VIDEO FOR LINUX USER GUIDE

DA_07601-001 | July 10, 2015
Advance Information | Subject to Change

Release R21.4

DOCUMENT CHANGE HISTORY

DA_07303-001_01

Version	Date	Authors	Description of Change
v1.0	13 March 2015	alevinson	Initial release.
v1.1	10 July 2015	mzensius/pengw	Added procedures for changing TPG resolution and calibrating MIPI D-PHY for CSI

TABLE OF CONTENTS

VIDEO FOR LINUX USER GUIDE

This document provides information on use of the MIPI Camera Serial Interface (CSI) on Tegra® K1, using software from the NVIDIA® Tegra® Linux Driver Package (also referred to as L4T). The MIPI CSI protocol, V4L2 API, Tegra K1 system architecture and method of attaching a CSI camera to Jetson TK1 are outside the scope of this document.

The V4L2 software implementation bypasses the Tegra ISP, and is suitable for use when Tegra ISP support is not required, such as with sensors or input devices that provide data in YUV format.

References to additional resources are provided, but the reader should already be familiar with Tegra K1, and have access to the *Tegra Technical Reference Manual* (TRM) and other documentation available at the Jetson Embedded Platform portal:

```
http://developer.nvidia.com/embedded-computing
```

V4L2/SOC_CAMERA OVERVIEW

V4L2 is the second version of Video4Linux or V4L, a video capture and output device API and driver framework in the Linux kernel. It supports many USB webcams, TV tuners, and other devices and is closely integrated with the Linux kernel. For a description of the APIs, see Linux Media Infrastructure APIs.

`soc-camera` is a set of drivers and a core module, that implement V4L2 functionality on embedded devices; typically a video-enabled embedded device: SoC with a capture interface and video data sources. The `soc-camera` includes host driver such as the Tegra V4L2 camera driver and client drivers (sensor drivers).

V4L2 ON JETSON TK1

Jetson TK1 is a powerful embedded development board including the NVIDIA Tegra K1 processor. The Tegra K1 processor has a video input interface (VI) and camera serial interface (CSI), so it can communicate with the external video input sources such as camera sensor module or other MIPI CSI compatible devices. VI/CSI of Tegra K1 also has 2 test pattern generators which can generate some data patterns like color bricks for testing purpose.

On the software side, Linux for Tegra (L4T) latest release R21.3 provides a Tegra V4L2 camera driver and sample drivers for a camera sensor and a built-in test pattern generator (TPG). With an open source V4L2 user space tool like Yavta, users can capture data from the TPG and sensors.

Test Pattern Generator

The test pattern generator is a configurable resource introduced to improve hardware verification capability for the Tegra CSI. There are two separate test pattern generators that can be configured to provide for the generation of synthetic image data, which is delivered to the PPA and PPB input FIFOs. The image data is multiplexed into the CSI data patch between lane-merging logic and the data FIFOs.

L4T provides a virtual V4L2 `soc_camera` sensor driver for exposing TPG functionality (`soc_camera_platform driver`). It can generate 1280x720 resolution RGBA32 color bricks data. There is no need to rebuild the kernel and the `soc_camera_platform` driver is provided as a loadable module.

To change TPG resolution

- TPG resolution is specified in the board file and programmed within the `tegra_v4l2_camera` driver. Modify the board file variables referenced in the following driver source code example:

```
TC_VI_REG_WT(cam, TEGRA_VI_CSI_0_CSI_IMAGE_SIZE_WC, image_size);
TC_VI_REG_WT(cam, TEGRA_VI_CSI_0_CSI_IMAGE_SIZE,
             (icd->user_height << 16) | icd->user_width);
```

  These registers are documented in the TRM. Refer to the Board File and Device Tree File Updates section of this document for details on modification of the board file.

To verify the TPG

- Remove the nvhost_vi module, an incompatible non-V4L2 VI driver used for other purposes and outside the scope of this document:

```
$ sudo rmmod nvhost_vi
```

- Install V4L2 driver modules:

```
$ sudo modprobe soc_camera_platform
$ sudo modprobe tegra_camera tpg_mode=2
```

- Use the yavta application to capture data (other V4L2 applications can be used, if preferred)

```
$ ./yavta /dev/video0 -c1 -n1 -s1280x720 -fRGB32 -Ftpg.rgba
```

- Copy `over tpg.rgba` to host and use ImageMagick to show the picture:

```
$ display -size 1280x720 -depth 8 tpg.rgba
```

MIPI D-PHY calibration for CSI

Tegra K1 has a separate block for MIPI D-PHY calibration which supports both CSI and DSI. In the R21.4 release of L4T, CSI MIPI calibration is performed by the `vi2_mipi_calibration()` function in the `tegra_v4l2_camera` driver.See that function for details.

To calibrate MIPI D-PHY for CSI

1. Setup MIPI calibration registers for CSI depending upon the CSI ports configured.
2. Start the automatic calibration.
3. Poll the status register bit.

   If status shows calibration is done (`CAL_DONE`), MIPI calibration is successful. If polling timeout occurs without `CAL_DONE` status reported, MIPI calibration is unsuccessful.

Example Sensor: IMX135

L4T provides a sample V4L2 sensor driver for the Sony IMX135 Bayer sensor. This driver can be used as a reference in creating a custom V4L2 sensor driver. NVIDIA does not provide a reference camera module at this time, so the following information is provided for example purposes, assuming you have an IMX135 sensor module connected to Jetson TK1.

The driver for IMX135 is neither built into kernel nor built as module. Please try following steps to test IMX135 in L4T on Jetson TK1.

- Hardware setup
  - Jetson TK1

- Jetson TK1 adapter board capable of connecting to an IMX135 camera module

- Enable IMX135 kernel driver and disable soc_camera_platform
  - CONFIG_SOC_CAMERA_IMX135=m
  - Disable CONFIG_SOC_CAMERA_PLATFORM
- Build kernel, flash Jetson TK1 and boot the Linux OS
- Use Yavta to capture a frame

```
$ sudo modprobe imx135_v4l2
$ sudo modprobe tegra_camera
$ ./yavta /dev/video0 -c1 -n1 -s1920x1080 -fSRGGB10 -Fimx135.raw
```

- Use raw2bmp to convert raw data to BMP file

```
https://gitorious.org/omap4-v4l2-
camera/yavta/source/5417d27b99b2a147e3a062a24f36fd7a71f49b40:raw2bmp
.c
```

- Capture color bar test patterns from IMX135

```
$ sudo modprobe imx135_v4l2 test_mode=2
$ sudo modprobe tegra_camera
$ ./yavta /dev/video0 -c1 -n1 -s1920x1080 -fSRGGB10 -Fimx135.raw
$ ./raw2bmp imx135.raw imx135.bmp 1920 1080 16 3
```

IMX135 and AR0261 Dual Capture Demo

Tegra K1 processor has 2 CSI ports: CSI_A and CSI_B and supports capture from these 2 ports simultaneously. On Jetson TK1, IMX135 connects to CSI_A via 4 data lanes (CIL_A and CIL_B) and AR0261 connects to CSI_B via 1 data lane (CIL_E).

L4T kernel source contains drivers for both of these 2 sensors. Try the following steps in L4T on Jetson TK1.

- Hardware setup
  - Jetson TK1
  - Jetson TK1 adapter board capable of connecting to an IMX135 camera module
- Enable IMX135 and AR0261 kernel driver and disable `soc_camera_platform`.
  - CONFIG_SOC_CAMERA_IMX135=m
  - CONFIG_SOC_CAMERA_AR0261=m
  - Disable CONFIG_SOC_CAMERA_PLATFORM
- Build the kernel, flash Jetson TK1, and boot into Ubuntu.

- Install the camera modules. Then /dev/video0 and /dev/video1 will appear:

```
$ sudo modprobe tegra_camera
```

- Use Yavta to capture from /dev/video0 and /dev/video1 at the same time

```
$ ./yavta /dev/video0 -c1000 -n4 -s1920x1080 -fSRGGB10 -F/dev/null &
$ ./yavta /dev/video1 -c1000 -n4 -s1920x1080 -fSRGGB10 -F/dev/null
```

V4L2 TEGRA DRIVER OVERVIEW

As V4L2 is a kernel video input framework, Tegra V4L2 stack contains several components. It controls hardware such as the Tegra VI/CSI hardware controller and external sensors. Additionally, it exports a generic device node named /dev/video<N> to user space, where <N> is a numeric value. User space application can use V4L2 standard API to control real hardware via /dev/video<N>.

This section will focus on Tegra K1-related drivers and code in L4T kernel source

Tegra V4L2 Camera Driver

Tegra V4L2 camera driver is a part of `soc_camera` and acts as a host driver. It directly controls Tegra K1 VI/CSI hardware. Normally users don't need to modify this driver, but developers should become familiar with it; it may require customization for some use cases.

- Source code

```
drivers/media/platform/soc_camera/Kconfig
drivers/media/platform/soc_camera/Makefile
drivers/media/platform/soc_camera/tegra_camera/*
include/media/tegra_v4l2_camera.h
```

- Kernel config

```
CONFIG_VIDEO_TEGRA=m
```

The module name is `tegra_camera.ko` and it won't be loaded by default after booting into L4T. There is another driver named nvhost_vi.ko installed by default and mutually-exclusive with `tegra_camera.ko`, so users must remove the nvhost_vi.ko before loading `tegra_camera.ko`.

- Input data format

Tegra K1 VI/CSI hardware supports 3 major input data format: YUV, RGB and Bayer RAW. However in this driver only the following have been implemented at the time this document was written (please review the driver for current details):

- RGB888
- RAW8
- RAW10

Note: YUV formats are also supported by hardware but software support is not present in the driver. Please refer to the Tegra TRM for details on supported input formats.

Study the source code then add new input data formats not listed here.

- All the formats are listed in `structs tegra_camera_yuv_formats`, `tegra_camera_rgb_formats` and `tegra_camera_bayer_formats` of `drivers/media/platform/soc_camera/tegra_camera/common.c`
- Add the format into function `tegra_camera_get_formats()` of `drivers/media/platform/soc_camera/tegra_camera/common.c`
- Add the format support into function `vi2_capture_setup_csi_0()` and `vi2_capture_setup_csi_1()` of `drivers/media/platform/soc_camera/tegra_camera/vi2.c`

Tegra V4L2 Sensor Driver

V4L2 sensor driver normally is an I2C device driver and in L4T it is also a V4L2 `soc_camera` client driver. It has several I2C register tables for different resolutions like 1920x1080, 1280x720 etc. When a user space application opens `/dev/video<N>`, the sensor driver will power on the sensor hardware and program it with the register table via I2C.

- Real sensor code

```
drivers/media/i2c/soc_camera/imx135_v4l2.c
include/media/imx135.h
drivers/media/i2c/soc_camera/ar0261_v4l2.c
include/media/ar0261.h
drivers/media/i2c/soc_camera/Kconfig
drivers/media/i2c/soc_camera/Makefile
```

- Test Pattern Generator virtual sensor driver source code

```
drivers/media/platform/soc_camera/soc_camera_platform.c
```

The `soc_camera_platform` driver does perform any real hardware operations like power control and I2C transactions. It is just a virtual driver to enable use of the TPG for testing.

- Kernel configs

```
CONFIG_SOC_CAMERA_AR0261
CONFIG_SOC_CAMERA_IMX135
CONFIG_SOC_CAMERA_PLATFORM
```

- Power controls

  Each sensor has its own power on/off sequence, clock settings and other hardware specific operations. L4T sensor driver put these power controls in the sensor driver itself. For more flexible driver design, these power controls need go to board files since each hardware board might have different power controls. Then sensor driver itself can be more generic. Normally power controls include:

  - GPIO for sensor reset, power on or power down
  - Regulators for sensor power supply
  - Clocks for sensor running like mclk or sensor local clock.

Board File

Before fully moving to device tree binding, a board file is the only way to describe platform-specific configurations within the Linux kernel. In L4T R21.3 release most hardware devices use device tree binding but V4L2 `soc_camera` still uses a board file approach.

- Source code

```
arch/arm/mach-tegra/board-ardbeg-sensors.c
```

- TPG board configs

  `soc_camera_platform_info` defines data format and resolution which should be matched with our TPG hardware.

```
static struct soc_camera_platform_info ardbeg_soc_camera_info = {
        .format_name = "RGB4",
        .format_depth = 32,
        .format = {
                .code = V4L2_MBUS_FMT_RGBA8888_4X8_LE,
                .colorspace = V4L2_COLORSPACE_SRGB,
                .field = V4L2_FIELD_NONE,
                .width = 1280,
                .height = 720,
        },
        .set_capture = ardbeg_soc_camera_set_capture,
};
```

  `tegra_camera_platform_data` is the most important data struct to describe the sensor connection. `.port` indicates which CSI port the sensor connects to:

TEGRA_CAMERA_PORT_CSI_A means the sensor uses CIL_A and CIL_B.

TEGRA_CAMERA_PORT_CSI_B means the sensor uses CIL_C and CIL_D.

TEGRA_CAMERA_PORT_CSI_C means the sensor uses CIL_E.

Tegra K1 internally just has 2 CSI channels (CSI_A and CSI_B). CSI_C is just a software alias to tell the driver that the sensor is using CIL_E.

TEGRA_CAMERA_PORT_CSI_A and TEGRA_CAMERA_PORT_CSI_B can support 1, 2 and 4 data lane sensors. TEGRA_CAMERA_PORT_CSI_C can only support 1 lane sensor.

```
static struct tegra_camera_platform_data ardbeg_camera_platform_data
= {
        .flip_v                 = 0,
        .flip_h                 = 0,
        .port                   = TEGRA_CAMERA_PORT_CSI_A,
        .lanes                  = 4,
        .continuous_clk         = 0,
};
```

- IMX135 board file configs

Real sensors don't require that sensor resolution or data format information be put into the board file like TPG `soc_camera_platform` driver, because that information is in the sensor driver itself.

- IMX135 connects to port CSI_A via 4 data lanes:

```
static struct tegra_camera_platform_data
ardbeg_imx135_camera_platform_data = {
        .flip_v                 = 0,
        .flip_h                 = 0,
        .port                   = TEGRA_CAMERA_PORT_CSI_A,
        .lanes                  = 4,
        .continuous_clk         = 0,
};
```

- IMX135 use I2C 2 bus and it's I2C address is 0x10:

```
static struct i2c_board_info ardbeg_imx135_camera_i2c_device = {
        I2C_BOARD_INFO("imx135_v4l2", 0x10),
        .platform_data = &ardbeg_imx135_data,
};

static struct soc_camera_link imx135_iclink = {
        .bus_id         = 0, /* This must match the .id of
tegra_vi01_device */
        .board_info     = &ardbeg_imx135_camera_i2c_device,
        .module_name    = "imx135_v4l2",
```

```
        .i2c_adapter_id = 2,
        .power          = ardbeg_imx135_power,
        .priv           = &ardbeg_imx135_camera_platform_data,
};
```

- Register IMX135 soc_camera platform device:

```
platform_device_register(&ardbeg_imx135_soc_camera_device);
```

- AR0261 board file configs:
  - AR0261 connects to port CSI_C via 1 data lane.

```
static struct tegra_camera_platform_data
ardbeg_ar0261_camera_platform_data = {
        .flip_v                 = 0,
        .flip_h                 = 0,
        .port                   = TEGRA_CAMERA_PORT_CSI_C,
        .lanes                  = 1,
        .continuous_clk         = 0,
};
```

  - AR0261 use I2C 2 bus and it's I2C address is 0x36:

```
static struct i2c_board_info ardbeg_ar0261_camera_i2c_device = {
        I2C_BOARD_INFO("ar0261_v4l2", 0x36),
        .platform_data = &ardbeg_ar0261_data,
};
static struct soc_camera_link ar0261_iclink = {
        .bus_id         = 1, /* This must match the .id of
tegra_vi01_device */
        .board_info     = &ardbeg_ar0261_camera_i2c_device,
        .module_name    = "ar0261_v4l2",
        .i2c_adapter_id = 2,
        .power          = ardbeg_ar0261_power,
        .priv           = &ardbeg_ar0261_camera_platform_data,
};
```

  - Register AR0261 soc_camera platform device:

```
platform_device_register(&ardbeg_ar0261_soc_camera_device);
```

Device Tree File

Device tree still provides regulator information required by the V4L2 sensor driver. Both the IMX135 and AR0261 sensor drivers use 3 regulators: `vana`, `vdig` and `vif`. They are defined in: `arch/arm/boot/dts/tegra124-platforms/tegra124-jetson_tk1-pmic-pm375-0000-c00-00.dtsi`.

IMX135 needs 2 extra regulators that are also defined in the same file.

HOW TO WRITE AND INTEGRATE A SENSOR DRIVER FOR L4T

Developers can write their own sensor driver for their specific device. Sensor drivers usually have very similar structures but different I2C register tables. Modification of the board file and the device tree file is required for different boards.

Sensor Driver Development

The IMX135 and AR0261 sensor drivers are a good start point for writing a new sensor driver. The following steps are recommended for developing a new driver:

- Import new I2C register tables

  Sensor vendors will provide I2C register settings as tables, which should be added to sensor driver. The following struct is a good example:

  ```
  static struct imx135_reg *mode_table[] = {
          [IMX135_MODE_4208X3120] = mode_4208x3120,
          [IMX135_MODE_1920X1080] = mode_1920x1080,
          [IMX135_MODE_1280X720]  = mode_1280x720,
          [IMX135_MODE_2616X1472]  = mode_2616x1472,
          [IMX135_MODE_3896X2192]  = mode_3896x2192,
          [IMX135_MODE_2104X1560]  = mode_2104x1560,
  };
  ```

- Power controls

  Different boards have different sensor power controls. It is better put those power controls into a board file. But it is simpler to implement them in a sensor driver. Please take a look at `imx135_power_on()` and `imx135_power_off()`.

- soc_camera and I2C interface

  The sensor driver implements `soc_camera ops` functions as well as I2C device probing/removing functions. Normally these are quite similar across different sensor drivers -- just reuse them in your driver and use `imx135_v4l2.c` as an example.

- KConfig and Makefile

  Add a SOC_CAMERA_IMX135 entry into the `Kconfig` and `Makefile` files.

- Kernel module parameters

Building a sensor driver as a module is beneficial for validating different module parameters. In the IMX135 sensor driver, a parameter for `test_mode` is passed when loading the module. Because IMX135 has a color bar test pattern generator inside, using this parameter can ask IMX135 to send out color bar data for testing and bypass those lens or focuser settings.

- Header file `include/media/sensor.h`

  This header contains some information for non-V4L2 NVIDIA camera stacks. The following structs can be reused if necessary:

```
struct imx135_power_rail {
        struct regulator *dvdd;
        struct regulator *avdd;
        struct regulator *iovdd;
        struct regulator *ext_reg1;
        struct regulator *ext_reg2;
};


struct imx135_platform_data {
        struct imx135_flash_control flash_cap;
        const char *mclk_name; /* NULL for default default_mclk */
        unsigned int cam1_gpio;
        unsigned int reset_gpio;
        unsigned int af_gpio;
        bool ext_reg;
        int (*power_on)(struct imx135_power_rail *pw);
        int (*power_off)(struct imx135_power_rail *pw);
};
```

Board File and Device Tree File Updates

A new project or new hardware board might have a new board file such as `board-ardbeg*.c` for Jetson TK1. If so, the new board file should include those settings for sensor drivers. Follow this template in the board file and replace “SENSOR” with your sensor name:

```
#if IS_ENABLED(CONFIG_SOC_CAMERA_SENSOR)
static int ardbeg_sensor_power(struct device *dev, int enable)
{
        return 0;
}
// NOTE: power controls can go here instead of sensor driver itself.

struct sensor_platform_data ardbeg_sensor_data;
```

```
static struct i2c_board_info ardbeg_sensor_camera_i2c_device = {
        I2C_BOARD_INFO("sensor_v4l2_driver_name", sensor_i2c_address),
        // sensor_v4l2_driver_name should match the sensor driver's
module name
        .platform_data = &ardbeg_sensor_data,
};

static struct tegra_camera_platform_data
ardbeg_sensor_camera_platform_data = {
        .flip_v                 = 0,
        .flip_h                 = 0,
        .port                   = TEGRA_CAMERA_PORT_CSI_X_for_sensor,
        .lanes                  = number_of_sensor_data_lanes,
        .continuous_clk         = 0,
};

static struct soc_camera_link sensor_iclink = {
        .bus_id         = 0, /* This must match the .id of
tegra_vi01_device */
        .board_info     = &ardbeg_sensor_camera_i2c_device,
        .module_name    = "sensor_v4l2_driver_name",
        .i2c_adapter_id = sensor_i2c_bus_number,
        .power          = ardbeg_sensor_power,
        .priv           = &ardbeg_sensor_camera_platform_data,
};

static struct platform_device ardbeg_sensor_soc_camera_device = {
        .name   = "soc-camera-pdrv",
        .id     = 0,
        .dev    = {
                .platform_data = &sensor_iclink,
        },
};
#endif
```

- Finally register the platform device in `ardbeg_camera_init()`:

  ```
  #if IS_ENABLED(CONFIG_SOC_CAMERA_SENSOR)
          platform_device_register(&ardbeg_sensor_soc_camera_device);
  #endif
  ```

- Device tree update

  Find the new device tree file for the new board and update regulator information appropriate to the hardware configuration of the new board. A good example to look at is:

  ```
  arch/arm/boot/dts/tegra124-platforms/tegra124-jetson_tk1-pmic-pm375-
  0000-c00-00.dtsi
  ```

Troubleshooting

- I2C transaction timeout error
  - I2C information is wrong

    Check the sensor I2C bus number and the sensor I2C device address in the board file.
  - Sensor power control sequence is wrong

    Check sensor MCLK setting.
    Check regulator operations.
    Check GPIO settings.
- Sync point timeout without error

  This means Tegra VI/CSI doesn't receive any data but no error occurs. Make sure the sensor is powered on and streaming data correctly before debugging the Tegra driver.
  - Change settle time value to see if there if some error shows up. These registers must be configured with the right values to get data from the sensor.

    ```
    TC_VI_REG_WT(cam, TEGRA_CSI_PHY_CILA_CONTROL0, 0x9);
    TC_VI_REG_WT(cam, TEGRA_CSI_PHY_CILB_CONTROL0, 0x9);
    ```

    or

    ```
    TC_VI_REG_WT(cam, TEGRA_CSI_PHY_CILC_CONTROL0, 0x9);
    TC_VI_REG_WT(cam, TEGRA_CSI_PHY_CILD_CONTROL0, 0x9);
    ```

    or

    ```
    TC_VI_REG_WT(cam, TEGRA_CSI_PHY_CILE_CONTROL0, 0x9);
    ```
  - Make sure CILA/B or CILC/D or CILE are not in deep power mode (DPD). DPD mode is normally disabled in sensor power on function. Please use `tegra_io_dpd_disable()` of `imx135_v4l2.c` as an example.
- Sync point timeout with error

  Capture the error message and look it up in Tegra K1 TRM for further debugging.

RESOURCES

Good resources for V4L integration are:

- Kernel documentation located in:

```
Documentation/video4linux/
```

- Linux TV website:

```
http://www.linuxtv.org/
```

- soc-camera slides:

```
http://elinux.org/images/f/f2/Soc-camera.pdf
```

- Yavta user space V4L2 tool

```
http://git.ideasonboard.org/yavta.git
```

- Jetson Embedded Platform page

```
http://developer.nvidia.com/embedded-computing
```

Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OR CONDITION OF TITLE, MERCHANTABILITY, SATISFACTORY QUALITY, FITNESS FOR A PARTICULAR PURPOSE AND ON-INFRINGEMENT, ARE HEREBY EXCLUDED TO THE MAXIMUM EXTENT PERMITTED BY LAW.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. NVIDIA Corporation products are not authorized for use as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright