NVIDIA

JETSON TX2 PLATFORM ADAPTATION AND BRING-UP GUIDE

DA_08477-002 | July 2, 2019

L4T Driver Package

Document Change History

DA_08477-002

Version	Date	Authors	Description of Change
v1.0	2 Mar 2017	twarren/bbasu/snath	Initial release for Jetson TX2
v1.1	14 Mar 2017	bbasu	Added Power Tree changes
v1.2	8 May 2017	bbasu	board configuration updates
v1.3	30 Jun 2017	jerchang/mzensius	GPIO-related updates
v1.4	29 Jan 2018	dliu/kstone	Device tree settings for QSPI_IO2
v1.5	30 May 2018	wwang/jsachs	Addition to “Required Device Tree Changes”
v1.6	6 Jun 2018	wwang/jsachs	New section, “To configure USB3.0 OTG”
v1.7	25 Jun 2018	wwang/jsachs	Add details to several sections: MB1 Configuration Changes, PinMxx Changes, USB-Lane Mapping.
v1.8	7 Dec 2018	wwang/jsachs	Correct USB Lane Mapping notes.
v1.9	2 Jul 2019	wchang/jsachs	Updates for L4T Release 32.1; new “Porting USB” section.

Table of Contents

List of Figures

List of Tables

Platform Adaptation and Bring-Up Guide

This document describes how to port the NVIDIA® Tegra® Linux Driver Package and the U-Boot bootloader from NVIDIA® Jetson™ TX2 Developer Kit to other hardware platforms.

The examples described include code for the Jetson TX2 Developer Kit (P2771).

For information on customizing the configuration files, refer to Tegra Linux Driver Package Development Guide, "MB1 Platform Configuration" and "Configuring Pinmux, GPIO and PAD" topics.

Board Configuration

The Jetson TX2 module consists of a P3310 main board that sits on a P2597 base board. The complete kit is named P2771 Jetson TX2 Developer Kit. The P3310 main board can be used without any software configuration modifications. The P3310 board sits on the P2597 I/O expansion base board. Both these boards have an EEPROM where the board ID is saved.

Before replacing the P2597 base board, verify the software programming of the Kernel device tables, MB1 configuration, ODM data, and flashing to de-configure the P2597 board with the custom configurations of your custom board. EEPROM ID for your custom board is not required.

Board Naming

To support your board in L4T, you must select a simple lower-case, alpha-numeric name for your board. The name can include dashes (-) or underscores (_) but cannot contain spaces. For example:

```
jetson-tx2
p2771-000-500
myboard
```

The name you select appears in:

- Filenames and pathnames
- U-Boot and Linux kernel source code
- User-visible device tree filenames

Additionally, this name is exposed to the user through the U-Boot command prompt and various Linux kernel `proc` files.

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

You must also select 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 re-use and modify the existing NVIDIA® Jetson™ TX2 Developer Kit code without selecting and using your own board name. If you do not use your own board name it will not be obvious to Jetson TX2 users whether the modified source code supports the original Jetson TX2 Developer Kit board or your board.**

Placeholders in the Porting Instructions

Placeholders are used throughout this document, substitute an appropriate value for each placeholder when executing 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 selected to represent your platform. For example, `p2771` is the name of the Jetson TX2 Developer Kit. NVIDIA `<board>` names use lower case letters.
- `<version>` is a board version number, such as `a00`. Files for NVIDIA reference boards include a version number. Files for customer platforms are not required to include a version number.
- `<vendor>` is the name of your organization, or the name of the vendor for your board.
- `<root>` is the device that holds root file system for the platform. The supported value is `emmc`.

MB1 Configuration Changes

MB1 provides the boot time configuration of the hardware applied by the bootloader. The MB1 boot configuration tables are available at:

```
<l4t_top>/bootloader/t186ref/BCT
```

Pinmux Changes

If your board schematic differs from that for Jetson™ TX2 Developer Kit board, you must change the pinmux configuration applied by the software.

The `Jetson-TX2-Generic-Customer-Pinmux-Template.xlsm` spreadsheet is provided to:

- Show the locations and default pinmux settings
- Define the pinmux settings in the source code or device tree

  The spreadsheet is available at:

  `https://developer.nvidia.com/embedded/downloads`

You must customize the spreadsheet for the configuration of your board.

GPIO Changes

If you designed your own carrier board, to translate from SOM-connector pins to actual GPIO numbers you must understand GPIO mapping formula below. The translated GPIO numbers can be controlled by the driver.

For example, to check the GPIO number of GPIO15/AP2MDM_READY. perform the following steps.

To check the GPIO number

1. Search for `GPIO15_AP2MDM_READY` in `Jetson_TX2_Generic_Customer_Pinmux_Release.xlsx`.
2. Confirm that the Customer Usage field is applied to `GPIO3_PBB.00`.
3. Confirm in tegra186-gpio.h that GPIO3_PBB.00 belongs to the main Tegra GPIO group, and that the port number is 21:

   ```
   #define TEGRA_MAIN_GPIO_PORT_BB 21
   ```

4. Because the SoC device registers GPIOs dynamically, search kernel messages to check GPIO allocation ranges for each GPIO group. The command and resulting output are similar to the following:

```
$ dmesg | grep gpiochip_add_data
[    1.247404] gpiochip_add_data: registered GPIOs 320 to 511 on
device: tegra-gpio
[    1.262595] gpiochip_add_data: registered GPIOs 256 to 319 on
device: tegra-gpio-aon
```

   As shown in the outpout above, there are 2 Tegra GPIO ports with different offsets:

   - tegra-gpio, offset = 320
   - tegra-gpio-aon, offset= 256

5. Because `PBB00` belongs to the tegra-gpio group, the port number from step 3 is 21, and the offset is 320. Use the following formula to calculate the GPIO number:

```
TEGRA_MAIN_GPIO(port, offset) =
((TEGRA_MAIN_GPIO_PORT_##port * 8) + offset)
```

Hence, the GPIO number of `GPIO15/AP2MDM_READY` is (21*8)+320 = 488.

PMIC Changes

The PMIC configuration file configures the initial PMIC in the P3310 board. Some GPIO expander-based GPIO regulator settings in the P2597 base board configurations are also defined. Review this configuration file to replace any references to the P2597 board to your custom board. If required, include any regulator information to enable this file.

For example, remove the following section that is writing to a slave on the I2C controller 0 address 0x74 in the P2597 base board. Additionally, update the number of blocks and array number for other entries of the block:

```
tegra186-mb1-bct-pmic-quill-p3310-1000-c04.cfg

# 5V0_HDMI_EN
pmic.generic.1.block[2].type = 1; # I2C Type
pmic.generic.1.block[2].i2c-controller-id = 0;
pmic.generic.1.block[2].slave-add = 0xE8; # 7BIt:0x74
pmic.generic.1.block[2].reg-data-size = 8;
pmic.generic.1.block[2].reg-add-size = 8;
pmic.generic.1.block[2].block-delay = 10;
pmic.generic.1.block[2].count = 2;
pmic.generic.1.block[2].commands[0].0x07.0xFF = 0xEF;
pmic.generic.1.block[2].commands[1].0x03.0xFF = 0x10;
```

Porting the Linux Kernel

It is assumed that you are using the CVM module provided by NVIDIA and that it has not been modified; the eMMC, PMIC, and DDR are the same with the same routing of lines. The modifications you are making are for the CVB baseboard that hosts all the peripherals. Consequently, based on the peripherals present on your baseboard, you can modify the `.dts` files by disabling/enabling the controllers and changing the supplies.

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

The configuration files available at:

```
<top>/hardware/nvidia/platform/t18x/
<top>/hardware_nvidia/soc/t18x
```

The final DTB file used is:

```
tegra186-quill-p3310-1000-a00-00-base.dts
```

By reading the above file, you see which other `.dtsi` files are referenced by include statements. Common `.dtsi` files that may be modified to reflect hardware design changes include:

Types of Changes	DTSI Filename or location
Power supply changes	tegra186-quill-power-tree-p3310-1000-a00-00.dtsi
Regulator parameter changes	tegra186-quill-spmic-p3310-1000-a00-00.dtsi
Display panel and node changes	Refer to the *Tegra Linux Driver Package Development Guide Display Configuration and Bringup* topic for details.
ODM data based feature configuration	tegra186-odm-data-plugin-manager.dtsi
NVIDIA SOC controller state to enable/disable a controller	soc/t18x/kernel-dts/tegra186-soc/
Panels related .dts files	tegra/common/kernel-dts/panels/

Verify that no other `.dts` or `.dtsi` file, including these `.dts` files, overrides any changes you make.

As a best practice, create your own set of `.dts` files based on the Quill files already present. Rename your newly created files to the name of your board.

Note: **Use `fdtdump` or `dtc` to generate a `.dts` from the final `.dtb` file and check if your changes have taken effect.**

The command usage is as follows:

```
dtc -I dtb -O dts tegra186-quill-p3310-1000-a00-00-base.dtb > tegra186-
quill-p3310-1000-a00-00-base.dts
fdtdump dts tegra186-quill-p3310-1000-a00-00-base.dtb > tegra186-quill-
p3310-1000-a00-00-base.dts
```

Power Tree Changes

The Jetson P2597 baseboard has a GPIO expander. Some of the pins on the GPIO expander are used as a GPIO regulator. One such usage is to enable `vbus-2-supply` which is powered using `vdd_usb2_5v` GPIO regulator. If your custom board does not have the `vdd_usb2_5v` supply, the `xhci` driver enumeration fails on the target system. To solve this situation, you must:

1. Change the supply with `battery_reg` using the `.dtsi` file located at:

   ```
   hardware/nvidia/platform/t18x/common/kernel-dts/t18x-common-
   platforms/tegra186-quill-power-tree-p3310-1000-a00-00.dtsi
   ```

2. Regenerate the DTB.
3. Flash with the correct DTB.

The modifications are as follows:

```
pinctrl@3520000 {
    vbus-0-supply = <&vdd_usb0_5v>;
    vbus-1-supply = <&vdd_usb1_5v>;
    vbus-2-supply = <&battery_reg>;
    vbus-3-supply = <&battery_reg>;
    vddio-hsic-supply = <&battery_reg>;
    avdd_usb-supply = <&spmic_sd3>;
    vclamp_usb-supply = <&spmic_sd2>;
    avdd_pll_erefeut-supply = <&spmic_sd2>;
};
```

To disable XHCI

4. Change the lane configuration.
5. Update the following node.

```
xhci@3530000 {
    status = "disabled";
    phys = <&tegra_xusb_padctl TEGRA_PADCTL_PHY_UTMI_P(0)>,
           <&tegra_xusb_padctl TEGRA_PADCTL_PHY_UTMI_P(1)>,
           <&tegra_xusb_padctl TEGRA_PADCTL_PHY_USB3_P(1)>;
```

```
    phy-names = "utmi-0", "utmi-1", "usb3-1";
};
```

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

Porting USB (Universal Serial Bus)

Jetson TX2 can support up to three SuperSpeed USB ports. In some implementations not all of these ports can be used because of UPHY lane sharing among PCIE, SATA, UFS, and XUSB. The Jetson P2597 carrier board is designed and verified for one USB3.0 port and one USB2.0 OTG port. If you designed your own carrier board, verify the UPHY lane mapping and compatibility between P2597 and your custom board by consulting the NVIDIA team.

USB Structure

An enhanced SuperSpeed USB port has nine pins:

- VBUS
- GND
- D+
- D−
- Two differential signal pairs for SuperSpeed data transfer
- One ground (GND_DRAIN) for drain wire termination and managing EMI, RFI, and signal integrity

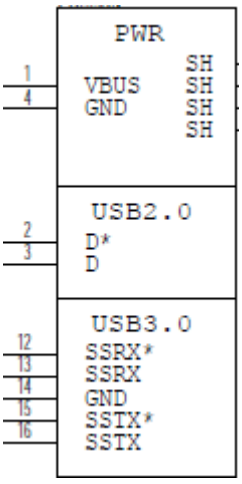

Figure 1. Enhanced SuperSpeed USB pin-out

The D+/D− signal pins connect to UTMI pads. The SSTX/SSRX signal pins connect to UPHY and are handled by a single UPHY lane. As UPHY lanes are shared between

PCIE, SATA, UFS, and XUSB, UPHY lanes must be assigned according to the custom carrier board's requirements.

UPHY Lane Assignment

UPHY is an acronym for **universal physical layer**, a physical I/O interface layer that can serve multiple types of interfaces, e.g. USB, PCIe, SATA, and UFS. A UPHY lane can support multiple types of interfaces.

The Jetson P2597 carrier board is designed and verified for one USB3.0 port and one USB2.0 OTG port. The possible use cases and their UPHY lane assignments are shown in Table 1.

Table 1. Available outputs for Jetson TX2

Config	Jetson TX2 Pin Names			PEX1/ USB_SS0	PEX_RFU	PEX2	USB_SS1	PEX0	SATA
	Tegra Lanes			Lane 0	Lane 1	Lane 3	Lane 2	Lane 4	Lane 5
	Available Outputs from Jetson TX2								
	USB 3.0	PCIe	SATA						
1	0	1x1 + 1x4	1	PCIe#2_0	PCIe#0_3	PCIe#0_2	PCIe#0_1	PCIe#0_0	SATA
2 *	1	1x4	1	USB_SS#0	PCIe#0_3	PCIe#0_2	PCIe#0_1	PCIe#0_0	SATA
3	2	3x1 ‡	1	PCIe#2_0	USB_SS#1	PCIe#1_0	USB_SS#2	PCIe#0_0	SATA
4	3	2x1 ‡	1	USB_SS#0	USB_SS#1	PCIe#1_0	USB_SS#2	PCIe#0_0	SATA
5	1	2x1 + 1x2	1	PCIe#2_0	USB_SS#1	PCIe#1_0	PCIe#0_1	PCIe#0_0	SATA
6	2	1x1 + 1x2	1	USB_SS#0	USB_SS#1	PCIe#1_0	PCIe#0_1	PCIe#0_0	SATA

* Configuration 2 represents the default on the carrier board. Lane 0 is assigned to USB SuperSpeed port 0, Lanes 1 through 4 are assigned to PCIe, and Lane 5 is assigned to SATA.

† PCIe Interface 2 can be brought to the PEX1 pins, or USB 3.0 port 1 to the USB_SS0 pins, depending on the setting of a multiplexor on the module. The selection is controlled by QSPI_IO2 configured as a GPIO. To switch USB_SS0 to PEX1, configure QSPI_I02 as follows:

```
pcie0_lane2_mux {
    gpio-hog;
    gpios = <TEGRA_MAIN_GPIO(R, 3) 0>;
    output-low;
    label = "pcie-lane2-mux";
-    status = "disabled";
+    status = "okay";
};
```

‡ Although PCIe is 3x1 and 2x1 for configurations 3 and 4, the actual setting in the device tree must be x2, x1, x1. Otherwise `pcie@1,0` does not work.

```
pci@1,0 {
    nvidia,num-lanes = <2>;
    status = "okay";
}
```

Jetson TX2 and the released software support all of the configurations described in Table 1. However, the device tree and ODMDATA must be set to support the configuration you want to use. For further information, see the *NVIDIA Jetson TX2 Technical Reference Manual* (TRM). Consult with NVIDIA before designing your custom board.

Lane assignment can be defined by the `uphy` node in the `bpmp-dtb` file or by ODMDATA, defined in `p2771-0000.conf.common`. If both sources define lane assignment, the assignments in ODMDATA take priority.

If a customer device requires custom UPHY lane assignments, NVIDIA recommends defining them through ODMDATA because it can set related properties, such as MUX function properties, at the same time. You may prefer to perform lane assignment by

modifying the `bpmp-dtb` file if you are thoroughly familiar with UPHY and UPHY lane assignment. Consult NVIDIA for further assistance if you are considering this.

bpmp-dtb

BPMP (**Boot and Power Management Processor**) is a Jetson TX2 processor that handles the boot process and offloads power management, clock management, and reset control tasks from the CPU. UPHY lane assignment is configured in the `bpmp-dtb` file under the device node `uphy`.

```
/ {
    uphy {
        lane0-owner = <TEGRA186_UPHY_LANE_XUSB>;
        lane1-owner = <TEGRA186_UPHY_LANE_PCIE>;
        lane2-owner = <TEGRA186_UPHY_LANE_PCIE>;
        lane4-owner = <TEGRA186_UPHY_LANE_PCIE>;
        lane5-owner = <TEGRA186_UPHY_LANE_SATA>;
        };
};
```

ODMDATA and Plugin Manager

ODMDATA and Plugin Manager support special properties of various products' device trees. While loading the BPMP firmware (BPMP-FW), Bootloader gets ODMDATA, checks the ODMDATA UPHY lane configuration bit, and updates the UPHY lane owners on `bpmp-dtb`. Later, BPMP-FW configures the UPHY lanes as defined by the updated DTB. This provides flexibility to maintain multiple board configurations during development

Table 2 shows the meanings of the ODMDATA bits that are related to UPHY lane assignment.

Table 2. ODMDATA bis for UPHY lane assignment

Bits	Name	Description
31:29	—	Reserved
28	UPHY_LANE5	Each lane's bits identify the function that owns the lane. Recognized values are: ▶ 0: PCIE ▶ 1: XUSB
27	UPHY_LANE4	
26	UPHY_LANE2	
25	UPHY_LANE1	
24	UPHY_LANE0	
23:0	—	Reserved

For example:

- ODMDATA=0x1090000 while flashing for Jetson TX2 for configuration 2
- ODMDATA=0x90000 for configuration 1
- ODMDATA=0x6090000 for configuration 3

Required Device Tree Changes

This section gives step-by-step guidance for checking schematics and configuring USB ports in the device tree. All the examples are based on the design of Jetson TX2 P2597 carrier board.

For a Host-Only Port

This section uses J19, a USB 3.0 type A connector, as an example of a host-only port.

Go Through the Schematics

Note: **The P2597 carrier board's schematic file, `P2597_C02_Concept_schematics.pdf`, is included in the Jetson TX1-TX2 Developer Kit Carrier Board Design Files, available at:**

http://developer.nvidia.com/embedded/dlc/jetson-tx1-tx2-developer-kit-carrier-board-c02-design-files

Check the USB connectors on the P2597 carrier board and find the wired socket location to the P3310 module board.

- USB2.0 signal pins D+/D- (DP and DN) wire out from J19 and lead to A39 (USB1_D) and A38 (USB1_D) on the CVM socket.

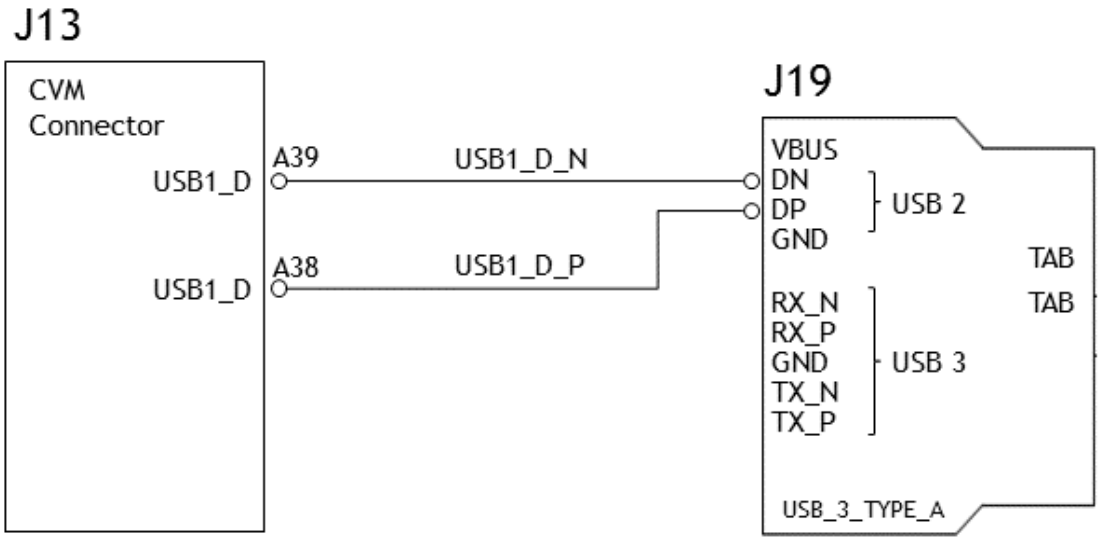

Figure 2. USB 2.0 pins

USB3.0 differential signal pairs (TX_* and RX_*) wire out from J19 and lead to C43 (USB_SS0_TX), C44 (USB_SS0_TX), F43 (USB_SS0_RX), and F44 (USB_SS0_RX) on the CVM socket.

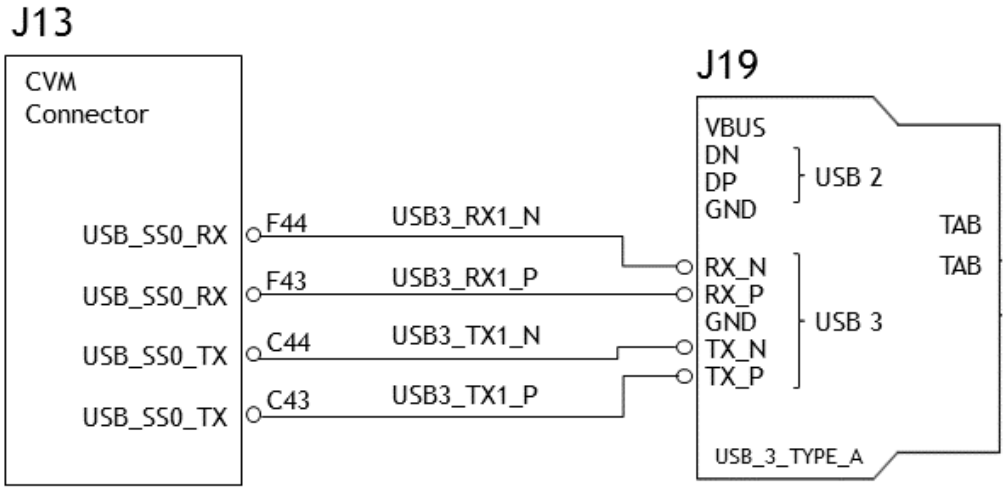

Figure 3. USB 3.0 pins

- VBUS is enabled by USB_VBUS_EN1 which lead to A18 (USB1_EN_OC) on the CVM socket through U21, the USB power-distribution switch.

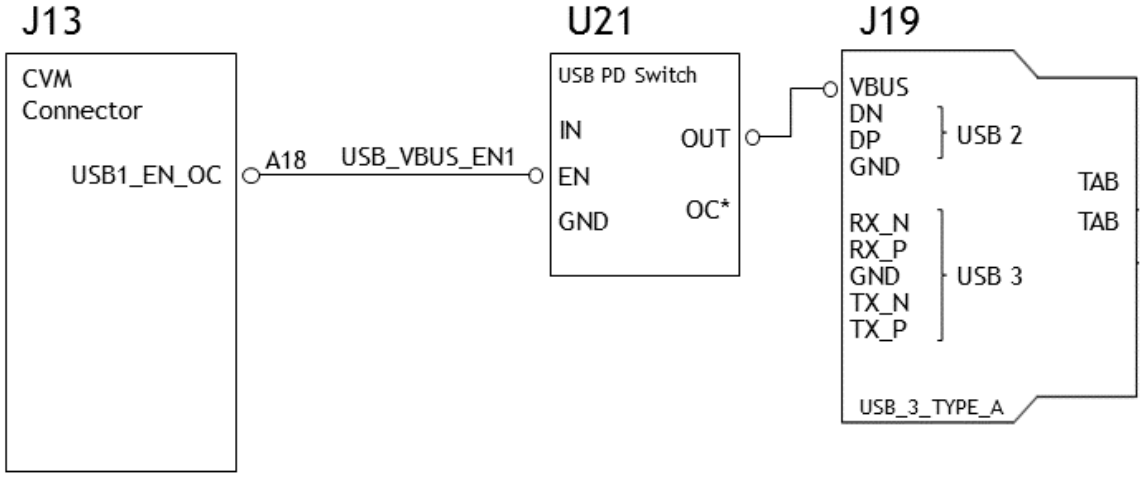

Figure 4. USB enable pin

Through the schematic, we can conclude that for J19:

- The USB2.0 signal pair is wired to UTMI pad 1 (USB2 port 1).
- The USB3.0 signal pairs are wired to UPHY lane 0 (USB3.0 port 0 according to UPHY lane mapping).
- The VBUS supply is controlled by USB1_EN_OC pin.

Under the Fixed-Regulators Node

The VBUS supply of the host driver (xHCI) is controlled with regulators. The device tree's `fixed-regulators` node follows the conventions of the `regulator.txt`, `fixed-regulator.txt`, and `gpio.txt` kernel documents. The node's properties are:

- `compatible`: If a regulator chip contains multiple regulators, and if the chip's binding contains a child node that describes each regulator, then this property indicates which regulator this child node configures. If this property is absent, the node's name is used instead.

  Note: **The VBUS supply of J19 is controlled by the USB1_EN_OC GPIO pin. Hence `compatible` should be `regulator-fixed-sync` for `fixed-regulator`.**

- `regulator-name`: A string used as a descriptive name for regulator outputs.
- `regulator-min-microvolt`: Smallest voltage consumers may set, in microvolts. Must be `"5000000"`.
- `regulator-max-microvolt`: Largest voltage consumers may set, in microvolts. Must be `"5000000"`.
- `gpio`: The GPIO which enables and disables the regulator.

Take J19 USB3.0 type A connector for example. Create a `pad` and `port` nodes and property list for J19 based on the device tree structure described above:

```
fixed-regulators {
    ...
    vdd_usb1_5v: regulator@5 {
        compatible = "regulator-fixed-sync";
        reg = <5>;
        regulator-name = "vdd-usb1-5v";
        regulator-min-microvolt = <5000000>;
        regulator-max-microvolt = <5000000>;
        gpio = <&tegra_main_gpio TEGRA_MAIN_GPIO(L, 5) 0>;
        gpio-open-drain;
        enable-active-high;
        ...
    };
};
```

Note: **Check the pinmux table for the GPIO that corresponds to the USB1_EN_OC pin.**

Create the xusb_padctl Node

The device tree's `xusb_padctl` node follows the conventions of the `pinctrl-bindings.txt` kernel document. It contains two groups named `pads` and `ports`, which describe USB2 and USB3 signals along with parameters and port numbers. The name of each parameter description subnode in `pads` and `ports` must be in the form

<type>-<port_number>, where <type> is "usb2" or "usb3" and <port_number> is the associated port number.

The pads Subnode

The single property defined in the pads subnode is:

- • nvidia,function: A string containing the name of the function to mux to the pin or group. Must be "xusb".

The ports Subnode

The properties defined in the ports subnode are:

- mode: A string that describes USB port capability. A port for USB2 must have this property. It must be one of the values "host", "device", or "otg".
- nvidia,usb2-companion: Specifies the USB2 port (0, 1, 2, or 3) to which the port is mapped. A port for USB3 must have this property.
- nvidia,oc-pin: Specifies the overcurrent VBUS pin the port is using. The value must be positive or zero.
- vbus-supply: The VBUS regulator for the corresponding UTMI pad. Set to "&battery_reg" for a dummy regulator.

For the detailed information about xusb_padctl, refer to the documentation at:

```
kernel/kernel-
4.9/Documentation/devicetree/bindings/pinctrl/nvidia,tegra186-
padctl.txt
```

Take J19 (a USB3.0 type A connector) as an example. Create pad and port nodes and property lists for J19 based on the device tree structure described above:

```
xusb_padctl: xusb_padctl@3520000 {
    ...
    pads {
        usb2 {
            lanes {
                usb2-1 {
                    nvidia,function = "xusb";
                    status = "okay";
                };
                ...
            };
        };
        usb3 {
            lanes {
                ...
                usb3-0 {
                    nvidia,function = "xusb";
```

```
                    status = "okay";
                };
                ...
            };
        };
    };
    ports {
        usb2-1 {
            mode = "host";
            vbus-supply = <&vdd_usb0_5v>;
            status = "okay";
            nvidia,oc-pin = <1>;
        };
        ...
        usb3-0 {
            nvidia,usb2-companion = <1>;
            status = "okay";
        };
        ...
    };
};
```

Create the xHCI Node

The Jetson TX2 xHCI controller complies with xHCI specifications, which support both USB 2.0 HighSpeed/FullSpeed/LowSpeed and USB 3.0 SuperSpeed protocols. The controller node's properties are:

- `phys`: Must contain an entry for each entry in `phy-names`.
- `phy-names`: Must contain an entry for each PHY used by the controller. Names must be of the form `<type>-<port_number>`, where <type> is `"usb2"` or `"usb3"`.
- `nvidia,boost_cpu_freq`: Specifies the value to which CPU frequency is boosted. This is only the minimum frequency; DVFS can scale up CPU frequency further based on need and CPU loading. CPU boost frequency through PMQOS is enabled for the xHCI controller only when this field's value is greater than zero. The boost is applicable only for bulk and ISOC transfers; other endpoints do not need to be boosted.
- `nvidia,boost_cpu_trigger`: Minimum buffer length of the bulk or ISOC transfers beyond which to boost frequency.
- `nvidia,xusb-padctl`: A pointer to the `xusb-padctl` node.

For the detailed information about xHCI, refer to the documentation at:

```
kernel/kernel-
4.9/Documentation/devicetree/bindings/pinctrl/nvidia,tegra186-xhci.txt
```

Take J19, a USB3.0 type A connector, as an example. Create an xHCI node and property list for J19 based on the device tree structure described above:

```
xhci@3530000 {
    ...
    phys = <&{/xusb_padctl@3520000/pads/usb2/lanes/usb2-1}>,
           <&{/xusb_padctl@3520000/pads/usb3/lanes/usb3-0}>;
    phy-names = "usb2-1", "usb3-0";
    nvidia,xusb-padctl = <&xusb_padctl>;
    status = "okay";
    ...
};
```

For an OTG (On-The-GO) Port

USB On-The-Go, often abbreviated USB OTG or just OTG, is a specification that allows USB to act as a host or a device on the same port. A USB OTG port can switch back and forth between the roles of host and device.

An OTG port adds a fifth pin to the standard USB connector, called the **ID pin**. An OTG cable has a type A plug on one end and a type B plug on the other end. The type A plug's ID pin is grounded, while the type B plug's ID pin is floating. A device with a type A plug inserted becomes and OTG type A device (a host), and a device with a type B plug inserted becomes a type B device (a device).

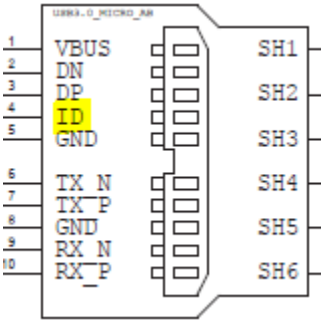

Figure 5. An OTG port connector

Go Through the Schematics

Note: **The P2597 carrier board's schematic file, `P2597_C02_Concept_schematics.pdf`, is included in Jetson TX2 Developer Kit Carrier Board Design Files, available at:**

http://developer.nvidia.com/embedded/dlc/jetson-tx1-tx2-developer-kit-carrier-board-c02-design-files

Check the USB connectors on the P2597 carrier board and find the wired socket location to the P3310 module board.

- USB2.0 signal pins D+/D− (DP and DN) wire out from J28 and lead to B40 (USB0_D) and B39 (USB0_D) on the CVM socket.

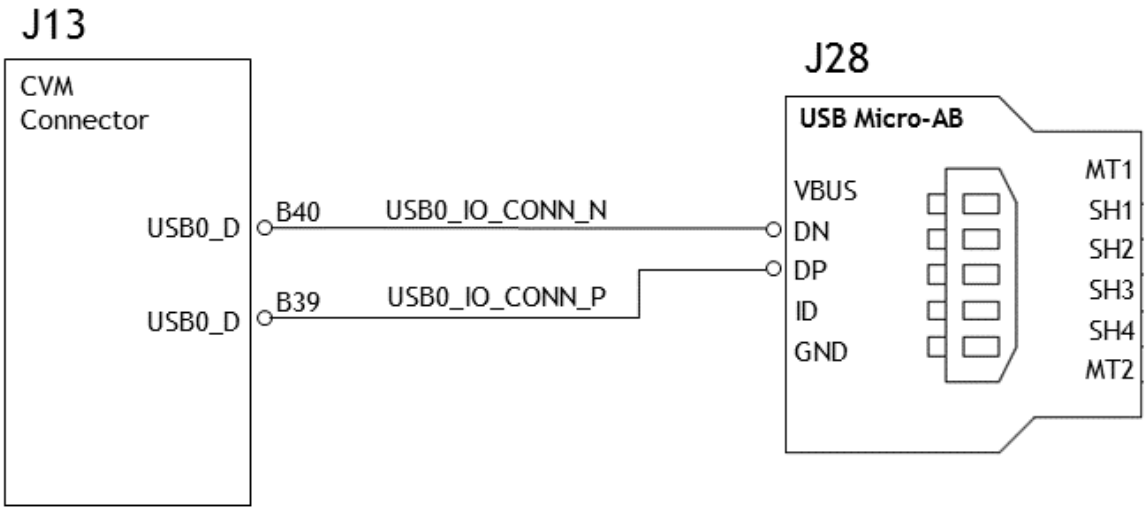

Figure 6. USB OTG signal pins

Note: **The OTG port, J28, only supports USB2.0 and hence does *not* have USB 3.0 differential signal pairs on connector.**

- VBUS is enabled by USB_VBUS_EN0 which leads to A17 (USB0_EN_OC) on the CVM socket through U25, the USB power distribution switch.
- VBUS also wires out from J28 and leads to B37 (USB0_VBUS_DET) on the CVM socket as the VBUS_DETECT pin.
- The ID pin wires out from J28 and leads to A36 (USB0_OTG_ID) on the CVM socket.

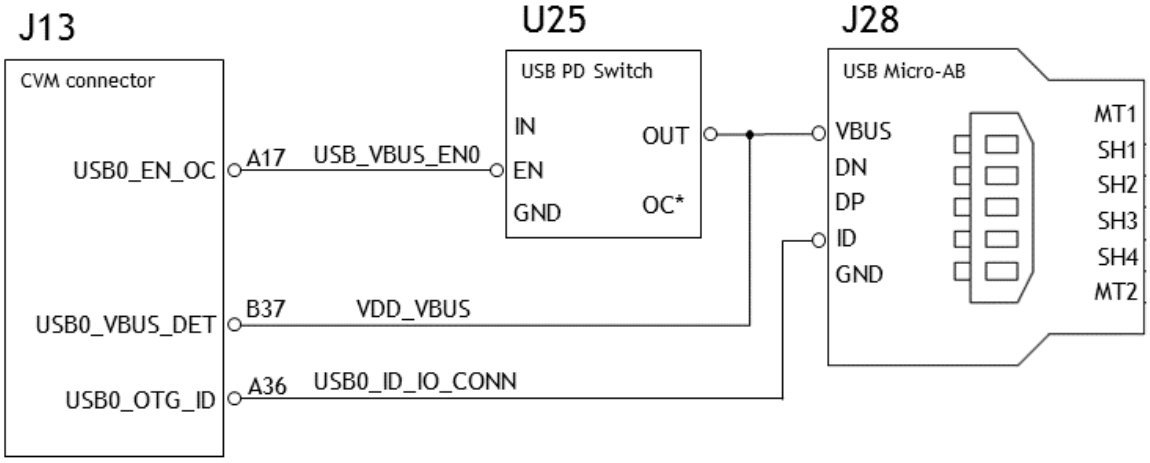

Figure 7. USB OTG ID pins

Through the schematic, we can conclude that for J28:

- The USB 2.0 signal pair is wired to UTMI pad 0 (USB2 port 0).
- The VBUS supply is controlled by USB0_EN_OC pin.
- The VBUS pin is also wired to the USB0_VBUS_DET pin as the VBUS_DETECT pin.
- The ID pin is wired to the USB0_OTG_ID pin

Under the fixed-regulators Node

The `fixed-regulators` settings for an OTG port are the same as for a host-only port.

Take J28 USB2.0 micro B connector as example. Create a `fixed-regulators` node and property list for J19 based on the device tree structure described above:

```
fixed-regulators {
    ...
    vdd_usb0_5v: regulator@4 {
        compatible = "regulator-fixed-sync";
        reg = <4>;
        regulator-name = "vdd-usb0-5v";
        regulator-min-microvolt = <5000000>;
        regulator-max-microvolt = <5000000>;
        gpio = <&tegra_main_gpio TEGRA_MAIN_GPIO(L, 4) 0>;
        gpio-open-drain;
        enable-active-high;
        ...
    };
};
```

Note: **Check the pinmux table for the GPIO that corresponds to the USB0_EN_OC pin.**

Under the xusb_padctl Node

The `xusb_padctl` settings for an OTG port are the same as for a host-only port except that the mode should be `"otg"`.

Taking J28, the USB2.0 micro-B connector, as an example, create `pad` and `port` nodes and property lists:

```
xusb_padctl: xusb_padctl@3520000 {
    ...
    pads {
        usb2 {
            lanes {
                usb2-0 {
                    nvidia,function = "xusb";
                    status = "okay";
                };
                ...
```

```
            };
        };
                ...
    };
    ports {
        usb2-0 {
            mode = "otg";
            vbus-supply = <&vdd_usb0_5v>;
            status = "okay";
        };
        ...
    };
};
```

Under the extcon Node

External connectors, which may have different types of cables attached (USB, TA, HDMI, analog A/V, and others), often have device drivers that detect state changes at the port, and separate device drivers that do something according to the state changes.

The **External Connector Class** (`extcon`), introduced in 2012, supports the use of a notifier for passing information such as state changes between device drivers.

Generally, port switching between the roles of an OTG port is controlled by the host driver (xHCI) and device driver (xUDC), and can be defined by the state of the ID pin and the VBUS_DETECT pin.

Taking J28, the USB2.0 micro B connector, as an example, the USB0_VBUS_DET GPIO pin acts as the VBUS_DETECT pin and USB0_OTG_ID as the ID pin, for example:

1. Find the corresponding GPIO states on the VBUS_DETECT pin and ID pin.

   Generally, the ID pin is designed as internal pull high (logical high). With a type A plug connected the ID pin is pulled to ground (logical low), while with a type B plug connected or no cable connected it remains logical high.

   The operation of the VBUS_DETECT pin depends on the device's design. Consider the schematic of USB 2.0 micro B connector J28, for example.

   VBUS_DETECT with a type B plug connected is logical low, because VBUS is provided from an external power supply. When no cable is connected it is logical high.

   Note: **VBUS_DETECT is initially logical high, then logical low because VBUS is provided by the host controller. Therefore, the state of the VBUS_DETECT pin does not matter when the OTG port is operating in host mode.**

2. Create the table of GPIO states and their corresponding output cable states:

USB0_OTG_ID	USB0_VBUS_DET	EXTCON_STATE
1	1	0x0 (EXCON_NONE)
0	0	0x2 (EXTCON_USB_HOST)
0	1	0x2 (EXTCON_USB_HOST)
1	0	0x1 (EXTCON_USB)

Under the xHCI Node

The xHCI settings for an OTG port are the same as for a host-only port except for the addition of `extcon` settings:

- `extcon-cables`: OTG support. Must contain a pointer to the `excon-cable` entry for the USB ID pin. When the `extcon` cable state is 0, the OTG port transitions to host mode.
- `extcon-cable-names`: Must be `"id"`.
- `#extcon-cells`: Number of cells in the `extcon` specifier. Must be 1.

Taking J28, the USB2.0 micro-B connector, as an example, create an xHCI node and property list based on the device tree structure described in Create the xHCI Node for a host-only port, plus the extcon settings above:

```
xhci@3530000 {
    ...
    extcon-cables = <&vbus_id_extcon 1>;
    extcon-cable-names = "id";
    #extcon-cells = <1>;
    phys = <&{/xusb_padctl@3520000/pads/usb2/lanes/usb2-0}>;
    phy-names = "usb2-0";
    nvidia,xusb-padctl = <&xusb_padctl>;
    status = "okay";
    ...
};
```

Under the xUDC Node

The Jetson TX2 xUDC controller supports both USB 2.0 HighSpeed/FullSpeed and USB 3.0 SuperSpeed protocols.

- `extcon-cables`: OTG support. Must contains an `excon-cable` entry which detects USB VBUS pin. When the `extcon` cable state is 1, OTG port transitions to device mode.
- `extcon-cable-names`: Must be `"vbus"`.
- `charger-detector`: USB charger detection support. Must be the phandle of the USB charger detection driver DT node.
- `phys`: An array; must contain pointers to the nodes that define each PHY in `phy-names`.

- `phy-names`: An array; must contain entries for each PHY used by the controller. Names must be in the form `<type>-<port_number>`, where `<type>` is one of `"usb2"` or `"usb3"`.
- `nvidia,boost_cpu_freq`: The value to which CPU frequency is to be boosted. This is only the minimum frequency; DVFS can scale up CPU frequency further based on need and CPU load. CPU boost frequency through PMQOS is enabled for the xUDC controller only when this field's value is greater than zero. The boost is applicable only to large bulk transfers on bulk endpoints; other endpoints do not need to be boosted.
- `nvidia,xusb-padctl`: Must be a pointer to the `xusb-padctl` node.

For the detailed information about xUDC, refer to the documentation at:

```
kernel/kernel-
4.9/Documentation/devicetree/bindings/pinctrl/nvidia,tegra186-xudc.txt
```

Taking J28, the USB2.0 micro B connector, as an example, create an xUDC node and property list for J28 based on the device tree structure described above:

```
xudc@3550000 {
    extcon-cables = <&vbus_id_extcon 0>;
    extcon-cable-names = "vbus";
    #extcon-cells = <1>;
    phys = <&{/xusb_padctl@3520000/pads/usb2/lanes/usb2-0}>;
    phy-names = "usb2";
    nvidia,xusb-padctl = <&xusb_padctl>;
    nvidia,boost_cpu_freq = <1200>;
    status = "okay";
};
```

USB Lane Mapping Issues

If you suspect a UPHY lane mapping issue, check the lane assignments programmed by BPMB firmware, based on ODMDATA:

3. UPHY lane 0: `./devmem2 0x02520284`
4. UPHY lane 1: `./devmem2 0x02530284`
5. UPHY lane 2: `./devmem2 0x02540284`
6. UPHY lane 3: `./devmem2 0x02550284`
7. UPHY lane 4: `./devmem2 0x02560284`
8. UPHY lane 5: `./devmem2 0x02570284`

Bits 0–2 identify the function that owns the lane:

- 0x00: XUSB
- 0x01: PCIe

- 0x02: SATA

If a target UPHY lane is not owned by the correct function, check the value of ODMDATA that was flashed to be sure that the target lane was assigned correctly.

Check the device tree values used at runtime to ensure that Plugin Manager did not override them unexpectedly.

For example, confirm that the proper properties are enabled by running the command:

```
ls /proc/device-tree/chosen/plugin-manager/odm-data/
```

For a custom board, configure ODMDATA properly and check all the values. This example shows the values under listed from `/proc/device-tree/chosen/plugin-manager/odm-data/`, which represent the properties generated from ODMDATA, for a Jetson TX2 P2597 carrier board:

```
android-build          enable-denver-wdt          enable-xusb-on-uphy-
lane0
disable-pmic-wdt       enable-pcie-on-uphy-lane1  name
disable-sdmmc-hwcq     enable-pcie-on-uphy-lane2  no-battery
disable-tegra-wdt      enable-pcie-on-uphy-lane4  normal-flashed
enable-debug-console   enable-sata-on-uphy-lane5
```

Note: **Before designing your custom board, verify the lane mapping by consulting the *Jetson TX2 OEM Product Design Guide*, available at:**

https://developer.nvidia.com/embedded/dlc/jetson-tx2-series-modules-oem-product-design-guide

The Flashing the Build Image

When flashing the build image, use your specific board name. The flashing script uses the configuration present in the `<board>.conf` file during the flashing process.

Setting Optional Environment Variables

The `flash.sh` script updates the following environment variables based on board EEPROM values and other parameters. If you want to override these environment variables' default values, set them in the board-specific file `board.conf`.

```
# Optional Environment Variables:
# BCTFILE ---------------- Boot control table configuration file to be used.
# BOARDID ---------------- Pass boardid to override EEPROM value
# BOARDREV --------------- Pass board_revision to override EEPROM value
# BOARDSKU --------------- Pass board_sku to override EEPROM value
# BOOTLOADER ------------- Bootloader binary to be flashed
# BOOTPARTLIMIT ---------- GPT data limit. (== Max BCT size + PPT size)
```

```
# BOOTPARTSIZE ----------- Total eMMC HW boot partition size.
# CFGFILE ---------------- Partition table configuration file to be used.
# CMDLINE ---------------- Target cmdline. See help for more information.
# DEVSECTSIZE ------------ Device Sector size. (default = 512Byte).
# DTBFILE ---------------- Device Tree file to be used.
# EMMCSIZE --------------- Size of target device eMMC (boot0+boot1+user).
# FLASHAPP --------------- Flash application running in host machine.
# FLASHER ---------------- Flash server running in target machine.
# INITRD ----------------- Initrd image file to be flashed.
# KERNEL_IMAGE ----------- Linux kernel zImage file to be flashed.
# MTS -------------------- MTS file name such as mts_si.
# MTSPREBOOT ------------- MTS preboot file name such as mts_preboot_si.
# NFSARGS ---------------- Static Network assignments.
#                          <C-ipa>:<S-ipa>:<G-ipa>:<netmask>
# NFSROOT ---------------- NFSROOT i.e. <my IP addr>:/exported/rootfs_dir.
# ODMDATA ---------------- Odmdata to be used.
# PKCKEY ----------------- RSA key file to used to sign bootloader images.
# ROOTFSSIZE ------------- Linux RootFS size (internal emmc/nand only).
# ROOTFS_DIR ------------- Linux RootFS directory name.
# SBKKEY ----------------- SBK key file to used to encrypt bootloader images.
# SCEFILE ---------------- SCE firmware file such as camera-rtcpu-sce.img.
# SPEFILE ---------------- SPE firmware file path such as bootloader/spe.bin.
# FAB -------------------- Target board's FAB ID.
# TEGRABOOT -------------- lowerlayer bootloader such as nvtboot.bin.
# WB0BOOT ---------------- Warmboot code such as nvtbootwb0.bin
```

Note: **All the parameters must be added below the reference to the file `<xxx>.conf.common` to be reflected in the flashed image.**

Here is an example of environment variable settings:

```
source "${LDK_DIR}/p2771-0000.conf.common";
PINMUX_CONFIG="tegra186-mb1-bct-pinmux-quill-p3310-1000-a00.cfg";
BPFDTB_FILE=tegra186-a02-bpmp-quill-p3310-1000-a00-00-te770d-ucm2.dtb;
DTB_FILE=tegra186-quill-p3310-1000-a00-00-edp.dtb;
TBCDTB_FILE=tegra186-quill-p3310-1000-a00-00-edp.dtb;
EMMC_BCT="P3310_A00_8GB_Samsung_8GB_lpddr4_204Mhz_A02.cfg";
MISC_COLD_BOOT_CONFIG="tegra186-mb1-bct-misc-si.cfg";
```

To flash the build image

- Execute the following command.

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

Hardware Bring-Up Checklist

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

Before Power-On

Make sure that the Jetson TX2 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_PWR_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_REQ entering and exiting and Deep Sleep (SC7). Ensure that CPU_PWR_REQ and associated power rail sequence meets Tegra Data Sheet requirements.	☐
Verify that all rails which must be OFF in Deep Sleep (SC7) are OFF.	☐
Verify that all rails which must be ON in Deep Sleep (SC7) are ON.	☐
Verify that required rails are back and at correct voltage under hardware control exiting Deep Sleep (SC7).	☐

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 (SC7).	☐
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 (SC7).	☐
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.	☐

PEX (Optional)

Verify proper connectivity by checking lanes.	☐

Verify that any implemented PEX interfaces transition to the lowest power state in Deep Sleep (SC7).	☐
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 (SC7).	☐
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 (SC7).	☐
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 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

If your replaced the SDRAM MB1 BCT with a new DDR, verify it.	☐
If you replaced the baseboard, verify the PMIC and pinmux configuration.	☐
If you replaced the eMMC, verify its operation.	☐
Obtain board schematics and component data sheets.	☐
Verify power tree and modify device tree, MB1 PMIC configuration accordingly, for the base board.	☐
Review board pinmux and modify MB1 pinmux and PAD configuration, accordingly.	☐

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	☐
SC7 (optional)	☐
CPU power down	☐
BCT, Full-speed	☐

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 NON-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, the NVIDIA logo, Tegra, and Jetson 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

© 2017-2019 NVIDIA Corporation. All rights reserved.