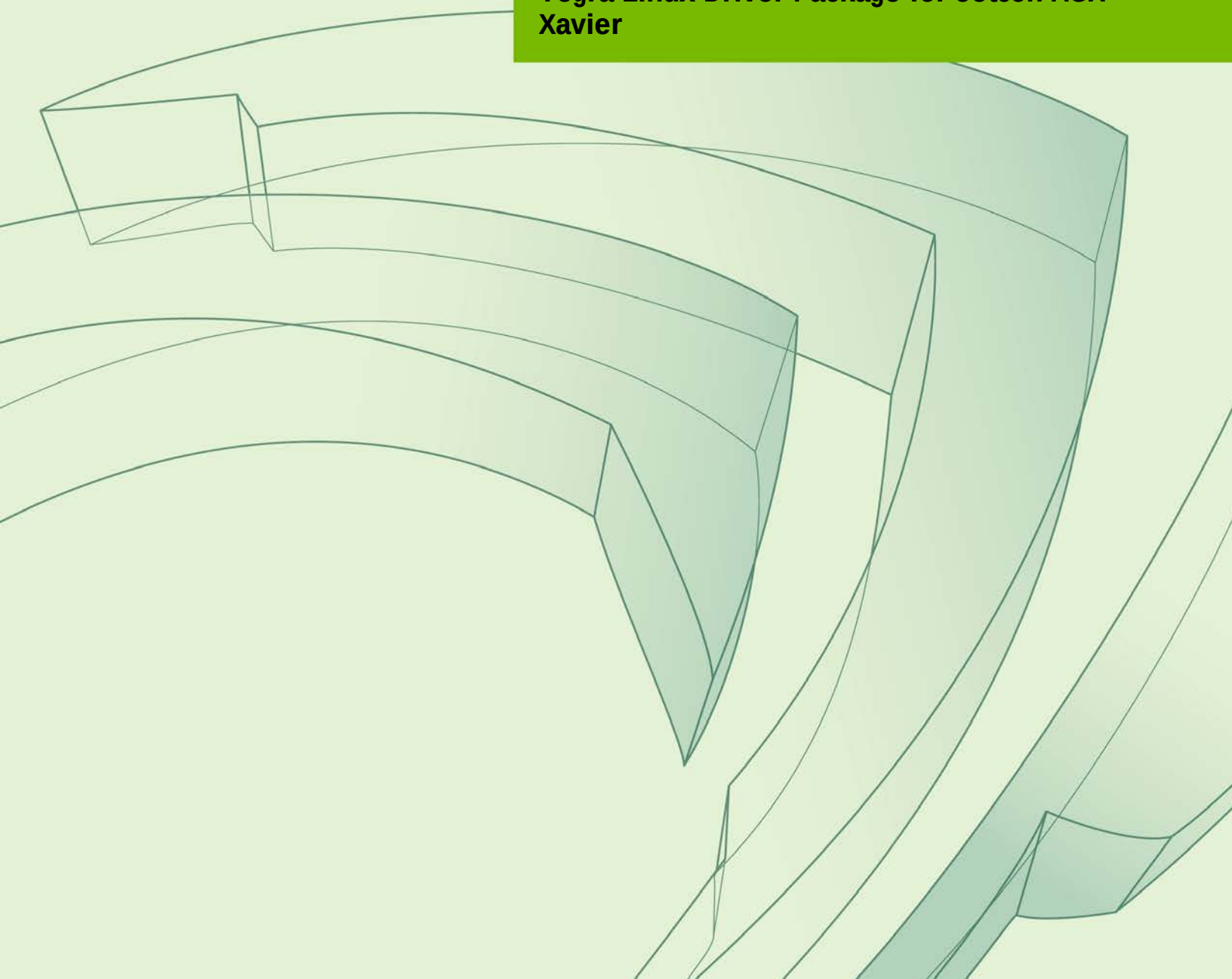
JETSON AGX XAVIER PLATFORM ADAPTATION AND BRING-UP GUIDE

DA_09237-003 | May 2, 2019

Tegra Linux Driver Package for Jetson AGX Xavier



Document Change History

DA_09237-003

Version	Date	Authors	Description of Change
v1.0	2 Nov 2018	bbasu/jsachs	Initial release for Jetson AGX Xavier.
v1.1	22 Jan 2019	wwang/jsachs	Remove a section that describes TX2 and must be updated to describe AGX Xavier.
v1.2	2 May 2019	wwang/jsachs	Updates for L4T r32.1. 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 (L4T) from NVIDIA® Jetson AGX Xavier™ Developer Kit to other hardware platforms.

The examples described include code for the Jetson AGX Xavier Developer Kit (P2972).

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

Board Configuration

The Jetson AGX Xavier Developer Kit consists of a P2888 System on Module (SOM) connected to a P2822 carrier board. The number P2972 refers to the complete Jetson AGX Xavier Developer Kit. Both SOM and carrier board have an EEPROM where the board ID is saved. The P2888 SOM can be used without any software configuration modifications.

The P2888 SOM sold for incorporation into customer products has a Thermal Transfer Plate (TTP) ready to accept a customer-provided thermal solution. The module shipped as part of the Developer Kit has no TTP; instead it has a thermal solution designed specifically for the Developer Kit. This thermal solution must not be removed from the module.

Before using the P2888 SOM with a carrier board other than P2822, change the kernel device tree, MB1 configuration, ODM data, and flashing configuration to include configuration for the new carrier board instead of for P2822. 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-xavier
p2972-0000-devkit
myboard
```

The name you select appears in:

- Filenames and pathnames
- User-visible device tree filenames

Additionally, this name is exposed to the user through 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 AGX Xavier 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 AGX Xavier users whether the modified source code supports the original Jetson AGX Xavier 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, `P2972` is the name of the Jetson AGX Xavier 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`.

Camera Connector Pin Differences

This table describes camera connector pin differences between the Jetson AGX Xavier module and the earlier Jetson TX1 and Jetson TX2 modules.

In summary, the Jetson AGX Xavier module:

- Adds four additional CSI lanes and places CSI6 where CSI5 was. CSI5 moves to where UART and DMIC were
- Removes the 1.2V and 5V rails (or changes 5V to 3.3V)
- Removes UART, SPI, DMIC, and I2S
- Removes Flash, Auto-Focus, and Strobe Control
- Removes Motion Int and Modem to AP Ready

Pin	TX1/TX2	Xavier	Notes
1	CSI0	CSI0	Equivalent
2	CSI1	CSI1	Equivalent
3	CSI0	CSI0	Equivalent
4	CSI1	CSI1	Equivalent
5	GND	GND	Equivalent
6	GND	GND	Equivalent
7	CSI0	CSI0	Equivalent
8	CSI1	CSI1	Equivalent
9	CSI0	CSI0	Equivalent
10	CSI1	CSI1	Equivalent
11	GND	GND	Equivalent
12	GND	GND	Equivalent
13	CSI0	CSI0	Equivalent
14	CSI1	CSI1	Equivalent
15	CSI0	CSI0	Equivalent
16	CSI1	CSI1	Equivalent
17	GND	GND	Equivalent
18	GND	GND	Equivalent
19	CSI2	CSI2	Equivalent
20	CSI3	CSI3	Equivalent

Pin	TX1/TX2	Xavier	Notes
21	CSI2	CSI2	Equivalent
22	CSI3	CSI3	Equivalent
23	GND	GND	Equivalent
24	GND	GND	Equivalent
25	CSI2	CSI2	Equivalent
26	CSI3	CSI3	Equivalent
27	CSI2	CSI2	Equivalent
28	CSI3	CSI3	Equivalent
29	GND	GND	Equivalent
30	GND	GND	Equivalent
31	CSI2	CSI2	Equivalent
32	CSI3	CSI3	Equivalent
33	CSI2	CSI2	Equivalent
34	CSI3	CSI3	Equivalent
35	GND	GND	Equivalent
36	GND	GND	Equivalent
37	CSI4	CSI4	Equivalent
38	CSI5	CSI6	Different CSI lane
39	CSI4	CSI4	Equivalent
40	CSI5	CSI6	Different CSI lane
41	GND	GND	Equivalent
42	GND	GND	Equivalent
43	CSI4	CSI4	Equivalent
44	CSI5	CSI6	Different CSI lane
45	CSI4	CSI4	Equivalent
46	CSI5	CSI6	Different CSI lane
47	GND	GND	Equivalent
48	GND	GND	Equivalent
49	CSI4	CSI4	Equivalent
50	CSI5	CSI6	Different CSI lane
51	CSI4	CSI4	Equivalent
52	CSI5	CSI6	Different CSI lane
53	GND	GND	Equivalent
54	GND	GND	Equivalent
55	RSVD	RSVD	Equivalent

Pin	TX1/TX2	Xavier	Notes
56	RSVD	RSVD	Equivalent
57	RSVD	RSVD	Equivalent
58	RSVD	RSVD	Equivalent
59	UART Present	CSI5	CSI replaces UART Present
60	NC	CSI7	CSI replaces SPI
61	UART	CSI5	CSI replaces UART
62	SPI	CSI7	CSI replaces SPI
63	UART	GND	CSI replaces UART
64	SPI	GND	CSI replaces SPI
65	UART	CSI5	CSI replaces UART
66	SPI	CSI7	CSI replaces SPI
67	UART	CSI5	CSI replaces UART
68	SPI	CSI7	CSI replaces SPI
69	GND	GND	Equivalent
70	GND	GND	Equivalent
71	DMIC	CSI5	CSI replaces DMIS
72	I2S	CSI7	CSI replaces I2S
73	DMIC	CSI5	CSI replaces DMIS
74	I2S	CSI7	CSI replaces I2S
75	CAM_I2C	I2C_GP3 (CAM_I2C)	Equivalent
76	I2S	NC	I2S removed
77	CAM_I2C	I2C_GP3 (CAM_I2C)	Equivalent
78	I2S	NC	I2S removed
79	GND	GND	Equivalent
80	GND	GND	Equivalent
81	2.8V (AVDD_CAM)	2.8V (AVDD_CAM)	Equivalent
82	2.8V (AVDD_CAM)	2.8V (AVDD_CAM)	Equivalent
83	2.8V (AVDD_CAM)	2.8V (AVDD_CAM)	Equivalent
84	3.3V (VDD_3V3_SLP)	NC	Functionality removed
85	CAM_AF_PWDN	NC	Functionality removed
86	CAM_VSYNC	NC	Functionality removed
87	I2C_PM	I2C_GP2	Equivalent
88	CAM1_MCLK	CAM1_MCLK03	Equivalent
89	I2C_PM	I2C_GP2	Equivalent
90	CAM1_PWDN1	GPIO15_CAM1_PWDN	Equivalent

Pin	TX1/TX2	Xavier	Notes
91	CAM0_MCLK	CAM0_MCLK02	Equivalent
92	CAM1_RST_L	GPIO16_CAM1_RST	Equivalent
93	CAM0_PWDN	CAM0_PWDN	Equivalent
94	CAM2_MCLK	CAM2_MCLK04	Equivalent
95	CAM0_RST_L	CAM0_RST	Equivalent
96	CAM2_PWDN	NC	CAM2 PWDN removed
97	FLASH_EN	NC	FLASH EN removed
98	CAM2_RST	NC	CAM2 RST removed
99	GND	GND	Equivalent
100	GND	GND	Equivalent
101	1.2V (DVDD_CAM_IO_1V2)	RSVD	1.2V removed
102	1.8V (DVDD_CAM_IO_1V8)	1.8V (VDD_1V8)	Equivalent
103	FLASH_INHIBIT	NC	Flash removed
104	TORCH_EN	NC	Torch removed
105	I2C_GP0	I2C_GP4	Equivalent
106	FLASH_STROBE	NC	Flash removed
107	I2C_GP0	I2C_GP4	Equivalent
108	3.3V (VDD_3V3_SLP)	3.3V (VDD_3V3)	Equivalent
109	5V	NC	5V removed
110	3.3V (VDD_3V3_SLP)	3.3V (VDD_3V3)	Equivalent
111	RSVD	NC	Equivalent
112	MOTION_INT_AP_L	NC	Motion Int removed
113	RSVD	NC	Equivalent
114	NC	NC	Equivalent
115	GND	GND	Equivalent
116	GND	GND	Equivalent
117	MDM2AP_READY	NC	Modem-AP Ready removed
118	5V (VDD_5V0_IO_SYS)	3.3V (VDD_3V3)	5V changed to 3.3V
119	VDD_SYS_EN	GPIO25_VDD_SYS_EN	Equivalent
120	5V (VDD_5V0_IO_SYS)	3.3V (VDD_3V3)	5V changed to 3.3V

Root Filesystem Configuration

Tegra Linux platforms can use any standard or customized Linux root filesystem (rootfs) that is appropriate for their targeted embedded applications.

However, certain settings must be configured in the rootfs's boot-up framework to set default configuration after boot, or some of the core functionalities will not run as expected.

For example:

1. The `nv.sh` and `nvfb.sh` boot-up scripts do some platform-specific configuration in the kernel.
2. The Xorg and X libraries must be correctly configured for the target device.
3. The nvpmodel clock and frequency must be configured for the target device.

These rootfs configurations and customizations are provided in this driver package in the directory and its subdirectories:

```
Linux_for_Tegra/nv_tegra/
```

You must incorporate the relevant customization for your target rootfs from this location.

Note: **For the sample Ubuntu root filesystem provided by NVIDIA, this customization is applied using the script `Linux_for_Tegra/apply_binaries.sh`.**

MB1 Configuration Changes

Multiple `.cfg` files define boot time configuration of the hardware. They are applied by 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 AGX Xavier Developer Kit board, you must change the pinmux configuration applied by the software.

The `Jetson-AGX-Xavier-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.

Once done, you must convert the `.dtsi` file generated by Excel to a `.cfg`. For instructions, see the README file at:

```
Linux_for_Tegra/kernel/pinmux/t19x/
```

You must perform the same conversion for `gpio.dtsi` and `padvoltage.dtsi`.

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_AGX_Xavier_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 Tegra 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 P2888 SOM. Some GPIO expander-based GPIO regulator settings in the P2822 carrier board configurations are also defined. Review this configuration file to replace any references to the P2822 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 P2822 carrier board. Additionally, update the number of blocks and array number for other entries of the block:

```
tegra194-mb1-bct-pmic-p2888-0001-a04-p2822-0000.cfg

# 5V0_HDMI_EN
pmic.system.block[0].type = 1; #I2C
pmic.system.block[0].controller-id = 4;
pmic.system.block[0].slave-add = 0x78; # 7BIt:0x3c
pmic.system.block[0].reg-data-size = 8;
pmic.system.block[0].reg-add-size = 8;
pmic.system.block[0].block-delay = 10;
pmic.system.block[0].commands[0].0x53.0x38 = 0x00; #SD4 FPS UP slot 0
pmic.system.block[0].commands[1].0x55.0x38 = 0x10; #GPIO2 FPS UP slot 2
pmic.system.block[0].commands[2].0x41.0x1C = 0x1C; #SLPEN=1, CLRSE = 11
```

Porting the Linux Kernel

It is assumed that you are using a P2888 SOM connected to a P2822 carrier board which have 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 P2888 SOM and P2822 carrier board. Consequently, based on the peripherals present on your carrier board, 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/t19x/
<top>/hardware_nvidia/soc/t19x
```

The final DTB file used is:

```
tegra194-p2888-001-p2822-0000.dtb
```

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	tegra194-power-tree-p2888-0001-p2822-1000.dtsi
Regulator parameter changes	tegra194-spmic-p2888-0001.dtsi
Display panel and node changes	For details, refer to the topic “Display Configuration and Bringup” in the *Tegra Linux Driver Package Development Guide*.
ODM data based feature configuration	tegra194- plugin-manager-p2888-0000.dtsi
NVIDIA SoC controller state to enable/disable a controller	soc/t19x/kernel-dts/tegra194-soc/
Panels related .dts files	platform/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 Galen 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 tegra194-p2888-0001-p2822-0000.dtb > tegra194-p2888-
0001-p2822-0000.dts
fdtdump dts tegra194-p2888-0001-p2822-0000.dtb > tegra194-p2888-0001-
p2822-0000.dts
```

PCIe Controller Configuration

The PCIe host controller is based on Synopsis Designware PCIe intellectual property, and thus inherits all the common properties defined in the information file at:

```
$(KERNEL_TOP)/Documentation/devicetree/bindings/pci/nvidia,tegra19x-pcie.txt
```

PCIe Controller Features

Jetson AGX Xavier has six PCIe controllers with these specifications:

- Speed: All controllers support up to Gen4 speed.
- Lane width:
  - C0, C5: up to x8
  - C4: up to x4
  - C1, C2, C3: x1
- Controllers C0, C4 and C5 support dual mode, that is, can be configured as endpoints.
- ASPM: All controllers support ASPM.

The Jetson AGX Xavier default PCIe configuration is:

- C5: x8
- C0: x4
- C1, C3: x1

These PCIe slots available on Jetson AGX Xavier:

- **M.2 Key M**: C0 controller operates in x4. Any M.2 Key M form factor NVMe cards can be connected.
- **eSATA controller**: C1 controller operates in x1. The eSATA port is available to connect any SATA drive.
- **M.2 Key E**: C3 controller operates in x1 mode. Any M.2 Key E form factor cards like Wi-Fi can be connected.
- **PCIe slot**: C5 controller operates in x8 mode. Any PCIe card can be connected. The PCIe slot is of x16 size to connect x16 card, but operates in x8 mode.

For information about Jetson AGX Xavier specific PCIe controller configuration, see the device tree documentation file at:

```
$(KERNEL_TOP)/Documentation/devicetree/bindings/pci/nvidia,tegra19x-pcie.txt
```

This file covers topics that include configuring maximum link speed and link width, and advertisement of different ASPM states.

To enable SMBus for PCIe slot

- In the file at:

```
$(TOP)/hardware/nvidia/platform/t19x/galen/kernel-
dts/common/tegra194-p2888-p2822-pcie-plugin-manager.dtsi
```

  Uncomment the following line:

```
/*&tegra_main_gpio TEGRA194_MAIN_GPIO(Y, 4) GPIO_ACTIVE_HIGH */ /*
I2C */
```

  Then flash a new DTB.

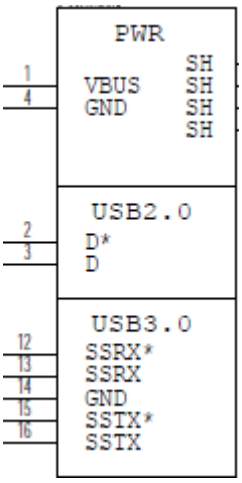
Porting USB (Universal Serial Bus)

Jetson AGX Xavier can support up to four enhanced 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 P2822 carrier board is designed and verified for three USB3.1 ports. If you designed your own carrier board, verify the UPHY lane mapping and compatibility between P2822 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



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 is a lane in UPHY which can support multiple types of interfaces.

The Jetson P2822 carrier board is designed and verified for three USB3.1 ports. The verified use cases and their UPHY lane assignments are shown in Table 1 and Table 2.

Table 1. Available outputs for the P2822 carrier board

Output type	Number of outputs
USB 3.1	3
PCIe	2 x1,1 X2, 1 X4
UFS	1 x1 UFS

Table 2. UPHY lane assignment use cases

Lane	Pin Names	Functions
0	UPHY_TX0 UPHY_RX0	PCIe x1 C1
1	UPHY_TX1 UPHY_RX1	USB3 1 P2
2	UPHY_TX2 UPHY_RX2	PCIe x4 C0 (L0/L1/L2/L3)
3	UPHY_TX3 UPHY_RX3	
4	UPHY_TX4 UPHY_RX4	
5	UPHY_TX5 UPHY_RX5	
6	UPHY_TX6 UPHY_RX6	USB3 1 P0

Lane	Pin Names	Functions
7	UPHY_TX7 UPHY_RX7	PCIe x1 C3
8	UPHY_TX8 UPHY_RX8	PCIe x2 C4 (L0/L1)
9	UPHY_TX9 UPHY_RX9	UFS
10	UPHY_TX10 UPHY_RX10	X1 (L1)
11	UPHY_TX11 UPHY_RX11	USB3 1 P3
NVHS [7:0]	NVHS0_TX* NVHS_SLVS_RX*	PCIe X8 C5

Jetson AGX Xavier is designed to support the configurations listed in these tables. Released software also supports these configurations. However, the table lists only one out of 64 possible configurations. For further information, consult the *NVIDIA Jetson AGX Xavier Technical Reference Manual* (TRM) and 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 `p2972-0000.conf.common`. If both sources define lane assignment, the assignments in ODMDATA take priority. If the customer device requires custom UPHY lane assignments, NVIDIA recommends defining them through ODMDATA, not by modifying the `bpmp-dtb` file, unless you are thoroughly familiar with UPHY and UPHY lane assignment.

bpmp-dtb

BPMP (Boot and Power Management Processor) is a Jetson AGX Xavier processor that is designed for handling the boot process and offloading 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 {
            status = "okay";
            pcie-xbar-config = "PCIE_XBAR_4_1_0_1_2";
            ufs-config = "UFS_x1_L1";
            nvhs-owner = "PCIE";
      };
};
```

ODMDATA and Plugin Manager

ODMDATA and Plugin Manager are designed to 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 3 shows the meanings of the ODMDATA bits that are related to UPHY lane assignment.

Table 3. ODMDATA bits for UPHY lane assignment

OMDDATA bits* / Usage	Options	
31:27 HSIO-PCIE XBAR configurations	b00000 = pcie-xbar-2-1-1-1-2 b00001 = pcie-xbar-4-1-0-1-2 b00010 = pcie-xbar-4-1-1-1-2 b00011 = pcie-xbar-4-0-0-1-2 b00100 = pcie-xbar-4-1-0-1-2-c1l6 b00101 = pcie-xbar-4-0-1-1-2 b00110 = pcie-xbar-4-1-1-1-2-c1l6 b00111 = pcie-xbar-2-1-1-0-4 b01000 = pcie-xbar-2-1-1-1-4 b01001 = pcie-xbar-4-1-0-0-4 b01010 = pcie-xbar-4-1-0-1-4 b01011 = pcie-xbar-4-1-1-0-4 b01100 = pcie-xbar-4-1-1-1-4	b01101 = pcie-xbar-8-1-0-0-0 b01110 = pcie-xbar-8-1-0-1-0 b01111 = pcie-xbar-8-0-0-0-2 b10000 = pcie-xbar-8-1-1-0-0 b10001 = pcie-xbar-8-1-1-1-0 b10010 = pcie-xbar-8-0-1-0-2 b10011 = pcie-xbar-8-1-0-0-1 b10100 = pcie-xbar-8-1-0-1-1 b10101 = pcie-xbar-8-1-0-0-2 b10110 = pcie-xbar-8-1-0-1-2 b10111 = pcie-xbar-8-1-1-0-1 b11000 = pcie-xbar-8-1-1-1-1
24:23 UFS configuration	00b: disable-ufs-uphy 01b: enable-ufs-uphy-l11	10b: enable-ufs-uphy-l10 11b: enable-ufs-uphy-l10-l11

OMDDATA bits* / Usage	Options
22 SATA enablement	0b: disable-sata 1b: enable-sata
Notes * ODMDATA = 0x8190000 while flashing for Jetson AGX Xavier carrier board. † ODMDATA bits 31:27 allocate lanes for PCIe XBAR to controllers 0 to 4.	

NVIDIA recommends peforming UPHY lane assignment through ODMDATA and Plugin Manager because it can set related properties, such as MUX function properties, at the same time.. If your product is designed to work without ODMDATA and Plugin Manager, consult NVIDIA team for further help.

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 AGX Xavier P2822 carrier board.

For a Host-Only Port

This section takes J513, a USB 3.1 type-C connector for example of a host-only port.

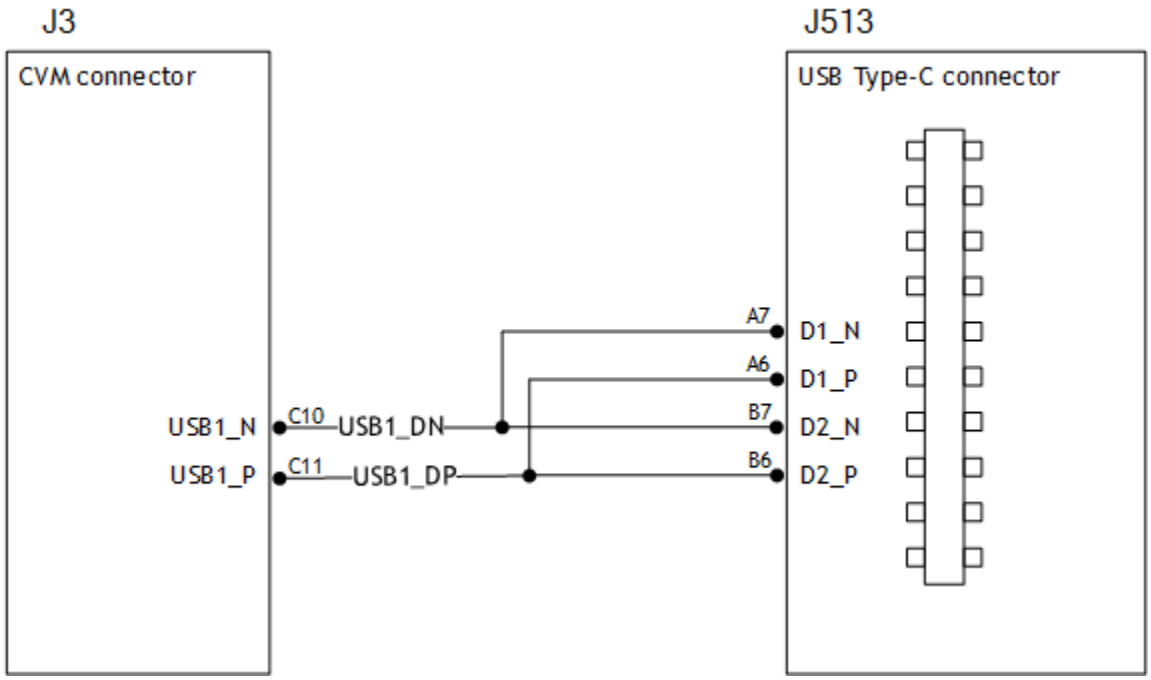
Go Through the Schematics

Note: **The P2822 carrier board's schematic file, `P2822_B02_Concept_schematics.pdf`, is included in *Jetson AGX Xavier Developer Kit Carrier Board Design Files*, available at:**

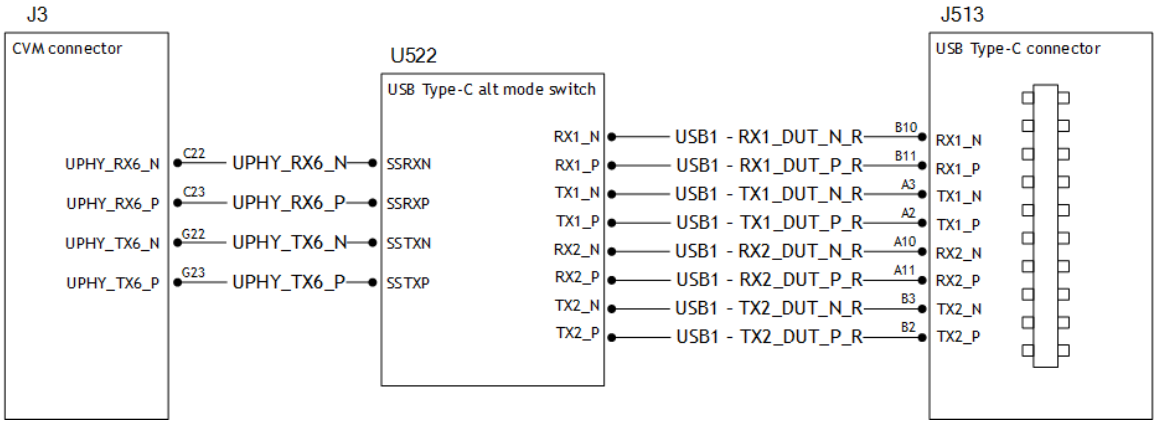
http://developer.nvidia.com/embedded/dlc/jetson-xavier-developer-kit-carrier-board-design-files-b03

Check the USB connectors on the P2822 carrier board and find the wired socket location to the P2888.

- USB2.0 signal pins D+/D- (USB1_D*) wire out from J513 and lead to C10 (USB1_D) and C11 (USB1_P) on the SOM socket.



- USB3.1 differential signal pairs (TX* and RX*) wire out from J513 and lead to K16 (UPHY_TX6_N), K17 (UPHY_TX6_P), B16 (UPHY_RX6_N), and B17 (UPHY_RX6_P) on the SOM socket through U523, the USB type-C alt mode switch.



Through the schematic, we can conclude that for J513:

- The USB2.0 signal pair is wired to UTMI pad 1 (USB2 port 1).
- The USB3.1 signal pairs are wired to UPHY lane 6 (USB3.1 port 0 according to UPHY lane mapping).

The xusb_padctl Node

The device tree's `xusb_padctl` node follows the conventions of the `pinctrl-bindings.txt` kernel document. It contains two sets of groups named `pads` and `ports`, which describe USB2 and USB3 signals along with parameters and port

numbers. The name of the 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

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

The ports Subnode

- `mode`: A string that describes USB port capability. A port for USB2 must have this property. It must be one of these values:
  - `host`
  - `device`
  - `otg`
- `nvidia,usb2-companion`: USB2 port (0/1/2/3) to which the port is mapped. A port for USB3 must have this property.
- `nvidia,oc-pin`: The overcurrent VBUS pin the port is using. The value must be positive or zero.

  Note: **Overcurrent detection and handling for J512 and J513 on the P2822 carrier board are controlled by U513, a Cypress Type-C controller. Therefore you need not set this property for J512 and J513 USB ports.**

- `vbus-supply`: VBUS regulator for the corresponding UTMI pad. Set to "`&battery_reg`" for a dummy regulator.

  Note: **The VBUS regulators for J512 and J513 are controlled by U513, a Cypress Type-C controller. Therefore you must set dummy regulators for those ports on the P2822 carrier board.**

- `nvidia,usb3-gen1-only`: A number (1/0) which describes whether or not to limit the port speed to USB3.1 gen1.

  Note: **J507, an eSATA port on the P2822 carrier board, only supports USB3.1 gen1 speed. Therefore you must set nvidia,usb3-gen1-only 1 (`true`) for J507.**

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

```
kernel/kernel-
4.9/Documentation/devicetree/bindings/pinctrl/nvidia,tegra194-
padctl.txt
```

Take J513 (USB3.1 type-C connector) for example. Create a pad/port node and property list for J513 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 = <&battery_reg>;
            status = "okay";
        };
        ...
        usb3-0 {
            nvidia,usb2-companion = <1>;
            status = "okay";
        };
        ...
    };
};
```

Under the xHCI Node

The Jetson AGX Xavier xHCI controller complies to xHCI specifications, which support both USB 2.0 HighSpeed/FullSpeed/LowSpeed and USB 3.1 SuperSpeed protocols.

- `phys`: Must contain an entry for each entry in `phy-names`.
- `phy-names`: Must include 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`: Set the value to which CPU frequency will be 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 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,tegra194-xhci.txt
```

Take J513 USB3.1 type-C connector for example. Create an xHCI node and property list for J513 based on the device tree structure described above:

```
tegra_xhci: xhci@3610000 {
    ...
    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 in the same port. A USB OTG port can switch back and forth between the roles of host and device.

This section takes J512, USB3.1 type-C connector, as an example of an OTG port.

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

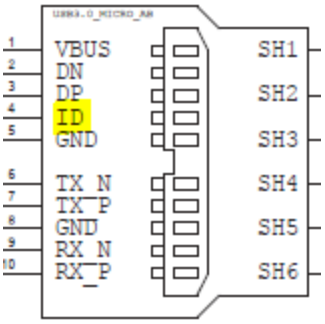


Figure 1. An OTG port connector

Note: **The roles of J512, the port switch, between the host driver (xHCI) and device driver (xUDC) are controlled by a U513 Cypress Type-C controller and ucsi_ccg driver in the Jetson AGX Xavier Developer Kit.**

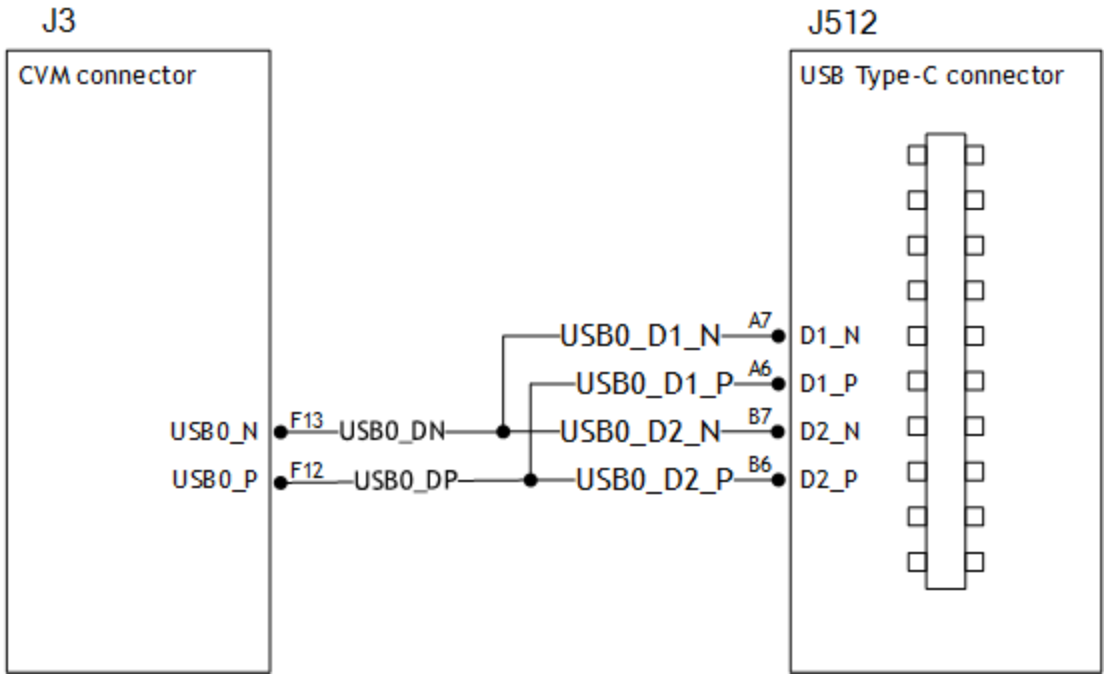
Go Through the Schematics

Note: **The P2822 carrier board's schematic file, `P2822_B02_Concept_schematics.pdf`, is included in *Jetson AGX Xavier Developer Kit Carrier Board Design Files*, available at:**

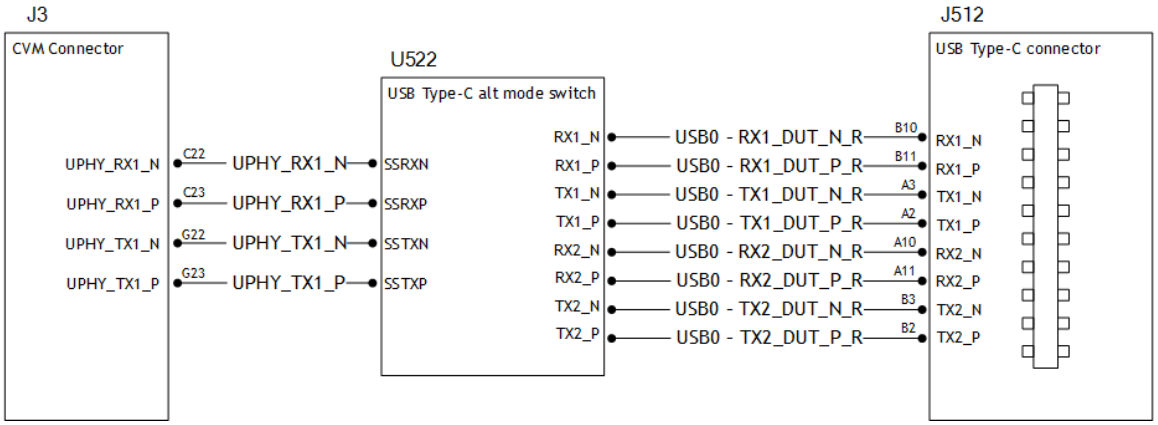
http://developer.nvidia.com/embedded/dlc/jetson-xavier-developer-kit-carrier-board-design-files-b03

Check the USB connectors on the P2822 carrier board and find the wired socket location to P2888.

- USB2.0 signal pins D+/D− (USB0_D*) wire out from J512 and lead to F12 (USB0_P) and F13 (USB0_N) on the SOM socket.



- USB3.1 differential signal pairs (TX* and RX*) wire out from J512 and lead to G22 (UPHY_TX1_N), G23 (UPHY_TX1_P), C22 (UPHY_RX1_N), and C23 (UPHY_RX1_P) on the SOM socket through U522, the USB type-C alt mode switch.



Through the schematic, we can that for J513:

- USB2.0 signal pair is wired to UTMI pad 0 (USB2 port 0).
- USB3.1 signal pairs are wired to UPHY lane 1 (USB3.1 port 2 according to UPHY lane mapping).

The External Connector Class (extcon)

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 GPIO_M3 as the VBUS_DETECT pin and GPIO_Q0 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 an A-plug connected the ID pin is pulled to ground (logical low), while with a 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 in Figure 2, for example:

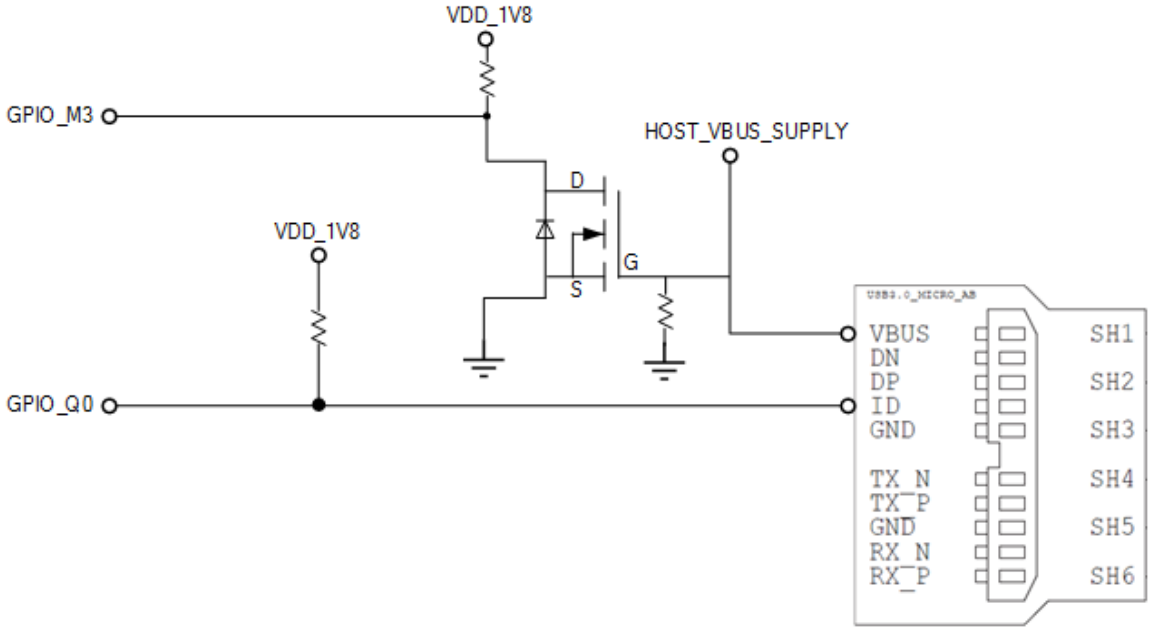


Figure 2. Example of general design for VBUS_DETECT pin

   With a B-plug connected VBUS_DETECT is logical low, because VBUS is provided from an external power supply, and 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:

GPIO_Q0 (ID)	GPIO_M3 (VBUS_DETECT)	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 extcon Node (Not Used on the P2822 Carrier Board)

Port switching between the roles of an OTG port is defined by the state of the ID pin and the VBUS_DETECT pin and the settings of the external connector class.

- `compatible`: Value must be `"extcon-gpio-states"`.
- `extcon-gpio,name`: Name of the `extcon` device.
- `gpios`: List of the GPIOs.
- `extcon-gpio,irq-flags`: IRQ flags for GPIO.
- `extcon-gpio,debounce`: Debounce time in milliseconds.
- `extcon-gpio,wait-for-gpio-scan`: Wait timeout for scanning all GPIOs' states after a GPIO state change is detected and debounce time has passed.
- `extcon-gpio,out-cable-names`: Output cable names.
- `extcon-gpio,cable-states`: GPIO states and their corresponding output cable states. The value is an array of byte values. Each even-numbered byte is a GPIO state, and the following odd-numbered byte is the corresponding output cable state.
- `cable-connected-on-boot`: Name of the output cable connected on boot, expressed as an index into `extcon-gpio,out-cable-names`. The first element is index 0, and so on. If not specified, the system assumes that no cable is to be connected. This property is valid if no GPIOs are provided for cable states.
- `wakeup-source`: A Boolean; `true` if the device can wake up the system.

For the detailed information about `extcon`, refer to the documentation at:

```
kernel/kernel-4.9/Documentation/devicetree/bindings/extcon/extcon-gpio-states.txt
```

Note: **OTG port switching between the host driver (xHCI) and device driver (xUDC) roles are controlled by the Cypress Type-C controller. Therefore this section is not a part of the device-tree for the Jetson AGX Xavier Developer Kit.**

- Create an `extcon` device node and property list based on the device tree structure described above and the table of GPIO states and corresponding output cable states for GPIO_Q0 and GPIO_M3:

```
vbus_id_extcon: extcon@1 {
    compatible = "extcon-gpio-states";
    extcon-gpio,name = "VBUS_ID";
```

```
    extcon-gpio,wait-for-gpio-scan = <0>;
    extcon-gpio,cable-states = <0x3 0x0
                                0x0 0x2
                                0x1 0x2
                                0x2 0x1>;
    gpios = <&tegra_main_gpio TEGRA194_MAIN_GPIO(M, 3) 0
            &tegra_main_gpio TEGRA194_MAIN_GPIO(Q, 0) 0>;
    extcon-gpio,out-cable-names =
            <EXTCON_USB EXTCON_USB_HOST EXTCON_NONE>;
    #extcon-cells = <1>;
};
```

For the example of `extcon`, refer to the device tree's source code at:

```
hardware/nvidia/platform/t19x/galen/kernel-dts/common/tegra194-
e3366-1199-a00.dtsi"
```

Note: **Check the pinmux table for the GPIO that corresponds to the ID pin and VBUS_DETECT pin.**

Under the ucsi_ccg Node

In the Jetson AGX Xavier Developer Kit role switching of port J512 between host driver (xHCI) and device driver (xUDC) modes is controlled by default by U513, a Cypress Type-C controller, and the `ucsi_ccg` driver.

typec-extcon

The subnode names must be in the form `port-<number>`, where `<number>` is a integer with value 0 or 1 which represents the type-C port (J512/J513).

- `#extcon-cells`: Number of cells in the `extcon` specifier. Must be 1.

typec-pd

The name of the subnode must be `pd`.

- `#extcon-cells`: Number of cells in the `extcon` specifier. Must be 1.

For the detailed information about ucsi_ccg, refer to the driver source code at:

```
kernel/kernel4.9/driver/usb/typec/ucsi/ucsi_ccg.c
```

Taking J512 USB3.1 type-C connector as an example, create a `ucsi_ccg` node and property list based on the device tree structure described above for J512:

```
ucsi_ccg: ucsi_ccg@8 {
    status = "okay";
    typec-extcon {
```

```
        typec_port0: port-0 {
            status = "okay";
            #extcon-cells = <1>;
        };
    };
    typec-pd {
        typec_pd: pd {
            status = "okay";
            #extcon-cells = <1>;
        };
    };
};
```

Under the xusb_padctl Node

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

Taking J512, the USB3.1 type-C connector, as an example, create a pad/port node and property list:

```
xusb_padctl: xusb_padctl@3520000 {
    ...
    pads {
        usb2 {
            lanes {
                usb2-0 {
                    nvidia,function = "xusb";
                    status = "okay";
                };
                ...
            };
        };
        usb3 {
            lanes {
                ...
                usb3-2 {
                    nvidia,function = "xusb";
                    status = "okay";
                };
                ...
            };
        };
    };
    ports {
        usb2-0 {
            mode = "otg";
            vbus-supply = <&battery_reg>;
            status = "okay";
        };
        ...
```

```
        usb3-2 {
            nvidia,usb2-companion = <0>;
            status = "okay";
        };
        ...
    };
};
```

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.

  Note: **The role of the Jetson AGX Xavier J512 port switch is controlled by U513, a Cypress Type-C controller, and the ucsi_ccg driver. Thus the `extcon-cables` entry must be "`<entry-of-ucsi_ccg> 1`" in the ucsi_ccg node, where 1 is the host mode detect status in ucsi_ccg.**

- `extcon-cable-names`: Must be `"id"`.
- `#extcon-cells`: Number of cells in the `extcon` specifier. Must be 1.

Taking J512, the USB3.1 type-C connector, as an example, create an xHCI node and property list based on the device tree structure described in Under the XHCI Node for a host-only port, plus the extcon settings above:

```
tegra_xhci: xhci@3610000 {
    ...
    extcon-cables = <&typec_port0 1>;
    extcon-cable-names = "id";
    #extcon-cells = <1>;
    phys = <&{/xusb_padctl@3520000/pads/usb2/lanes/usb2-0}>,
           <&{/xusb_padctl@3520000/pads/usb3/lanes/usb3-2}>;
    phy-names = "usb2-0", "usb3-2";
    nvidia,xusb-padctl = <&xusb_padctl>;
    status = "okay";
    ...
};
```

Under the xUDC Node

The Jetson AGX Xavier xUDC controller supports both USB 2.0 HighSpeed/FullSpeed and USB 3.1 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 will transition to device mode.

  Note: **The role of Jetson AGX Xavier J512 port switch is controlled by U513 Cypress Type-C controller and the `ucsi_ccg` driver. Hence, the `extcon` cable entry should be "<entry-of-ucsi_ccg> <host mode detect status in ucsi_ccg>" in ucsi_ccg node, where the <host mode detect status in ucsi_ccg> should be 0.**

- `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 a pointer to the node that defines each PHY in `phy-names`.
- `phy-names`: An array; must contain an entry 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,tegra194-xudc.txt
```

Taking J512, the USB3.1 type-C connector, as an example, create an xUDC node and property list for J512 based on the device tree structure described above:

```
tegra_xudc: xudc@3550000 {
    extcon-cables = <&typec_port0 0>;
    extcon-cable-names = "vbus";
    #extcon-cells = <1>;
    phys = <&{/xusb_padctl@3520000/pads/usb2/lanes/usb2-0}>,
           <&{/xusb_padctl@3520000/pads/usb3/lanes/usb3-2}>;
    phy-names = "usb2", "usb3";
    nvidia,xusb-padctl = <&xusb_padctl>;
    nvidia,boost_cpu_freq = <1200>;
    status = "okay";
};
```

To resolve possible UPHY lane mapping issues

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

1. UPHY Lane 0: `./devmem2 0x02d20388`
2. UPHY Lane 1: `./devmem2 0x02d30388`
3. UPHY Lane 2: `./devmem2 0x02d40388`
4. UPHY Lane 3: `./devmem2 0x02d50388`
5. UPHY Lane 4: `./devmem2 0x02d60388`
6. UPHY Lane 5: `./devmem2 0x02d70388`
7. UPHY Lane 6: `./devmem2 0x02da0388`
8. UPHY Lane 7: `./devmem2 0x02db0388`
9. UPHY Lane 8: `./devmem2 0x02de0388`
10. UPHY Lane 9: `./devmem2 0x02df0388`
11. UPHY Lane 10: `./devmem2 0x02e20388`
12. UPHY Lane 11: `./devmem2 0x02e30388`

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

- Bit 0 = 1: XUSB
- Bit 1 = 1: PCIe
- Bit 2 = 1: 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 PCIe XBAR is enabled by running the command:

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

The expected value is `pcie-xbar-4-1-0-1-2` for a P2822 carrier board. If another value is found, look for a problem in Plugin Manager.

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 AGX Xavier carrier board:

```
bootloader_unlocked       disable-sata             name
disable-pcie-c0-endpoint  disable-ufs-uphy         no-battery
disable-pcie-c4-endpoint  enable-debug-console     normal-build
```

```
disable-pcie-c5-endpoint  enable-denver-wdt        pcie-xbar-4-1-0-1-2
disable-pmic-wdt          enable-nvhs-uphy-pcie-c5
```

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

https://developer.nvidia.com/embedded/dlc/jetson-xavier-oem-product-designguide

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.

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 AGX Xavier 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 Suspend (LP0). 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 (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 Suspend (LP0).	☐
Verify that power-on/off sequencing of rails associated with each display meets manufacturer's requirements.	☐
Verify DSI or eDP timing (see *Tegra DC and DSI Debugging Guide* for details on how and what to verify).	☐
Probe DSI 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 for 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	☐

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 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	☐
eSATA	☐
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	☐
BCT, Full-speed	☐

Notice

Trademarks

Copyright