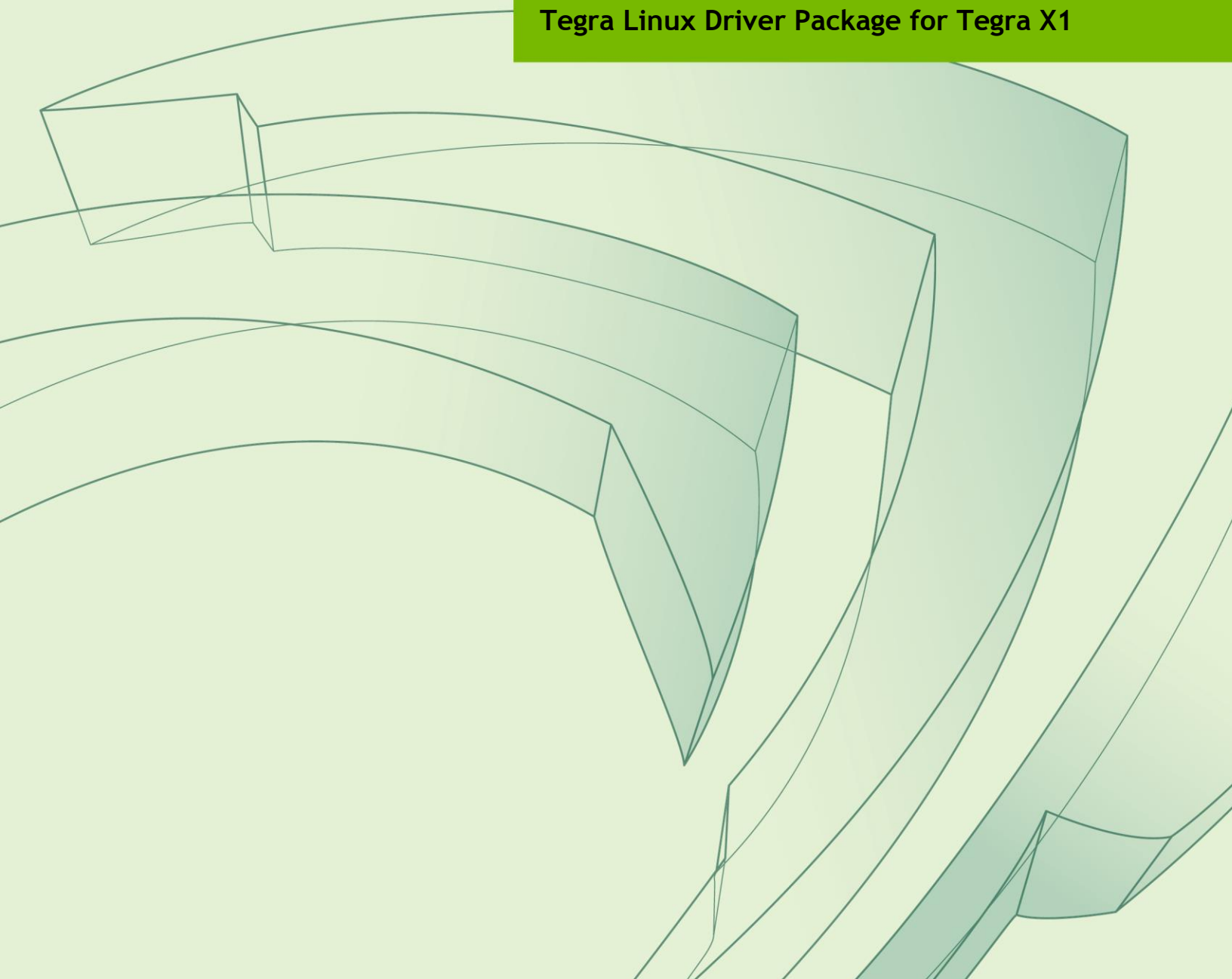
PLATFORM ADAPTATION AND BRING-UP GUIDE

DA_07839-001 | February 28, 2018

Tegra Linux Driver Package for Tegra X1



Document Change History

DA_07839-001

Version	Date	Authors	Description of Change
v1.0	1 Mar 2016	msum, jsachs	Initial release.
v1.1	11 Aug 2016	mzensius	Adds USB Lane Mapping section.
V1.2	3 Mar 2017	mzensius	Adds camera sensor bring-up checklist.
V1.3	30 June 2017	hlang	generate final PDF for 28.1 release.
V1.4	3 Oct 2017	hlang/vsagar/danel	Update porting the linux kernel and USB lane mapping sections.
v1.5	15 Nov 2017	hlang/danel	update the R28 device tree changes and example topics with updates for PCIe.

Table of Contents

Platform Adaptation and Bring-Up

This document is for software developers whose target is the NVIDIA® Jetson™ X1 module. It describes how to port the NVIDIA® Tegra® Linux Driver Package and the U-Boot boot loader from NVIDIA® Jetson™ TX1 Developer Kit to other hardware platforms.

For all of the procedures in this document, the NVIDIA® Tegra® Linux Driver Package release includes code for the Jetson X1 Developer Kit (P2371-2180) that can serve as an example.

Board Naming

To support your board in L4T, you must choose a simple lower-case, alpha-numeric name for your board, possibly including dashes (-) or underscores (_) but containing no spaces, such as the following examples:

```
jetson-tx1
p2371-2180
beaver
```

The name you choose will appear in file names and path names in U-Boot and Linux kernel source code, user-visible device tree file names, and be exposed to the user via the U-Boot command prompt and various Linux kernel `/proc` files.

In this document, `<board>` represents your board name.

You must also choose a similarly-constructed vendor name. The same character set rules apply, such as the following example:

```
nvidia
```

In this document, <vendor> represents your vendor name.

Note: **Do not simply re-use and modify the existing NVIDIA® Jetson™ TX1 Developer Kit code without choosing and using your own board name. If you do not use your own board name it will not be obvious to Jetson TX1 users whether modified source code supports the original Jetson TX1 Developer Kit carrier board or your board.**

Placeholders in the Porting Instructions

The following sections refer to filenames and pathnames that contain these placeholders. Substitute an appropriate value for each placeholder when you enter the commands.

- <function> is a functional module name, which may be power-tree, pinmux, sdmmc-drv, keys, comm (WIFI/BT), camera, etc.
- <board> is a name you have chosen to represent your platform. For example, p2597 is the name of the Jetson TX1 Developer Kit carrier board. Note that NVIDIA <board> names use lower case letters only.
- <som> is a System on a Module (SOM) board name, such as 2180.
- <version> is a board version number, such as a00. Files for NVIDIA reference boards include a version number. Files for customer platforms need not include one.
- <vendor> is your organization's name, or the name of your board's vendor.
- <root> is the device that holds the platform's root file system. At present the only supported value is emmc.

Pinmux Changes

If your board schematic differs from that for Jetson™ TX1 Developer Kit carrier board, you must change the pinmux configuration applied by the software.

To define your board's pinmux configuration, you must obtain Jetson_TX1_customer_pinmux_release.xlsm from NVIDIA and customize it for the configuration of your board using the following procedures.

To customize the pinmux spreadsheet

1. Create a copy of the file with a name based on your board name, e.g. <board>_pinmux.xlsm.
2. Ensure that the new file is writable.
3. On a Windows PC, open the new file in Microsoft Excel.
4. If Microsoft Excel displays any warnings such as "PROTECTED VIEW" or "SECURITY WARNING," click Enable Editing or Enable Content, so that you can save your changes to the new file.
5. Rename the **Jetson TX1 Configuration** tab based on the name of your board:

- Right-click on the **Jetson TX1 Configuration** tab at the bottom of the window.
- Click the **Rename** menu option.
- Type your board name into the tab, then press ENTER.

6. Modify columns AE through AO of the spreadsheet as required by the pinmux configuration for your board, based on the schematic.

Once the spreadsheet reflects the configuration you want, export the configuration data in a format that U-Boot and the Linux kernel can use.

Exporting Pinmux for U-Boot

U-Boot uses a header file to define the pinmux configuration. This header file may be generated using the `tegra-pinmux-scripts` tool.

To customize tegra-pinmux-scripts for your board

1. Obtain `tegra-pinmux-scripts` by running the following commands on your Linux system:

```
$ git clone https://github.com/NVIDIA/tegra-pinmux-scripts.git
$ cd tegra-pinmux-scripts
```

2. In the `tegra-pinmux-scripts` directory, open `csv-to-board.py` in a text editor.
3. Locate the definition of the `supported_boards` data structure, at approximately line 50.
4. Add an entry for your board to the `supported_boards` data structure like the following example:

```
'<board>': {
    # <board>_pinmux.xlsm worksheet <board>
    'filename': 'csv/<board>.csv',
    'rsvd_0based': False,
},
```

5. Save the file and exit the editor.
6. Commit this change to your local Git repository:

```
$ git commit -a -m "Add support for <board>" -s
```

The `tegra-pinmux-scripts` read a CSV (Comma Separated Values) version of the pinmux spreadsheet as input.

To save the spreadsheet in CSV format

1. In Microsoft Excel, click the File tab.
2. On the File tab, click Save As.
3. From Save as type, choose `CSV (MS-DOC) (*.csv)`.
4. Verify that the file name ends in `.csv`, but otherwise matches the file name in your changes to `csv-to-board.py`.
5. Click Save.
6. Copy the CSV file to the `csv/` directory of `tegra-pinmux-scripts` on your Linux system.

To generate the U-Boot pinmux header file

1. Enter the following command in the `tegra-pinnmux-scripts` directory to import the data into the `tegra-pinmux-script` internal format:

   ```
   $ ./csv-to-board.py <board>
   ```

   Optionally, use the `--csv <csv_file_name>` command line option to specify the CSV file to import. This allows you to copy the CSV file to an arbitrary location on your Linux system if you wish.

2. Enter the following command to generate the U-Boot pinmux header file:

   ```
   $ ./board-to-uboot.py <board> > pinmux-config-<board>.h
   ```

Later, you will copy `pinmux-config-<board>.h` into the U-Boot source tree.

Exporting Pinmux for the L4T Linux Kernel

The Linux kernel uses device tree files to define the pinmux configuration, which you can generate directly from the Excel spreadsheet.

To generate device tree files for your pinmux configuration

1. In the spreadsheet, click Generate DT.
2. Answer “yes” to the prompt asking whether you wish to generate the DT files and provide the name of your board when prompted.

The device tree files are saved in the same location as the Excel spreadsheet. After the file is generated, make sure that the file name matches what you use in your kernel code. Correct the file name if there is a mismatch. Later, you will copy the device tree files into the Linux kernel source tree.

Porting U-Boot

Perform the following actions in the U-Boot source code to add support for your board.

1. Copy `include/configs/jetson-p2371-2180.h` to `include/configs/<board>.h`.
2. Edit the set of enabled devices and features in `<board>.h` as appropriate for your board. For example, you must change the following:

```
#define CONFIG_TEGRA_BOARD_STRING       "NVIDIA P2371-2180"
```

3. Copy `arch/arm/dts/tegra210-p2371-2180.dts` to `arch/arm/dts/tegra210-<board>.dts`.
4. Edit the set of enabled devices and their parameters (e.g. GPIO and IRQ IDs) in `tegra210-<board>.dts` as appropriate for your board.

   Nodes and properties might need to be added, removed, or edited.

   Note: **U-Boot and the Linux kernel do not always use the exact same device tree schema (bindings) to represent the same data. Follow examples from U-Boot rather than from the Linux kernel.**

5. Add `tegra210-<board>.dtb` to `arch/arm/dts/Makefile`.
6. Copy `configs/p2371-2180_defconfig` to `configs/<board>_defconfig`.
7. Edit `<board>_defconfig` to refer to your board name.
8. Edit `arch/arm/mach-tegra/tegra210/Kconfig` to add target config and Kconfig.
9. Copy the `board/nvidia/p2371-2180/` directory to `board/<vendor>/<board>/`.
10. Edit all of the files in `board/<vendor>/<board>/` to refer to your board name rather than the P2371-2180. The files in this directory contain many instances of the P2371-2180 board name.
11. Edit `board/<vendor>/<board>/MAINTAINERS` to provide the correct maintainer contact information for your board.
12. Edit `board/<vendor>/<board>/<board>.c` to provide the correct PMIC programming for your board, if required.
13. Copy the pinmux header you generated (`pinmux-config-<board>.h`) to the `board/<vendor>/<board>/` directory.

Porting the Linux Kernel

To port the kernel configuration code (the device tree) to your platform, modify one of the distributed configuration files to describe your platform's design.

The configuration files are in `arch/arm64/boot/dts/`. Their names have the form `tegra210-jetson-cv-base-<board>-<som>.dts`, where `<board>` refers to one of the NVIDIA reference boards, such as `p2597`.

NVIDIA recommends that you use this file and the files included, which describes the device tree of reference board P2597:

```
arch/arm64/boot/dts/tegra210-jetson-cv-base-p2597-2180-a00.dts
```

This device tree file includes many `.dtsi` files for various types of hardware. To configure the kernel to work on your platform, make copies of the `.dts` file and the `.dtsi` files it references, and modify the copies to correspond to your platform's design.

The following procedure will guide you through this process.

1. Copy the `.dts` file you have chosen to this location:

   ```
   arch/arm64/boot/dts/tegra210-<board>.dts
   ```

2. From the `arch/arm64/boot/dts/tegra210-platforms/` directory, copy each file whose name has the form:

   ```
   tegra210-jetson-cv-<function>-<board>-<som>-<version>.dtsi
   ```

   Copy each file to:

   ```
   arch/arm64/boot/dts/tegra210-platforms/tegra210-<board>-
   <function>.dtsi
   ```

   You may rename the copies if that makes your work easier.

3. Edit your copy of the `.dts` file to refer to your copies of the `.dtsi` files.
4. Edit the set of enabled devices and their parameters (e.g. GPIO and IRQ IDs) in each copied file as appropriate for your board.

   You may need to add, remove, or edit `.dtsi` file nodes and properties.

   Note: **U-Boot and the Linux kernel do not always use the exact same device tree schema (bindings) to represent the same data. Follow examples from the Linux kernel rather than from U-Boot.**

5. Replace the content of `tegra210-<board>-gpio.dtsi` and `tegra210-<board>-pinmux.dtsi` with the content you generated from the kernel pinmux files earlier.
6. Edit `arch/arm64/boot/dts/Makefile` to add an entry for your board, modeled after the existing Jetson™ TX1 entry.

7. Copy this file:

```
Linux_for_Tegra/bootloader/t210ref/p2371-2180/extlinux.conf.emmc
```

   To this location:

```
Linux_for_Tegra/bootloader/t210ref/<board>-<som>/extlinux.conf.emmc
```

8. Copy the generated DTB to the following directory for flashing:

```
Linux_for_Tegra/kernel/dtb/
```

   For rel-28 releases, to provide plugin manager support, the kernel DTB is not included in the file system along with the kernel image in the boot directory. Instead, the kernel DTB is selected from the DTB partition and modified by Cboot. Cboot passes it on to Uboot, which in turn passes it on to the kernel without diluting any of the data.

   - To flash only the DTB, execute the command:

```
sudo ./flash.sh  -k DTB <platform> mmcblk0p1
```

   - To flash only Uboot, execute the command:

```
sudo ./flash.sh  -k LNX <platform> mmcblk0p1
```

   - To flash only the kernel:

     1. copy the kernel image to the following directory.

```
Linux_for_Tegra/kernel/
```

     2. Execute the command:

```
sudo ./flash.sh  <platform> mmcblk0p1
```

        Optionally, you can perform a secure copy to copy the kernel image to the boot partition of the target system and reboot.

9. Copy `Linux_for_Tegra/jetson-tx1.conf` to `Linux_for_Tegra/<board>.conf`.
10. Edit `SYSBOOTFILE` and `DTB_FILE` in `<board>.conf` to refer to your board.

Following are some examples of modifications that you may have to make.

Regulator

VDDIO of SDMMC1 comes from PMU LDO2. That section looks like this:

```
    regulators {
        ldo2 {
            regulator-name = "vddio-sdmmc1";
        };
    };
```

If there is any change for this power tree, this item should be changed.

Pad Power Detection

Pad power on the T210 defaults to 3.3 V. Unlike some other Tegra models, the T210 does not have auto power detect cells. I/O pads that are powered at 1.8 V must be set manually in DTS to 1.8 V.

```
gpio {
nvidia,io-pad-init-voltage = <IO_PAD_VOLTAGE_1_8V>;
};
```

GPIO

There are many GPIO configurations in different hardware modules. To change the GPIO setting, check the related device tree file.

For example, VDD of SDMMC is controlled by a GPIO pin (GPIO_PZ3). That is a power tree module, so the definition of this part is in:

```
tegra210-platforms/tegra210-jetson-cv-power-tree-p2597-2180-a00.dtsi
```

And looks like this:

```
    en_vdd_sd: regulator@4 {
        gpio = <&gpio TEGRA_GPIO(Z, 3) 0>;
    };
```

You can change this setting according to platform circuit.

Interrupt

For modules that make interrupt requests, the interrupt requests can also be declared in the `.dts` file.

For example, this file:

```
tegra210-platforms/tegra210-comms-p2530-0930.dtsi
```

Describes a WIFI interrupt like this:

```
    bcmdhd_wlan {
        compatible = "android,bcmdhd_wlan";
        interrupt-parent = <&gpio>;
        interrupts = <TEGRA_GPIO(H, 2) 0x14>;
        wlan-pwr-gpio = <&gpio TEGRA_GPIO(H, 0) 0>;
        status = "okay";
    };
```

This specifies GPIO_PH02 (same pin as WIFI_WAKE_AP) as the interrupt request pin from the WIFI module.

Key

The key is defined in:

```
tegra210-platforms/tegra210-keys-p2530-0930.dtsi
```

The power key is defined as:

```
    power {
        label = "Power";
        gpios = <&gpio TEGRA_GPIO(X, 5) GPIO_ACTIVE_LOW>;
        linux,code = <KEY_POWER>;
        gpio-key,wakeup;
    };
```

Note: **This example is meant only to show how to define a key in a `.dts` file. 'Power key' is a special key, and *cannot* be changed.**

For the detailed information about `.dts` files, refer to the documentation at `Documentation/devicetree/bindings` in the NVIDIA released Linux kernel source package.

USB Lane Mapping

USB lane mapping and the required device tree changes are due to kernel version changes.

- Release 24.X (R24) kernel version is 3.10
- Release 28.X (R28) kernel version is 4.4

USB 3.0 has 4 superspeed ports. Not all can be used in the same implementation because of lane sharing between PCIE, SATA, and XUSB. Possible combinations for USB 3.0 are shown in the tables below.

Jetson TX1 Pin Names		PEX1	PEX_RFU	PEX2	USB_SS1	PEX0	USB_SS0	NA	SATA
Tegra X1 Lanes		Lane 0	Lane 1	Lane 2	Lane 3	Lane 4	Lane 5	Lane 6	SATA
Use Case	1 (Carrier)	PCIe#1_0	PCIe#0_3	PCIe#0_2	PCIe#0_1	PCIe#0_0	USB_SS#1	USB_SS#0 On-Jetson TX1 For LAN	SATA
	2	PCIe#1_0	PCIe#0_3	PCIe#0_2	PCIe#0_1	PCIe#0_0	USB_SS#1		USB_SS#3
	3	USB_SS#2	PCIe#0_3	PCIe#0_2	PCIe#0_1	PCIe#0_0	USB_SS#1		SATA
	4	PCIe#1_0	-	-	USB_SS#2	PCIe#0_0	USB_SS#1		SATA
	5	PCIe#1_0	-	-	USB_SS#2	PCIe#0_0	USB_SS#1		USB_SS#3

Use Case	Available Outputs from Jetson TX1		
	USB 3.0	PCIe	SATA
1 (Carrier)	1	1x1 + 1x4	1
2	2	1x1 + 1x4	0
3	2	1x4	1
4	2	2x1	1
5	3	2x1	0

The customer pinmux spreadsheet contains all Tegra X1 pin names and ball names for determining which ball names are used for the super-speed connector, and the pinmux configuration of those pins.

An example configuration is given in section 5.1 of the *Jetson TX1 OEM Product Design Guide.* Each external super-speed connector has both USB 2.0 (DP, DN) and USB 3.0 lines (TX+-, RX+-) linked to the connector. A possible exception is where a fixed, on-board device is connected to super-speed lines and does not require USB 2.0 compatibility.

R24 Required Device Tree Changes

The following R24 device tree properties must change when USB configuration changes.

Property	Description
XHCI	
nvidia,portmap	Lists all ports, 2.0 and 3.0, available for the XUSB controller. Each bit represents a port and the bit is set for ports controlled by XUSB. ▶ Bits 0-7 represent USB 3.0 ports with Port 0 on the LSB. ▶ Bits 8-15 represent USB 2.0 ports with Port 0 on Bit 8. ▶ Bits 16-23 represent HSIC ports with Port 0 on Bit 16. For example, <0x0e02> represents USB 3.0 Port 1 and USB 2.0 Port 1, 2, and 3 are enabled.
PADCTL	
nvidia,ss_portmap	Mapping between USB 2.0 USB 2.0 port lines and USB 3.0 por lines available on a single connector. ▶ Each nibble represents a USB 3.0 port starting from LSB. The matching USB 2.0 port must be entered. ▶ Enter 7 if the superspeed port is NOT in use or is not available. For example, <0x7730> represents the following mapping: ▶ ssport0-usb2port0 ▶ ssport1-usb2port3 ▶ ssport2-disabled ▶ ssport3-disabled USB 3.0 standalone ports without a USB 2.0 port require mapping to a USB 2.0 port that is a valid host port. For standalone ports, two USB 3.0 ports can be mapped to a single USB 2.0 port with the same role and may not represent the end-connector mapping.
PEX/HSIO PEX/HSIO lanes used by USB3.0 ports	
nvidia,lane_owner	Each nibble represents a USB 3.0 port starting from the LSB. A matching lane number must be entered. ▶ Enter 0xF if the port is not in use or the lane is not mapped. For example, <0xFF56> represents the following mapping: ▶ ssport0-lane6 ▶ ssport1-lane5 ▶ ssport2-not in use ▶ ssport3-not in use

R24 Example 1

A case where USB 2.0 Port 3 and USB 3.0 Port 2 are linked to an external connector on a carrier board matches usecase 3 in possible mappings of superspeed ports. In that case the USB_SS#2 Port is using Lane 0 (USB_SS1 pins, ball names E41, E42, H41, H42). USB

2.0 Port 3, ball names B42 and B43, is mapped to an external connector; the default configuration uses these lines for M2.Key.

- PCIe x1 must be disabled
- USB_SS#2 port must be enabled
- Lane mapping should indicate that lane 0 is in use by USB

In this example, the following device tree properties must be changed:

- **nvidia,ss_portmap**

  Super-speed port 2 must be enabled. This portmap should match USB 3.0 and USB 2.0 lines on the same connector. For a standalone USB 3.0 port, any valid USB 2.0 port is sufficient. In this case USB 2.0 port 3 is matched with USB 3.0 port 2, so the value changes from <0x7721> to <0x7321>.

- **nvidia,portmap**

  Control super-speed port 2 with XHCI by replacing <0x0e03> with <0x0e07>.

- **nvidia,lane_owner**

  The lane owner for lane 0 is XUSB and is used by super-speed port 2. The property value of <0xff56> becomes <0xf056>.

- **nvidia,lane-map**

  This property indicates PEX lanes are available in the PCIe module and how they are configured. Because lane 0 (PEX1) is in use by USB, the lane map is modified from <0x14> to <0x4>.

Overall device tree properties patch in this case are as follows:

```
        pcie-controller {
[...]
-                       nvidia,lane-map = <0x14>;
+                       nvidia,lane-map = <0x4>;
[...]
       };
        xusb_pad_ctl: xusb_padctl { /* Put common control config here
*/
[...]
-                       nvidia,ss_portmap = <0x21>;
-                       nvidia,lane_owner = <0xff56>; /* Use 0xF to
disable lane assign */
-                       nvidia,lane-map = <0x14>;
+                       nvidia,ss_portmap = <0x321>;
+                       nvidia,lane_owner = <0xf056>; /* Use 0xF to
disable lane assign */
+                       nvidia,lane-map = <0x4>;
[...]
       };
        xusb@70090000 {
```

```
[...]
-                       nvidia,portmap = <0x0e03>;
+                       nvidia,portmap = <0x0e07>;
[...]
       };
```

R24 Example 2

USB 3.0 Port 2 is used as a standalone port using lane 3. This matches usecase 4/5 in possible mappings of superspeed ports. USB_SS#2 port is using lane 3 (PEX1 pins, ball names G42, G43, D42, D43). PCIe configuration changes from 1x1+1x4 to 1x1+1x1. USB_SS#2 is enabled and lane mapping indicates that lane 3 is in use by USB.

The following device tree properties must change:

- **nvidia,ss_portmap**

  Superspeed port 2 is enabled. This portmap must match USB 3.0 and USB 2.0 lines on the same connector. For a standalone USB 3.0 port, specify any valid USB 2.0 Port. In this case, a USB 2.0 standalone port matches a valid USB 2.0 port. This value changes from <0x7721> to <0x7221>.

- **nvidia,portmap**

  Control superspeed Port 2 with XHCI by replacing <0x0e03> with <0x0e07>.

- **nvidia,lane_owner**

  The lane owner for lane 3 is XUSB and is used by superspeed Port 2. The property value of <0xff56> becomes <0xf356>.

- nvidia,lane-map

  This property indicates that PEX lanes are available in the PCIe module and describes how they are configured. Because lane 3 USB_SS1 is in use by USB, the lane map is modified from <0x14> to <0x11>.

Overall device tree properties patch for this use case are as follows:

```
        pcie-controller {

-                       nvidia,lane-map = <0x14>;
+                       nvidia,lane-map = <0x12>;
[...]
       };
        xusb_pad_ctl: xusb_padctl { /* Put common control config here
*/
[...]
-                       nvidia,ss_portmap = <0x7721>;
-                       nvidia,lane_owner = <0xff56>; /* Use 0xF to
disable lane assign */
```

```
-                       nvidia,lane-map = <0x14>;
+                       nvidia,ss_portmap = <0x7221>;
+                       nvidia,lane_owner = <0xf356>; /* Use 0xF to
disable lane assign */
+                       nvidia,lane-map = <0x11>;
[...]
        };
         xusb@70090000 {
[...]
-                       nvidia,portmap = <0x0e03>;
+                       nvidia,portmap = <0x0e07>;
                        nvidia,common_padctl = <&xusb_pad_ctl>;
[...]
        };
```

R28 Required Device Tree Changes

The following R28 device tree properties must change when USB configuration changes.

Property	Description
pinctrl@7009f000/pinmux	
nvidia,lanes	Identifies the lane used on this port. ▶ For USB 2.0 port, set this property to otg-<N>. Where <N> is the number of USB 2.0 port. ▶ For USB 3.0/PCIe/SATA ports, set this property to uphy-lane-<N>. Where <N> is the lane number for this port.
nvidia,function	Specifies the function running on this port. Possible values include: ▶ xusb: indicates this port is used by USB 2.0 controller. ▶ usb3: indicates this port is used by USB 3.0 controller. ▶ pcie: indicates this port is used by PCIe controller.
nvidia,port-cap	If this USB 2.0/3.0 port supports host mode, set this property to <TEGRA_PADCTL_PORT_HOST_ONLY>.
nvidia,usb3-port	Set this property on USB 3.0 port only because it specifies which USB 3.0 port is bound to the lane set by the nvidia,lanes property.
nvidia,usb2-map	set this property on USB 3.0 port because it specifies which USB 2.0 port is used with this USB 3.0 on the same connector.
nvidia,pcie-lane-select	Specifies that this PCIe port supports x1 or x4.
xusb@70090000	
phys	Identifies the PHY handles to the USB 2.0 and 3.0 ports used by XUSB controller.
phy-names	Identifies the PHY names of the USB 2.0 and 3.0 phys used by XUSB controller.

	The sequence in phys and phy-names must match each other.
pcie-controller@1003000	
nvidia,num-lanes	This property must be present in each root port sub-node of the PCIe controller parent node. For example: ▶ pci@1,0, pci@2,0 and pci@3,0 nodes. It indicates the maximum link width of that root port. For example: ▶ For controller-0, use 0x4for all cases. ▶ For controller-1, only 0x1 is supported.

R28 Example 1

A case where USB 2.0 port 3 and USB 3.0 port 2 are linked to an external connector on a carrier board matches use case 3 in possible mappings of super-speed ports. In that case the USB_SS#2 port is using Lane 0 (USB_SS1 pins, ball names E41, E42, H41, H42). USB 2.0 port 3 (ball names B42, B43) is mapped to an external connector (the default configuration uses these lines for M2.Key). PCIe x1 should be disabled, USB_SS#2 port should be enabled and lane mapping should now indicate that lane 0 is in use by USB.

In this example, the following device tree properties must be changed:

1. Disable pcie-m2 sub node.

   Disable PCIe x1 on M2.Key by adding status = "disabled" to the `pinctrl@7009f000/pinmux/pcie-m2` sub node.

2. Create two new sub-nodes under pinctrl@7009f000/pinmux.

   Create two sub-nodes, such as usb2-m2 to represent USB 2.0 port 3 and usb3-m2 as an instance of USB3.0 port 2, then set the following properties of them.

- **nvidia,lanes**

  In this case, USB 3.0 port 2 is using Lane 0, so set nvidia,lanes = "uphy-lane-0" under usb3-m2 sub-node. Since USB 2.0 port 3 is used on M2.Key, set nvidia,lanes = "otg-3" of usb2-m2 sub-node.

- **nvidia,function**

  Set nvidia,function = "xusb" in usb2-m2 and nvidia,function = "usb3" for usb3-m2.

- **nvidia,port-cap**

  Set `nvidia,port-cap = <TEGRA_PADCTL_PORT_HOST_ONLY>` in both usb2-m2 and usb3-m2 sub-nodes because USB connector on M2.Key supports only host mode.

- **nvidia,usb3-port**

This property must be set in usb3-m2 to indicate which USB 3.0 port is bound to the lane set in nvidia,lanes. In this case, it is USB 3.0 port 2, so the value is nvidia,usb3-port = <2> in usb3-m2.nvidia,ss_portmap.

- **nvidia,usb2-map**

  This nvidia,usb2-mapportmap must match USB 3.0 and USB 2.0 lines on the same connector and only be set in usb3-m2 sub-node. For a standalone USB 3.0 port, any valid USB 2.0 port is sufficient. In this case, USB 2.0 port 3 is matched with USB 3.0 port 2, so the value must be set to `nvidia,usb2-map = <3>.`

- **phys**

  Add `<&tegra_padctl_uphy TEGRA_PADCTL_UPHY_UTMI_P(3)>` for USB 2.0 port 3 and `<&tegra_padctl_uphy TEGRA_PADCTL_UPHY_USB3_P(2)>` for USB3.0 port2.

- **phy-names**

  Add "utmi-3" for USB 2.0 port 3 and "usb3-2" for USB 3.0 port 2.

The overall device tree properties patch in this case are as follows:

```
        pinctrl@7009f000 {
[...]
         pinmux {
[...]
            pcie-m2 {
              nvidia,lanes = "uphy-lane-0";
              nvidia,function = "pcie";
              nvidia,pcie-controller = <1>;
              nvidia,pcie-lane-select = <TEGRA_PADCTL_PCIE_LANE_X1>;
+             status = “disabled”;
            };
+            usb2-m2 {
+             nvidia,lanes = "otg-3";
+             nvidia,function = "xusb";
+             nvidia,port-cap = <TEGRA_PADCTL_PORT_HOST_ONLY>;
+            };
+            usb3-m2 {
+             nvidia,lanes = "uphy-lane-0";
+             nvidia,function = "usb3";
+             nvidia,usb3-port = <2>;
+             nvidia,usb2-map = <3>;
+             nvidia,port-cap = <TEGRA_PADCTL_PORT_HOST_ONLY>;
+             };
          };
        };
[...]
        xusb@70090000 {
[...]
           phys = <&tegra_padctl_uphy TEGRA_PADCTL_UPHY_UTMI_P(2)>,
```

```
                    <&tegra_padctl_uphy TEGRA_PADCTL_UPHY_USB3_P(1)>,
                    <&tegra_padctl_uphy TEGRA_PADCTL_UPHY_UTMI_P(1)>,
                    <&tegra_padctl_uphy TEGRA_PADCTL_UPHY_USB3_P(0)>,
                    <&tegra_padctl_uphy TEGRA_PADCTL_UPHY_UTMI_P(0)>,
+                   <&tegra_padctl_uphy TEGRA_PADCTL_UPHY_UTMI_P(3)>,
+                   <&tegra_padctl_uphy TEGRA_PADCTL_UPHY_USB3_P(2)>;
            phy-names = "utmi-2", "usb3-1", "utmi-1", "usb3-0", "utmi-0",
+                       "utmi-3", "usb3-2";
 [...]
        };
```

R28 Example 2

USB 3.0 port 2 is used as a standalone port using lane 3. This matches use case 4/5 in possible mappings of super speed ports. USB_SS#2 port is using lane 3 (PEX1 pins, ball names G42, G43, D42, D43). PCIe configuration changes from 1x1+1x4 to 1x1+1x1. USB_SS#2 is enabled and lane mapping indicates that lane 3 is in use by USB.

The following device tree properties must change:

1. Create two new sub-nodes under pinctrl@7009f000/pinmux.

   Create two sub-nodes, such as usb2-standalone to represent USB 2.0 port 3 and usb3-standalone as an instance of USB3.0 port 2, then set the following properties of them.

- **nvidia,lanes**

  In this case, USB 3.0 port 2 is using Lane 3, so set nvidia,lanes = "uphy-lane-3" under usb3-standalone sub-node and set nvidia,lanes = "otg-3" of usb2- standalone sub node.

  Since PCIe configuration changes from 1x1+1x4 to 1x1+1x1, change nvidia,lanes of sub-node pcie from 4 lanes("uphy-lane-1", "uphy-lane-2", "uphy-lane-3", "uphy-lane-4") to "uphy-lane-4".

- **nvidia,function**

  Set nvidia,function = "xusb" in usb2-standalone and nvidia,function = "usb3" for usb3-standalone.

- **nvidia,port-cap**

  Set `nvidia,port-cap = <TEGRA_PADCTL_PORT_HOST_ONLY>` in both usb2-standalone and usb3-standalone sub-nodes because USB connector supports only host mode.

- **nvidia,usb3-port**

This property must be set in usb3-standalone to indicate which USB 3.0 port is bound to the lane set in nvidia,lanes. In this case, it is USB 3.0 port 2, so the value is nvidia,usb3-port = <2> in usb3-standalone.

- **nvidia,usb2-map**

  This nvidia,usb2-map must match USB 3.0 and USB 2.0 lines on the same connector and only be set in usb3-m2standalone sub-node. In this case, USB 2.0 port 3 is matched with USB 3.0 port 2, so the value must be set to `nvidia,usb2-map = <3>`.

- **nvidia,pcie-lane-select**

  PCIe configuration changes from 1x1+1x4 to 1x1+1x1, it must always be set to `<TEGRA_PADCTL_PCIE_LANE_X4>` for controller 0.

- **phys**

  Add `<&tegra_padctl_uphy TEGRA_PADCTL_UPHY_UTMI_P(3)>` for USB 2.0 port 3 and `<&tegra_padctl_uphy TEGRA_PADCTL_UPHY_USB3_P(2)>` for USB3.0 port2.

- **phy-names**

  Add "utmi-3" for USB 2.0 port 3 and "usb3-2" for USB 3.0 port 2.

The overall device tree properties patch for above case would look like:

```
        pinctrl@7009f000 {
[...]
          pinmux {
[...]
            pcie {
-             nvidia,lanes =  "uphy-lane-1", "uphy-lane-2",
-                             "uphy-lane-3", "uphy-lane-4";
+             nvidia,lanes =  "uphy-lane-4";
              nvidia,function = "pcie";
              nvidia,pcie-controller = <0>;
              nvidia,pcie-lane-select =
<TEGRA_PADCTL_PCIE_LANE_X4>
            };
            pcie-m2 {
              nvidia,lanes = "uphy-lane-0";
              nvidia,function = "pcie";
              nvidia,pcie-controller = <1>;
              nvidia,pcie-lane-select = <TEGRA_PADCTL_PCIE_LANE_X1>;
+             status = “disabled”;
            };
+            usb2-standalone {
+             nvidia,lanes = "otg-3";
+             nvidia,function = "xusb";
+             nvidia,port-cap = <TEGRA_PADCTL_PORT_HOST_ONLY>;
+            };
```

```
+             usb3-standalone {
+              nvidia,lanes = "uphy-lane-3";
+              nvidia,function = "usb3";
+              nvidia,usb3-port = <2>;
+              nvidia,usb2-map = <3>;
+              nvidia,port-cap = <TEGRA_PADCTL_PORT_HOST_ONLY>;
+              };
          };
        };
[...]
         xusb@70090000 {
[...]
          phys = <&tegra_padctl_uphy TEGRA_PADCTL_UPHY_UTMI_P(2)>,
                 <&tegra_padctl_uphy TEGRA_PADCTL_UPHY_USB3_P(1)>,
                 <&tegra_padctl_uphy TEGRA_PADCTL_UPHY_UTMI_P(1)>,
                 <&tegra_padctl_uphy TEGRA_PADCTL_UPHY_USB3_P(0)>,
                 <&tegra_padctl_uphy TEGRA_PADCTL_UPHY_UTMI_P(0)>,
+                <&tegra_padctl_uphy TEGRA_PADCTL_UPHY_UTMI_P(3)>,
+                <&tegra_padctl_uphy TEGRA_PADCTL_UPHY_USB3_P(2)>;
          phy-names = "utmi-2", "usb3-1", "utmi-1", "usb3-0", "utmi-0",
+                     "utmi-3", "usb3-2";
 [...]
        };
```

Other Considerations When Porting

Other considerations and recommendations to consider when porting are as follows.

To flash the build image

- When flashing the build image, use your specific board name:

```
$ sudo ./flash.sh <board>-<som> mmcblk0p1
```

To flash with BOARDID if the design does not use EEPROM

BOARDID is either passed using an XML file during flashing or is read from EEPROM. The flashing software uses the BOARDID from the XML file if provided; otherwise it uses the EEPROM value. The file `board_config_p2597-devkit.xml`, shown below, illustrates the XML file format.

```
<?xml version="1.0"?>
<!-- Nvidia Tegra board info configuration file -->
<board_configs>
    <board type="proc" id="2180" sku="1000" fab="0" />
    <board type="display" id="0000" sku="0000"/>
    <board type="pmu" id="2180" sku="0000" />
</board_configs>
```

This flashing config file `p2371-2180-devkit.conf` passes the name of the XML file as an option:

```
BCFFILE="bootloader/${target_board}/cfg/board_config_p2597-devkit.xml";
```

The file contains the processor module ID (`type="proc"`), display board ID (`type="display"`), and power management unit ID (`type="pmu"`). Since the processor and PMU are on the same module in the development kit, they have the same ID.

If you add new values for the `board` tag's `id` property, you must add them to the list of valid values in `nvtboot`.

To change the UART port to UARTA

1. In `Linux_for_Tegra/<board>.conf`, modify the ODMDATA assignment:

   ```
   ODMDATA=0x60084000;
   ```

2. In the U-Boot boot loader, locate the following lines in `/include/configs/jetson-tx1.h`:

   ```
   #define CONFIG_TEGRA_ENABLE_UARTD
   #define CONFIG_SYS_NS16550_COM1             NV_PA_APB_UARTD_BASE
   ```

3. Modify those lines to specify UARTA:

   ```
   #define CONFIG_TEGRA_ENABLE_UARTA
   #define CONFIG_SYS_NS16550_COM1             NV_PA_APB_UARTA_BASE
   ```

4. In the kernel, modify the `debug_uartport` assignment:

   ```
   debug_uartport=lsport,0
   ```

Hardware Bring-Up Checklist

This section provides a checklist for the platform hardware bring-up process.

Before Power-On

Make sure that the Jetson TX1 is connected to the BTB connector correctly and securely.	☐
Verify that power supplies are not shorted to ground or to other power supplies.	☐

Initial Power-On

Verify that VDD_IN from carrier board is in the 6 V to 19 V range.	☐
Verify that CARRIER_WR_ON goes to HIGH when power is turned on.	☐
Verify that system can enter force recovery.	☐

Initial Software Flashing

Verify that system can be flashed with TegraFlash.	☐
Verify that TegraBoot and U-boot run to completion by checking log output.	☐
Verify that OS runs to desktop.	☐
Verify that any UARTs intended for debugging are enabled and functional.	☐

Power

Verify that all supplies required on at power-on are enabled appropriately.	☐
Verify that all supplies required off at power-on are not enabled initially.	☐
Verify that each controllable supply can be enabled and disabled, and different voltage levels can be set if applicable.	☐
Verify that carrier board power-on sequence starts after CARRIER_PWR_ON signal is asserted.	☐

Power Optimization

Capture CPU PWR Request entering and exiting Suspend (LP1) and Deep Sleep (LP0). Ensure that CPU PWR Request and associated power rail sequence meets Tegra Data Sheet requirements.	☐
Verify that all rails which must be OFF in Deep Sleep (LP0) are OFF.	☐
Verify that all rails which must be ON in Deep Sleep (LP0) are ON.	☐
Verify that required rails are back and at correct voltage under hardware control exiting Deep Sleep (LP0).	☐

USB 2.0 PHY

Verify that USB0 supports USB Recovery (device mode).	☐
Verify that USB0 device mode works with intended peripheral types, if supported.	☐
Verify USB0, USB1 and or USB2 Host mode, if implemented.	☐
Verify USB0 Device/Host detection, if supported.	☐
Verify that USB PHYs go to lowest power mode when not used or when the system is in low power mode.	☐
Verify that AVDD_USB and AVDD_PLL_UTMIP are off during Deep Sleep (LP0).	☐
Capture USB0_D+/D- signals at both ends of link (connector and test points near Tegra).	☐
Capture USB2_D+/D- signals at both ends of link (connector and test points near Tegra).	☐
Using USB-IF procedures, verify that signals meet requirements (correct eye height/width, etc.).	☐
If USB signals do not meet requirements, use the *Tegra USB Tuning Guide* to adjust settings until requirements are met.	☐

USB 3.0

Verify USB 3.0 Host mode.	☐
Verify USB 3.0 Device mode, if enabled.	☐
Verify that the USB 3.0 interface goes to the lowest power mode when not used or when the system is in low power mode.	☐

HDMI

Verify that HDMI-compatible display works at 1080p.	☐
Verify that display is detected properly (HPD).	☐
Verify that HDMI reads and writes to the display using DDC interface.	☐
Verify that HDMI related rails are powered off when not used or system is in Deep Sleep (LP0) or Suspend (LP1).	☐
Capture HDMI signals at the connector (using appropriate test fixture and termination).	☐
Verify that signal quality is acceptable (meets EYE diagram, etc.). Consult *Tegra HDMI Tuning Guide* for details.	☐
If HDMI signals do not meet requirements, use the *Tegra HDMI Tuning Guide* to adjust settings until requirements are met.	☐

Audio

Verify reads and writes on I2C interface used for Audio Codec.	☐

Verify that playback works properly on speakers, headphones, and headset.	☐
Verify that capture works properly: Sound is recorded from microphone/headset if supported.	☐
Verify that tones, voice, etc. can be heard from speakers or headphones/headset.	☐
Verify that Audio Codec goes to lowest power mode when not in use or system enters low power mode.	☐
Capture signals at receiver end of link, if accessible, for each I2S I/FT used.	☐
Verify that signal quality is acceptable. Look for excessive over/undershoot and glitches on signal edges.	☐

UART

Verify that Tegra TX/RX/CTS/RTS connects to device RX/TX/RTS/CTS for each UART used.	☐
Verify that signal quality is acceptable. Look for excessive over/undershoot and glitches on signal edges.	☐

SD Card (SDMMC1)

Verify proper connectivity by setting Tegra pins to GPIOs, if necessary, to debug.	☐
Verify that basic SD commands operate properly.	☐
Verify reads and writes for a variety of SD Cards.	☐
Verify that SD Card insertion detection works and wakes system, if supported.	☐
Verify that SD Card Write Protect works, if implemented.	☐
Verify that SD Card goes to low power mode or rails are powered off when not used or in low power system state.	☐
Verify that signal quality is acceptable when probed at receiver end (socket or test points near BTB connector or both for bidirectional signals). Look for excessive over/ undershoot and glitches on signal edges and abnormal Clock duty cycle.	☐

Sensors I2C: General

Verify that addresses of all I2C devices appear correctly, and no unknown ghost devices appear.	☐
Verify that signal quality is acceptable, including rise times of signals, when probed at BTB connector and devices.	☐

Sensors I2C: Touch Screen (Optional)

Verify that Reads/Writes on I2C or SPI to Touch Screen controller are functional (reading device ID or a similar register is successful).	☐

Verify that interrupts are generated properly.	☐
Verify functionality of Touch Screen.	☐
Verify that Touch Screen Controller goes to lowest power mode when not used, or system is in low power state.	☐

PEX (Optional)

Verify proper connectivity by checking lanes.	☐
Verify that any implemented PEX interfaces transition to the lowest power state in Deep Sleep (LP0) and Suspend (LP1).	☐
Verify that signal quality is acceptable when probed at receiver end of link near Tegra and device. Look for excessive over/ undershoot and glitches on signal edges.	☐

SATA (Optional)

Verify proper connectivity by checking diff lines.	☐
Verify that any implemented SATA interfaces transition to the lowest power state in Deep Sleep (LP0) and Suspend (LP1).	☐
Verify that signal quality is acceptable when probed at receiver end of link near Tegra and device. Look for excessive over/ undershoot and glitches on signal edges.	☐

Embedded Display(s) (Optional)

Verify that I2C or other control interface is able to perform writes/reads to display.	☐
Verify that each embedded display shows correct colors.	☐
Verify that each embedded display’s backlight is enabled when in normal display mode.	☐
Verify that each embedded display’s backlight brightness can be adjusted properly.	☐
Verify that each embedded display’s backlight is disabled when in a low power mode.	☐
Verify that each embedded display (and any display bridge) transitions to the lowest power state in Deep Sleep (LP0) and Suspend (LP1).	☐
Verify that power-on/off sequencing of rails associated with each display meets manufacturer's requirements.	☐
Verify DSI, LVDS or eDP timing (see *Tegra DC and DSI Debugging Guide* for details on how and what to verify).	☐
Probe DSI, LVDS or eDP signals near panel driver, or at connector/test points if access to driver is not possible, and verify that signal quality is acceptable. Look for excessive over/undershoot and glitches on signal edges.	☐

Imager(s) (Optional)

Verify that I2C interface writes/reads work to all cameras.	☐
Verify that preview displays properly for all cameras.	☐
Verify that still capture works on all cameras.	☐
Verify that video capture works on all cameras.	☐
Verify that all flashes operate properly.	☐
Verify that any available autofocus mechanism functions properly.	☐
Verify that privacy LED operates properly, if implemented.	☐
Verify that cameras and related circuitry enter lowest power mode when not used or system is in a low power mode.	☐
Verify that power-on/off sequencing of rails associated with imager module meets manufacturer's requirements.	☐
Probe MCLK output at recommended test points, and verify that signal quality is acceptable. Look for excessive over/undershoot and glitches on signal edges.	☐
Look for excessive over/undershoot and glitches on signal edges.	☐

Software Bring-Up Checklist

This section provides a checklist for the software bring-up process.

Preparation

Verify Board BCT.	☐
Verify operation eMMC with the NVIDIA Diagnostic Tool.	☐
Obtain board schematics and component data sheets.	☐
Verify power tree.	☐
Review board pinmux.	☐

Bring-up Hardware Validation

Power and Reset Sequence, Power Rail Check	☐
Recovery Mode	☐
NvTest (Tegra MODS) DDR, eMMC, CPU	☐
JTAG connection check	☐

U-Boot Port and Boot Validation

TegraFlash	☐
UART output	☐
KBD connection	☐
Board config/PMIC regulator config/Pinmux/Review device tree	☐
Verify FS support/Config boot scripts (bootcmd)	☐
Boot to U-boot	☐
Boot to kernel	☐
Boot to kernel command line or custom desktop	☐

Kernel and Peripherals, Port and Validation

Device tree review, Pinmux, GPIO, Wake pins	☐
PMU and regulator drivers	☐
Display/HDMI	☐
Audio codec	☐
Microphone and speaker	☐
USB	☐
SD card	☐
Thermal Sensor	☐
EMC DFS table	☐
Ethernet	☐
SATA	☐
PCIe	☐

System Power and Clocks

CPU/CORE/GPU DVFS	☐
EMC DFS table	☐
CPU/CORE EDP	☐
GPU EDP	☐
System EDP (Contain Current monitor & Voltage comparator)	☐
Power Off	☐
LP0 (optional)	☐
CPU power down (LP2)	☐
BCT, Full-speed	☐

Notice

Trademarks

Copyright