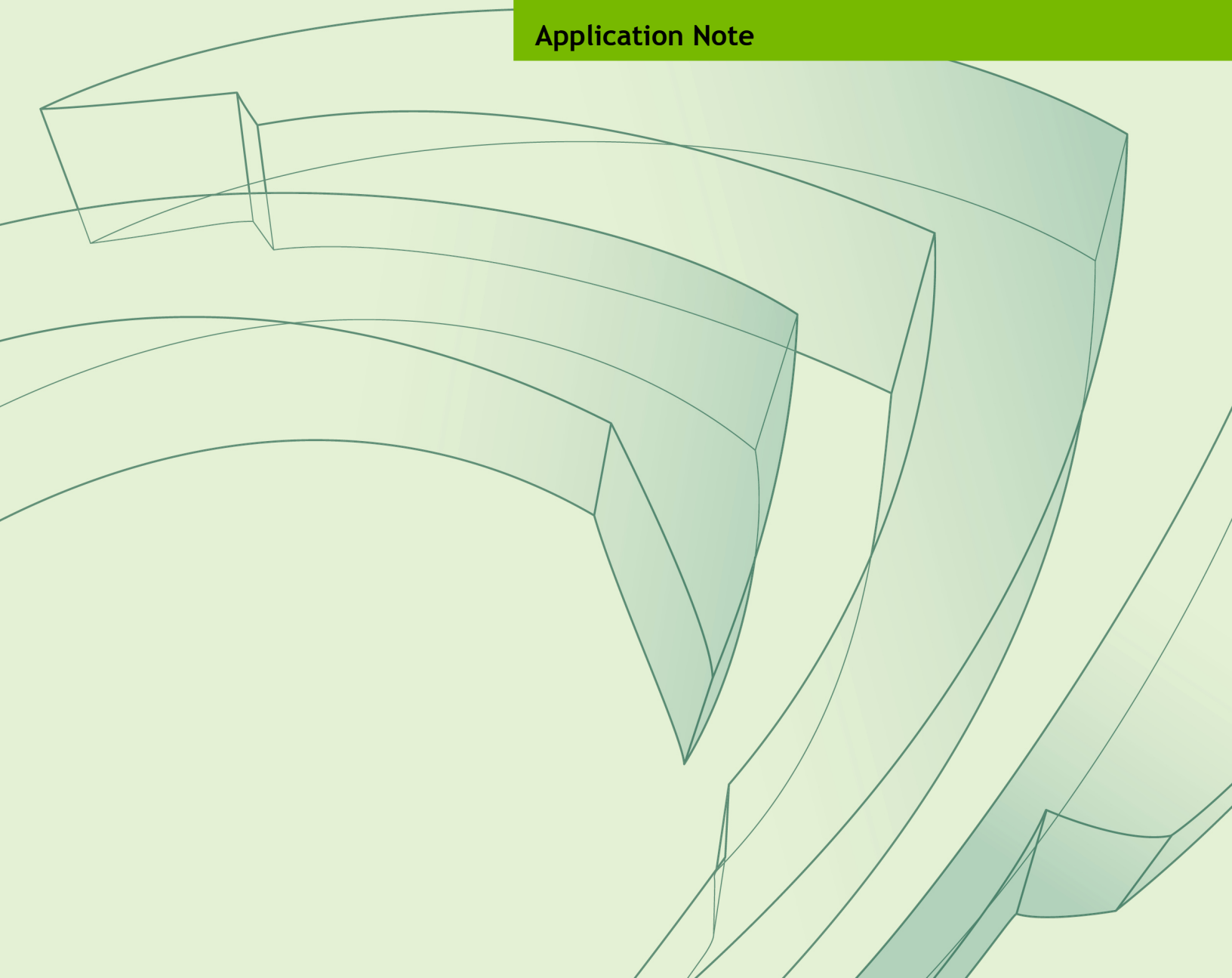
RUNTIME BOOTLOADER UPDATE PROCESS FOR JETSON TK1

DA_07695-001_01 | August 24, 2016

Application Note



DOCUMENT CHANGE HISTORY

DA_07695-001_01

Version	Date	Authors	Description of Change
v1.0	May 29, 2015	hlang	R21.3 Release
v2.0	July 29, 2015	hlang/msum	Addition of bootloader redundancy feature
v2.1	August 24, 2016	mzensius	Minor non-content related updates to links and references.

TABLE OF CONTENTS

INTRODUCTION

This document provides guidelines for enabling runtime boot loader updates with data redundancy features to be used as part of a failure-tolerant system update procedure. The topics discussed are specific to the NVIDIA® Tegra® Linux Driver Package (L4T) R21.5 release and the U-Boot boot loader on the Jetson™ TK1 Developer Kit. Adaptation of these instructions may be required for use on other Tegra K1-based platforms. Implementation of a specific update mechanism or update procedure is outside the scope of this document.

PREREQUISITES

For a description of the Tegra boot flow, see the NVIDIA® Tegra® Public Application Notes. This document assumes an understanding of the Tegra boot process.

For information on the Linux Tegra Driver Package, refer to the NVIDIA Embedded Developer Zone. This document assumes an understanding of the L4T software and the Jetson TK1 Developer Kit.

BCT AND BOOTLOADER REDUNDANCY

The Tegra BootROM validates the BCT through an integrated checksum. If the calculated checksum does not match the checksum value within the BCT, the BootROM searches for the next BCT and attempts to validate. Up to 64 copies of the BCT are searched, at mod-16KiB boundaries. The NVIDIA flashing utility, `nvflash`, writes up to 64 copies of the BCT based on the space allocated for the BCT partition.

A BCT can contain up to four entries for indicating the location (offset and size) and checksum of the boot loader. The BootROM computes and validates the checksum for each entry. When the first valid checksum is located, the BootROM transfers control (jumps) to the specified boot loader.

OVERVIEW

The procedures for updating the runtime boot loader are as follows:

- Implement Boot loader Redundancy:
  - Modify the Linux kernel to expose the eMMC boot0 and boot partitions for runtime access.
  - Modify the partition configuration file to make the boot loader partitions visible to the GPT partition table.
  - Modify the U-Boot configuration file to locate the proper boot partition.
  - Modify the U-Boot boot script file, `extlinux.conf`, to specify the proper root file system partition.
- Deploy devices with boot loader redundancy.
- Prepare new boot loader and BCT images for update:
  - Extract the BCT from the deployed system for offline modification.
  - Use offline, host-based tools to regenerate the BCT and U-Boot boot loader images.
- Download the new BCT and U-Boot images into the target device.
- Update the new BCT and U-Boot images:
  - Copy the BCT and U-Boot images to eMMC partitions.
  - Reboot to execute the updated boot loader.

FAILURE-TOLERANT BCT AND BOOT LOADER UPDATE

The standard L4T release does not enable boot loader redundancy features. As part of a failure-tolerant boot loader update, you must first implement and deploy boot loader redundancy and then follow the boot loader update procedure.

IMPLEMENTING BOOT LOADER REDUNDANCY

Enabling boot loader redundancy requires modification of the following components:

- Kernel eMMC driver
- Partition configuration file
- U-Boot configuration file
- U-Boot start script

After these modifications are made, flashed, and deployed, then initial boot loader redundancy is enabled.

Modifying the Linux Kernel

You must modify the Linux kernel to expose the eMMC `boot0` and `boot1` partitions for runtime access. By default, eMMC boot partitions are not exposed during runtime by the Linux kernel.

To expose the eMMC boot partitions

1. Navigate to the kernel driver:

```
<kernel>/drivers/mmc/host/sdhci-tegra.c
```

2. Comment out the following line:

```
host->mmc->caps2 |= MMC_CAP2_BOOTPART_NOACC;
```

When the kernel is booted, the write-protected BCT partition is visible at `/dev/mmcblk0boot0`. This modification also enables access to `/dev/mmcblk0boot1`, which is beyond the scope of this document.

Making the Boot Loader Partitions Visible

You must modify the partition configuration file to make the bootloader partitions visible to the GPT partition table.

To modify the partition table

1. Modify the BSP configuration file:

```
bootloader/ardbeg/cfg/gnu_linux_fastboot_emmc_full.cfg
```

   The BSP configuration file contains the partitioning information for both the Tegra partition table and the GPT.

   - Partitions defined after GP1 are visible to Linux.
   - Partition EBT contains the bootloader.
   - Partitions EB1, EB2, and EB3 store additional boot loaders for redundancy. These partitions must be defined after GP1 and before APP to become visible within the GPT.

2. Verify that EB3 is defined first and EBT is defined last, ensuring that EBT is the first in the Boot ROM search order and EB3 is the last.
3. Modify the `gnu_linux_fastboot_emmc_full.cfg` file to move the GP1 entry immediately after the PT entry.
4. Modify all partition entry ID=values to be contiguous.

   For example:

```
…
[partition]
name=PT
id=4
type=partition_table
allocation_policy=sequential
filesystem_type=basic
size=2097152
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
```

```
percent_reserved=0

[partition]
name=GP1
id=5
type=GP1
allocation_policy=sequential
filesystem_type=basic
size=2097152
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0

[partition]
name=EB3
id=6
type=bootloader
allocation_policy=sequential
filesystem_type=basic
size=4194304
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0
filename=fastboot.bin

[partition]
name=EB2
id=7
type=bootloader
allocation_policy=sequential
filesystem_type=basic
size=4194304
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0
filename=fastboot.bin

[partition]
name=EB1
id=8
type=bootloader
allocation_policy=sequential
filesystem_type=basic
size=4194304
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0
filename=fastboot.bin
```

```
[partition]
name=EBT
id=9
type=bootloader
allocation_policy=sequential
filesystem_type=basic
size=4194304
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0
filename=fastboot.bin
…

[partition]
name=APP
id=15
type=data
allocation_policy=sequential
filesystem_type=basic
size=1073741824
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0
filename=system.img

…
```

This modification causes the bootloader partitions to appear under Linux as `/dev/mmcblk0p1` through `/dev/mmcblk0p4`, makes other partitions visible, and causes the root file system partition to move. Further modifications are required to inform the U-Boot and the Linux kernel of the location of the new root file system.

Modifying U-Boot

U-Boot contains a partition number from which the U-Boot configuration file, `extlinux.conf`, is loaded.

To update the U-Boot configuration file

1. Find out the new the GPT partition number of the `APP` partition, which is calculated as follows:

   ```
   ID of APP partition – ID of GP1 partition
   ```

   In the above example, the new partition number is 15 – 5 = 10.

2. Navigate to the U-Boot source code file:

```
<U-Bboot Source PATH>/include/config_distro_bootcmd.h
```

3. Change `bootpart=1\0` to `bootpart=<N>\0`.

   Where `<N>` is the value as calculated in the previous step (e.g., `0xa`).

   U-Boot interprets the boot partition number as a HEX value. Verify that the HEX number `0xa` is used instead of the decimal number `10`.

   The U-Boot build system creates `u-boot-dtb-tegra.bin` as output. L4T expects that output to be used and named as `u-boot.bin`.

Modifying the U-Boot Boot Script File

You must modify the U-Boot boot script file, `extlinux.conf`, to specify the proper root file system partition in the kernel `cmdline` parameter.

To specify the root file system partition

1. Using the same GPT partition number as above (e.g., 10), modify the default kernel command line passed from U-Boot to the Linux kernel to specify the proper root partition.
2. Navigate to the U-Boot boot-script source file:

```
<BSP>/bootloader/ardbeg/jetson-tk1_extlinux.conf.emmc
```

3. Change `root=<value>` to the proper value. Using the example above, the root value is as follows:

```
root=/dev/mmcblk0p10
```

ENABLING BOOT LOADER REDUNDANCY

Flash the Jetson TK1 platform with the above modifications to verify proper loading and functioning of U-Boot, the Linux kernel, and the proper location specified for the root file system. To ensure that all build components are functional, it is recommended that you verify the bootloader and kernel independently prior to enabling bootloader redundancy.

For flashing instructions, see the *Developer Guide* for your device.

PREPARING BOOT LOADER AND BCT IMAGES FOR UPDATE

For convenience, this document refers to BCT image names with four appended numeric values, which denote the bootloader version described by the BCT image. The numeric versions are arbitrary but indicate the bootloader update process.

For example, after flashing, with initial bootloader redundancy enabled, all bootloader versions are identical and referred to as `bct_1111`. When U-Boot is updated and the BCT modified to contain two new images in slots 0 and 1 (positions 1 and 2), the boot version is referred to as `bct_2211`.

Likewise, for descriptive purposes, a similar numeric value is appended to the name of the BCT binary.

Extracting the BCT from the System

Before updating the bootloader, **read and maintain a copy of the BCT (`bct_1111`) flashed on the production device**. This BCT is used for later modifications to update the bootloader entries (location, size, and hash) within the BCT.

To retrieve the binary BCT

1. Place a production device into forced-recovery mode.

   For instructions on how to place a device in recovery mode, see the *Setting Up Your Platform* topic in the *Developer Guide* for your device.

2. Use the `nvflash` utility to read `bct_1111` from the device.

```
# cd <BSP>/Linux_for_Tegra/bootloader
# ./nvflash --read 2 bct_1111 --bl fastboot.bin --go
```

Regenerating the BCT and U-Boot Images

NVIDIA provides a host-based tool, `mkbctpart`, for offline modifications and updates of the BCT. Use `mkbctpart` to specify a new location, size, and hash value for an updated bootloader binary.

The `mkbctpart` syntax is as follows:

```
Usage:  mkbctpart [options] [new BCT file]
where
  <options> are:
    -b|--bctpartition <input BCT file> ------------ default=bct.dump
```

```
    -i|--instances <BL update entry CSV> ---------- default=0,1
    -l|--listbcts --------------------------------- default=N/A
    -p|--paddedfile <flashable padded BL file> ---- default=<BL>.padded
    -t|--tegratype <Tegra type> ------------------- default=T124
    -B|--Bootloader <new BL file name> ------------ default=u-boot.bin
    -V|--Verbose ---------------------------------- default=0
  <new BCT file> is:
    Output file name for updated BCT partition.

  <BL>.padded is:
    The new U-Boot file padded to have a size of modulo 16 bytes, which
is crypto block size.
```

For example:

```
$ ./mkbctpart -b bct_1111 -i 0,1 -B u-boot.bin.2 -V bct_2211
```

In the above example, `mkbctpart`:

- Takes `bct_1111` as input, describing four version- 1 U-Boot binaries originally flashed in the production device and new U-Boot version-2 (`u-boot.bin.2`) as its input.
- Generates the new BCT file `bct_2211`, which describes U-Boot version-2 in slots 0 and 1, and U-Boot version-1 in slots 2 and 3.
- Generates a new padded U-Boot file `u-boot.bin.2.padded`. Both `bct_2211` and `u-boot.bin.2.padded` are downloaded to the device prior to the update process.

DOWNLOADING THE UPDATED BCT AND U-BOOT IMAGES

You must copy the updated `bct_2211` and padded bootloader image `u-boot.bin.2.padded` to the target device (e.g., with the `scp` command via removable storage devices, etc.). The mechanism for performing this task is beyond the scope of this document.

COPYING THE BCT AND U-BOOT IMAGES INTO EMMC

The following provides example target commands to overwrite the BCT and bootloader partitions (EBT and EB1) with the updated binaries.

```
# dd if=u-boot.bin.2.padded of=/dev/mmcblk0p4
# dd if=u-boot.bin.2.padded of=/dev/mmcblk0p3
# echo 0 >/sys/block/mmcblk0boot0/force_ro
```

```
# dd if=bct_2211 of=/dev/mmcblk0boot0
# sync
# sync
# echo 1 >/sys/block/mmcblk0boot0/force_ro
```

EXECUTING THE UPDATED BOOT LOADER

To execute the updated BCT and bootloader, you must reboot the system.

MKBCTPART UTILITY

The `mkbctpart` utility generates the updated BCT partition file and padded bootloader file from the provided BCT partition file and the new bootloader file, as follows:

1. Reads in the BCT partition file and validates it.
2. Reads in the new bootloader, pads it to be mod-16 bytes, and calculates the hash.
3. Updates the bootloader entries in the BCT as specified by the `-i` option.

   Entries (also known as slots) not specified for modification remain the same, assuming the location, size, and hash of the bootloader being replaced.

4. Writes out the new updated BCT partition file and the padded bootloader file.

   For the `mkbctpart` syntax and an example, see Regenerating the BCT and U-Boot Images.

Notice

Trademarks

Copyright