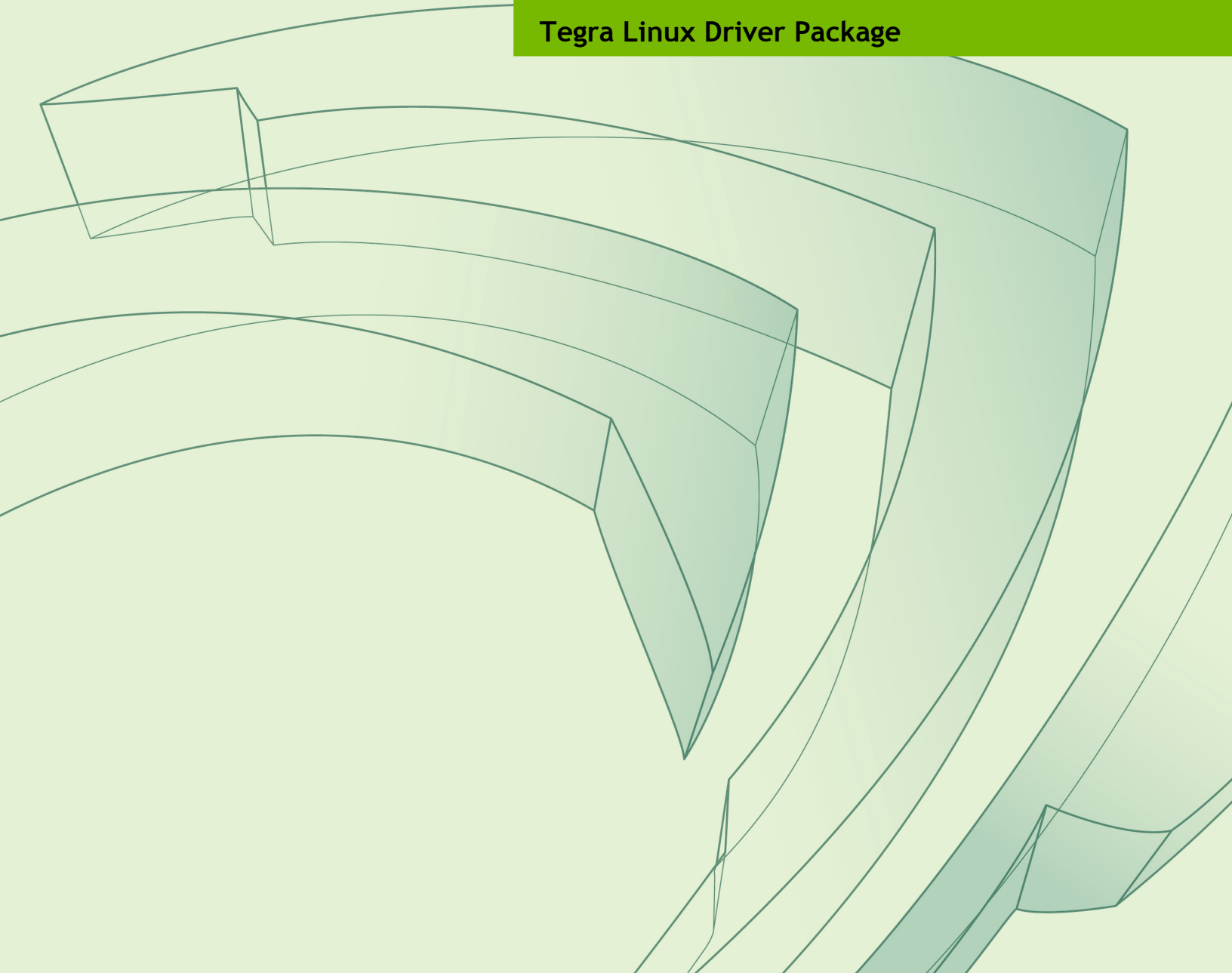




PLATFORM ADAPTATION AND BRING-UP GUIDE

DA_07378-001_01 | June 2, 2015
Advance Information | Subject to Change

Tegra Linux Driver Package



DOCUMENT CHANGE HISTORY

DA_07378-001_01

Version	Date	Authors	Description of Change
v1.0	6 Nov 2014	swarren/mzensius	Initial release
v1.1	2 Jun 2015	mzensius	Added “Other Considerations When Porting” section

TABLE OF CONTENTS

Platform Adaptation and Bring-Up Guide	1
Porting Linux for Tegra (L4T) to Your Design	1
Board Naming	1
Pinmux Changes	2
Exporting Pinmux for U-Boot.....	2
Exporting Pinmux for the L4T Linux Kernel	4
Porting U-Boot	4
Porting the Linux Kernel	5
Other Considerations When Porting.....	6
Hardware Bring-Up Checklist.....	7
Software Bring-Up Checklist.....	14

PLATFORM ADAPTATION AND BRING-UP GUIDE

This document describes how to port the NVIDIA® Tegra® Linux Driver Package, using the U-Boot boot loader, from NVIDIA® Jetson™ TK1 to other hardware platforms.

PORTING LINUX FOR TEGRA (L4T) TO YOUR DESIGN

For all procedures below, the Linux for Tegra (L4T) R21.1 release Jetson TK1 (jetson-tk1) code can serve as an example.

Board Naming

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

```
jetson-tk1  
beaver
```

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

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

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

```
nvidia
```

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



Note: Do not simply re-use and modify the existing Jetson TK1 support without choosing and using your own board name. If you do not use your own board name it will not be obvious to Jetson TK1 users if modified source code supports the original Jetson TK1 board or your board.

Pinmux Changes

If your board schematic differs from that for Jetson TK1, you must change the pinmux configuration applied by the software.

To define your board's pinmux configuration, you must obtain `Jetson_TK1_customer_pinmux_release.xlsm` from NVIDIA and customize it for the configuration of your board using the following procedures.

To customize the pinmux spreadsheet

1. Create a copy of the file with a name based on your board name, e.g. `<board>_pinmux.xlsm`.
2. Ensure that the new file is writable.
3. On a Windows PC, open the new file in Microsoft Excel.
4. If Microsoft Excel displays any warnings such as "PROTECTED VIEW" or "SECURITY WARNING", click Enable Editing or Enable Content, so that you can save your changes to the new file.
5. Rename the "Jetson TK1 Configuration" tab based on the name of your board:
 - Right-click on the "Jetson TK1 Configuration" tab at the bottom of the window.
 - Click the "Rename" menu option.
 - Type your board name into the tab, then press `[enter]`.
6. Modify columns AE through AO of the spreadsheet as required by the pinmux configuration for your board, based on the schematic.

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

Exporting Pinmux for U-Boot

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

To customize tegra-pinmux-scripts for your board

1. Obtain `tegra-pinmux-scripts` by running the following commands on your Linux host system (or any Linux system):

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

2. In the `tegra-pinmux-scripts` directory, open `csv-to-board-tegra124-xlsx.py` in a text editor.
3. Locate the definition of the `supported_boards` data structure, at approximately line 50.
4. Add an entry for your board to the `supported_boards` data structure similar to the following example:

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

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

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

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

To save the spreadsheet in CSV format

1. In Microsoft Excel, click the File tab.
2. On the File tab, click Save As.
3. From Save as type, choose "CSV (MS-DOC) (*.csv)".
4. Verify that the file name ends in ".csv", but otherwise matches the file name in your changes to `csv-to-board-tegra124-xlsx.py`.
5. Click Save.
6. Copy the CSV file to the `csv/` directory of `tegra-pinmux-scripts` on your Linux system.

To generate the U-Boot pinmux header file

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

```
$ ./csv-to-board-tegra124-xlsx.py <board>
```

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

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

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

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

Exporting Pinmux for the L4T Linux Kernel

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

To generate device tree files for your pinmux configuration

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

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

Porting U-Boot


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

1. Copy `include/configs/jetson-tk1.h` to `include/configs/<board>.h`.
2. Edit the set of enabled devices and features in `<board>.h` as appropriate for your board. For example, you must change the following:


```
#define CONFIG_TEGRA_BOARD_STRING          "NVIDIA Jetson TK1"
```

3. Copy `arch/arm/dts/tegra124-jetson-tk1.dts` to `arch/arm/dts/tegra124-<board>.dts`.
4. Edit the set of enabled devices and their parameters (e.g. GPIO and IRQ IDs) in `tegra124-<board>.dts` as appropriate for your board.

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

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


5. Add `tegra124_<board>.dtb` to `arch/arm/dts/Makefile`.
6. Copy `configs/jetson-tk1_defconfig` to `configs/<board>_defconfig`.
7. Edit `<board>_defconfig` to refer to your board name.
8. Edit `arch/arm/cpu/armv7/tegra124/Kconfig` to add target config and `Kconfig`.
9. Add `#ifdef` construct for your board in `board/nvidia/venice2/as3722_init.h`.
10. Copy the `board/nvidia/jetson-tk1/` directory to `board/<vendor>/<board>/`
11. Edit all of the files in `board/<vendor>/<board>/` to refer to your board name rather than Jetson TK1. There are many instances of the Jetson TK1 board name in the files in this directory.
12. Copy the pinmux header you generated (`pinmux-config-<board>.h`) to the `board/<vendor>/<board>/` directory.
13. In `Makefile`, edit the path to `as3722_init.o` to specify `../../../../nvidia/venice2/as3722_init.o`.

 **Note: Depending on the Tegra SKU that your board uses, you may need to edit `board/nvidia/venice2/as3722_init.*` to adjust the default CPU voltage rail settings.**

Porting the Linux Kernel

1. Copy `arch/arm/boot/dts/tegra124-jetson_tk1-pm375-000-c00-00.dts` to `arch/arm/boot/dts/tegra124-<board>.dts`.
2. For each file `arch/arm/boot/dts/tegra-platforms/tegra124-jetson-tk1-XXX-pm375-0000-c00-00.dtsi`, copy that file to `arch/arm/boot/dts/tegra-platforms/tegra124-<board>-XXX.dtsi`.
3. Edit each copied `.dts` file to refer to the correct/renamed `.dtsi` file if any.
4. Edit the set of enabled devices and their parameters (e.g. GPIO and IRQ IDs) in each copied file as appropriate for your board.

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

 **Note: U-Boot and the Linux kernel do not always use the exact same device tree schema (bindings) to represent the same data. Follow examples from the Linux kernel rather than from U-Boot. You may also refer to the official schema documentation in the `Documentation/devicetree/bindings/` directory in the Linux kernel source.**

5. Replace the content of `tegra124-<board>-gpio.dtsi` and `tegra124-<board>-pinmux.dtsi` with the content you generated from the kernel pinmux files earlier.
6. Edit `arch/arm/boot/dts/Makefile` to add an entry for your board, modeled after the existing Jetson TK1 entry.
7. For each file

```
Linux_for_Tegra/bootloader/ardbeg/jetson-tk1_extlinux.conf.XXX,
```

copy that file to the following:

```
Linux_for_Tegra/bootloader/ardbeg/<board>_extlinux.conf.XXX
```

8. Edit the following statements in each copied file

```
FDT /boot/tegra124-jetson_tk1-pm375-000-c00-00.dtb
```

to refer to your board.

9. Copy `Linux_for_Tegra/jetson-tk1.conf` to `Linux_for_Tegra/<board>.conf`
10. Edit `SYSBOOTFILE` and `DTB_FILE` in `<board>.conf` to refer to your board.

Other Considerations When Porting

This section describes other considerations and recommendations to consider when porting.

To flash the build image

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

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

To calculate BOOTPARTSIZE and EMMC_SIZE

1. Flash the device, using fastboot.
2. During flashing, copy the following three lines from the debug port:

```
Region=1 SD Erase start 512B-sector=0,512B-sector-num....
Region=2 SD Erase start 512B-sector=0,512B-sector-num....
Region=0 SD Erase start 512B-sector=0,512B-sector-num....
```

3. Modify `jetson-tk1.conf` to include the following assignments:

```

BOOTPARTSIZE=<(sector_number_from_region1 +
sector_number_from_region2) * 512>;
EMMC_SIZE=<(sector_number_from_region1 + sector_number_from_region2 +
sector_number_from_region3) * 512>;

```

- Flash the device again, specifying the root file system size:

```
$ sudo ./flash.sh -S <filesystem_size>MiB jetson-tk1 mmcblk0p1
```

To flash with BOARDID if the design does not use EEPROM

- Un-comment BOARDID in `Linux_for_Tegra/<board>.conf` if your design does not use EEPROM.

IF BOARDID is left commented-out, you might encounter boot issues. For more information, see “Support for BOARDID” in the NVIDIA *Tegra Linux Driver Package U-Boot User Guide*.

To change the UART port to UARTA

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

```
ODMDATA=0x60084000;
```

- In the U-Boot boot loader, locate the following lines in `/include/configs/jetson-tk1.h`:

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

- Modify those lines to specify UARTA:

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

- In the kernel, modify the `debug_uartport` assignment:

```
debug_uartport=1sport,0
```

Hardware Bring-Up Checklist

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

Before Power-On
Verify Board BCT.
Verify correct Solder Profile used, especially for Tegra and other fine-pitch BGA components.
Verify power supplies are not shorted to ground or to other power supplies.
Verify components are correctly placed with respect to component pin 1, polarity, and other placement requirements.
Verify strapping matches boot device.
Verify any stuffing options are properly handled.
Verify battery or other power source is sufficiently charged or otherwise adequate to boot platform.
Initial Power-On
Measure total system power. Ensure power is within reasonable range for default powered devices.
Verify VDD_CORE and VDD_RTC are correct initial voltage prior to SYS_RESET_N going high.
Verify VDDIO_DDR, VDDIO_DDR_MCLK, and VDDIO_DDR_HS are powered prior to SYS_RESET_N going high.
Verify AVDD_OSC (1.8V), AVDD_PLLn (1.05V), AVDD_USB (3.3V), and AVDD_USB_PLL (1.8V) are powered prior to SYS_RESET_N going high.
Verify VDDIO_SYS (1.8V), VDDIO_SDMMC4 (1.8V), and VDDIO_GMI (1.8V/3.3V), are powered prior to SYS_RESET_N going high.
Verify basic power sequence meets Tegra K1 requirements. <ul style="list-style-type: none"> ▶ Verify above power measurements are not excessive including spikes as rails are enabled. ▶ Capture power sequence including critical power rails, SYS_RESET_N, CLK_32K_IN and XTAL_OUT. ▶ Capture power ramp times for all rails Sheet max ramp rate specs. (1 V/ms) ▶ Verify there are no voltage dips or spikes on the Tegra K1 key power rails as each rail is enabled.
Verify 32 KHz CLK is present at least 4 cycles before SYS_RESET_N goes high.
Verify correct clock frequency is present at XTAL_OUT (Xtal or external source) before SYS_RESET_N goes high.
Verify JTAG can connect to AVP and Main CPUs.
Verify any UARTs intended for debugging are enabled and functional.
Verify DRAM with NVIDIA Diagnostic tool.
Verify SPI NAND with NVIDIA Diagnostic tool (If implemented).
Verify eMMC with Nvidia Diagnostic tool.
If boot device is EMMC, verify that bus operating at 8 bits wide and at 48 MHz or higher (i.e. HS200)
Verify system can enter USB recovery including USB (PID / VID) is recognized by the PC.

Initial Software Flashing
Verify System can be flashed with NvFlash.
Verify boot loader runs and displays splash screen.
Verify OS runs to desktop.
Power
Verify all supplies required at power-on are enabled appropriately.
Verify all supplies required off at power-on are not enabled initially.
Verify each supply that can be controlled can be enabled and disabled and different voltage level set if applicable.
Capture power sequence including critical power rails, SYS_RESET_N, CLK_32K_IN and XTAL_OUT.
Verify above sequence meets Tegra Datasheet requirements.
Capture power ramp times for all rails Sheet max ramp rate specs. (1 V/ms)
Verify above ramp rate meets Data Sheet specs. (1 V/ms)
Verify Fuel Gauge can be accessed and voltage read back is proper. (if implemented)
Verify charging works when USB wall plug adapter or AC adapter is connected. (if implemented)
Verify charging works when USB Host/Host charger or AC adapter are connected
Power Optimization
Capture Power consumption per rail.
Verify above power measurements are within expected limits.
PWR_I2C used to interface with PMU and external CPU supply, if present, functions properly with DVFS software.
Capture CPU PWR Request entering and exiting Suspend (LP1) and Deep Sleep (LP0). Ensure CPU PWR Request and associated power rail sequence meets Tegra Data Sheet requirements.
Verify all rails that must be OFF in Deep Sleep (LP0) are OFF.
Verify all rails that must be ON in Deep Sleep (LP0) are ON.
Verify required rails are back and at correct voltage under hardware control exiting Deep Sleep (LP0).
Verify Fuel Gauge Alert Interruption works for EDP capping.
Clocks
Verify 32KHz CLK is present.
Verify clock meets Tegra Data Sheet requirements.
Capture SYSTEM clock at XTAL_OUT.
Verify system clock is 12 MHz.
Boot Device
Verify optimum boot device timing settings are known, tested, and used in BCT.
Verify boot device write protection (if supported) functions properly.

DRAM
Run DRAM Shmoo (Consult Tegra Memory Char. App Note) and update BCTs with optimum timing values
Verify DRAM Stress tests pass across temperature
USB 2.0 PHY
Verify USB0 supports USB Recovery (device mode).
Verify 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 USB PHYs go to lowest power mode when not used or when the system is in low power mode.
Verify 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 USB1_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 signals meet requirements (correct eye height/width, etc.).
If USB signals do not meet requirements, use the <i>Tegra USB Tuning Guide</i> to adjust settings until requirements are met.
USB 3.0
Verify USB 3.0 (USB3_TX0/RX0) Host mode.
Verify USB 3.0 (USB3_TX1/RX1) Host mode.
Verify USB 3.0 interface goes to the lowest power mode when not used or when the system is in low-power mode
HDMI
Verify HDMI-compatible display works at 1080p.
Verify display detected properly (HPD).
Verify reads and writes to the display using DDC interface.
Verify HDMI related rails powered off when not used or system in Deep Sleep (LP0) or Suspend (LP1).
Capture HDMI signals at the connector (using appropriate test fixture and termination).
Verify signal quality is acceptable (meets EYE diagram, etc.). Consult <i>Tegra HDMI Tuning Guide</i> for details.
If HDMI signals do not meet requirements, use <i>Tegra HDMI Tuning Guide</i> to adjust settings until requirements are met.
Audio
Verify reads and writes on I2C interface used for Audio Codec.
Verify playback works properly on speakers, headphones, headset
Verify capture works properly: Sound is recorded from microphone/headset if supported.

Verify tones, voice, etc. can be heard from speakers or headphones/headset
Verify Audio Codec goes to lowest power mode when not in use or system enters low power mode
Capture (for each I2S I/FT used), signals at receiver end of link (if accessible.)
Verify signal quality is acceptable. Look for excessive over/undershoot, glitches on signal edges.
UART
Verify (for each UART used), Tegra TX/RX/CTS/RTS connects to device RX/TX/RTS/CTS.
Verify signal quality is acceptable. Look for excessive over/undershoot, glitches on signal edges.
eMMC (SDMMC4)
Verify I/O and Core power supplies activate before SYS_RESET_N goes high (if boot device).
Verify eMMC can be written to with NvFlash.
Capture SDMMC4 signals at receiver end (socket or test points near Tegra or both for Bidirectional signals). Add to Bring-up bug.
Verify signal quality is acceptable. Look for excessive over/undershoot, glitches on signal edges and abnormal Clock duty cycle.
SD Card (SDMMC3)
Verify proper connectivity by setting Tegra pins to GPIOs (if necessary to debug.)
Verify basic SD commands operate properly.
Verify reads and writes for a variety of SD Cards.
Verify SD Card insertion detection works and wakes system, if supported.
Verify SD Card Write Protect works, if implemented.
Verify SD Card goes to low power mode or rails powered off when not used or in low power system state.
Capture SDMMC3 signals at receiver end (socket or test points near Tegra or both for Bidirectional signals).
Verify signal quality is acceptable. Look for excessive over/undershoot, glitches on signal edges and abnormal Clock duty cycle.
Sensors I2C: General
Verify addresses of all I2C devices appear correctly, and no unknown ghost devices appear.
Capture I2C clock near device, and data at both device and Tegra end of interface. Add to Bring-up bug
Verify signal quality is acceptable including rise times of signals.
Sensors I2C: Touch Screen (Optional)
Verify Reads/Writes on I2C or SPI to Touch Screen controller are functional (possibly read device ID or similar register.)
Verify Interrupts generated properly.
Verify functionality of Touch Screen.

Verify Touch Screen Controller goes to lowest power mode when not used, or system in low power state.
Sensors I2C: Thermal sensor
Verify Reads/Writes on I2C interface to Thermal Sensor are functional (possibly read device ID or similar register)
Verify Alert and Therm outputs are generated at correct die temperatures
PEX (Optional)
Verify proper connectivity by checking lanes.
Verify any PEX interfaces implemented transition to lowest power state in Deep Sleep (LP0) and Suspend (LP1).
Capture PEX signals at receiver end of link near Tegra and device.
Verify signal quality is acceptable. Look for excessive over/undershoot, glitches on signal edges.
SATA (Optional)
Verify proper connectivity by checking diff lines
Verify any SATA interfaces implemented transition to lowest power state in Deep Sleep (LP0) and Suspend (LP1).
Capture SATA signals at receiver end of link near Tegra and device.
Verify signal quality is acceptable. Look for excessive over/undershoot, glitches on signal edges.
HSIC (Optional)
Verify HSIC interfaces to Baseband, hub or other peripheral functions properly.
Verify any HSIC interfaces implemented transition to lowest power state in Deep Sleep (LP0) and Suspend (LP1).
Capture HSIC signals at receiver end of link near Tegra and device.
Verify signal quality is acceptable. Look for excessive over/undershoot, glitches on signal edges.
Embedded Display(s) (Optional)
Verify I2C or other control I/F is able to perform writes/reads to display.
Verify embedded display(s) shows correct colors.
Verify embedded display(s) backlight is enabled when in normal display mode.
Verify embedded display(s) backlight brightness can be adjusted properly.
Verify embedded display(s) backlight is disabled when in a low power mode.
Verify embedded displays (and any display bridge) transition to lowest power state in Deep Sleep (LP0) and Suspend (LP1).
Verify Power-on/off sequencing of rails associated with display meet manufacturer's requirements.
Verified DSI, LVDS or eDP timing (See "Tegra DC and DSI Debugging Guide" for details on how and what to verify).

Capture DSI, LVDS or eDP signals near panel driver, or connector/test points if access to driver not possible.
Verify signal quality is acceptable. Look for excessive over/undershoot, glitches on signal edges.
Imager(s) (Optional)
Verify I2C interface writes/reads work to all cameras.
Verify preview displays properly for all cameras.
Verify still capture works on all cameras.
Verify video capture works on all cameras.
Verify all flashes operate properly.
Verify any available autofocus mechanism functions properly.
Verify privacy LED operates properly if implemented.
Verify cameras and related circuitry enter lowest power mode when not used or system in a low power mode.
Verify Power-on/off sequencing of rails associated with imager module meet manufacturer's requirements.
Capture MCLK output at recommended test points. Verify signal quality is acceptable.
Verify signal quality is acceptable. Look for excessive over/undershoot, glitches on signal edges.
Recommended Signal Captures for Debugging
Capture Power sequence including critical power rails, SYS_RESET_N, CLK_32K_IN and XTAL_OUT.
Capture power ramp times for all rails Sheet max ramp rate specs. (1V/ms)
Capture time from Power button pressed to VDD_CORE ON.
Capture time from Power button pressed to SYS_RESET_N de-asserted.
Capture time from Power button pressed to BOOTROM hand over to boot loader.
Capture time from Power button pressed to CPU_PWR_REQ asserted.
Capture Power consumption per rail.
Capture CORE_PWR_REQ , CPU_PWR_REQ and SYS_CLK_REQ entering/exiting Deep Sleep (LP0).
Capture CPU PWR Request entering/exiting LP1/LP2.
Capture SYSTEM clock at XTAL_OUT.
Capture USB0_D+/D- signals at both ends of link (connector and test points near Tegra).
Capture USB1_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).
Capture HSIC signals at receiver end of link near Tegra and device.
Capture DSI, LVDS or eDP signals near panel driver, or connector/test points if access to driver not possible.
Capture HDMI signals at the connector (using appropriate test fixture and termination).
Capture MCLK output at recommended test points. Verify signal quality is acceptable.

Capture (for each I2S I/FT used), signals at receiver end of link (if accessible).
Capture SDMMC4 signals at receiver end (socket or test points near Tegra or both for Bidirectional signals).
Capture SDMMC3 signals at receiver end (socket or test points near Tegra or both for Bidirectional signals).
Capture SDMMC1 signals at receiver end (test points near Tegra and device).
Capture SYSTEM clock at XTAL_OUT.

Software Bring-Up Checklist

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

Preparation
Verify Board BCT.
Verify eMMC with Nvidia Diagnostic tool.
Obtain board schematics and component datasheets.
Power tree.
Board pinmux.
Bring-up Hardware Validation
Power and Reset Sequence, Power Rail Check.
Recovery Mode.
NvTest (Tegra MODS) DDR, eMMC, AVP, CPU.
JTAG connection check.
U-Boot port and Boot Validation
NvFlash
UART output
KBD connection
Board config/PMIC regulator config/Pinmux/Review device tree
Verify FS support/Config boot scripts (bootcmd)
Boot to U-boot
Boot to kernel
Boot to kernel command line or custom desktop
Kernel and Peripherals, Port and Validation
Device tree review, Pinmux, GPIO, Wake-pins
PMU and regulator drivers
Display/HDMI
Audio codec
Microphone and speaker

USB
SD card
Thermal Sensor
EMC DFS table
Ethernet
SATA
PCIe
System Power and Clocks
CPU/CORE/GPU DVFS
EMC DFS table
CPU/CORE EDP
GPU EDP
System EDP (Contain Current monitor & Voltage comparator)
Power Off
LP0 (optional)
CPU power down (LP2)
BCT, Full-speed

Notice

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

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

Trademarks

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

Copyright

© 2015 NVIDIA Corporation. All rights reserved.