ROScube Pico BSP Quick Start Guide

Document Generated Date : 2021-03-18
Document Released for: copyright

More information are available at ADLINK Technology Inc. Advanced Robotic Platform Group

Preface

Copyright

Disclaimer

Environmental Responsibility

ADLINK is committed to fulfill its social responsibility to global environmental preservation through compliance with the European Union's Restriction of Hazardous Substances (RoHS) directive and Waste Electrical and Electronic Equipment (WEEE) directive. Environmental protection is a top priority for ADLINK. We have enforced measures to ensure that our products, manufacturing processes, components, and raw materials have as little impact on the environment as possible. When products are at their end of life, our customers are encouraged to dispose of them in accordance with the product disposal and/or recovery programs prescribed by their nation or company.

Trademarks

TABLE OF CONTENTS

CHAPTER ONE

REVISIONS

Document version: l4t-32.4.4-kernel-1.0.7

Compatible with ROScube Pico BSP version:

- L4T-32.4.3-Kernel-1.0.7
- L4T-32.4.4-Kernel-1.0.7

Author	Revision
Chester Tseng	L4T-32.4.3-Kernel-1.0.7
Chester Tseng	L4T-32.4.4-Kernel-1.0.7

1.1 BSP MFI Image Naming Rule

ROScube-Pico BSP MFI file follows the nameing rule.

`mfi_<model name>_<filesystem and distro type>_<L4T version>-Kernel-<kernel version>.tbz2`

For example: `mfi_npn2_nvidia-ubuntu-rootfs-bionic_L4T-32-4-4-Kernel-1-0-7.tbz2`.

1.1.1 Model Name

- `npn1` - ROScube Pico with Nvidia Nano module.
- `npn2` - ROScube Pico with Nvidia NX module.

1.1.2 FileSystem Type

- `nvidia-sample-rootfs` - Nvidia provided sample filesystem. It is based on Ubuntu 18.04
- `custom-slim-rootfs-bionic` - Custom rootfs build from Ubuntu base image 18.04.5. This file type is much smaller.
- `custom-slim-rootfs-focal` - Custom rootfs build from Ubuntu base image 20.04.1. This file type is much smaller.

1.1.3 L4T Version

The L4T version, for example 32.4.3 or 32.3.4

1.1.4 Kernel Version

ROScube-Pico BSP version, which is affect by kernel and device tree from ADLINK's internal engineering development.

1.2 Release Note

Version is based on BSP version (e.g: 1.0.x)

1.2.1 v1.0.0

- NPN2 (ROScube Pico NX)
  - Bring up ROScube Pico Pinmux, generial I/Os.
  - Add FAN and GPIO TachoMeter driver and device tree.
  - Add CanBus device tree.
  - Add Tegra GPIO device tree.
  - Set up Tegra internal ethernet device tree.
  - Set up USB lanes device tree settings.

1.2.2 v1.0.1

- NPN2 (ROScube Pico NX)
  - Add board SD card device tree.
  - Set up pinmux AUX to I2C on HDMI port.
  - Add SX1509 PinCtrl Driver and device tree on 40 pin external header.
  - Add USB OTG device tree and enable USB Gadget Linux Driver.

1.2.3 v1.0.2

- NPN2 (ROScube Pico NX)
  - Fix LAN7800 LAN chip driver interrupt.
  - Use Ubuntu Base as base image.
  - Add extra LXDE desktop environment from Ubuntu base image.

1.2.4 v1.0.3

- NPN2 (ROScube Pico NX)
  - Add ethtools, i2c-tools and pci-utils to custom slim rootfs.
  - Lock kernel version, to provent from upgrading to Nvidia' kernel.

1.2.5 v1.0.4

- NPN2 (ROScube Pico NX)
  - Bring up audio subsystem.

1.2.6 v1.0.5

- NPN2 (ROscube Pico NX)
  - Fix SPI channel number.

1.2.7 v1.0.6

- NPN1 (ROScube Pico Nano)
  - First release ROScube Pico Nano
- NPN2 (ROScube Pico NX)
  - Add OTA server for future use.

1.2.8 L4T-32.4.3-Kernel-v1.0.7

- NPN1 (ROScube Pico Nano)
  - Drive all external GPIO and PWM pins to 0.
  - Enable audio subsystem.
  - Enable SD card. Set SD card detection pin into pull up when boot up.
  - Add OTA server for future use.
  - Add Intel AC9260 install script in BSP.
- NPN2 (ROScube Pico NX)
  - Apply thermal table from thermal team suggested.
  - Add alsa sound setting (Audio out + MIC in)
  - Config SD card detect pin default state to pull up.
  - Drive all GPIO to 0 when boot up.
  - Load mttcan kernel module when boot up.
  - Add Intel AC9260 install script in BSP.

1.2.9 L4T-32.4.4-Kernel-v1.0.7

- NPN1 (ROScube Pico Nano)
  - Upgrade to Nvidia L4T 32.4.4
- NPN2 (ROScube Pico AGX NX)
  - Upgrade to Nvidia L4T 32.4.4

CHAPTER

TWO

FLASHING ROSCUBE PICO IMAGE

ROScube Pico provides 2 skus, which are

1. Board level ROScube Pico (Without chassis.)
2. System level ROScube Pico (With chassis.)

2.1 Prerequisite

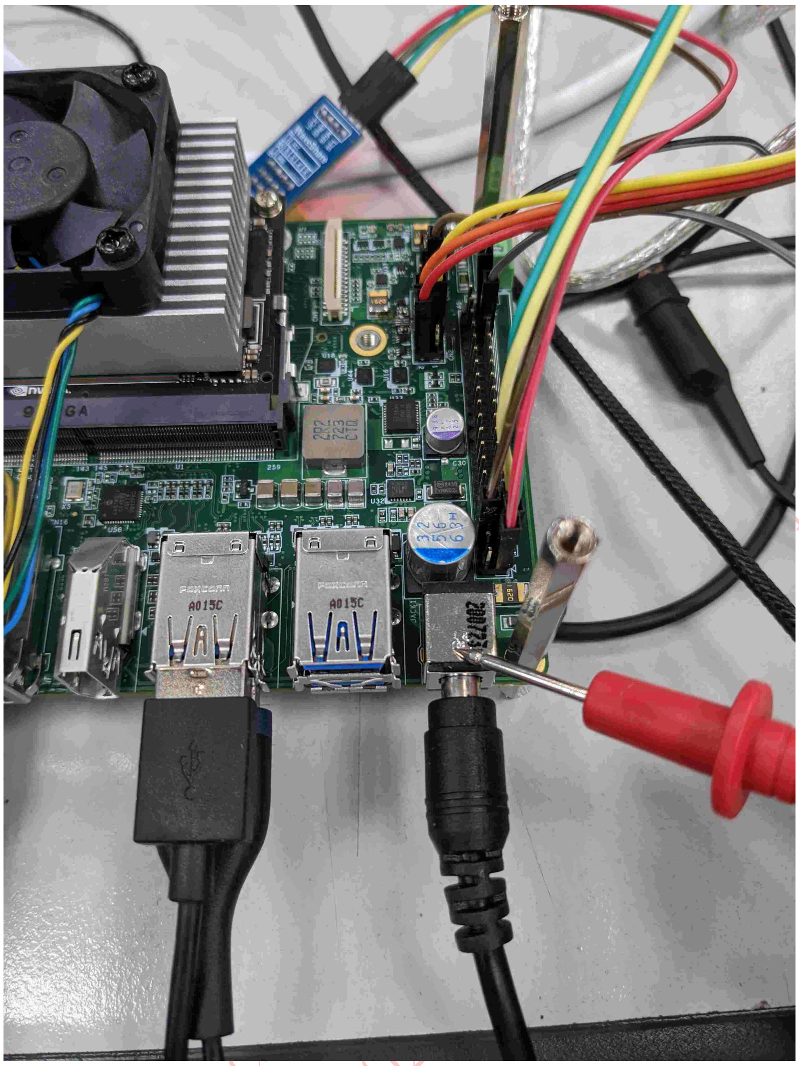
Before flashing image to ROScube Pico, you should prepare the following items:

- **Host PC** with **Ubuntu 18.04 or 20.04** operating system.
- A good quality micro-usb for connecting to ROScube Pico.

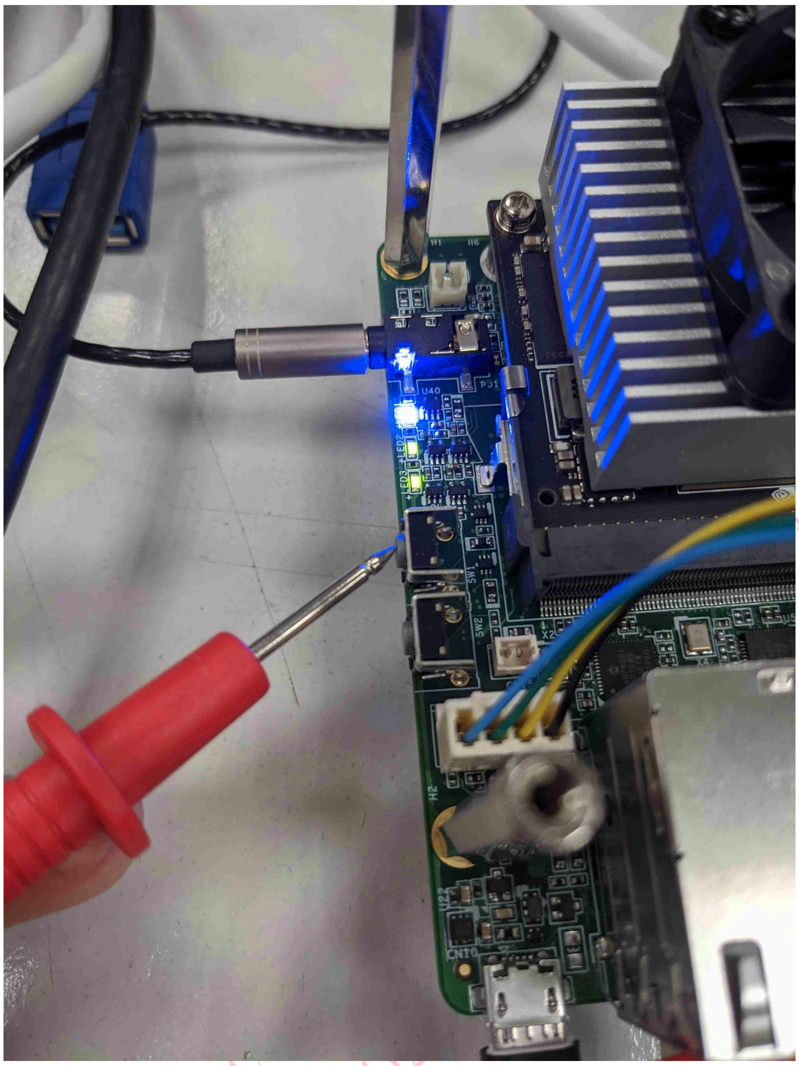
2.2 Board Level ROScube Pico Flashing Image

2.2.1 1. Set ROScube Pico into recovery mode

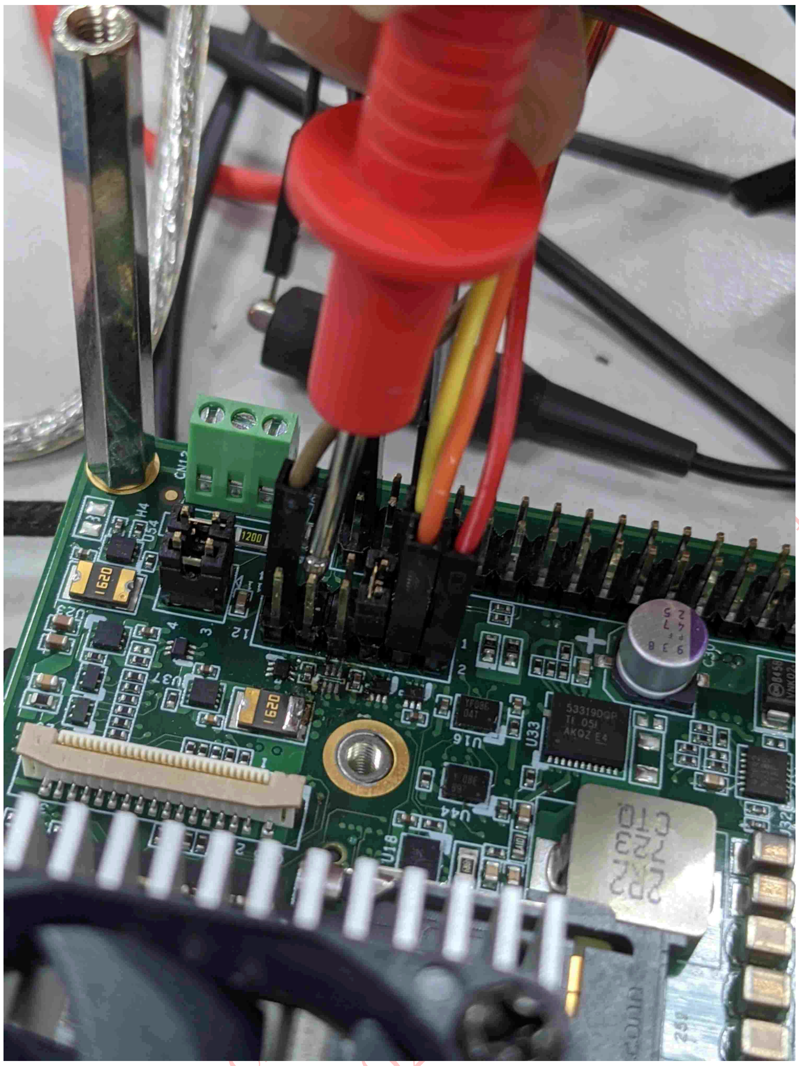
1. Connect power cable to ROScube Pico.



2. Power on ROScube Pico. (Press power button)

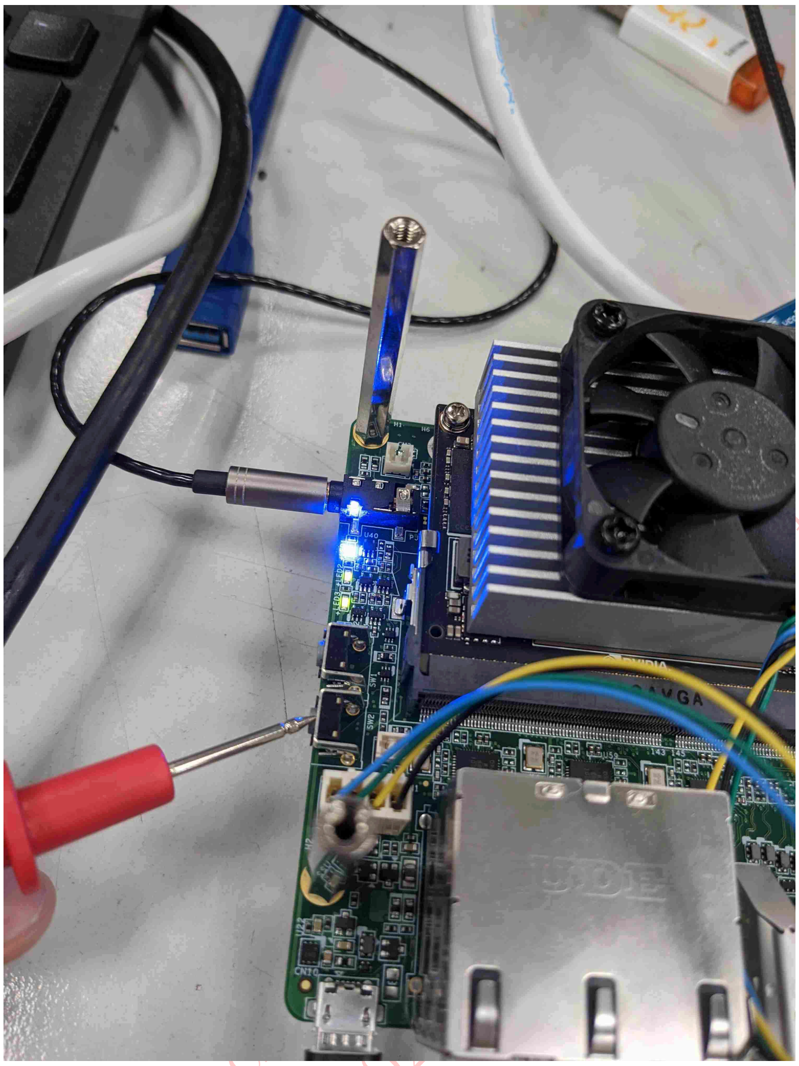


3. Short **Pin 09 and Pin 10** and **hold short status**.



	Pin header		
PC_LED_K	01	02	PC_LED_A (+V5P0)
3V3_UART2_RX	03	04	3V3_UART2_TX
LATCH_SET	05	06	LATCH_SET_BUT
GND	07	08	SYS_RST#
GND	09	10	FORCE_RECOVERY#
GND	11	12	PWR_BTN#

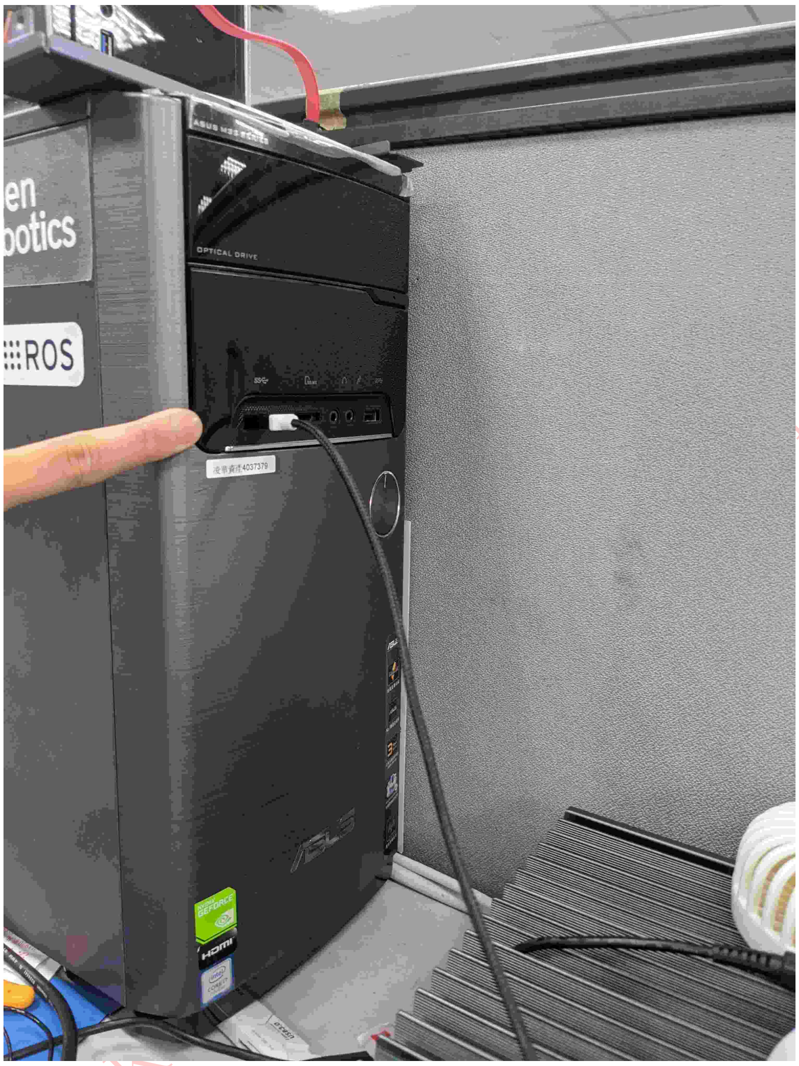
4. **Hold Pin 09 and Pin 10 short status** and press **Reset** button.

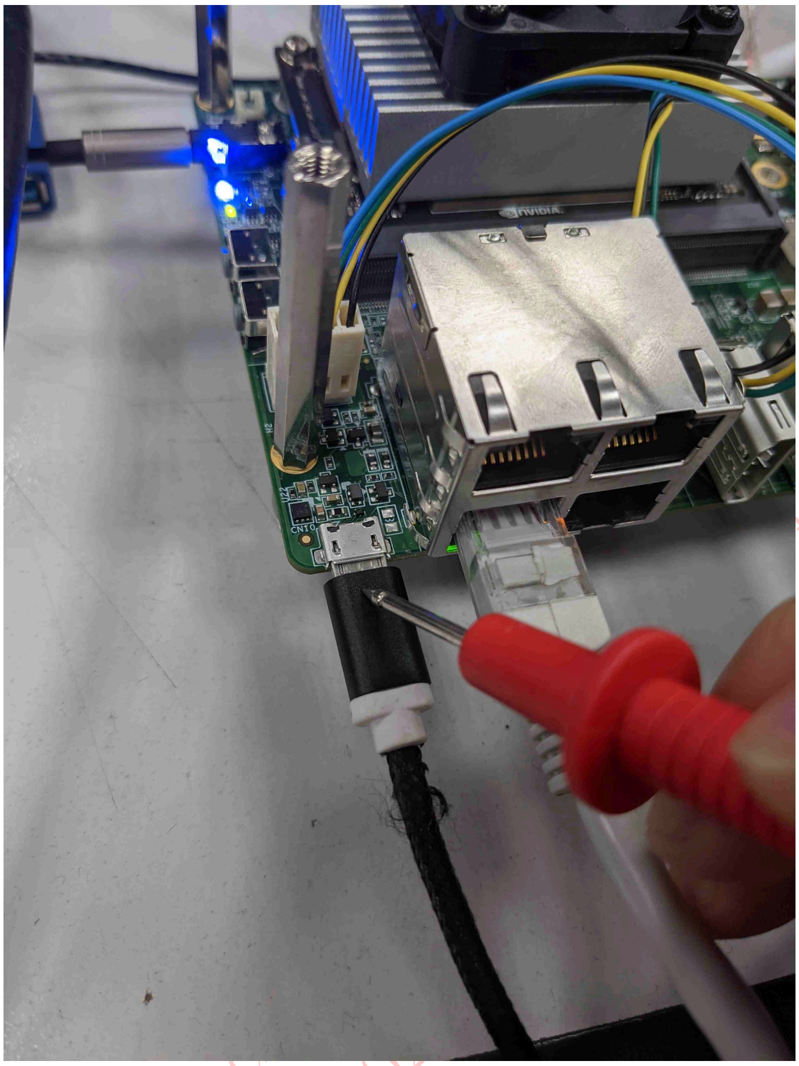


5. Release **Short pins** and **pull out short pins!!**.

Now ROScube Pico is in recovery mode.

2.2.2 2. Connect Host PC and ROScube Pico with micro usb cable.





2.2.3 3. Prepare released image on Host PC

Assuming image's file name is `mfi_npn2_nvidia-sample-rootfs_L4T-32-4-3-Kernel-1-0-6.tbz2`. Un-archive this file first.

```
$> tar xvf mfi_npn2_nvidia-sample-rootfs_L4T-32-4-3-Kernel-1-0-6.tbz2
```

Then, use BSP internal tool, `nvmflash.sh` to run the flashing procedure. **Please make sure your Host PC has attached to ROScube Pico**.

```
$> cd mfi_rqp_nx
$> sudo ./nvmflash.sh
```

Note: You may need to input your **host PC**'s root password when flashing the image.

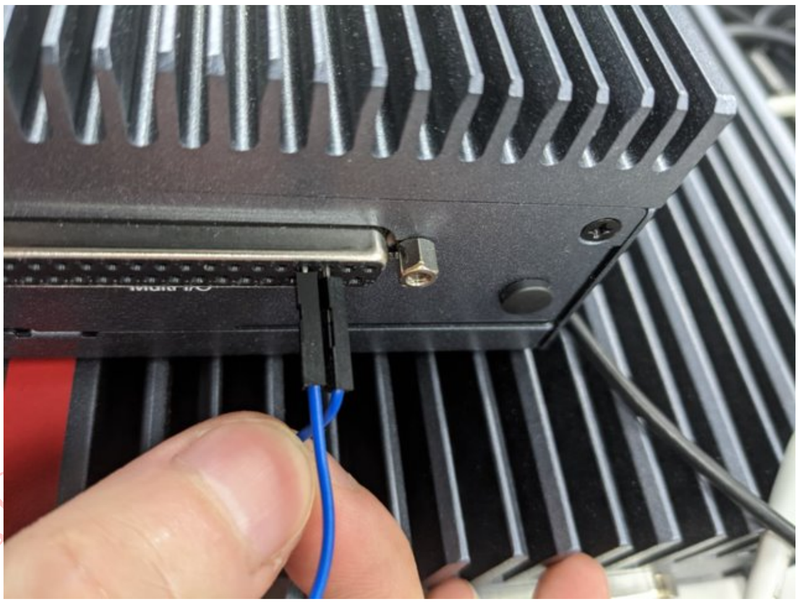
Note: The flashing procedure might take 8 ~ 10 minutes.

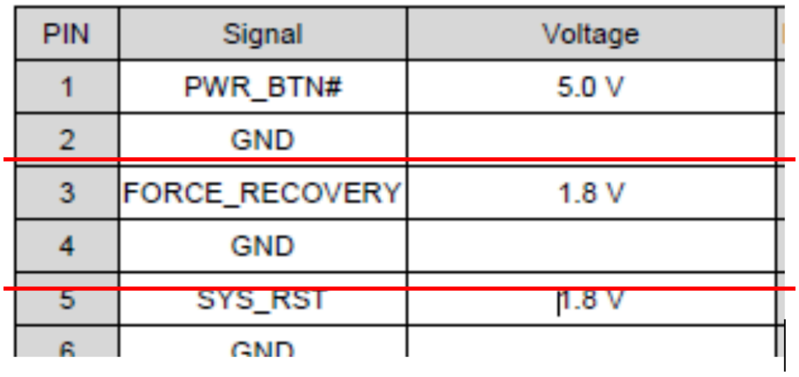
2.3 System Level ROScube Pico Flashing Image

2.3.1 1. Set ROScube Pico into recovery mode

1. Connect power cable to ROScube Pico.

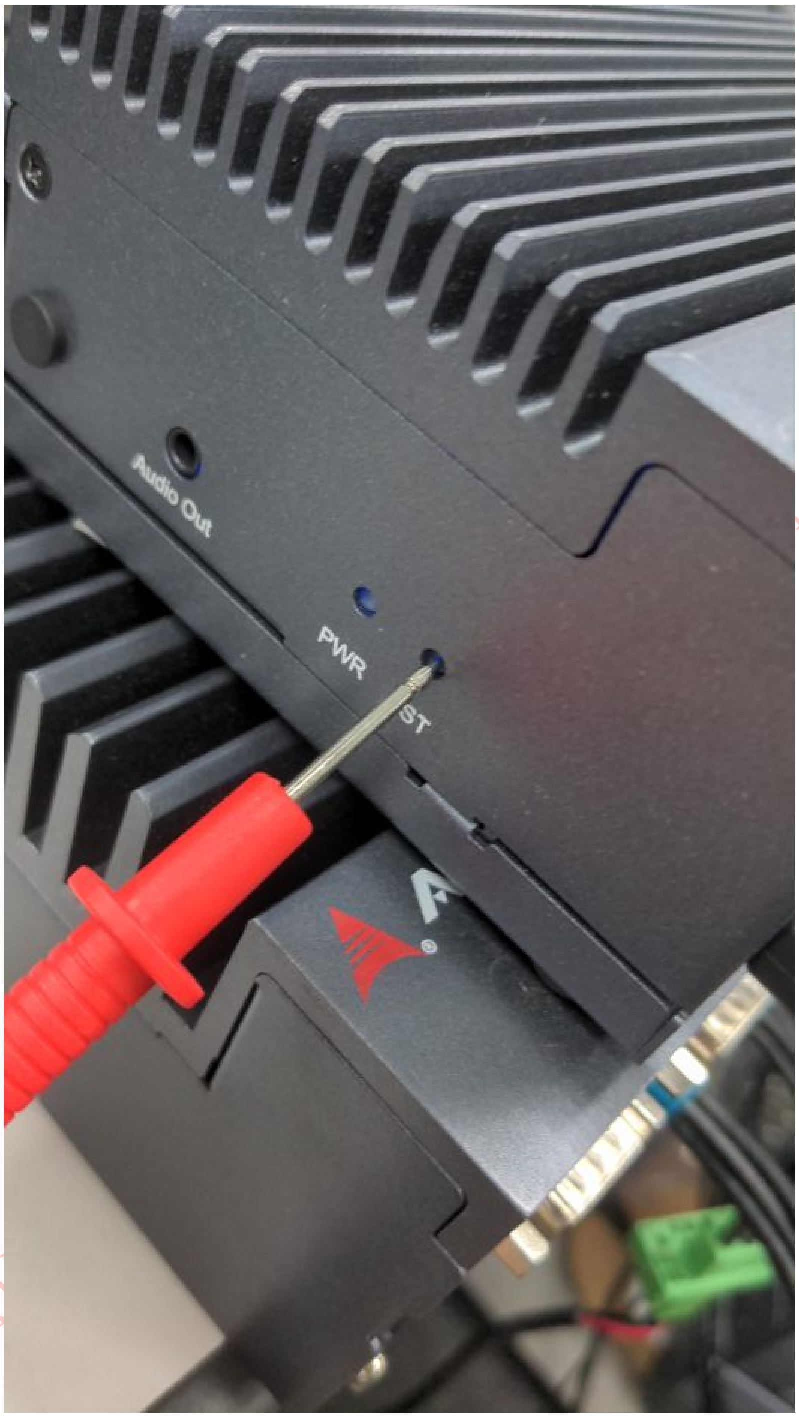
3. Short **Pin 03 and Pin 04** and **hold short status**.



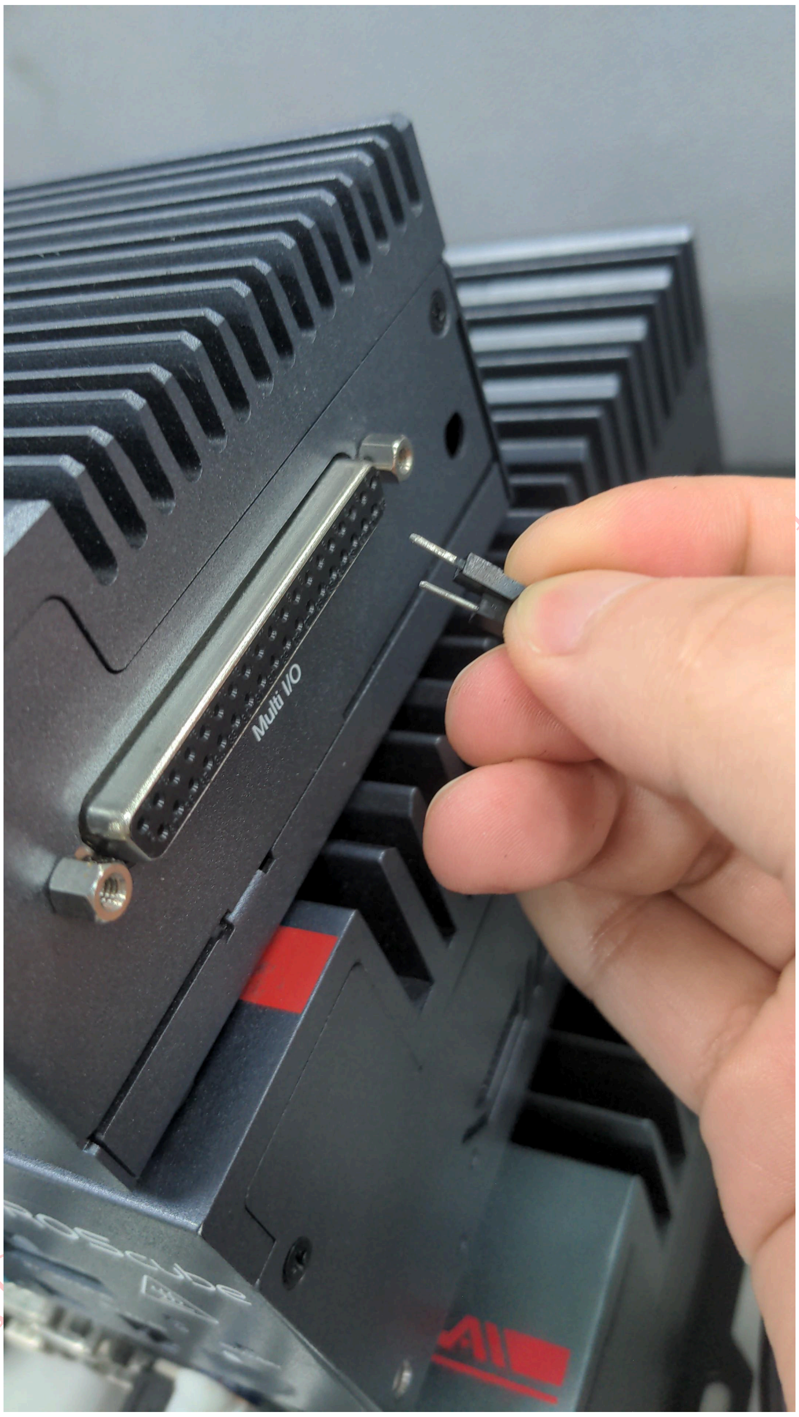


PIN	Signal	Voltage
1	PWR_BTN#	5.0 V
2	GND	
3	FORCE_RECOVERY	1.8 V
4	GND	
5	SYS_RST	1.8 V
6	GND	

4. **Keep Pin 03 and Pin 04 short status** and press **Reset** button.

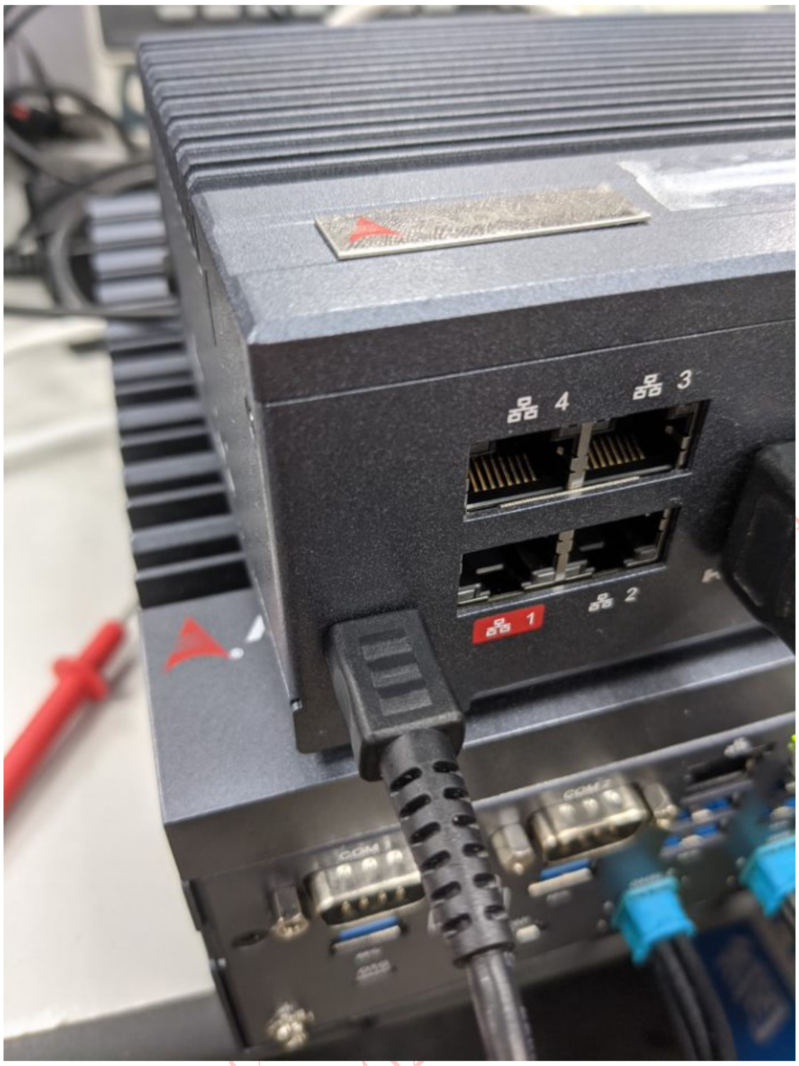


5. Release **Short pins** and **pull out cable !!**.



Now ROScube Pico is in recovery mode.

2.3.2 2. Connect Host PC and ROScube Pico with micro usb cable.



2.3.3 3. Prepare released image on Host PC

Assuming image's file name is `mfi_npn2_nvidia-sample-rootfs_L4T-32-4-3-Kernel-1-0-6.tbz2`. Un-archive this file first.

```
$> tar xvf mfi_npn2_nvidia-sample-rootfs_L4T-32-4-3-Kernel-1-0-6.tbz2
```

Then, use BSP internal tool, `nvmflash.sh` to run the flashing procedure. **Please make sure your Host PC has attached to ROScube Pico**.

```
$> cd mfi_rqp_nx
$> sudo ./nvmflash.sh
```

Note: You may need to input your **host PC**'s root password when flashing the image.

Note: The flashing procedure might take 8 ~ 10 minutes.

2.4 Massive Flashing

`nvmflash.sh` supports massive flashing, which means you can attach multiple ROScube Pico and run nvmflash.sh to flash the image to multiple boards at the same time.

Warning: Please do not attach different models (e.g. ROScube Pico NX and ROScube Pico Nano) to your host computer when using nvmflash.sh.

CHAPTER

THREE

ROSCUBE-PICO BSP QUICK START

3.1 Default User and Password

ROScube-Pico BSP can login as root user within initial stage.

- username: **root**
- password: **adlinkros**

3.2 Check BSP Version

You can use the following command to check BSP and L4T version.

To check L4T version, please use the following command.

```
$> cat /etc/nv_tegra_release
```

To check kernel version, please use the following command.

```
$> uname -a
```

3.3 Hardware Monitor

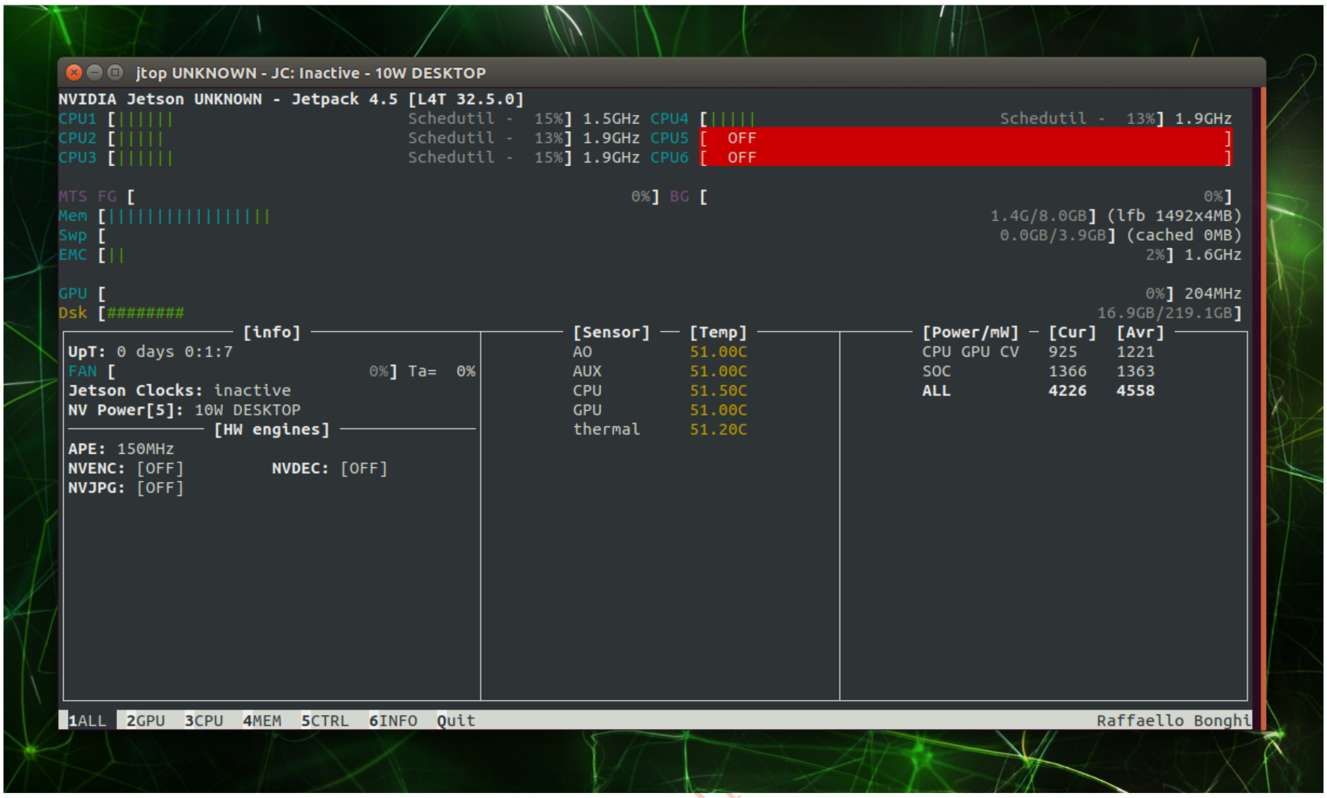
Since generic x86 computer with Nvidia GPU PCIe card under linux has a tool `nvidia-smi` to check GPU status, but ROscube-Pico and ROScube-Pico are ARM based CPU, there's no `nvidia-smi` tool available. You could use a third party tool: `jetson-stats` to monitor Jetson's CPU, GPU and memory status.

```
$> sudo apt install python-pip python-dev build-essential
$> sudo pip install --upgrade pip
$> sudo pip install jetson-stats

# Then reboot system to apply jetson-stats systemcv service.

$> sudo jtop
```

- Overall Hardware Monitor Screenshoot



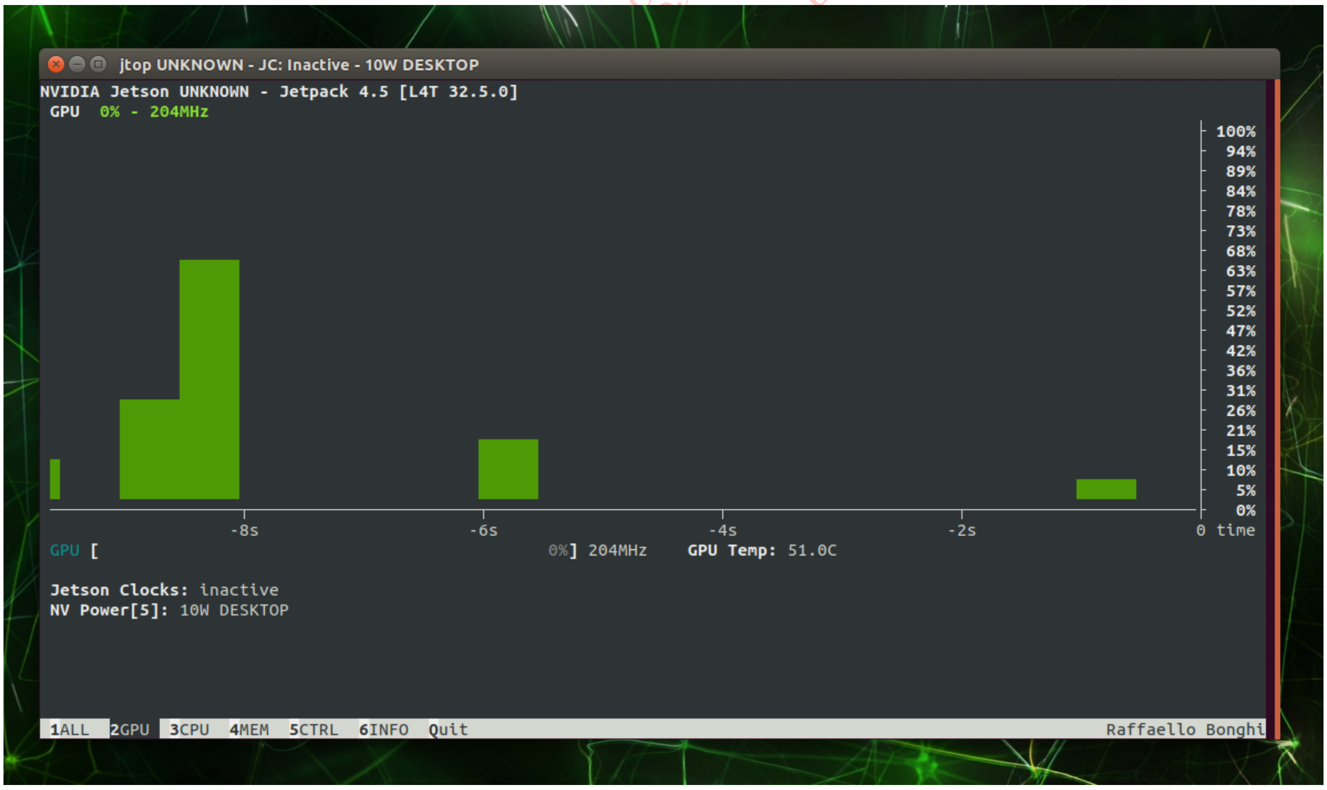
```
jtop UNKNOWN - JC: Inactive - 10W DESKTOP
NVIDIA Jetson UNKNOWN - Jetpack 4.5 [L4T 32.5.0]
CPU1 [||||||                Schedutil -  15%] 1.5GHz CPU4 [|||||                    Schedutil -  13%] 1.9GHz
CPU2 [|||||                 Schedutil -  13%] 1.9GHz CPU5 [  OFF                                      ]
CPU3 [||||||                Schedutil -  15%] 1.9GHz CPU6 [  OFF                                      ]

MTS FG [                                        0%] BG [                                         0%]
Mem [||||||||||||||||                                           1.4G/8.0GB] (lfb 1492x4MB)
Swp [                                                           0.0GB/3.9GB] (cached 0MB)
EMC [||                                                                  2%] 1.6GHz

GPU [                                                                    0%] 204MHz
Dsk [########                                                      16.9GB/219.1GB]
 ──────────── [info] ────────────────── [Sensor] ─ [Temp] ─────────── [Power/mW] ─ [Cur]  [Avr] ──
 UpT: 0 days 0:1:7                    AO          51.00C              CPU GPU CV    925    1221
 FAN [                0%] Ta=  0%     AUX         51.00C              SOC           1366   1363
 Jetson Clocks: inactive              CPU         51.50C              ALL           4226   4558
 NV Power[5]: 10W DESKTOP             GPU         51.00C
 ────────── [HW engines] ──────────   thermal     51.20C
 APE: 150MHz
 NVENC: [OFF]        NVDEC: [OFF]
 NVJPG: [OFF]

 1ALL  2GPU  3CPU  4MEM  5CTRL  6INFO  Quit                                         Raffaello Bonghi
```

- GPU RealTime Monitor Screenshoot

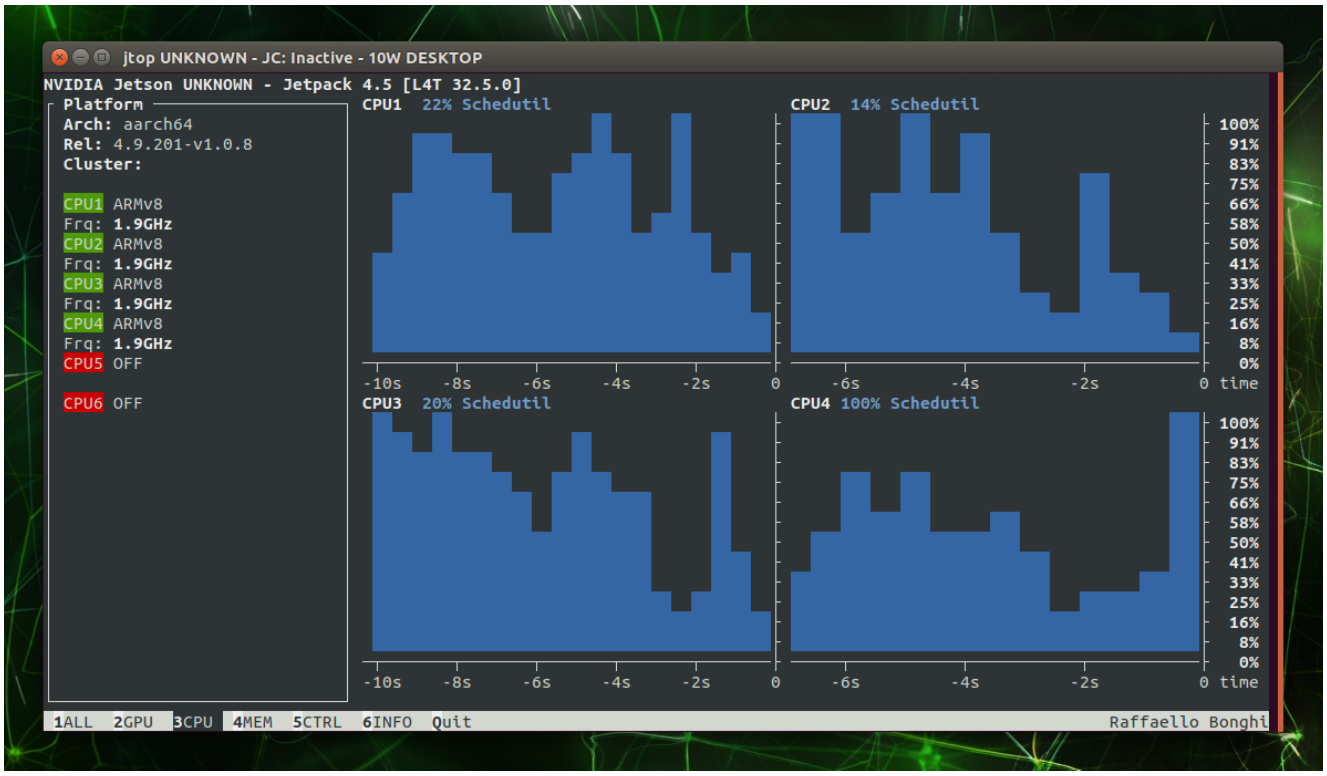


```
jtop UNKNOWN - JC: Inactive - 10W DESKTOP
NVIDIA Jetson UNKNOWN - Jetpack 4.5 [L4T 32.5.0]
 GPU  0% - 204MHz
                                                                                         100%
                                                                                          94%
                                                                                          89%
                                                                                          84%
                                                                                          78%
                                                                                          73%
                                                                                          68%
                                                                                          63%
                                                                                          57%
                                                                                          52%
                                                                                          47%
                                                                                          42%
                                                                                          36%
                                                                                          31%
                                                                                          26%
                                                                                          21%
                                                                                          15%
                                                                                          10%
                                                                                           5%
                                                                                           0%
               -8s                -6s                -4s                -2s             0 time
 GPU [                                    0%] 204MHz    GPU Temp: 51.0C

 Jetson Clocks: inactive
 NV Power[5]: 10W DESKTOP

 1ALL  2GPU  3CPU  4MEM  5CTRL  6INFO  Quit                                         Raffaello Bonghi
```

- CPU RealTime Monitor Screenshoot



```
NVIDIA Jetson UNKNOWN - Jetpack 4.5 [L4T 32.5.0]
 Platform           CPU1  22% Schedutil          CPU2  14% Schedutil
 Arch: aarch64
 Rel: 4.9.201-v1.0.8
 Cluster:

 CPU1 ARMv8
 Frq: 1.9GHz
 CPU2 ARMv8
 Frq: 1.9GHz
 CPU3 ARMv8
 Frq: 1.9GHz
 CPU4 ARMv8
 Frq: 1.9GHz
 CPU5 OFF

 CPU6 OFF
                    CPU3  20% Schedutil          CPU4 100% Schedutil

1ALL  2GPU  3CPU  4MEM  5CTRL  6INFO  Quit                       Raffaello Bonghi
```

Note: Before using jetson-stats, you should install nvidia-jetpack first.

CHAPTER

FOUR

DRIVER PACKAGE USAGE

Driver Package is a package contains several assets that runs on host computer, which allows you to do:

1. Customize system file system.
2. Flash system image to target machine.
3. System image back up.
4. Create a custom mfi image.

User can modify their rootfs on their host computer before flashing the image to target machine.

You could use tar tool to unarchive driver_package.

```
$> tar xvf RQX-580-Driver-Package-L4T-32-4-4-Kernel-1-0-8.tar.gz
```

4.1 Customize System FileSystem

4.1.1 Prepare Filesystem

Driver package allows user to customize their filesystem. Driver package provides a script (`create_image.sh`) to create a board specific image base on several rootfs selection, here's what we support.

1. nvidia_sample_rootfs - Nvidia provided sample file system.

Please download the correct L4T version from https://developer.nvidia.com/embedded/linux-tegra-archive. For example, if you are using ADLINK provided driver package 32.4.4, you should download https://developer.nvidia.com/embedded/L4T/r32_Release_v4.4/r32_Release_v4.4-GMC3/T186/Tegra_Linux_Sample-Root-Filesystem_R32.4.4_aarch64.tbz2

2. custom_slim_rootfs_bionic - Ubuntu provided arm64 18.04.05 minimun image.

Please download ubuntu 18.04.5 base image from Ubuntu's website. https://cdimage.ubuntu.com/ubuntu-base/releases/18.04.5/release/. The binary link is at: https://cdimage.ubuntu.com/ubuntu-base/releases/18.04.5/release/ubuntu-base-18.04.5-base-arm64.tar.gz

3. custom_slim_rootfs_focal - Ubuntu provided arm64 20.04.01 minimun image. (Experimental, not fully tested yet)

Please download ubuntu 20.04.1 base image from Ubuntu's website. https://cdimage.ubuntu.com/ubuntu-base/releases/20.04.1/release/. The binary link is at: https://cdimage.ubuntu.com/ubuntu-base/releases/20.04.1/release/ubuntu-base-20.04.1-base-arm64.tar.gz

Note: Before customizing the rootfs, all these rootfs binaries should be downloaded from their official website. ADLINK doesn't provide them.

After you download rootfs binary successfully, put the binary under the driver package folder. Then, run `./create_image.sh`.

```
$> cd <driver_package_path>
$> sudo ./create_image.sh nvidia_sample_rootfs
```

4.1.2 Modify Filesystem under Chroot

Users can chroot into filesystem, and install packages or put files to filesystem. That allows user to put their software and configurations into rootfs.

Mount host environment to filesystem.

```
$> sudo apt update qemu-user-static
$> cd <driver package>
$> cd rootfs
$> sudo cp /usr/bin/qemu-aarch64-static usr/bin/
$> sudo mount --bind /dev/ dev/
$> sudo mount --bind /sys/ sys/
$> sudo mount --bind /proc/ proc/

# Copy host environment dns settings.
$> sudo cp /etc/resolv.conf etc/resolv.conf.host
$> sudo mv etc/resolv.conf etc/resolv.conf.saved
$> sudo mv etc/resolv.conf.host etc/resolv.conf
```

Now you can chroot into filesystem and modify filesystem.

```
$> sudo LC_ALL=C LANG=C.UTF-8 chroot . /bin/bash
$(chroot)> apt update
$(chroot)> apt install wget vim

# or you can install jetpack sdk
$(chroot)> apt install nvidia-l4t-jetson-multimedia-api=32.4.4-20201016123640
$(chroot)> apt install nvidia-jetpack
```

Once you finish modification, you can exit chroot environment by pressing ctrl+D. Then, remove cache files, e.g. apt cache to save storage space.

```
$(chrooot)> exit
```

(continues on next page)

(continued from previous page)

```
$> sudo umount ./proc
$> sudo umount ./sys
$> sudo umount ./dev
$> sudo rm usr/bin/qemu-aarch64-static

# Restore dns setting.
$> sudo rm etc/resolv.conf
$> sudo mv etc/resolv.conf.saved etc/resolv.conf

# Remove caches and logging files.
$> sudo rm -rf var/lib/apt/lists/*
$> sudo rm -rf dev/*
$> sudo rm -rf var/log/*
$> sudo rm -rf var/tmp/*
$> sudo rm -rf var/cache/apt/archives/*.deb
$> sudo rm -rf tmp/*
```

4.2 Flash System Image to Target Machine

Once you prepared rootfs in step: *customize rootfs*. You can use *flash.sh* tool to flash image to target machine. **Remember to put machine into recovery mode before flashing.**

```
# Put machine into recovery mode first.
$> sudo ./flash.sh <target name> mmcblk0p1
```

Model	Target Name
RQX-580	rqx_580
RQX-58G	rqx_58g
NPN-1 (Pico Nano)	rqp_nano
NPN-2 (Pico AGX NX)	rqp_nx

4.3 Prepare a Bootable Disk

Driver package provides a tool `./tools/jetson-disk-image-creator.sh` to pack rootfs into a bootable image. This image could be put in `sd card` or `usb dongle`. Then, after a proper `extlinux.conf` setting, Linux Kernel will mount the bootable disk to it's root filesystem.

```
$> sudo ./tools/jetson-disk-image-creator.sh -o <image-name>  -b <target>
```

`<image-name>` is the output disk file, it could be any string, and `<target>` is the target name of the model. For example:

```
$> sudo ./tools/jetson-disk-image-creator.sh -o sd-blob.img  -b rqx_580
```

Then, flash the image to the bootable disk with `dd`.

```
$> sudo dd if=<image-name> of=/dev/<disk name> bs=1M oflag=direct
```

For example, flash the bootable image to sd card.

```
$> sudo dd if=sd-blob.img of=/dev/mmcblk0p1 bs=1M oflag=direct
```

4.4 Boot from the Bootable Disk

Modify `extlinux.conf` in the target machine. `extlinux.conf` is located at `/boot/extlinux/extlinux.conf`. Find the `primary` LABEL,

```
TIMEOUT 30
DEFAULT primary

MENU TITLE L4T boot optiions

LABEL primary
    MENU LABEL primary kernel
    LINUX /boot/Image
    INITRD /boot/initrd
    APPEND ${cbootargs} quiet root=mmcblk0p1 rw rootwait rootfstype=ext4 console=ttyTCU0,
↪115200n8 console=tty0 fbcon=map:0 net.ifnames=0
```

Replace `root=mmcblk0p1` with `root=mmcblk1p1`. Then, reboot machine.

CHAPTER

FIVE

NVIDIA JETSON SOFTWARE STACK

5.1 Version Table

L4T	Linux Kernel	Jetpack SDK	CUDA	TensorRT	CUDNN	VisionWork	OpenCV	VPI	DeepStream SDK
32.4.3	4.9.190	4.4.0	10.2	7.1.3	8.0	1.6	4.1.1	1.0.3	5.0
32.4.4	4.9.190	4.4.1	10.2	7.1.3	8.0	1.6	4.1.1	1.0.3	5.0

5.2 JetPack SDK Installation

Since Nvidia L4T 32.4.x support L4T OTA, that users can download Nvidia JetPack from Nvidia's official APT repository. For ROScube-Pico users, you can use `apt` to install Nvidia JetPack compoments, e.g. tensorrt, cuda.

Before installing JetPack SDK, you should install `nvidia-l4t-jetson-multimedia-api` in a proper version coresponding to L4T version (e.g. 32.4.3 or 32.4.4).

```
$> sudo apt update
$> sudo apt install nvidia-l4t-jetson-multimedia-api=32.4.4-20201016123640
```

To install the whole JetPack SDK

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

To install particular JetPack SDK compoment, e.g. nvidia-tensorrt.

```
$> sudo apt update
$> sudo apt install nvidia-tensorrt
```

> **Warning:** It's not suggested to install `nvidia-jetpack` directly, because nvidia-jetpack is a meta package which will install `ALL` JetPack SDK compoments. All JetPack SDK compomenets occupy `8GB`.

5.3 JetPack SDK Example Usage

5.3.1 YOLO Object Detection

1. Set up your environment

```
$> export CUDA_HOME=/usr/local/cuda
$> export PATH=$CUDA_HOME/bin:$PATH
$> export LD_LIBRARY_PATH=$CUDA_HOME/lib64:$LD_LIBRARY_PATH
```

2. Check `nvcc` *is workable*

```
$> nvcc --version
...
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2019 NVIDIA Corporation
...
```

3. Retrive darknet repository from github

```
$> sudo apt update
$> sudo apt install git
$> git clone https://github.com/AlexeyAB/darknet.git
```

4. Build darknet with CUDNN and OpenCV support

Modify darknet's Makefile with the following..

- GPU=1
- CUDNN=1
- OPENCV=1

```
$> sudo apt update
$> sudo apt install make build-essential
$> cd darknet
$> # Edit Makefile GPU=1, CUDNN=1, OPENCV=1
$> make
```

5. Download the pre-trained weights

```
$> wget https://pjreddie.com/media/files/yolov3-tiny.weights
```

Warning: ADLINK doesn't own the pre-train data. Thsi pre-trained data is a contribution from original author in community.

6. Run object detector from example image.

```
$> ./darknet detect cfg/yolov3-tiny.cfg yolov3-tiny.weights data/dog.jpg
```

5.4 DeepStream SDK Installation

Download the DeepStream 5.0 for Jetson

https://developer.nvidia.com/deepstream-getting-started

```
$> sudo apt install ./deepstream_sdk_5.0_arm64.deb
```

Note: Please refer *jetson software version table* to download proper DeepStream SDK version.

Note: The total Nvidia L4T Ubuntu System + JetPack SDK + DeepStream SDK occupy 17GB, which might not sufficient for ROScube-Pico's internal eMMC storage space. User might consider a larger space SD Card and use it as a bootable disk.

5.5 DeepStream SDK Example Usage

After finished DeepStream SDK installation, there will be some DeepStream examples and tools available under `/opt/nvidia/deepstream/deepsteeam-<version>`

5.5.1 DeepStream Sample App

```
$> deepstream-app -c /opt/nvidia/deepstream/deepstream-5.0/samples/configs/deepstream-app/
↪<config file>
```

Where <config file> could be replaced with the following table:

Model	Config File
RQX-580	source12_1080p_dec_infer-resnet_tracker_tiled_display_fp16_tx2.txt
RQX-58G	source12_1080p_dec_infer-resnet_tracker_tiled_display_fp16_tx2.txt
NPN-1 (Pico Nano)	source8_1080p_dec_infer-resnet_tracker_tiled_display_fp16_nano.txt
NPN-2 (Pico AGX NX)	source12_1080p_dec_infer-resnet_tracker_tiled_display_fp16_tx2.txt