TI81XX PSP PCI Express Endpoint Boot Driver User Guide

TI81XX PCI Express Endpoint Boot Driver User Guide

Hemant Pedanekar

Introduction

This document is applicable to DM816x/AM389x, DM814x/AM387x and DM813x family of devices referred hereafter as TI816X, TI814X and TI813x respectively.

Descriptions common across these device families use the term TI81XX/ti81xx.

TI81XX devices have a PCI Express hardware module which can either be configured to act as a Root Complex or a PCIe Endpoint. When configured as PCIe Endpoint (EP), the device can be set to boot over PCIe. This document caters to the Endpoint boot operation and describes the Driver needed to configure and operate on TI81XX PCI Express Endpoint to perform booting. A sample application (part of release package) is used to carry out boot operation.

NOTE1: Support for TI814X Root Complex is added from 04.01.00.06 release, hence the references/setups described in this document related to TI814X as Root Complex are not applicable for prior TI814X releases.

NOTE2: Support for TI813X Root Complex or EP mode is currently not part of any release and is applicable only for latest kernel used from ti81xx-master branch on Arago git.

NOTE3: Various code snippets now use term `ti81xx` when referring to code common for TI81XX devices. For releases prior to 04.00.00.12 (TI816X) please consider the code snippets as having `ti816x` prefix, otherwise, refer the PDF of this user guide from respective release package.

Scope

This document covers the setup involving two TI816X/Dm814x EVMs connected together over PCIe using a PCIe cable having male connectors on both the ends. Here one device is set to boot in any other peripheral or memory boot mode (e.g., NAND) which will be designated as Root Complex (RC) and other TI81XX device configured in PCIe Boot mode (EP).

It is also possible to connect a TI816X EP device to any other PCIe RC. For the scope of this document, we will consider a x86 PC running Linux.

Note: In case of TI814X/TI813X EVM with device in PCIe boot mode, booting from TI816X or TI814X Root Complex is supported currently (post 04.01.00.06 release for TI814X RC, latest kernel on Arago for TI813X).

Also note that only 32-bit PCIe mode is supported currently on both RC and EP.

Conventions

- This document refers TI816X device set up in PCIe boot mode as EP (Endpoint) and the TI816X device acting as PCIe Root Complex as RC.
- The shell commands executed on Linux host machine as preceded with `LINUXHOST$` and the commands to be executed on TI816X RC are preceded with `ti8168-evm#`.
- It is assumed that the same Linux host machine is used as Root Complex when having PC to TI816X EVM EP connectivity.

NOTE 1: The only major difference between TI816X and TI814X/TI813X PCIe is TI816X has a x2 link while TI814X/TI813X has x1 link. Since this doesn't lead to any differences in topology and software execution impact on TI81XX Root Complex, all of the descriptions considering TI816X as Root Complex in rest of the document apply equally to TI814X/TI813X as RC as well, unless otherwise stated.

NOTE 2: Though we cover EP device as TI816X mainly, most of the description is applicable to TI814X/TI813X EP device also except for sections covering TI816X EVM modifications and booting from PC. Any other specific differences with TI816X devices are highlighted wherever applicable.

Known Issues/Restrictions (Read this before you proceed)

This section lists various constraints applicable on TI81XX PCIe EP boot operation. Most the the points listed below are covered in respective sections of this document.

- Only 32-bit PCIe boot mode is supported, 64-bit mode not supported currently.
- At the minimum, 3 BARs - BAR0, BAR1 and BAR2 are needed for boot operation and hence only the switch setting mentioned in next section is supported on EP.
- Other possible switch settings which may set up additional BARs may lead to issues in case RC is not able to allocate those BARs.
- Boot driver and application cannot handle multiple TI81XX EPs. Will only boot the first detected EP. If the setup involves combination of TI816X/TI814X/TI813X devices, TI816X device will be detected and configured for boot.
- In case ramdisk image is used as filesystem on EP, it cannot exceed 8MB. This restriction is imposed by the sample boot application to avoid exceeding the minimum BAR2 size (8MB) on TI816X.
- *Booting EP from a non-TI81XX RC is possible for* ***TI816X EP*** but will require the boot driver and application to be built on respective RC. *Note that the boot driver and application are written and tested to work with Linux kernel version 2.6.32 onwards.* So make sure that the RC is running Linux kernel as supported. This document will cover steps for building these components on a PC as RC running Linux. **In addition, you may need to do hardware modifications/configurations as described in Clocking Schemes Guide**
- Current configuration of boot application skips checking boot flag clear indication from EP U-Boot (indicating DDR setup done) and proceeds to load other images to DDR. This may fail in case EP U-Boot is yet to set up the DDR. Try increasing boot flag check delay in this case. Refer Boot Application section below.
- Similarly, EP U-Boot doesn't check for any boot flag indication from the boot application (to notify about completion of subsequent download) and may proceed to load kernel (or ramdisk or 2nd stage U-Boot where applicable) even before it is completely loaded by boot application. In such cases, you can precede 'sleep 3' (or more) in the boot script or bootcmd. Alternately, you can also tune CONFIG_BOOTDELAY in U-Boot header file for corresponding board to a higher value to delay auto booting.
- Though the boot driver supports static build (building into kernel), some filesystem changes are required if you are using Arago/OE based filesystem with udev device cache. Please refer Updating the Filesystem section for details.

Setup

This document considers following distinct setups:

1. **TI816X EVM RC - TI816X EVM EP:** This setup requires two DM8168 boards (EVMs). Ensure that serial port (UART2) on each board is connected to host terminals.
2. **TI816X EVM RC - TI814X EVM EP:** This setup requires one DM8168 EVM set up as RC while DM8148 EVM set up as EP with PCIe-32-bit boot mode. Ensure that serial port on each board (UART2 on DM8168 EVM and UART0 on DM8148 EVM) is connected to host terminals.
3. **TI814X EVM RC - TI814X EVM EP:** This setup requires two DM8148 boards (EVMs). Ensure that serial port (UART0) on each board is connected to host terminals.
4. **TI814X EVM RC - TI816X EVM EP:** This setup requires one DM8148 EVM set up as RC while DM8168 EVM set up as EP with PCIe-32-bit boot mode. Ensure that serial port on each board (UART0 on DM8148 EVM and UART2 on DM8168 EVM) is connected to host terminals.
5. **x86 PC as RC - TI816X EVM EP:** In this setup, the PC is RC and is running Ubuntu 10.04 LTS with Linux kernel 2.6.32. Ensure that serial port (UART2) on TI816X EVM used as EP is connected to host serial port.
6. **TI816X EVM RC - TI813X EVM EP:** This setup requires one DM8168 EVM set up as RC while TI813X EVM set up as EP with PCIe-32-bit boot mode. Ensure that serial port on each board (UART2 on DM8168 EVM and UART0 on TI813X EVM) is connected to host terminals.

Note: In any of the cases covered here, it is advised to power on the EP device first and then RC.

Conflict with NAND/NOR and TI814X/TI813X/TI816X PCIe Boot Mode setting

Note: This section applies when using U-Boot and/or kernel from PSP releases 04.01.00.06 (TI814X) or 04.00.01.13 (TI816X) or later. For older releases, this issue does not exist and you can skip this section.

The GPMC related boot pin CS0BW used for PCIe boot mode configuration is checked by u-boot and kernel to determine the 8-bit/16-bit NAND/NOR type connected on the board.

This will lead to conflict in few cases such as:

1. PCIe 32-bit boot mode with CS0BW=1 for BAR2 will require the DIP switch associated with CS0BW pin to be set to ON but U-Boot will take it as 8-bit NAND even if actual NAND on board was 16-bit. This will lead to NAND detection failures
2. Similar issue would occur when using 8-bit NOR where CS0BW=1 will be taken by NOR driver on Linux or U-Boot as NOR being 16-bit

Thus, with kernel and/or U-Boot from above mentioned releases, NAND and/or NOR may not be accessible when using PCIe boot mode of TI816X/TI814X/TI813X EP.

Setting up the TI816X RC

- Flash the TI816X RC on board NAND with U-Boot. Refer PSP Flashing Tools Guide document for details.
- Set the DM8168 EVM switch settings to enable NAND boot mode. Refer U-Boot User Guide for details about building U-Boot for NAND boot mode and switch settings.
- Use Male-Male (x2 or x4) PCIe cable to connect RC and PCIe slot on EP (minimum x4).
- Alternatively, you can set up RC to boot in any other (non-PCIe) boot mode.

Setting up the PC as RC

- Use Male-Male (x2 or x4) PCIe cable to connect to the PCIe slot on PC motherboard and the DM8168 Board (EP).

Note: The cable needs to be modified as described in Clocking Schemes Guide

Setting up TI816X EP

- Set the board designated as PCIe EP in PCIe 32-bit Boot mode as follows:

```
SW3[5:1] ---> BTM[4:0] = 01000 and SW3[8] ---> CS0BW = 1. Rest all switches should be '0' (zero) or 'OFF'
```

- This results into following boot pin configuration for TI816X device

```
SYSBOOT[4:0] = 01000 ==> 32-bit PCIe Boot mode
CS0BW        = 1     ==> for 8MB BAR2, in addition to default 8MB BAR1 and 4KB BAR0
```

- Note: Setting up any other combination of switch settings with PCIe Boot mode is not supported/validated.

Note: When using with non-TI816X device as RC, DM8168 EVM to be set as EP needs to be modified as described in Clocking Schemes Guide.

Taking care of PERSTn

On DM8168 EVMs the SW5 DIP switch has a switch for "PCIe RST". This corresponds to the in/out mode of the PERSTn line of the PCIe slot which in turn is tied to PCIe_PORz. The switch position 'OFF' (or '0') means the pin is set as INPUT while switch position 'ON' means the pin is in OUTPUT mode. Thus, having this switch in ON state (OUTPUT) on EP EVM OFF state (INPUT) on RC EVM will lead to conflicting configuration and the RC may not boot.

- Ensure that the SW5 "PCIe RST" switch on both EP and RC EVMs is set in OFF (INPUT) state.
- For setup involving PC as RC, it is mandatory is set SW5-1 to OFF (INPUT) state otherwise PC may not boot.
- For setup where DM8168 is RC and TI814X/TI813X device is EP, it is mandatory is set SW5-1 to OFF (INPUT) state otherwise the EP will not get detected from RC.

```
SW5[1] ---> PCIe RST = 0
```

Alternately, you can set the above switch on both the EVMs in ON state (OUTPUT).

Note: Setting SW5 PCIe RST as OFF (INPUT) on EP and ON (OUTPUT) on RC is not supported due to issue SDOCM00077550 (refer Release Notes for the release you are using)

Setting up the TI814X RC

- Flash the TI814X RC on board NAND with U-Boot. Refer PSP Flashing Tools Guide document for details.
- Set the DM8148 EVM switch settings to enable NAND boot mode. Refer U-Boot User Guide for details about building U-Boot for NAND boot mode and switch settings.
- Use Male-Male (x2 or x4) PCIe cable to connect RC and with the PCIe slot (minimum x4) on EP.
- Alternatively, you can set up RC to boot in any other (non-PCIe) boot mode.

Setting up TI814X/TI813X EP

- Set the TI814X/TI813X EVM designated as PCIe EP in PCIe 32-bit Boot mode as follows:

```
Boot mode select switch S1[5:1] = 01000 and S1[8] = 1. Rest all switches should be '0' (zero) or 'OFF'
```

- This results into following boot pin configuration for TI814X/TI813X device

```
BTMODE[[4:0]/SYSBOOT[4:0] = 01000 ==> 32-bit PCIe Boot mode
BTMODE[12]/CS0BW          = 1     ==> for 16MB BAR2, in addition to default 8MB BAR1 and 4KB BAR0
```

- **Note: Setting up any other combination of switch settings with PCIe Boot mode is not supported/validated.**

Keeping the Required Modules Ready

Following section describes various modules involved in EP boot operation and steps for preparing binaries and U-Boot script required to be present on PCIe RC to boot TI81XX EP. In addition, it is assumed that a ramdisk filesystem or NFS based filesystem for EP are available.

Note: Though the kernel (uImage) built with default config for TI81XX RC will boot on TI81XX EP, it will lead to conflicting configuration of the system since the kernel booting on EP will switch the PCIe mode from EP to RC. Due to this, the communication between PCIe RC and EP will no longer happen. For this, the kernel needs to be re-built with PCI support removed for booting on EP.

Also note that the sections below assume you have extracted the TI81XX release package to a directory referred as $PKGDIR and build environment is setup as described in TI81XX PSP User Guide.

PCIe EP Boot Driver

Note: The default kernel build configuration does not select the driver and needs to be enabled for respective kernels (e.g., TI816X, TI814X, TI813X, x86 etc) as explained in respective sections below.

This driver runs on TI81XX RC or an x86 PC running Linux kernel 2.6.32 onwards. It configures the first TI81XX EP device detected in the system and configures it to be able to carry boot operation.

Features Supported

- Support for detecting and configuring TI816X/TI814X/TI813X devices
- Provides character device interface on Linux Kernel to PCIe boot user-space application
- Provide mmap support to enable the boot application to copy image files (U-Boot, kernel etc) to EP memory
- Can be built as loadable module or into kernel

Features NOT Supported

- Operate more than one TI81XX EP. If more than one TI81XX EPs are connected in the system, this driver operates only on the first detected TI816X/TI814X/TI813X device detected in order.
- No interrupt support
- Not validated on any other RC than TI81XX and x86 PC. This driver will require porting in case you are using Linux kernel prior to 2.6.32 or any other h/w platform than TI81XX or x86 PC.
- No support for power management (e.g., suspend/resume)

Supported IOCTLs

- TI81XX_PCI_GET_BAR_INFO: Returns the size in bytes of the specified BAR.

```
int dev_desc;
dev_desc = open("/dev/ti81xx_pcie_ep", O_RDWR);
...
struct ti81xx_bar_info bar;
bar.num = bar_number;
ioctl(dev_desc, TI81XX_PCI_GET_BAR_INFO, &bar);
...
```

In the above code fragment, the driver returns BAR size in 'size' field of the 'bar' structure object on success.

- TI81XX_PCI_SET_BAR_WINDOW: Application can specify the internal address on EP for specified BAR. For example, the boot application sets BAR1 to OCMC1 start on EP (0x40400000) for TI816X EP using this ioctl.

```
...
struct ti81xx_bar_info bar;
bar.num = 1;
bar.addr = 0x40400000;
ioctl(dev_desc, TI81XX_PCI_SET_BAR_WINDOW, &bar);
...
```

- TI81XX_PCI_SET_DWNLD_DONE: Set the bootflag on EP. The driver writes '1' to the location 0x4043FFFC on TI816X EP or at address 0x4031B7FC in case of TI814X EP and waits for the flag to be cleared till maximum of the time (in seconds) as specified by the application. Timeout of '0' means no wait for the flag clearing.

```
...
ioctl(dev_desc, TI81XX_PCI_SET_DWNLD_DONE, 3);
...
```

The IOCTL and data structure declarations are in drivers/char/ti81xx_pcie_bootdrv.h file in TI81XX kernel source.

Source Files

The driver files are present at following path relative to extracted kernel source directory for TI81XX.

- drivers/char/ti81xx_pcie_bootdrv.h
- drivers/char/ti81xx_pcie_bootdrv.c

Steps for Building for TI81XX Kernel

- The steps listed below cover building details for TI816X but similar steps are applicable for building TI814X/TI813X kernel with only differences being for TI814X, you will need to use TI814X kernel release (04.01.00.06 onwards) and the EVM default configuration name is `ti8148_evm_defconfig`, while for TI813X, latest kernel from *ti81xx-master* [1] branch on Arago needs to be built with configuration `dm385_evm_defconfig`
- Navigate to TI81XX kernel source directory and extract the kernel source and create default TI8168 EVM configuration. Note that the release version suffix used in example below can be any later PSP version.

```
LINUXHOST$ cd $PKGDIR/src/kernel
LINUXHOST$ tar -xzf linux-04.00.00.09.tar.gz
LINUXHOST$ cd linux-04.00.00.09
LINUXHOST$ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm ti8168_evm_defconfig
```

- Enter the kernel configuration menu

```
LINUXHOST$ make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm menuconfig
```

- This will open the kernel configuration menu
- Use DOWN Arrow key to till "Device Drivers" is highlighted

```
    General setup  --->
[*] Enable loadable module support  --->
...
Device Drivers  --->
```

- Press ENTER to go inside this section. Use DOWN Arrow key to select "Character devices"

```
    Generic Driver Options  --->
...
    Character devices  --->
```

- Again press ENTER and scroll down to select "TI81XX PCIe Endpoint Boot Driver". Press 'm' to enable to build as a loadable module.

```
-*- Virtual terminal
...
<M> TI81XX PCIe Endpoint Boot Driver
```

Note that the above option will be shown only if PCI Bus support is enabled in kernel configuration. This is default in ti8168_evm_defconfig.

- Use RIGHT Arrow key to bring focus on 'Exit' and press ENTER. Repeat this to return to previous menu(s) and eventually back to the shell. Make sure to save the changes you have made when prompted at exit.
- Build the driver

```
LINUXHOST$ make modules.
```

- The module will be built as drivers/char/ti81xx_pcie_bootdrv.ko
- The above steps build the driver as loadable kernel module but it can also be built into kernel by pressing 'y' instead of 'm' in kernel menu configuration mentioned above.

Steps for Building for x86 Kernel

Following assumptions are made before proceeding to build the boot driver for PC:

- The PC is running x86 Linux Kernel 2.6.32 or latest (till 2.6.37)
- The source for the kernel running on the PC is available. We will assume this source is extracted in directory `/home/builduser/kernel/src` on the PC which is being used as RC.
- The native GNU toolchain installed on the PC.
- The user has 'root' or 'sudo' privileges on the PC host.

Building the boot driver as loadable kernel module:

- Transfer the boot driver files (.c and .h) to `drivers/char` folder inside kernel source directory, e.g.,

```
/home/builduser/kernel/src/drivers/char/ti81xx_pcie_bootdrv.h
/home/builduser/kernel/src/drivers/char/ti81xx_pcie_bootdrv.c
```

- Navigate to the kernel source directory

```
LINUXHOST$ cd /home/builduser/kernel/src
```

- Open the `Makefile` in `drivers/char` and add following line at the end:

```
obj-m                                   += ti81xx_pcie_bootdrv.o
```

- You may as well choose to build the driver into kernel in which case, the above line would be changed as:

```
obj-y                                   += ti81xx_pcie_bootdrv.o
```

- Build the module:

```
LINUXHOST$ make modules
```

- You might get some warning related to type casting and/or section mismatch, these can be safely ignored.
- The driver should get built as `ti81xx_pcie_bootdrv.ko` inside `/home/builduser/kernel/src` directory
- Though the above steps consider x86 kernel, same should be applicable for any 32-bit kernel (2.6.32 or later till 2.6.37).

U-Boot

For PCIe boot, the ROM on the EP directly jumps to the start of OCMC RAM when indicated by the boot driver (on RC) that the boot image is copied. **This means that the raw U-Boot binary (without any preceding header) must be used for booting.**

TI816X

For booting TI816X EP, the pre-built U-Boot binary u-boot.bin from $PKGDIR/images/u-boot/ti816x directory of TI816X release can be used.

In case you need to make any changes or add/update autoboot delay/command, follow steps mentioned in TI81XX PSP U-Boot User Guide to obtain u-boot.bin generated as part of MMC/SD image build.

TI814X

For booting TI814X EP, you will need U-Boot binaries for 2 stages:

- For 1st stage, you will need to build the U-Boot binary with following steps:

  1. Extract the U-Boot sources and navigate to the base directory of U-Boot source
  2. Make sure that the complete build environment as required for U-Boot build is setup on Linux host. For details, refer U-Boot User Guide from the release package or online here
  3. Prepare the configuration for build

```
make ti8148_evm_min_sd
```

  4. Build U-Boot

```
make
```

  5. This will create u-boot.bin file, rename this to MLO

- For 2nd stage, you can use the pre-built U-Boot binary u-boot.bin from $PKGDIR/images/u-boot/ti814x directory

Note: You can even use the pre-built MLO image from the $PKGDIR/images/u-boot/ti814x in release package provided the first 8 bytes are removed using binary editor.

TIP: Use following command at Linux shell to strip off 8 bytes -

```
# dd skip=8 if=MLO of=MLO.no_header bs=1
# mv MLO.no_header MLO
```

TI813X

For booting TI813X EP, you will need U-Boot binaries for 2 stages:

- For 1st stage, you will need to build the U-Boot binary with following steps:

    1. Extract the U-Boot sources and navigate to the base directory of U-Boot source

    2. Make sure that the complete build environment as required for U-Boot build is setup on Linux host. For details, refer U-Boot User Guide from the release package or online here

    3. Prepare the configuration for build

```
make ti813x_evm_min_sd
```

    4. Build U-Boot

```
make
```

    5. This will create u-boot.bin file, rename this to MLO

- For 2nd stage, re-configure U-Boot as

```
make ti813x_evm_config_sd
```

- Build u-boot.bin by followng command

```
make
```

PCIe EP Boot Application

This application runs on TI81XX/x86 (or any other) RC running Linux and accesses the interfaces provided by the EP Boot Driver to download U-Boot, Kernel, etc images to EP and trigger EP boot.

You need not follow the build steps mentioned below if you want to use pre-built saBootApp ELF present in $PKGDIR/images/examples/pcie directory. Note that, this ELF is built to be executed on TI81XX RC.

Features Supported

- Download U-Boot, U-Boot bootscript, Kernel and (optionally) filesystem images to EP memory
- Uses 2 stage boot loading in case of TI814X/TI813X EP -
- Requires only 3 BARs (BAR0, BAR1 and BAR2) to perform complete boot operation. Uses EP boot driver to move internal EP windows to access OCMC and DDR.

Features NOT Supported

- Cannot operate without EP Boot Driver
- Does not support ramdisk filesystem size greater than 8MB
- Kernel image size limit is 8MB and bootscript size if 2MB maximum

Steps for Building for TI81XX RC

- Navigate to TI81XX examples source directory

```
LINUXHOST$ cd $PKGDIR/examples/pcie
```

- Open file Makefile and set the path to kernel source directory as applicable. For more details, refer README file present in the directory.
- Build the application by typing 'make' at the shell. This will create an ELF file named 'saBootApp'.

Steps for Building for PC

- Note all that assumptions listed earlier for building boot driver on x86 PC are applicable.
- Create a directory for boot application source and Makefile. For simplicity, we will create a directory inside kernel source prepared earlier for building boot driver on PC.

```
LINUXHOST$ cd /home/builduser/kernel/src/
LINUXHOST$ mkdir pci
```

- Copy saBootApp.c and Makefile to this directory
- Edit Makefile to change following files

```
CC=arm-none-linux-gnueabi-gcc
KERNEL_DIR=../../linux-omap-2.6
CFLAGS=-I $(KERNEL_DIR)/include -I $(KERNEL_DIR)/arch/arm/include -I $(KERNEL_DIR)/drivers/char
```

- respectively, as follows:

```
CC=gcc
KERNEL_DIR=../
CFLAGS=-I $(KERNEL_DIR)/include -I $(KERNEL_DIR)/arch/x86/include -I $(KERNEL_DIR)/drivers/char
```

- Build the application

```
LINUXHOST$ make -C pci
```

- The boot application `saBootApp` should get built inside `pci` directory.

Tuning Boot Application

In some cases, the defaults chosen by the boot application may require to be changed. Various such options are covered below with their default settings. Note that in such case, you cannot use the pre-built Boot Application ELF file and will need to build the application as mentioned above.

Following options are defined in saBootApp.c file: -

- CONFIG_IGNORE_BOOTFLAG_FAIL: This flag is enabled by default (set to '1') which directs the boot application to proceed even if the boot flag is not cleared. This is enabled currently since present U-Boot doesn't clear the boot flag and having this flag enables the boot application to proceed to load kernel and boot EP. You can set this flag to '0' if you update the U-Boot to clear the boot flag after setting DDR (refer Boot ROM spec for location of boot flag).

- CONFIG_BOOTFLAG_FAIL_DELAY: Time in seconds to wait for boot flag to be cleared. Default is 3 sec. You can lower/increase this value if you face booting issues

Additional configuration options such as load addresses for various images, etc., are present in Boot Driver header file ti81xx_pcie_bootdrv.h located in $PKGDIR/src/kernel/linux-04.00.00.09/driver/char directory and are used by the boot application.

PCIe EP U-Boot Boot Script

In PCIe boot scenario, the U-Boot environment storage may not be available on EP. In such cases loading a U-Boot script file (referred as 'bootscript') on EP with various environment variables set helps simulating environment storage. This also adds flexibility in a way that the various boot parameters such as kernel boot arguments, filesystem type, paths for EP can be changed though this script without any need to rebuild the U-Boot or need to explicitly set the U-Boot environment variables on EP. Also, setting U-Boot parameters on EP will require to have UART console connected to EP every time.

Creating U-Boot Script

- Refer sample script file named bootscript.txt is provided in $PKGDIR/src/examples/pcie directory.
- Edit/add parameters as per your requirement. The default script sets up kernel to use RAMDISK image.
- You will need U-Boot's 'mkimage' utility to build the script image from this file. Refer U-Boot User Guide for details about 'mkimage'
- Ensure that the PATH is set to point to 'mkimage'
- Build the boot script as

```
LINUXHOST$ cd $PKGDIR/src/examples/pcie

LINUXHOST$ mkimage -A arm -O linux -T script -C none -n 'PCIe Boot Script' -d bootscript.txt boot.scr
```

- Alternatively, for booting TI816X EP, you can use pre-built boot.scr present inside $PKGDIR/images/examples/pcie directory. This is built from sample bootscript.txt

Note: The sample boot script packaged in release is for TI816X EP only and needs to be changed and re-built for TI814X/TI813X EP. Especially, ttyO2 needs to be changed to ttyO0 for TI814X/TI813X boot arguments.

Auto Booting with the Bootscript

This approach requires modifying and re-building U-Boot. The release version numbers used in examples below will vary depending upon the TI81XX PSP release being used.

Note: The current U-Boot default boot command is set to load the bootscript from MMC. Thus, pre-built u-boot.bin U-Boot binary from the release package, loaded on EP using boot application will not auto boot the kernel image downloaded over PCIe. The same issue applies if the U-Boot is built from source for SD boot. In case of TI814X/TI813X, other 1st stage U-Boot modes select autoload of 2nd stage from respective boot medium/methods, thus will bypass loading U-Boot downloaded over PCIe.

This can be resolved by directing the U-Boot running on EP to load the bootscript from DDR location 0x80400000 (as loaded by the boot application) in case of TI816X U-Boot as well as TI814X/TI813X 2nd stage U-Boot. In addition, for TI814X/TI813X where 2 stage boot loading is used, the 1st stage U-Boot needs to be directed to autoload 2nd stage U-Boot from DDR.

This can be done as described in following sections:

TI816X

- Navigate to $PKGDIR/src/u-boot directory and extract U-Boot source tarball

```
LINUXHOST$ cd $PKGDIR/src/u-boot
LINUXHOST$ tar -xzf u-boot-04.00.00.09.tar.gz
LINUXHOST$ cd u-boot-04.00.00.09
```

- Edit include/configs/ti8168_evm.h file to change `CONFIG_BOOTCOMMAND` as below:

```
#define CONFIG_BOOTCOMMAND "source 0x80400000"
```

- Build the U-Boot

```
LINUXHOST$ make CROSS_COMPILE=arm-none-linux-gnueabi- ti8168_evm_config
LINUXHOST$ make CROSS_COMPILE=arm-none-linux-gnueabi-
```

- Note that in the 1st step above, you can chose U-Boot configuration for any boot mode other than NOR boot mode.

TI814X/TI813X

Extract the U-Boot source as per the approach described above and open following file to update:

- TI814X: include/configs/ti8148_evm.h
- TI813X: include/configs/ti813x_evm.h

For 1st stage U-Boot, change `bootcmd` as part of `CONFIG_EXTRA_ENV_SETTINGS` for respective boot mode configuration being used:

E.g., for TI813X when using SD boot 1st stage U-Boot,

- Go inside the CONFIG_TI813X_MIN_CONFIG section in the above file and look for following

```
# elif defined(CONFIG_SD_BOOT)          /* Autoload the 2nd stage from SD */
```

- In CONFIG_EXTRA_ENV_SETTINGS, set the bootcmd (removing original line for bootcmd) as

```
"bootcmd=go 0x80800000\0" \
```

- Save the file and build U-Boot

```
LINUXHOST$ make clean
LINUXHOST$ make ti813x_evm_min_sd
LINUXHOST$ make
```

- Use the u-boot.bin built in the above step as 1st stage U-Boot (MLO)

For 2nd stage U-Boot, change `bootcmd` as part of `CONFIG_BOOTCOMMAND`:

E.g., for TI813X,

- Go inside the #else part of CONFIG_TI813X_MIN_CONFIG section in the EVM header file mentioned above and look for following

```
# define CONFIG_BOOTCOMMAND \
```

- Replace the #define as follows to run bootscript

```
#define CONFIG_BOOTCOMMAND "source 0x80400000"
```

- Save and build the 2nd stage for desired medium (e.g., SD)

```
LINUXHOST$ make clean
LINUXHOST$ make ti813x_evm_config_sd
LINUXHOST$ make
```

- The u-boot.bin built in above step can be used as 2nd stage U-Boot.

Note: bootcmd set explicitly in storage medium used will take precedence over the bootcmd set above in the source.

Saving Boot Command on EP Storage

This approach requires access to U-Boot console and availability of persistent storage such as NAND on EP.

In case of TI814X/TI813X, this is only possible for 2nd stage U-Boot.

- Boot U-Boot on EP using PCIe Boot procedure described in next sections
- Halt U-Boot countdown by pressing ENTER at EP console
- Set 'bootcmd' at U-Boot prompt on EP as:

```
set bootcmd 'source 0x80400000'
saveenv
```

- Note that this is a one time operation and U-Boot will jump to correct script location on subsequent resets.

Building Kernel for EP

Note: The change mentioned below is not needed for TI814X when using release 04.01.00.05 or older and the kernel uImage built with ti8148_evm_defconfig can be used. For 04.01.00.06 and onwards release of TI814X kernel, the steps mentioned below should be followed since they have Root Complex support included. For TI813X, the steps below are only needed if you use latest kernel from Arago.

As mentioned earlier in this section, the kernel built with PCIe RC support will result into conflicting configuration when booted on EP. To avoid this, it is recommended to rebuild the kernel with PCIe RC support disabled.

- As explained before, extract kernel source tarball and enter kernel configuration menu.

```
    General setup  --->
[*] Enable loadable module support  --->
-*- Enable the block layer  --->
   System Type  --->
   Bus support  --->
...
```

- Scroll down with the DOWN Arrow key till "Bus Support" gets selected. Press ENTER. This will show following menu:

```
[*] PCI support
[*] Message Signaled Interrupts (MSI and MSI-X)
...
```

- Disable "PCI support" by pressing 'n' key or SPACE key. This will automatically hide other dependent configurations and the above configuration menu will look like as shown below:

```
[ ] PCI support
< > PCCard (PCMCIA/CardBus) support  --->
```

- Use RIGHT Arrow key to bring focus on 'Exit' and press ENTER. Repeat this to return to previous menu(s) and eventually back to the shell. Make sure to save the changes you have made when prompted at exit.

- Build the kernel.

Updating the Filesystem

The kernel may not boot or provide shell prompt when the boot driver is built into kernel. This issue is due to the device node numbers (major numbers) used for console device(s) not getting updated as udev in the filesystem still uses stale nodes from the device cache in the filesystem even on detection or removal of EP device across RC reboots.

This issue can be avoided by forcing the rebuild of device cache on detecting change in boot time device detection. This can be achieved by updating the filesystem as follows:

- Add following check in /etc/rcS.d/S03udev inside DEVCACHE block

```
[ -r /proc/devices ] && cat /proc/devices > /tmp/devices || touch /tmp/devices
```

- Similarly, add comparison for saved device list to detect changes in devices

```
cmp -s /tmp/devices /etc/udev/saved.devices && \
```

- Now the devcache block will look as follows

```
if [ "$DEVCACHE" != "" ]; then
        # Invalidate udev cache if the kernel or its bootargs/cmdline have changed
        [ -x /bin/uname ] && /bin/uname -mrspv > /tmp/uname || touch /tmp/uname
        [ -r /proc/cmdline ] && cat /proc/cmdline > /tmp/cmdline || touch /tmp/cmdline
        [ -r /proc/devices ] && cat /proc/devices > /tmp/devices || touch /tmp/devices
        [ -r /proc/atags ] && cat /proc/atags > /tmp/atags || touch /tmp/atags
        if [ -e $DEVCACHE ] && \
           cmp -s /tmp/uname /etc/udev/saved.uname && \
           cmp -s /tmp/cmdline /etc/udev/saved.cmdline && \
           cmp -s /tmp/devices /etc/udev/saved.devices && \
           cmp -s /tmp/atags /etc/udev/saved.atags; then
                (cd /; tar xf $DEVCACHE > /dev/null 2>&1)
                not_first_boot=1
        fi
fi
```

- Update udev devcache script /etc/rcS.d/S12udev-cache to save device list by adding following line

```
mv /tmp/devices /etc/udev/saved.devices
```

- Update the same file with a line to delete temporary device list

```
rm -f /tmp/devices
```

- Now the dev cache code in this file will look like

```
if [ "$DEVCACHE" != "" ]; then
        echo -n "Populating dev cache"
        (cd /; tar cf $DEVCACHE dev)
        mv /tmp/uname /etc/udev/saved.uname
        mv /tmp/cmdline /etc/udev/saved.cmdline
        mv /tmp/atags /etc/udev/saved.atags
        mv /tmp/devices /etc/udev/saved.devices
        echo
else
        rm -f /tmp/uname
        rm -f /tmp/cmdline
        rm -f /tmp/atags
        rm -f /tmp/devices
fi
```

- The above changes will ensure that udev will rebuild the device cache depending upon detection of PCIe EP presence or removal across RC reboots.

Other simpler (but not recommended) option is to turn of the device cache altogether. This can be done by commenting the DEVCACHE line in /etc/default/udev of the filesystem.

Updating the RC Kernel

```
Note: You will need RC kernel source to follow this section.
```

If you are using a non-TI81XX Root Complex device, you will need to patch the kernel to assign a valid class code to the TI816X EP detected. This is required because, TI816X devices configured to boot as PCIe Endpoint have class code = 0. This makes kernel PCI bus code to skip allocating BARs to these devices resulting into following type of error when trying to enable them:

```
"Device 0000:01:00.0 not available because of resource collisions"
```

The device cannot be operated because of the above issue.

Following patch would be required to be applied on RC kernel. Copy and save the text below in a file and apply patch using `patch -p 1 -i <filename>` command (replace <filename> with the name of the file where the patch is saved) in the RC kernel source tree.

```
diff --git a/drivers/pci/quirks.c b/drivers/pci/quirks.c
index 53a786f..2087f3d 100644
--- a/drivers/pci/quirks.c
+++ b/drivers/pci/quirks.c
@@ -2791,6 +2791,17 @@ DECLARE_PCI_FIXUP_EARLY(PCI_VENDOR_ID_INTEL, 0x342e, vtd_mask_spec_errors);
 DECLARE_PCI_FIXUP_EARLY(PCI_VENDOR_ID_INTEL, 0x3c28, vtd_mask_spec_errors);
 #endif

+static void __devinit fixup_ti81xx_class(struct pci_dev* dev)
+{
+       /* TI 81xx devices do not have class code set when in PCIe boot mode */
+       if (dev->class == PCI_CLASS_NOT_DEFINED) {
+               dev_info(&dev->dev, "Setting PCI class for 81xx PCIe device\n");
+               dev->class = PCI_CLASS_MULTIMEDIA_VIDEO;
+       }
+}
+DECLARE_PCI_FIXUP_EARLY(PCI_VENDOR_ID_TI, 0xb800, fixup_ti81xx_class);
+DECLARE_PCI_FIXUP_EARLY(PCI_VENDOR_ID_TI, 0xb801, fixup_ti81xx_class);
+
 static void pci_do_fixups(struct pci_dev *dev, struct pci_fixup *f,
                          struct pci_fixup *end)
 {
```

Note 1: You may need to manually resolve conflicts if RC kernel version is not 2.6.37.
Note 2: The above patch is not required to be applied if the RC is TI81XX and kernel used is from release 04.00.00.09 or onwards since it already comes with above patch applied.

Note: The above mentioned patch is needed only when you are relying on resource assignment from kernel. If the kernel is just using BIOS allocated resources (which is normal case on PCs), you need not apply this patch. Of course your device will be labeled as "unclassified" device when using a kernel without this patch.

For example:

```
LINUXHOST$ sudo lspci -v
...
 01:00.0 Non-VGA unclassified device: Texas Instruments Device b800 (rev 01)
    Flags: bus master, fast devsel, latency 0, IRQ 11
    Memory at fe9ff000 (32-bit, non-prefetchable) [size=4K]
    Memory at f0800000 (32-bit, prefetchable) [size=8M]
    Memory at f0000000 (32-bit, prefetchable) [size=8M]
    Memory at <unassigned> (32-bit, prefetchable)
    Capabilities: [40] Power Management version 3
    Capabilities: [50] Message Signalled Interrupts: Mask- 64bit+ Queue=0/0 Enable-
    Capabilities: [70] Express Endpoint, MSI 00
    Capabilities: [100] Advanced Error Reporting <?>
...
```

Ensuring the PCIe System Initialization

Before proceeding to using the TI81XX PCIe EP in the system, we must ensure that the RC is up and has detected and configured the PCIe EP. Steps mentioned below should be carried out before actually accessing the EP and proceeding for boot.

- Make sure that the board setup instructions mentioned in earlier section are followed.
- Power on EP
- Power on RC
- Once shell is available at RC, type 'lspci' command, it should show following two devices (EP in this example is TI816X):

```
ti8168-evm# lspci -v
00:00.0 Class 0604: Device 104c:8888 (rev 01)
        Flags: bus master, fast devsel, latency 0
        Memory at <ignored> (32-bit, non-prefetchable)
        Memory at <ignored> (32-bit, prefetchable)
        Bus: primary=00, secondary=01, subordinate=01, sec-latency=0
        Memory behind bridge: 21000000-210fffff
        Prefetchable memory behind bridge: 20000000-20ffffff
        Capabilities: [40] Power Management version 3
        Capabilities: [50] MSI: Enable- Count=1/1 Maskable- 64bit+
        Capabilities: [70] Express Root Port (Slot-), MSI 00
        Capabilities: [100] Advanced Error Reporting
```

```
01:00.0 Class 0004: Device 104c:b800 (rev 01)
        Flags: fast devsel, IRQ 48
        Memory at 21000000 (32-bit, non-prefetchable) [size=4K]
        Memory at 20000000 (32-bit, prefetchable) [size=8M]
        Memory at 20800000 (32-bit, prefetchable) [size=8M]
        Memory at <unassigned> (32-bit, prefetchable)
        Capabilities: [40] Power Management version 3
        Capabilities: [50] MSI: Enable- Count=1/1 Maskable- 64bit+
        Capabilities: [70] Express Endpoint, MSI 00
        Capabilities: [100] Advanced Error Reporting
```

- If 'lspci' is not available in your filesystem, look into `/sys/bus/pci/devices` directory to see if EP is detected. You should see two sub directories in this directory, e.g., see following command and its output:

```
cat /sys/bus/pci/devices/0000\:0*/device
 0x8888
 0xb800
```

- The first device is RC itself and the second device is EP
- Note that, in case the x86 PC is used as RC, the "lspci" would only show TI816X EP along with other PCI/PCIe devices connected to PC motherboard.
- Notice that the EP is allocated three BARs - BAR0, BAR1 and BAR2 of size 4KB, 8MB and 8MB respectively.
- In case the EP is TI814X/TI813X device, BAR2 size will be 16MB

Load Addresses

The boot application loads various images at following addresses on EP:

Image	**EP Device**	**Load Address**
U-Boot 1st Stage	TI814X/TI813X	0x40300000
U-Boot	TI816X	0x40400000
	TI814X/TI813X (2nd Stage)	0x80800000
Kernel	TI816X	0x80900000
	TI814X/TI813X	
Ramdisk	TI816X	0x81000000
	TI814X/TI813X	
Bootscript	TI816X	0x80400000
	TI814X/TI813X	

Booting EP

- Download PCIe EP Boot driver, U-Boot, U-Boot script, kernel image and ramdisk file (optional) on RC.
- These files can be downloaded over network using TFTP or can be copied to NFS mounted filesystem
- In our example, we will use following commands on RC:

```
ti8168-evm# tftp -m binary 192.168.0.1 -c get ti81xx_pcie_bootdrv.ko
ti8168-evm# tftp -m binary 192.168.0.1 -c get saBootApp
ti8168-evm# tftp -m binary 192.168.0.1 -c get u-boot.bin
ti8168-evm# tftp -m binary 192.168.0.1 -c get boot.scr
ti8168-evm# tftp -m binary 192.168.0.1 -c get uImage
ti8168-evm# tftp -m binary 192.168.0.1 -c get ramdisk.gz
```

Note: Additionally, transfer the 1st stage U-Boot MLO used for MMC/SD boot in case of TI814X/TI813X EP.

- Make boot application as executable:

```
ti8168-evm# chmod a+x saBootApp
```

- Insert boot driver:

```
ti8168-evm# insmod ti81xx_pcie_bootdrv.ko
 ti81xx_pcie_ep: Found TI816x PCIe EP @0xc781ec00
 pci 0000:01:00.0: This driver supports booting the first TI816x or TI814x target found on the bus
 pci 0000:01:00.0: Major number 253 assigned
 pci 0000:01:00.0: Added device to the sys file system
 pci 0000:01:00.0: BAR Configuration -
            Start         |       Length  |       Flags
 pci 0000:01:00.0:       0x21000000      |       4096    |       0x00040200
 pci 0000:01:00.0:       0x20000000      |       8388608 |       0x00042208
 pci 0000:01:00.0:       0x20800000      |       8388608 |       0x00042208
 pci 0000:01:00.0: TI81XX registers mapped to 0xc8886000
 pci 0000:01:00.0: TI81XX OCMC1 mapped to 0xca000000
 pci 0000:01:00.0: TI81XX DDR mapped to 0xcb000000
```

Note: Due to an errata on PCIe ROM boot mode of TI813X devices, the execution on EP needs to be reset (from RC) before carrying out normal boot process. Since the TI813X EP comes up with device ID same as TI814X, the EP driver cannot distinguish between TI814X and TI813X and user needs to pass module parameter `eprst=1` in case TI813X PCIe ROM boot mode is used. This will direct the driver to execute errata workaround to reset EP.

E.g., the `insmod` command on RC when using TI813X EP in ROM PCIe boot mode should then be,

```
 ti8168-evm# insmod ti81xx_pcie_bootdrv.ko eprst=1
```

- Or, if the Linux PC is set up as as RC (assuming the source directory mentioned in earlier sections is used):

```
LINUXHOST$ cd /home/builduser/kernel/src/
LINUXHOST$ sudo insmod drivers/char/ti81xx_pcie_bootdrv.ko
```

- The above log shows that three BARs on the EP are set up with 4KB, 8MB and 8MB size respectively.
- You may need to create node "/dev/ti81xx_pcie_ep" with the major number displayed when PCIe Boot Driver module is inserted. This depends on the filesystem and configuration, e.g., 'udev' should do this automatically.
- In case the device node is not created automatically, use following command sequence:

```
ti8168-evm# cat /proc/devices
 Character devices:
    1 mem
    :
    :
    253 ti81xx_pcie_ep            <--- Our device major number is 253
    :
ti8168-evm# mknod /dev/ti81xx_pcie_ep c 253 0
```

- Run the boot application providing U-Boot, bootscript, kernel and ramdisk names as:

```
ti8168-evm# ./saBootApp u-boot.bin boot.scr uImage ramdisk.gz
```

- For TI814X/TI813X EP, ensure that MLO binary is present in same directory as boot application
- Or on PC (assuming all required binaries are present in current directory):

```
LINUXHOST$ sudo ./saBootApp u-boot.bin boot.scr uImage ramdisk.gz
```

- Alternatively, you can use NFS root system on EP by updating the bootscript.txt accordingly and using the corresponding bootscript image. In this case, you need not download the ramdisk image.

```
ti8168-evm# ./saBootApp u-boot.bin boot.scr uImage
```

- The boot application will use boot driver to configure the EP and download the images to complete boot operation.

Note: You will see some messages about "Bootflag clear timed out" during the boot process. This is normal since the current U-Boot doesn't support clearing the boot flag in OCMC1 RAM at EP (location 0x40403FFC in case of TI816X EP and 0x4031b7fc for TI814X/TI813X) as set by the PCIe Boot code running on RC. The application will still proceed to complete boot operation.

References

[1] http://arago-project.org/git/projects/?p=linux-omap3.git;a=shortlog;h=refs/heads/ti81xx-master

Article Sources and Contributors

TI81XX PSP PCI Express Endpoint Boot Driver User Guide *Source*: http://processors.wiki.ti.com/index.php?oldid=119725 *Contributors*: Hemantp, Kevinsc, RK

Image Sources, Licenses and Contributors

Image:TIBanner.png *Source*: http://processors.wiki.ti.com/index.php?title=File:TIBanner.png *License*: unknown *Contributors*: Nsnehaprabha

License