Application Report
SPRAC65A–February 2017–Revised November 2017

Using Peripheral Boot and DFU for Rapid Development on Jacinto 6 Devices

Venkateswara Rao Mandela

ABSTRACT

This application report describes how to use peripheral boot and Device Firmware Upgrade (DFU) to reduce the time required to load updated binaries to various cores of a Jacinto 6 (DRA7xx) family device [1].

Contents

Trademarks

1 Introduction

On embedded hardware, adding new features to the first and second stage bootloaders can be a time consuming process. The bootloader needs to be copied to a SD card or other media each time the source code is modified and then the media needs to be plugged into the board. This process can be tedious and time consuming.

This document shows how to use the peripheral boot feature of DRA7xx devices along with the DFU utility to enable rapid iteration and reduce development time.

1.1 Peripheral Boot

Peripheral Boot is one of the many boot modes supported by DRA7xx devices. Peripheral boot mode allows DRA7xx ROM to download the first stage bootloader over USB and run it.

1.2 Device Firmware Upgrade (DFU)

Wikipedia states "Device Firmware Upgrade (DFU) is a vendor- and device-independent mechanism for upgrading the firmware of USB devices with improved versions provided by their manufacturers, offering (for example) a way for firmware bugfixes to be deployed."

The U-Boot provided by TI with the DRA7xx devices has support for DFU. TI U-Boot provides a way to flash bootloader, kernel and device tree to various media via DFU. For more details, see *Flashing Binaries to DRA7xx Factory Boards Using DFU*. This facility is useful when flashing the hardware with stable binaries. During development, however, the time taken in flashing the binaries can be tedious.

Using the steps described in this document, one can eliminate the need to flash bootloader or kernel or device tree during development. While the same can be achieved with tftp, use of DFU is faster and more flexible for transferring and booting binaries in various combinations.

2 Prerequisites

2.1 Hardware Requirements

- Linux PC running Ubuntu 14.04 LTS or any other version supported by Processor SDK Linux Automotive.
- Micro-USB to USB cable for connecting EVM to the PC. This is used for transferring the binaries from the PC to the board.
- Mini-USB to USB cable to displaying UART logs on the PC.

2.2 Software Requirements

This application report is intended for use with Processor SDK Linux Automotive 3.02 or later. The patches described below apply on top of the U-Boot commit used in the SDK. For the same functionality on GLSDK 7.04, see the previous version on this document.

2.2.1 Bootswitch

The bootswitch tool is used to transfer the first stage boot loader to the EVM. To install the tool, clone the git repository below and follow the steps detailed in the README.md in the repository.

git://git.ti.com/glsdk/dra7xx-bootswitch.git

This tool can also be used on windows. For compilation instructions, please refer to the file README.win.md in the repository.

This tool can also be used to control the media (for example, QSPI, eMMC, SD Card) from which the EVM is booting, but this feature of the tool will not be used in this document. For more information, see the documentation in the dra7xx-bootswitch git repository.

2.2.2 DFU

DFU tool is available as part of the ubuntu repositories. It can be installed by:

```
host $ sudo apt-get install dfu-util
```

dfu-util is used to transfer binaries to the EVM.

2.2.3 u-boot-tools

The fdtput binary is used from the u-boot-tools package for manipulating the device tree. This package can be installed by running the below command.

```
host $ sudo apt-get install u-boot-tools
```

2.2.4 device-tree-compiler

The dtc binary is used from the device-tree-compiler package to pad the device tree. This creates space in the device tree blob for setting different attributes in the bootloader in the target. This is required when loading the DSP and M4(IPU) cores on the SOC.

```
host $ sudo apt-get install device-tree-compiler
```

3 Testing/Debugging Various Binaries

This section covers the following:

- MLO development/testing
  - How to transfer the first stage bootloader (MLO/SPL) to the EVM using peripheral boot
  - Potential changes to be made to the MLO when booting using peripheral boot instead of other media.
- Code changes to enable transferring U-Boot/Kernel/Remotecore binaries via DFU
- How to transfer and start the below binaries via DFU
  - U-Boot
  - Kernel and device tree
  - Remotecores (DSP, IPU)
  - Kernel, device tree and ramdisk

3.1 Transferring First Stage Bootloader

To load the first stage bootloader on to the EVM, the peripheral boot feature of DRA7xx SoC is used. DRA7xx ROM supports receiving the first stage bootloader over USB. The bootswitch tool is used to perform the transfer from the host side. Please ensure that you have setup bootswitch tool as specified in README.md

1. Place the EVM in peripheral boot mode by setting SW2[0:5] to 01 0000.
2. Connect a USB cable from connector P2 on the EVM to the PC.
3. Modify the configuration file used by the bootswitch tool (/tmp/bootsetting.txt) to point to the MLO/SPL binary. If the U-boot source is in the /home/user/u-boot, the content of the file should be

```
1:5
/home/user/u-boot/spl/u-boot-spl.bin
```

4. Reboot the EVM.

On reboot, bootswitch tool transfers the u-boot-spl.bin file to the EVM. DRA7xx ROM switches to executing the binary as it would with a binary read from QSPI/SD/eMMC. The time taken from rebooting the EVM to DRA7xx starting the execution of the transferred binary is around 500 ms. Compare this to 10-15 seconds it would take to extract the microSD card out of the EVM, copy the MLO binary, insert it back into the EVM and power cycle the board.

If the default U-Boot provided with the Processor SDK Linux Automotive 3.02 release is run, you will see the following message on the serial port console.

```
U-Boot SPL 2016.05 ...
DRA722-GP ES1.0
*** Warning - bad CRC, using default environment
Trying to boot from MMC2
```

3.1.1 Controlling the 2nd Stage Boot Media

The media in which SPL/MLO looks for u-boot.img/kernel is determined by the device it booted from. When debugging MLO, you may want to force it to obtain u-boot.img/kernel from a different boot media. The below example patch shows how to force the MLO/SPL to pick up u-boot.img/kernel from the SD card.

Table 1. Patch to Force 2nd Stage Media for U-Boot 2016.05

S. No	URL	Headline
1	http://review.omapzoom.org/38669	spl: dra7xx: hardcode boot device to mmc

3.1.2 Debugging MLO

The patches shown in Table 2 can be used for debugging SPL/MLO itself using a JTAG. The first patch in the table adds an infinite while loop function that can be invoked from any location in the code. Once SPL execution is halted, you can connect to the A15 via a JTAG, exit the infinite loop and step through the code. Take care to remove any GEL files in the A15 configuration in CCS as the GEL files would reset the A15.

Table 2. Debug Patches for U-Boot 2016.05

S. No	URL	Headline
1	http://review.omapzoom.org/38670	spl: dra7xx: add an infinite loop function for debug
2	http://review.omapzoom.org/38671	spl: dra7xx: enable debug flags for CCS stepping

If you would like to debug SPL/MLO from the beginning of the C code, enable the call to wait_for_debugger() in board_init_f() in the file arch/arm/cpu/armv7/omap-common/hwinit-common.c.

The 2nd patch above enables debug flags and disables optimizations on files that might need to be stepped through. Customize this as per your debug needs.

3.2 Code Changes for Transferring Other Binaries

For transferring the second stage bootloader or kernel or other core binaries, the first stage bootloader needs to be modified to receive binaries over USB. For this purpose, the Device Firmware Upgrade(DFU) support in U-Boot will be enhanced. Apply the following patches on top of the u-boot and rebuild MLO and U-Boot (see Table 3). These patches provide the ability to load kernel, device tree or uboot over USB to DDR and execute them.

Table 3. DFU Patches for U-Boot 2016.05

S. No	URL	Headline
1	http://review.omapzoom.org/38423	spl: dra7xx: dfu: enable non-FIT kernel loading
2	http://review.omapzoom.org/38424	spl: dra7xx: defconfig: enable dfu_ram

The additional patches shown in Table 4 provide the ability to load the remotecores as well from MLO before starting the kernel.

Table 4. DFU Late Attach Patches for U-Boot 2016.05

S. No	URL	Headline
3	http://review.omapzoom.org/38425	spl: dfu: add support for receiving remotecore images
4	http://review.omapzoom.org/38426	spl: early boot: autoset late attach properties on dtb
5	http://review.omapzoom.org/38427	spl: dra7xx: early boot: handle dma pool when autosetting attributes
6	http://review.omapzoom.org/38428	spl: dra7xx: defconfig: enable late attach

These modifications produce a MLO that enable transfers of U-Boot/Kernel/Remotecore binaries in Peripheral Boot and functions normally in standard boot. It is necessary to use the u-boot-spl.bin built in this step for the rest of the steps in this document.

3.2.1 Testing the Modified First Stage Bootloader

To test the modified bootloader, transfer it to the EVM using the same steps as described in Section 3.1.

You will see the following messages on the serial port console:

```
U-Boot SPL 2016.05 ..
DRA722-GP ES1.0
Trying to boot from USB DFU
Using default environment
```

At this point, the first stage bootloader is waiting to receive binaries from the PC via DFU. The binaries that can be transferred to the target over DFU are listed using the command dfu-util -l.

```
host $ sudo dfu-util -l
dfu-util 0.8

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2014 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to dfu-util@lists.gnumonks.org

Found DFU: [0451:d022] ver=0223, devnum=12, cfg=1, intf=0, alt=7, name="ramdisk",
serial="UNKNOWN"
Found DFU: [0451:d022] ver=0223, devnum=12, cfg=1, intf=0, alt=6, name="dsp1", serial="UNKNOWN"
Found DFU: [0451:d022] ver=0223, devnum=12, cfg=1, intf=0, alt=5, name="ipu2", serial="UNKNOWN"
Found DFU: [0451:d022] ver=0223, devnum=12, cfg=1, intf=0, alt=4, name="dsp2", serial="UNKNOWN"
Found DFU: [0451:d022] ver=0223, devnum=12, cfg=1, intf=0, alt=3, name="ipu1", serial="UNKNOWN"
Found DFU: [0451:d022] ver=0223, devnum=12, cfg=1, intf=0, alt=2, name="fdt", serial="UNKNOWN"
Found DFU: [0451:d022] ver=0223, devnum=12, cfg=1, intf=0, alt=1, name="uboot", serial="UNKNOWN"
Found DFU: [0451:d022] ver=0223, devnum=12, cfg=1, intf=0, alt=0, name="kernel",
serial="UNKNOWN"
```

The output indicates the various binaries that can be transferred. Note that transferring U-boot can only be done when not transferring the kernel.

3.3 Transferring U-Boot

To transfer U-Boot to the EVM and start executing, run the following command on the host PC.

```
host $ sudo dfu-util -a uboot -R -D /home/user/u-boot/u-boot.img
```

The breakdown of the command line arguments is as follows:

- "-a uboot" indicates u-boot is being transferred.
- "-D /home/user/u-boot/u-boot.img" indicates the path from which the u-boot binary is being read on the host.
- "-R" indicates that MLO should exit the DFU mode after the u-boot transfer is complete.

Once MLO exits DFU mode, it checks whether the kernel or u-boot binaries were transferred. As only u-boot was transferred, it jumps to u-boot.

3.4 Transferring Kernel and Device Tree

Instead of U-Boot, jump straight to the kernel using the following commands.

```
host $ sudo dfu-util -a fdt -D "/tftp/dra7-evm-lcd-lg.dtb"
host $ sudo dfu-util -a kernel -R -D "/tftp/uImage"
```

First, the device tree is sent and then the kernel. The -R flag is used when sending the kernel so that MLO exits the DFU mode and jumps to the kernel.

Note that the kernel image is used in uImage format. This can be created from the default zImage using the following command:

```
host $ mkimage -A arm -O linux -C none  -T kernel -a 0x80008000 -e 0x80008000
-n 'Linux uImage' -d zImage uImage
```

Note that as you are booting directly to the kernel from the first stage bootloader, the kernel boot arguments need to be passed via the device tree. This could be done by modifying the device tree file before compilation or using fdtput after compilation.

```
host $ fdtput -t s "/tftp/dra7-evm-lcd-
lg.dtb" "/chosen" bootargs "elevator=noop console=ttyS0,115200n8 vram=16M fixrtc
omapdrm.num_crtc=2 consoleblank=0 root=/dev/nfs rw rootwait
nfsroot=172.24.145.235:/tfs/3_02_00_04 ip=dhcp
```

3.5 Booting Remotecores

In usecases where functionality needs to be available in less than a second (for example, rear view camera), it is typical to boot one of the remotecores (DSP1, DSP2, IPU1 and IPU2) of the DRA7xx SOC before booting Linux on the A15. This functionality is called “Early Boot - Late Attach” [4] and [5]. In the production boot flow, the binaries are read from flash. As flashing the binaries is time consuming in development, DFU functionality can be used to boot the remote cores from bootloader and verify the functionality.

The first four lines shown below show the commands necessary to transfer the binaries for the remotecores to the EVM.

```
host $ sudo dfu-util -a ipu1 -D "/tftp/dra7-ipu1-fw.xem4"
host $ sudo dfu-util -a ipu2 -D "/tftp/dra7-ipu2-fw.xem4"
host $ sudo dfu-util -a dsp1 -D "/tftp/dra7-dsp1-fw.xe66"
host $ sudo dfu-util -a dsp2 -D "/tftp/dra7-dsp2-fw.xe66"
host $ sudo dfu-util -a fdt -D "/tftp/dra7-evm-lcd-lg.dtb"
host $ sudo dfu-util -a kernel -R -D "/tftp/uImage"
```

Note that:

- You can transfer any or all or none of the remotecore binaries via DFU.
- The binaries do not start to execute until the ‘-R’ switch is invoked for the ‘dfu-util’ command.
- The remotecore binaries are first copied to temporary locations in DDR where they are parsed and then copied to their final locations.

If you face any issues with early boot of remotecore binaries, please refer to the Processors Wiki article on debugging early boot and late attach [6].

The device tree requires specific attributes to be set on the remotecore nodes to enable the attach functionality. When using DFU to transfer the binaries, these attributes are set automatically in MLO. No late attach specific device tree modifications are required during the kernel build. However, manipulation of the device tree in MLO requires free space in the device tree. Ensure that there is enough free space in the device tree by padding it using the dtc command. A padding of 4 KB is sufficient for most scenarios.

```
host $ dtc -I dtb -O dtb -o "/tftp/dra7-evm-lcd-lg.dtb" -p 4096 "/tftp/dra7-evm-lcd-lg.dtb"
```

3.5.1 Booting Baremetal Remotecore Binaries

Baremetal binaries do not have a resource table which is used by U-Boot to translate binary load addresses to physical addresses. In such a scenario, use the below patch to treat the load addresses as physical addresses.

Table 5. Patch for Loading Binaries Without Resource Table

S.No	URL	Headline
1	http://review.omapzoom.org/38666	spl: dra7xx: early boot: add function to set/print ipu ammu config
2	http://review.omapzoom.org/38665	spl: dra7xx: early boot: handle binaries without resource table

3.6 Booting Kernel, Device Tree and ramdisk

If you would like to use a ramdisk for the initial file system, the ramdisk can be transferred along with the kernel and device tree.

```
host $ sudo dfu-util -a ramdisk -D "/tftp/initramfs.cpio.gz"
host $ sudo dfu-util -a fdt -D "/tftp/dra7-evm-lcd-lg.dtb"
host $ sudo dfu-util -a kernel -R -D "/tftp/uImage"
```

When using a ramdisk, it is expected that the location of the ramdisk in DDR is indicated via the device tree. Below is a simple bash script that takes the ramdisk path and device tree path as arguments and update the device tree with the ramdisk size.

```
IFS_FULL_PATH=$1
DTB_PATH=$2
SIZE=$(du -b $IFS_FULL_PATH | cut -f1)
START_D=$(echo "obase=10; ibase=16; 83000000" | bc)
END=$(echo "ibase=10;obase=16; $START_D + $SIZE " | bc)
fdtput -v -t x $DTB_PATH "/chosen" linux,initrd-start 0x83000000
fdtput -v -t x $DTB_PATH "/chosen" linux,initrd-end "0x$END"
```

The utility fdtput can be installed as part of u-boot-tools. Assuming that above script is saved as update-dtb.sh, the following commands will update the device tree with ramdisk locations.

```
host $ sudo apt-get install u-boot-tools
host $ update-dtb.sh /tftp/initramfs.cpio.gz /tftp/dra7-evm-lcd-lg.dtb
```

3.7 Automating the Binary Loading

Instead of manually typing the DFU commands on each boot of the EVM, they can be automated using udev. We can create an udev rule to launch a script when a USB device with the vendor id and product id used by the target for DFU is detected. As an example, the below udev rule would launch the script /home/user/d-transfer.sh each time a device with vendor id “0403” and product id “bd00” is detected.

```
SUBSYSTEM=="usb",ATTRS{idVendor}=="0403",ATTRS{idProduct}=="bd00",MODE:="777",RUN+="/home/user/d
-transfer.sh"
```

The commands to be run can now be placed in the file /home/user/d-transfer.sh. Example content of the file to load DSP2 from MLO and then jump to the kernel is shown below:

```
sudo dfu-util -a dsp2 -D "/tftp/dra7-dsp2-fw.xe66”
dtc -I dtb -O dtb -o "/tftp/dra7-evm-lcd-lg.dtb" -p 4096 "/tftp/dra7-evm-lcd-lg.dtb"
sudo dfu-util -a fdt -D "/tftp/dra7-evm-lcd-lg.dtb"
sudo dfu-util -a kernel -R -D "/tftp/uImage"
```

The product id and vendor id to be used can be found from the output of ‘dfu-util -l’ command.

```
host $ sudo dfu-util -l
...
Found DFU: [0451:d022] ver=0223, devnum=12, cfg=1, intf=0, alt=0, name="kernel",
serial="UNKNOWN"
```

4 Summary

An approach to reduce the time required to update binaries during u-boot development is covered in this application report. This document also shows how the same approach can be used to load the DSP's and IPU's on DRA7xx as well as load Linux onto the A15. This approach can also be used for test automation by providing complete control of the kernel and device tree used to the host PC.

For support on this application note, please post your queries to the DRAx Infotainment SOCs E2E Forum.

https://e2e.ti.com/support/arm/automotive_processors/f/1020

5 References

1. *Overview for "Jacinto" DRAx Infotainment SoCs*
2. *Flashing Binaries to DRA7xx Factory Boards Using Device Firmware Upgrade (DFU)*
3. USB wiki
4. *DRA7xx GLSDK Software Developers Guide: Using the Late Attach Functionality*
5. *DRA7xx Processor SDK Linux Automotive Software Developers Guide: Using the Late Attach Functionality*
6. *DRA7xx Processor SDK Linux Automotive Software Developers Guide: Early Boot and Late Attach in Linux*

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Original (February 2017) to A Revision **Page**

- Moved supported SDK version to 3.02 or later 2
- Added new Section 3.1.1. 3
- Added new Section 3.1.2 showing patches to debug MLO. 4
- Added new Section 3.5.1 showing how to load baremetal DSP/IPU binaries via DFU. 7

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265