



## **Installing Tools on Aerial Devkit**

# Table of contents

Prerequisites

---

Configure BIOS Settings

---

Install Ubuntu 22.04 Server

---

Verify BIOS Settings

---

Configure the Network Interfaces

---

Disable Auto Upgrade

---

Install the Low-Latency Kernel

---

Configure Linux Kernel Command-line

---

Apply the Changes and Reboot to Load the Kernel

---

Disabling Nouveau

---

Install Dependency Packages

---

Install RSHIM and Mellanox Firmware Tools on the Host

---

Install CUDA Driver

---

Install GDRCopy Driver

---

Install Docker CE

---

Install the Nvidia Container Toolkit

---

Install ptp4l and phc2sys

---

Setup the Boot configuration service

---

Install Matlab (Optional)

---

This chapter describes how to install the required kernel, driver, and tools on the host. This is a one-time installation and can be skipped if the system has been configured already.

- In the following sequence of steps, the target host is the [Aerial Devkit](#).
- Depending on the release, tools that are installed in this section may need to be upgraded in the [Installing and Upgrading Aerial cuBB](#) section.
- After everything is installed and updated, refer to the *cuBB Quick Start Guide* on how to use the Aerial cuBB.

## Prerequisites

1. Install the GPU card and CX6-DX NIC.
2. Connect the CX6-DX port 0 on both servers using a 100GbE cable.
3. Connect the Internet port to the local network.

## Configure BIOS Settings

Change the following system BIOS settings to improve network performance:

1. Set the Power Policy to **Best Performance**.
2. Set the Power Policy SpeedStep (Pstates) option to “Disabled”.
3. Disable **HyperThreading**. Ensure you have performed Step 1 above before doing this step.
4. Save the BIOS settings, and then reboot the system.

## Install Ubuntu 22.04 Server

After installing Ubuntu 22.04 Server, verify the following:

- The system time is correct, to avoid apt update error. If not, see [How to fix system time](#).

- The LVM volume uses the whole disk space. If not, see [How to resize LVM volume](#).
- Check that the GPU and NIC are detected by the OS:

Use the following commands to determine whether the GPU and NIC are detected by the OS:

```
$ lspci |grep -i nvidia # If the system has A100 40G GPU installed b6:00.0 3D controller: NVIDIA Corporation Device 20f1 (rev a1) # If the system has A100 80G GPU installed b6:00.0 3D controller: NVIDIA Corporation Device 20b5 (rev a1) # If the system has A100X GPU installed bb:00.0 3D controller: NVIDIA Corporation Device 20b8 (rev a1) $ lspci |grep -i mellanox b5:00.0 Ethernet controller: Mellanox Technologies MT2892 Family [ConnectX-6 Dx] b5:00.1 Ethernet controller: Mellanox Technologies MT2892 Family [ConnectX-6 Dx]
```

## Verify BIOS Settings

Run the following command to verify that the BIOS setting update took effect:

```
$ lscpu
```

Verify that hyperthreading is disabled:

```
# Thread(s) per core: 1
```

## Configure the Network Interfaces

The following installation steps need an Internet connection. Ensure that you have the proper netplan config for your local network.

The network interface names could change after reboot. To ensure persistent network interface names after reboot, create a persistent net link files under `/etc/systemd/network`, one for each interface.

To find the MAC address of the CX6-DX NIC, run `lshw` to check for network devices and look for the `ConnectX-6 Dx` entries.

```
$ sudo apt-get install jq -y $ sudo lshw -json -C network | jq '.[] | "\(.product), MAC: \(.serial)"' | grep "ConnectX-6 Dx" "MT2892 Family [ConnectX-6 Dx], MAC: b8:ce:f6:xx:xx:xx" "MT2892 Family [ConnectX-6 Dx], MAC: b8:ce:f6:yy:yy:yy"
```

Create files at `/etc/systemd/network/` with the desired name for the interface and the MAC address found in the previous step.

```
$ sudo nano /etc/systemd/network/20-aerial00.link [Match]
MACAddress=b8:ce:f6:xx:xx:xx [Link] Name=aerial00 $ sudo nano
/etc/systemd/network/20-aerial01.link [Match] MACAddress=b8:ce:f6:yy:yy:yy [Link]
Name=aerial01
```

To apply the change:

```
$ sudo netplan apply
```

## Disable Auto Upgrade

Edit the `/etc/apt/apt.conf.d/20auto-upgrades` system file, and change the “1” to “0” for both lines. This prevents the installed version of the low latency kernel from being accidentally changed with a subsequent software upgrade.

```
$ sudo nano /etc/apt/apt.conf.d/20auto-upgrades APT::Periodic::Update-Package-Lists "0"; APT::Periodic::Unattended-Upgrade "0";
```

## Install the Low-Latency Kernel

If the low latency kernel is not installed, you must remove the old kernels and keep only the latest generic kernel. Enter the following command to list the installed kernels:

```
$ dpkg --get-selections | grep -i 'linux-image' | awk '/ii/{ print $2}' # To remove old kernel $ sudo apt-get purge linux-image-<old kernel version> $ sudo apt-get autoremove
```

Install the low-latency kernel with the specific version listed in the release manifest.

```
$ sudo apt-get update $ sudo apt-get install -y linux-image-5.15.0-1042-nvidia-lowlatency
```

Update the GRUB to change the default boot kernel:

```
# Update grub to change the default boot kernel $ sudo sed -i 's/^GRUB_DEFAULT=.*\/GRUB_DEFAULT="Advanced options for Ubuntu>Ubuntu, with Linux 5.15.0-1042-nvidia-lowlatency"/' /etc/default/grub
```

## Configure Linux Kernel Command-line

To set kernel command-line parameters, edit the `GRUB_CMDLINE_LINUX_DEFAULT` parameter in the GRUB file `/etc/default/grub` and append or update the parameters described below. The following kernel parameters are optimized for Aerial DevKit with 24 cores Xeon Gold 6240R and 96GB memory.

To automatically append the GRUB file with these changes, enter this command:

```
$ sudo sed -i 's/^GRUB_CMDLINE_LINUX_DEFAULT="[^\"]*/& default_hugepagesz=1G hugepagesz=1G hugepages=16 tsc=reliable clocksource=tsc intel_idle.max_cstate=0 mce=ignore_ce processor.max_cstate=0 intel_pstate=disable audit=0 idle=poll isolcpus=managed_irq,domain,2-21 nohz_full=2-21 rcu_nocbs=2-21 rcu_nocb_poll nosoftlockup iommu=off intel_iommu=off irqaffinity=0-1,22-23 numa_balancing=disable/' /etc/default/grub
```

The CPU-cores-related parameters need to be adjusted depending on the number of CPU cores on the system. In the example above, the “2-21” value represents CPU core numbers 2 to 21; you may need to adjust this parameter depending on the HW configuration. By default, only one DPDK thread is used. The isolated CPUs are used by

the entire cuBB software stack. Use the `nproc --all` command to see how many cores are available. Do not use core numbers that are beyond the number of available cores.

### **Warning**

These instructions are specific to Ubuntu 22.04 with a 5.15 low-latency kernel provided by Canonical. Make sure the kernel commands provided here are suitable for your OS and kernel versions and revise these settings to match your system if necessary.

## Apply the Changes and Reboot to Load the Kernel

```
$ sudo update-grub $ sudo reboot
```

After rebooting, enter the following command to verify that the system has booted into the low-latency kernel:

```
$ uname -r 5.15.0-1042-nvidia-lowlatency
```

Enter this command to verify that the kernel command-line parameters are configured properly:

```
$ cat /proc/cmdline BOOT_IMAGE=/vmlinuz-5.15.0-1042-nvidia-lowlatency
root=/dev/mapper/ubuntu--vg-ubuntu--lv ro default_hugepagesz=1G
hugepagesz=1G hugepages=16 tsc=reliable clocksource=tsc intel_idle.max_cstate=0
mce=ignore_ce processor.max_cstate=0 intel_pstate=disable audit=0 idle=poll
isolcpus=managed_irq,domain,2-21 nohz_full=2-21 rcu_nocbs=2-21 rcu_nocb_poll
nosoftlockup iommu=off intel_iommu=off irqaffinity=0-1,22-23
numa_balancing=disable
```

Enter this command to check if hugepages are enabled:

```
$ grep -i huge /proc/meminfo AnonHugePages: 0 kB ShmemHugePages: 0 kB
FileHugePages: 0 kB HugePages_Total: 16 HugePages_Free: 16 HugePages_Rsvd: 0
HugePages_Surp: 0 Hugepagesize: 1048576 kB Hugetlb: 16777216 kB
```

## Disabling Nouveau

Enter this command to disable Nouveau:

```
$ cat <<EOF | sudo tee /etc/modprobe.d/blacklist-nouveau.conf blacklist nouveau
options nouveau modeset=0 EOF
```

Regenerate the kernel initramfs and reboot the system:

```
$ sudo update-initramfs -u $ sudo reboot
```

## Install Dependency Packages

Enter these commands to install prerequisite packages:

```
$ sudo apt-get update $ sudo apt-get install -y build-essential linux-
headers-$(uname -r) dkms unzip linuxptp pv
```

## Install RSHIM and Mellanox Firmware Tools on the Host

### Note

1. Aerial has been using the Mellanox inbox driver instead of MOFED since the 23-4 release. MOFED must be removed if it is installed on the system.
2. RSHIM package is shared via a PID account after the official product release.



Determine if there is an existing MOFED installed on the host system.

```
$ ofed_info -s MLNX_OFED_LINUX-23.07-0.5.0.0:
```

Uninstall MOFED if it is present.

```
$ sudo /usr/sbin/ofed_uninstall.sh
```

Download the rshim package [here](#) and copy it to the local file system on the server.

Enter the following commands to install the rshim driver.

```
# Install rshim $ sudo apt-get install libfuse2 $ sudo dpkg -i  
rshim_2.0.17.g0caa378_amd64.deb
```

Enter the following commands to install Mellanox firmware tools.

```
# Install Mellanox Firmware Tools $ export MFT_VERSION=4.26.1-3 $ wget  
https://www.mellanox.com/downloads/MFT/mft-$MFT_VERSION-x86_64-deb.tgz $  
tar xvf mft-$MFT_VERSION-x86_64-deb.tgz $ sudo mft-$MFT_VERSION-x86_64-  
deb/install.sh # Verify the install Mellanox firmware tool version $ sudo mst version  
mst, mft 4.26.1-3, built on Nov 27 2023, 15:24:39. Git SHA Hash: N/A $ sudo mst  
start # check NIC PCIe bus addresses and network interface names $ sudo mst status -v  
MST modules: ----- MST PCI module is not loaded MST PCI configuration  
module loaded PCI devices: ----- DEVICE_TYPE MST PCI RDMA NET NUMA  
ConnectX6DX(rev:0) /dev/mst/mt4125_pciconf0.1 b5:00.1 mlx5_1 net-aerial00  
ConnectX6DX(rev:0) /dev/mst/mt4125_pciconf0 b5:00.0 mlx5_0 net-aerial01 0
```

Enter these commands to check the link status of port 0:

```
# Here is an example if port 0 is connected to another server via a # 100GbE cable. $  
sudo mlxlink -d b5:00.0 Operational Info ----- State : Active Physical state :  
LinkUp Speed : 100G Width : 4x FEC : Standard RS-FEC - RS(528,514) Loopback Mode  
: No Loopback Auto Negotiation : ON Supported Info ----- Enabled Link Speed
```

(Ext.) : 0x000007f2 (100G\_2X,100G\_4X,50G\_1X,50G\_2X,40G,25G,10G,1G) Supported  
Cable Speed (Ext.) : 0x000002f2 (100G\_4X,50G\_2X,40G,25G,10G,1G) Troubleshooting  
Info ----- Status Opcode : 0 Group Opcode : N/A Recommendation : No  
issue was observed Tool Information ----- Firmware Version : 22.35.1012  
amBER Version : 2.22 MFT Version : mft 4.26.1-3

## Install CUDA Driver

### Note

Aerial has been using the open-source GPU kernel driver (OpenRM) since the 23-4 release.

If the system has an older driver installed, unload the current driver modules and uninstall the old driver first.

```
# Unload the current driver modules $ for m in $(lsmod | awk "/^[^[:space:]]*"
(nvidia|nv_|gdrdrv)/ {print \$1}"); do echo Unload $m...; sudo rmmod $m; done #
Remove the driver if it was installed by runfile installer before. $ sudo /usr/bin/nvidia-
uninstall
```

Run the following commands to install the **NVIDIA open-source GPU kernel driver** (OpenRM).

```
# Install CUDA driver $ wget https://download.nvidia.com/XFree86/Linux-
x86_64/535.54.03/NVIDIA-Linux-x86_64-535.54.03.run $ sudo sh NVIDIA-Linux-
x86_64-535.54.03.run --silent -m kernel-open # Verify if the driver is loaded
successfully $ nvidia-smi +-----+
---+ | NVIDIA-SMI 535.54.03 Driver Version: 535.54.03 CUDA Version: 12.2 | |-----+
-----+-----+-----+-----+ | GPU Name Persistence-M
| Bus-Id Disp.A | Volatile Uncorr. ECC | | Fan Temp Perf Pwr:Usage/Cap | Memory-
Usage | GPU-Util Compute M. | | | | MIG M. |
```

```

=====+=====+=====
| 0 NVIDIA A100-PCIE-40GB Off | 00000000:B6:00.0 Off | 0 | | N/A 26C P0 38W /
250W | 4MiB / 40960MiB | 44% Default | | | Disabled | +-----
-----+-----+-----+-----+-----
-----+ | Processes: | | GPU GI CI PID Type Process name GPU Memory |
| ID ID Usage |
|=====
| No running processes found | +-----
-----+

```

## Install GDRCopy Driver

Run the following commands to install the GDRCopy driver. If the system has older version installed, remove the old driver first.

### Warning

GDRCopy driver must be installed after CUDA.

```

# Check the installed GDRCopy driver version $ apt list --installed | grep gdrdrv-dkms
# Remove the driver if you have the older version installed. $ sudo apt purge gdrdrv-
dkms $ sudo apt autoremove # Install GDRCopy driver $ wget
https://developer.download.nvidia.com/compute/redist/gdrcopy/CUDA%2012.2/ubun
dkms_2.4-1_amd64.Ubuntu22_04.deb $ sudo dpkg -i gdrdrv-dkms_2.4-
1_amd64.Ubuntu22_04.deb

```

## Install Docker CE

The full official instructions for installing Docker CE can be found on the Docker website: <https://docs.docker.com/engine/install/ubuntu/#install-docker-engine>. The following instructions are one supported way of installing Docker CE:

## Warning

To work correctly, the CUDA driver must be installed before Docker CE or nvidia-container-toolkit installation. It is recommended that you install the CUDA driver before installing Docker CE or the nvidia-container-toolkit.

```
$ sudo apt-get update $ sudo apt-get install -y ca-certificates curl gnupg $ sudo
install -m 0755 -d /etc/apt/keyrings $ curl -fsSL
https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg $ sudo chmod a+r /etc/apt/keyrings/docker.gpg $ echo
\ "deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \ "$( /etc/os-release && echo
"$VERSION_CODENAME)" stable" | \ sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null $ sudo apt-get update $ sudo apt-get install -y docker-ce docker-ce-cli
containerd.io docker-buildx-plugin docker-compose-plugin $ sudo docker run hello-
world
```

## Install the Nvidia Container Toolkit

Locate and follow the nvidia-container-toolkit [install instructions](#).

Or use the following instructions as an alternate way to install the nvidia-container-toolkit. Version **1.14.1-1** is supported.

## Warning

To work correctly, the CUDA driver must be installed before Docker CE or nvidia-container-toolkit installation. It is recommended that you install the CUDA driver before installing Docker CE or the nvidia-container-toolkit.

```
$ curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor
-o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg \ \&& curl -s -L
https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-container-toolkit.list |
\ sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-
keyring.gpg] https://#g' | \ sudo tee /etc/apt/sources.list.d/nvidia-container-
toolkit.list \ \&& \ sudo apt-get update $ sudo apt-get install -y nvidia-container-
toolkit $ sudo nvidia-ctk runtime configure --runtime=docker $ sudo systemctl
restart docker $ sudo docker run --rm --runtime=nvidia --gpus all ubuntu nvidia-smi
```

## Install ptp4l and phc2sys

Enter these commands to configure PTP4L assuming the `aerial00` NIC interface and CPU core **21** are used for PTP:

```
$ cat <<EOF | sudo tee /etc/ptp.conf [global] dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128 maxStepsRemoved 255 logAnnounceInterval -3
logSyncInterval -4 logMinDelayReqInterval -4 G.8275.portDS.localPriority 128
network_transport L2 domainNumber 24 tx_timestamp_timeout 30 slaveOnly 1
clock_servo pi step_threshold 1.0 egressLatency 28 pi_proportional_const 4.65
pi_integral_const 0.1 [aerial00] announceReceiptTimeout 3 delay_mechanism E2E
network_transport L2 EOF $ cat <<EOF | sudo tee /lib/systemd/system/ptp4l.service
[Unit] Description=Precision Time Protocol (PTP) service Documentation=man:ptp4l
After=network.target [Service] Restart=always RestartSec=5s Type=simple
ExecStartPre=ifconfig aerial00 up ExecStartPre=ethtool --set-priv-flags aerial00
tx_port_ts on ExecStartPre=ethtool -A aerial00 rx off tx off ExecStartPre=ifconfig
aerial01 up ExecStartPre=ethtool --set-priv-flags aerial01 tx_port_ts on
ExecStartPre=ethtool -A aerial01 rx off tx off ExecStart=taskset -c 21 /usr/sbin/ptp4l
-f /etc/ptp.conf [Install] WantedBy=multi-user.target EOF $ sudo systemctl daemon-
reload $ sudo systemctl restart ptp4l.service $ sudo systemctl enable ptp4l.service
```

Include 'slaveOnly 1' in the beginning of the ptp.conf file on the server that runs cuphycontroller, as shown below:

```
[global] slaveOnly 1
```

The server without the 'slaveOnly 1' configuration becomes the master clock, as shown below:

```
$ sudo systemctl status ptp4l.service • ptp4l.service - Precision Time Protocol (PTP)
service Loaded: loaded (/lib/systemd/system/ptp4l.service; enabled; vendor preset:
enabled) Active: active (running) since Thu 2022-02-03 22:41:35 UTC; 4min 47s ago
Docs: man:ptp4l Main PID: 1112 (ptp4l) Tasks: 1 (limit: 94582) Memory: 904.0K
CGroup: /system.slice/ptp4l.service    1112 /usr/sbin/ptp4l -f /etc/ptp.conf Feb 03
22:41:44 devkit-01 taskset[1112]: ptp4l[135.371]: selected local clock
b8cef6.ffff.33fdee as best master Feb 03 22:41:44 devkit-01 taskset[1112]:
ptp4l[135.371]: assuming the grand master role Feb 03 22:41:44 devkit-01
taskset[1112]: ptp4l[135.745]: selected local clock b8cef6.ffff.33fdee as best master
Feb 03 22:41:44 devkit-01 taskset[1112]: ptp4l[135.745]: assuming the grand master
role Feb 03 22:41:44 devkit-01 taskset[1112]: ptp4l[135.780]: port 1: link up Feb 03
22:41:44 devkit-01 taskset[1112]: ptp4l[135.804]: port 1: FAULTY to LISTENING on
INIT_COMPLETE Feb 03 22:41:44 devkit-01 taskset[1112]: ptp4l[135.855]: port 1: new
foreign master b8cef6.ffff.33fe16-1 Feb 03 22:41:44 devkit-01 taskset[1112]:
ptp4l[136.105]: selected best master clock b8cef6.ffff.33fe16 Feb 03 22:41:44
devkit-01 taskset[1112]: ptp4l[136.105]: assuming the grand master role Feb 03
22:41:44 devkit-01 taskset[1112]: ptp4l[136.105]: port 1: LISTENING to
GRAND_MASTER on RS_GRAND_MASTER
```

The other server running cuphycontroller becomes the secondary, follower clock, as shown below:

```
$ sudo systemctl status ptp4l.service • ptp4l.service - Precision Time Protocol (PTP)
service Loaded: loaded (/lib/systemd/system/ptp4l.service; enabled; vendor preset:
enabled) Active: active (running) since Fri 2023-02-17 17:01:46 UTC; 37s ago Docs:
man:ptp4l Main PID: 2225310 (ptp4l) Tasks: 1 (limit: 598864) Memory: 624.0K
CGroup: /system.slice/ptp4l.service    2225310 /usr/sbin/ptp4l -f /etc/ptp.conf Feb
17 17:02:14 devkit-02 taskset[2225310]: ptp4l[1992342.927]: rms 6 max 9 freq -8277
+/- 5 delay 220 +/- 0 Feb 17 17:02:15 devkit-02 taskset[2225310]:
ptp4l[1992343.927]: rms 4 max 8 freq -8265 +/- 5 delay 219 +/- 1 Feb 17 17:02:16
devkit-02 taskset[2225310]: ptp4l[1992344.927]: rms 4 max 7 freq -8260 +/- 4 delay
219 +/- 1 Feb 17 17:02:17 devkit-02 taskset[2225310]: ptp4l[1992345.927]: rms 4
```

```
max 7 freq -8268 +/- 5 delay 219 +/- 1 Feb 17 17:02:18 devkit-02 taskset[2225310]:
ptp4l[1992346.927]: rms 4 max 11 freq -8268 +/- 6 delay 221 +/- 1 Feb 17 17:02:19
devkit-02 taskset[2225310]: ptp4l[1992347.927]: rms 4 max 9 freq -8259 +/- 5 delay
219 +/- 1 Feb 17 17:02:20 devkit-02 taskset[2225310]: ptp4l[1992348.927]: rms 9
max 17 freq -8280 +/- 10 delay 220 +/- 1 Feb 17 17:02:21 devkit-02 taskset[2225310]:
ptp4l[1992349.927]: rms 5 max 13 freq -8282 +/- 6 delay 219 +/- 1 Feb 17 17:02:22
devkit-02 taskset[2225310]: ptp4l[1992350.927]: rms 6 max 9 freq -8270 +/- 8 delay
218 +/- 0 Feb 17 17:02:23 devkit-02 taskset[2225310]: ptp4l[1992351.927]: rms 4
max 9 freq -8269 +/- 6 delay 217 +/- 1
```

Enter the commands to turn off NTP:

```
$ sudo timedatectl set-ntp false $ timedatectl Local time: Thu 2022-02-03 22:30:58
UTC Universal time: Thu 2022-02-03 22:30:58 UTC RTC time: Thu 2022-02-03
22:30:58 Time zone: Etc/UTC (UTC, +0000) System clock synchronized: no NTP
service: inactive RTC in local TZ: no
```

Run PHC2SYS as service:

PHC2SYS is used to synchronize the system clock to the PTP hardware clock (PHC) on the NIC.

Specify the network interface used for PTP and system clock as the slave clock.

```
# If more than one instance is already running, kill the existing # PHC2SYS sessions. #
Command used can be found in /lib/systemd/system/phc2sys.service # Update the
ExecStart line to the following $ cat <<EOF | sudo tee
/lib/systemd/system/phc2sys.service [Unit] Description=Synchronize system clock
or PTP hardware clock (PHC) Documentation=man:phc2sys Requires=ptp4l.service
After=ptp4l.service [Service] Restart=always RestartSec=5s Type=simple # Gives
ptp4l a chance to stabilize ExecStartPre=sleep 2 ExecStart=/bin/sh -c "taskset -c 21
/usr/sbin/phc2sys -s /dev/ptp$(ethtool -T aerial00 | grep PTP | awk '{print $4}') -c
CLOCK_REALTIME -n 24 -O 0 -R 256 -u 256" [Install] WantedBy=multi-user.target EOF
```

After the PHC2SYS config file is changed, run the following:

```
$ sudo systemctl daemon-reload $ sudo systemctl restart phc2sys.service # Set to
start automatically on reboot $ sudo systemctl enable phc2sys.service # check that
the service is active and has converged to a low rms value (<30) and that the correct NIC
has been selected (aerial00): $ sudo systemctl status phc2sys.service
phc2sys.service - Synchronize system clock or PTP hardware clock (PHC) Loaded:
loaded (/lib/systemd/system/phc2sys.service; enabled; vendor preset: enabled)
Active: active (running) since Fri 2023-02-17 17:02:35 UTC; 7s ago Docs:
man:phc2sys Main PID: 2225556 (phc2sys) Tasks: 1 (limit: 598864) Memory: 372.0K
CGroup: /system.slice/phc2sys.service 2225556 /usr/sbin/phc2sys -a -r -n 24 -R
256 -u 256 Feb 17 17:02:35 devkit-02 phc2sys[2225556]: [1992363.445]
reconfiguring after port state change Feb 17 17:02:35 devkit-02 phc2sys[2225556]:
[1992363.445] selecting CLOCK_REALTIME for synchronization Feb 17 17:02:35
devkit-02 phc2sys[2225556]: [1992363.445] selecting aerial00 as the master clock
Feb 17 17:02:36 devkit-02 phc2sys[2225556]: [1992364.457] CLOCK_REALTIME rms
15 max 37 freq -19885 +/- 116 delay 1944 +/- 6 Feb 17 17:02:37 devkit-02
phc2sys[2225556]: [1992365.473] CLOCK_REALTIME rms 16 max 42 freq -19951 +/-
103 delay 1944 +/- 7 Feb 17 17:02:38 devkit-02 phc2sys[2225556]: [1992366.490]
CLOCK_REALTIME rms 13 max 31 freq -19909 +/- 81 delay 1944 +/- 6 Feb 17 17:02:39
devkit-02 phc2sys[2225556]: [1992367.506] CLOCK_REALTIME rms 9 max 27 freq
-19918 +/- 40 delay 1945 +/- 6 Feb 17 17:02:40 devkit-02 phc2sys[2225556]:
[1992368.522] CLOCK_REALTIME rms 8 max 24 freq -19925 +/- 11 delay 1945 +/- 9
Feb 17 17:02:41 devkit-02 phc2sys[2225556]: [1992369.538] CLOCK_REALTIME rms 9
max 23 freq -19915 +/- 36 delay 1943 +/- 8
```

Verify that the system clock is synchronized:

```
$ timedatectl Local time: Thu 2022-02-03 22:30:58 UTC Universal time: Thu 2022-02-
03 22:30:58 UTC RTC time: Thu 2022-02-03 22:30:58 Time zone: Etc/UTC (UTC,
+0000) System clock synchronized: yes NTP service: inactive RTC in local TZ: no
```

## Setup the Boot configuration service

Create the directory `/usr/local/bin` and create the `/usr/local/bin/nvidia.sh` file to run the commands with every reboot. The command for “`nvidia-smi lgc`” expects just one GPU device (`-i 0`). This needs to be modified if the system uses more than one GPU.



```
$ cat <<"EOF" | sudo tee /usr/local/bin/nvidia.sh #!/bin/bash mst start nvidia-smi -i 0
-lgc $(nvidia-smi -i 0 --query-supported-clocks=graphics --
format=csv,noheader,nounits | sort -h | tail -n 1) nvidia-smi -mig 0 echo -1 >
/proc/sys/kernel/sched_rt_runtime_us EOF
```

Create a system service file to be loaded after network interfaces are up.

```
$ cat <<EOF | sudo tee /lib/systemd/system/nvidia.service [Unit]
After=network.target [Service] ExecStart=/usr/local/bin/nvidia.sh [Install]
WantedBy=default.target EOF
```

Create a system service file for nvidia-persistenced to be run at startup.

### **Note**

This file was created following the sample from  
[/usr/share/doc/NVIDIA\\_GLX-1.0/samples/nvidia-persistenced-  
init.tar.bz2](#)

```
cat <<EOF | sudo tee /lib/systemd/system/nvidia-persistenced.service [Unit]
Description=NVIDIA Persistence Daemon Wants=syslog.target [Service]
Type=forking ExecStart=/usr/bin/nvidia-persistenced ExecStopPost=/bin/rm -rf
/var/run/nvidia-persistenced [Install] WantedBy=multi-user.target EOF
```

Then set the file permissions, reload the systemd daemon, enable the service, restart the service when installing the first time, and check status


```
sudo chmod 744 /usr/local/bin/nvidia.sh sudo chmod 664
/lib/systemd/system/nvidia.service sudo chmod 664 /lib/systemd/system/nvidia-
persistenced.service sudo systemctl daemon-reload sudo systemctl enable nvidia-
persistenced.service sudo systemctl enable nvidia.service sudo systemctl restart
```

```
nvidia.service sudo systemctl restart nvidia-persistenced.service sudo systemctl
status nvidia.service sudo systemctl status nvidia-persistenced.service
```

The output of the last command should look like this:

```
aerial@server:~$ sudo systemctl status nvidia.service  nvidia.service Loaded:
loaded (/lib/systemd/system/nvidia.service; enabled; vendor preset: enabled) Active:
inactive (dead) since Tue 2024-01-23 19:54:49 UTC; 5min ago Process: 103785
ExecStart=/usr/local/bin/nvidia.sh (code=exited, status=0/SUCCESS) Main PID:
103785 (code=exited, status=0/SUCCESS) CPU: 849ms Jan 23 19:54:48 server
nvidia.sh[103787]: Starting MST (Mellanox Software Tools) driver set Jan 23 19:54:48
server nvidia.sh[103787]: Loading MST PCI module - Success Jan 23 19:54:48 server
nvidia.sh[103787]: [warn] mst_pciconf is already loaded, skipping Jan 23 19:54:48
server nvidia.sh[103787]: Create devices Jan 23 19:54:49 server nvidia.sh[103787]:
Unloading MST PCI module (unused) - Success Jan 23 19:54:49 server
nvidia.sh[104663]: GPU clocks set to "(gpuClkMin 1410, gpuClkMax 1410)" for GPU
00000000:B6:00.0 Jan 23 19:54:49 server nvidia.sh[104663]: All done. Jan 23
19:54:49 server nvidia.sh[104664]: Disabled MIG Mode for GPU 00000000:B6:00.0
Jan 23 19:54:49 server nvidia.sh[104664]: All done. Jan 23 19:54:49 server
systemd[1]: nvidia.service: Deactivated successfully. aerial@server:~$ sudo
systemctl status nvidia-persistenced.service  nvidia-persistenced.service - NVIDIA
Persistence Daemon Loaded: loaded (/lib/systemd/system/nvidia-
persistenced.service; enabled; vendor preset: enabled) Active: active (running) since
Tue 2024-01-23 19:54:48 UTC; 6min ago Process: 103781 ExecStart=/usr/bin/nvidia-
persistenced (code=exited, status=0/SUCCESS) Main PID: 103782 (nvidia-persiste)
Tasks: 1 (limit: 598790) Memory: 324.0K CPU: 7ms CGroup: /system.slice/nvidia-
persistenced.service  103782 /usr/bin/nvidia-persistenced Jan 23 19:54:48 server
systemd[1]: Starting NVIDIA Persistence Daemon... Jan 23 19:54:48 server nvidia-
persistenced[103782]: Started (103782) Jan 23 19:54:48 server systemd[1]: Started
NVIDIA Persistence Daemon.
```

## Install Matlab (Optional)

 **Note**

This step is optional. Matlab is not required to generate TV files if using Aerial Python mcore module. See Generating TV and Launch Pattern Files section in *cuBB Quick Start Guide*.

Refer to <https://www.mathworks.com/downloads/> for downloading and installing Matlab. Follow the established IT process for the license and installation at your site.

Matlab is used to run the test-vector generator script. This can be run on any machine that has a graphical display and a graphical UI capable operating system that Matlab supports. The generated test-vector files can then be copied to the cuBB server.

If you would like to run the test-vector generator Matlab script on the same Ubuntu server machine that runs cuBB, then Matlab should be run in console mode

```
matlab -nosplash -nodesktop .
```

The following Matlab components are required:

- Matlab 2020b or later
- Matlab licenses:
  - MATLAB
  - Communications Toolbox
  - DSP System Toolbox
  - Signal Processing Toolbox
  - Fixed-Point Designer (optional)
    - Call half function to accelerate testing/simulation
    - Can be disabled by setting SimCtrl.fp16AlgoSel = 1
  - Parallel Computing Toolbox (optional)

- Accelerate testing/simulation automatically
  
- 5G Toolbox
  - Not required for TV generation (can be disabled by setting Chan.use5Gtoolbox = 0 in /nr\_matlab/config/cfgChan.m)
  
  - Required for waveform compliance test and performance simulation

© Copyright 2024, NVIDIA.. PDF Generated on 06/06/2024