节点
添加 usb3-2相关设置

```
padctl@3520000 {
□□□status = "okay";

□□□pads {
□□□□usb2 {
□□□□□lanes {
□□□□□□usb2-0 {
□□□□□□□nvidia,function = "xusb";
□□□□□□□status = "okay";
□□□□□□};

□□□□□□usb2-1 {
□□□□□□□nvidia,function = "xusb";
□□□□□□□status = "okay";
□□□□□□};

□□□□□□usb2-2 {
□□□□□□□nvidia,function = "xusb";
□□□□□□□status = "okay";
□□□□□□};
□□□□□};
□□□□};
```

```
□□□□usb3 {
□□□□□□lanes {
□□□□□□□□usb3-0 {
□□□□□□□□□□nvidia,function = "xusb";
□□□□□□□□□□status = "okay";
□□□□□□□□};

□□□□□□□□usb3-1 {
□□□□□□□□□□nvidia,function = "xusb";
□□□□□□□□□□status = "okay";
□□□□□□□□};
□□□□□□□□
□□□□□□□□usb3-2 {
□□□□□□□□□□nvidia,function = "xusb";
□□□□□□□□□□status = "okay";
□□□□□□□□};
□□□□□□};
□□□□};
□□□};

□□□ports {
□□□□□/* recovery port */
□□□□□usb2-0 {
□□□□□□□mode = "otg";
□□□□□□□vbus-supply = <&vdd_5v0_sys>;
□□□□□□□status = "okay";
□□□□□□□usb-role-switch;
□□□□□};

□□□□□/* hub */
□□□□□usb2-1 {
□□□□□□□mode = "host";
□□□□□□□vbus-supply = <&vdd_1v1_hub>;
□□□□□□□status = "okay";
□□□□□};

□□□□□/* M.2 Key-E */
□□□□□usb2-2 {
□□□□□□□mode = "host";
□□□□□□□vbus-supply = <&vdd_5v0_sys>;
```

```
        status = "okay";
    };

    /* hub */
    usb3-0 {
        nvidia,usb2-companion = <1>;
        status = "okay";
    };

    /* J5 */
    usb3-1 {
        nvidia,usb2-companion = <0>;
        status = "okay";
    };
    usb3-2 {
        nvidia,usb2-companion = <2>;
        status = "okay";
    };
  };

  usb@3550000 {
    status = "okay";

    phys = <&{/bus@0/padctl@3520000/pads/usb2/lanes/usb2-0}>,
           <&{/bus@0/padctl@3520000/pads/usb3/lanes/usb3-1}>;
    phy-names = "usb2-0", "usb3-0";
  };

  usb@3610000 {
    status = "okay";

    phys = <&{/bus@0/padctl@3520000/pads/usb2/lanes/usb2-0}>,
           <&{/bus@0/padctl@3520000/pads/usb2/lanes/usb2-1}>,
           <&{/bus@0/padctl@3520000/pads/usb2/lanes/usb2-2}>,
           <&{/bus@0/padctl@3520000/pads/usb3/lanes/usb3-0}>,
           <&{/bus@0/padctl@3520000/pads/usb3/lanes/usb3-1}>,
           <&{/bus@0/padctl@3520000/pads/usb3/lanes/usb3-2}>;
    phy-names = "usb2-0", "usb2-1", "usb2-2", "usb3-0",
        "usb3-1", "usb3-2";
```

```
□□};
```

?????????????

2.3 ????

> 此步骤可能需要进行半个小时以上

- 创建输出目录

```
cd ../../
mkdir kernel_out
```

- 编译安装内核

```
export CROSS_COMPILE=$HOME/l4t-gcc-toolchain/aarch64--glibc--stable-2022.08-1/bin/aarch64-buildroot-linux-gnu- #设置交叉编译工具
make -C kernel #构建 Jetson Linux 内核镜像
sudo -E make install -C kernel #安装内核模块和树内模块
```

- 将内核镜像复制到刷机目录下

```
cp kernel/kernel-jammy-src/arch/arm64/boot/Image ../kernel/
```

- 构建 NVIDIA树外模块（驱动程序）

```
export CROSS_COMPILE=$HOME/l4t-gcc-toolchain/aarch64--glibc--stable-2022.08-1/bin/aarch64-buildroot-linux-gnu-
export KERNEL_HEADERS=$PWD/kernel/kernel-jammy-src
make modules
```

- 安装到刷机目录下

```
export INSTALL_MOD_PATH=$HOME/nvidia/nvidia_sdk/JetPack_6.2.1_Linux_JETSON_ORIN_NANO_TARGETS/Linux_for_Tegra/rootfs/ #根据实际目录进行修改
sudo -E make modules_install
```

- 开始构建 DTB

```
export CROSS_COMPILE=$HOME/l4t-gcc-toolchain/aarch64--glibc--stable-2022.08-1/bin/aarch64-buildroot-linux-gnu-
export KERNEL_HEADERS=$PWD/kernel/kernel-jammy-src
make dtbs
```

- 将生成的 dtb文件拷贝到刷机目录

```
cp kernel-devicetree/generic-dts/dtbs/* ../kernel/dtb/
```

???????

- 普通模式

```
sudo ./tools/kernel_flash/l4t_initrd_flash.sh --external-device nvme0n1p1 \
-c tools/kernel_flash/flash_l4t_t234_nvme.xml -p "-c
bootloader/generic/cfg/flash_t234_qspi.xml" \
--showlogs --network usb0 jetson-orin-nano-devkit internal
```

- super模式

```
sudo ./tools/kernel_flash/l4t_initrd_flash.sh --external-device nvme0n1p1 \
-c tools/kernel_flash/flash_l4t_t234_nvme.xml -p "-c
bootloader/generic/cfg/flash_t234_qspi.xml" \
--showlogs --network usb0 jetson-orin-nano-devkit-super internal
```

???QSPI???

- 普通模式

```
sudo ./flash.sh -c bootloader/t186ref/cfg/flash_t234_qspi.xml --no-systemimg jetson-orin-nano-devkit nvme0n1p1
```

- super模式

```
sudo ./flash.sh -c bootloader/t186ref/cfg/flash_t234_qspi.xml --no-systemimg jetson-orin-nano-devkit-super nvme0n1p1
```