Application Report
SPRACN3–September 2019

Using ARM ROM Bootloader on Keystone II Devices

Catalog Processors

ABSTRACT

The application report is a comprehensive technical manual to leverage the ROM bootloader to boot applications or secondary bootloaders on Keystone II devices like 66AK2H14 and 66AK2E05. This document provides boot examples along with description of boot utilities and setting up TI supported evaluation platforms to test the sample software.

Additionally, some of the frequently asked questions are addressed around this topic along with additional references to Processor SDK in the SDK documentation on ti.com to design your system boot.

The document is meant to be used as an addendum document to *ARM Bootloader User Guide for KeyStone II Devices*. The Processor SDK also provides secondary bootloaders that enable developers to enable two stage application booting, which allows for customization of the system boot beyond the capabilities of the ARM® ROM bootloader

Contents

List of Figures

List of Tables

Trademarks

1 Keystone2 Boot loader Overview

Keystone II bootloader overview and details are covered in the collateral and training slides mentioned below:

- *KeyStone II Architecture ARM Bootloader User's Guide*
- Keystone Boot loader Training slides

It is recommended that you review the material provided above before getting started with building the boot examples.

2 Boot Examples Package Download

GIT Repo for Boot Examples

In order to download the software, execute the following in the git bash environment:

```
git clone git@git.ti.com:keystone2_boot_examples/keystone2_boot_examples.git boot_examples
```

3 Software Dependencies

Processor SDK RTOS or MCSDK 3.x (Legacy SDK)

MinGW for Windows with msys and make tools installed. OR Linux® machine with Ubuntu 12.04 LTS with wine utility.

TI Arm compiler (typically packaged with Code composer Studio).

4 Supported Hardware

- K2H EVM rev 1.1 and later
- K2E EVMs rev 1.0.2.0 and later

NOTE: Ensure that the EVM has the latest UCD and BMC updates as described in the Setup Hardware section.

5 Software Features

- Boot Utilities to create and format boot images
- Examples that demonstrate booting K2H and K2E devices from Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C), Universal Asynchronous Receiver/Transmitter (UART), Ethernet and NAND. Examples also demonstrate the following features of the BootROM
  - Single and Multi-Stage booting
  - Using Boot Parameter tables to speed up booting from a boot media
  - Using Boot Configuration tables to initialize DDR
- Examples to demonstrate formatting uboot for above mentioned boot modes

6 Directory Structure

Figure 1 shows the directory structure for the software package. The examples directory is organized to follow the convention.

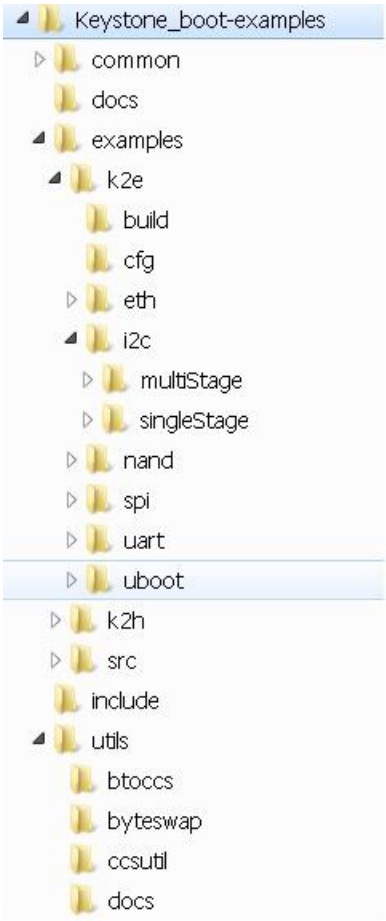

Figure 1. Boot Example Directory Structure

```
Device Name (k2h, k2e) ---> Boot Mode (eth, i2c,nand,spi, uart) ---
> Example Type (singleStage, multiStage).
```

Detailed description of the directory structure is given below:

- docs - directory contains ReadMe documents and the software manifest for the package.
- common - directory contains source and header files for helper functions that are used by the examples to configure MMU, CACHE, PLLs, UART and to set user defined entry point for their applications.
- include - directory contains all boot header files that applies to all the devices in the Keystone family.
- examples - directory contains source for single stage and multi stage boot examples for different boot modes.
  - k2e - directory contains boot examples for K2E devices.
    - build - directory contains linker command files that specify the memory configuration for each examples
    - cfg - directory contains header files that contain ddr configuration tables, boot parameter tables for different boot modes and tiboot.h file defined in the BootROM.
    - [peripheral] - directory contains boot example code for each peripheral (eth, i2c,nand,spi, uart)
      - [example type] - directory contains single stage/mulitstage example binaries.
    - uboot - directory contains build files required to create uboot binaries for different boot media.

  - k2h - directory contains boot examples for K2H devices.
    - build - directory contains linker command files that specify the memory configuration for each examples
    - cfg - directory contains header files that contain ddr configuration tables, boot parameter tables for different boot modes and tiboot.h file defined in the BootROM.
    - [peripheral] - directory contains boot example code for each peripheral (eth, i2c,nand,spi, uart)
      - [example type] - directory contains single stage/multi-stage example binaries.
    - uboot - directory contains build files required to create uboot binaries for different boot media.
  - src - directory contains source for sample applications that demonstrate single stage and multistage booting.
    - singleStage: directory contains source files for Single stage boot example
    - mulitStage: directory contains source files for two stage boot example
- utils - directory contains the utilities for building boot images
  - byteswap : directory contains the source and binaries for byteswapccs utility.
  - btoccs : directory contains source and binaries for b2ccs,catccs, ccs2bin utilities
  - ccsutil: directory contains source and binaries for ccsAddGphdr,ccsAddGptlr,ccspad utilities

Source files:

- Stage1.c
- Stage2.c

Configuration files:

- paramTables.h Boot Parameter table used for multi Stage boot examples are provided in this file.
- ddrConfigTable.h: DDR configuration table used in examples initialize are provided in this file.
- tiboot.h: BootROM header file that contains details of boot parameter structures, configuration tables and BOOTROM call table.

7 Building the Examples

1. Setting up host environment.

   For Windows Environment: The top level directory in the software package contains a file setupMsysEnv that is used to set the path to the environment variables required to build the examples. Set the path to MinGW installation (TOOL_MINGW_DST), TI Arm compiler (ARM_COMPILER_FOLDER) and the platform development package (PDK_PACKAGES) and optionally to the uboot source directory.

   NOTE: PDK_packages must point to path to the packages folder in the PDK directory.

   For Linux Environment: The top level directory in the software package contains a file setupLinuxEnv that is used to set the path to the environment variables required to build the examples. Set the path to TI Arm compiler (ARM_COMPILER_FOLDER) and the platform development package (PDK_PACKAGES) and optionally to the uboot source directory.

   NOTE: For Linux environment, using the boot utilities requires wine utility. If you do not have this installed, install the utility by executing following instruction on your Ubuntu machine.

   ```
   sudo apt-get install wine
   ```

   PDK_packages must point to path to the packages folder in the PDK directory.

2. In order to build the examples for all the supported keystone platforms, you can execute "make all". In order to build uboot binaries in addition to boot examples, you can execute "make k2h_uboot" or "make k2e_uboot".

   Top level make file supports the following targets:

```
k2h_examples : Builds examples for K2H devices
k2e_examples : Builds examples for K2E devices
utils : Builds boot utlities
k2h_uboot : Builds uboot binaries for K2H devices
k2e_uboot : Builds uboot binaries for K2E devices
```

NOTE: In Linux environment, if you see errors related to environment variables, check your path to the tools. If that is accurate, ensure that you have saved the file in unix format.

```
:set fileformat=unix
:wq
```

8 Description of the Examples

The software packages contains two types of examples single stage boot examples and multistage boot examples.

8.1 Single Stage Boot Examples

Single stage boot examples demonstrate booting the K2 devices directly from the boot media specified by the boot mode pins using the default settings specified by the boot parameter loaded by the BootROM. For default boot parameter settings used by the bootROM, see the device-specific data manual. The sample application is designed to wake up the secondary Arm core upon boot and then initializes the UART to print message when each core wakes up. Each Arm core will send a message over UART: "Core n standing by..." where n is the Arm core number.

8.2 Multi-Stage Boot Example

The multistage boot example demonstrates two stage booting the K2 devices using functions in BootROM call table. The first stage is loaded from the boot media specified by the boot switches using default bootROM parameter tables. The first stage code replaces the default fields in the boot parameter table and re-enters boot to load the second stage. This process is used in-order to modify the default boot process in cases where you need to speed up the boot speeds, boot from a different offset in the flash memory, load code into DDR memory or to perform time critical application specific initialization that may be required without waiting for the entire application to load. The second stage initializes wakes up secondary Arm core and initializes the UARTs and prints messages when each Arm core wakes up.

For details regarding the organization of the boot image for specific boot mode, see the makefile in the path examples/<device>/<peripheral>/multiStage.

8.3 Boot Media-Specific Details

8.3.1 SPI Boot Example

SPI boot is a general-purpose boot mode in which the Boot ROM configures the PLL in bypass mode. The Boot ROM loads the image from the base of the SPI NOR flash. In the multi-stage boot example, the first stage loads the boot parameter table that forces the Boot ROM to configure the PLL and load the seconds stage at 100 Khz. The second stage is loaded at an offset of 0x1000 in the SPI NOR Flash.

8.3.2 I2C Boot Examples

I2C boot is a general-purpose boot mode in which the Boot ROM configures the PLL in bypass mode. The Boot ROM loads the image from the base of the I2C EEPROM. In the multi-stage boot example, the first stage loads the boot parameter table that forces the Boot ROM to configure the PLL and load the seconds stage at 100 Khz. The second stage is loaded at an offset of 0x2000 in the I2C EEPROM connected on bus addr 50 on the EVM.

8.3.3 NAND Examples

NAND boot is a general-purpose boot mode in which the Boot ROM configures the PLL based on the reference clock setting provided from the boot switches. The Boot ROM loads the image from the first page of block 1. The example uses a DDR configuration table to initialize the DDR memory. The current package does not provide a multi-stage example as the boot mode does not require initialization of PLLs or DDR configuration.

8.3.4 UART Boot Examples

UART boot is a blob boot in which the Boot ROM configures the PLL based on the reference clock setting provided from the boot switches. The boot ROM configures the UART to the baud rate of 115200 bps and sends a ping character to indicate the device is ready to accept the image. The device loads the blob boot image at the base of the MSMC. On K2E devices, you have the option to change the load address for the boot images but on K2H the second stage will also need to be loaded at the base of MSMC. It was also observed that the you need to enable MMU table on the Arm on blob boot modes.

NOTE: On K2H devices, the boot ROM does not invalidate the Instruction cache so the example on K2H loads the two stages in non-overlapping memory. However, the first instruction that redirects the device to the entry point of the application needs to execute from the base of MSMC, so the instruction cache in the first stage needs to also be invalidated. To create non-overlapping 2 stages, use a 1K random memory segment in the first stage.

8.3.5 Ethernet boot examples

Ethernet boot is a blob boot in which the Boot ROM configures the PLL based on the reference clock setting provided from the boot switches. The boot ROM configures the PHY and booot switch and sends a bootp packet to indicate it is ready to receive the boot image. You need to run a dhcp server on the host to load the boot image on the target K2 device. The device loads the blob boot image is loaded at the base of MSMC. On K2E devices, you have the option to change the load address for the boot images but, on K2H the second stage will also need to be loaded at base of MSMC. It was also observed that the users need to enable MMU table on the Arm on blob boot modes.

NOTE: On K2H devices, the boot ROM does not invalidate the Instruction cache so the example on K2H loads the 2 stages in non-overlapping memory. However, the first instruction that redirects the device to the entry point of the application needs to execute from the base of MSMC, so the instruction cache in the first stage needs to also be invalidated. To create non-overlapping 2 stages, use a 1K random memory segment in the first stage.

8.3.6 K2E Ethernet Boot Errata Workaround

On K2E devices, there is an errata associated with ethernet boot that prevents the device to directly boot the ethernet. For more information, see the KeyStoneII.BTS_errata_advisory.25 in the K2E device errata here. The example included in the path examples/k2e/eth/multistage shows how to apply the fix to the Ethernet ROM code and PHY on the K2E EVM. The first stage is built by default for UART and I2C , but the makefile can be modified to rebuild the image into one that can be loaded onto NAND or NOR. The first stage is designed to apply the fix. After this image is loaded and finished executing, the device transmits the BOOTP packets at a regular intervals, which uses the MAC ID of the device to load an image over ethernet.

The PHY workaround code in ethWard.c can be safely removed if this example is not running on the K2E/K2L EVM. This code starts with the comment "EVM PHY Workaround" and ends at "End of EVM PHY Workaround." The function call 'setup_Marvell_Phy(1);' in main() can also be removed, if desired.

8.4 Flashing and Running Boot Examples

8.4.1 Dip Switch Settings

- No Boot: SW1 (pin1, pin2, pin3, pin4): off, off, off, on
- UART: SW1 (pin1, pin2, pin3, pin4): off, on, off, off
- NAND: SW1 (pin1, pin2, pin3, pin4): off, off, off, off
- SPI: SW1 (pin1, pin2, pin3, pin4): off, off, on, off
- I2C: SW1 (pin1, pin2, pin3, pin4): off, off, on, on
- Ethernet: SW1 (pin1, pin2, pin3, pin4): off, on, off, on

8.4.2 Running I2C EEPROM example

- Location of boot image binaries:
  - examples/<k2 device>/i2c/singleStage/bin/i2cImage.dat
  - examples/<k2 device>/i2c/multiStage/bin/i2cImage.dat
- I2C Flash Writer software:

  The I2C EEPROM flash writer can be located in MCSDK under the path \mcsdk_bios_3_01_00_01\tools\writer\eeprom\evmk2X. In order to refer to the detailed documentation of the flash writer, see the ReadMe document included in mcsdk_bios_3_01_00_01\tools\writer\eeprom\docs.
- Flashing the I2C boot examples:
  1. Copy the boot binary (i2cImage.dat) to the \mcsdk_bios_3_01_00_01\tools\writer\eeprom\evmk2X\bin folder and rename the file to app.dat.
  2. Open the file eepromwriter_input and modify the parameters as shown below:

     ```
     file_name = app.dat
     bus_addr = 0x50
     start_addr = 0
     swap_data = 1
     ```
  3. Configure the EVM to "No Boot mode" as shown in Section 8.4.1 and connect to the EVM using and emulator and Code Composer studio.

     NOTE: Details to connect to the EVM using Code Composer Studio™ (CCS) are discussed in the Hardware Setup Guide.
  4. Connect to and initialize the Arm core using the Device GEL file found in the emualation pack found here.
  5. Connect to the C66x core and load the eepromwriter_evmk2e.out on the device.
  6. Go to the "View" Menu and open a memory browser and select "Load Memory" in the memory browser. Browse to the boot image and select "Use the file header to set the start address and size of the memory block to be loaded.

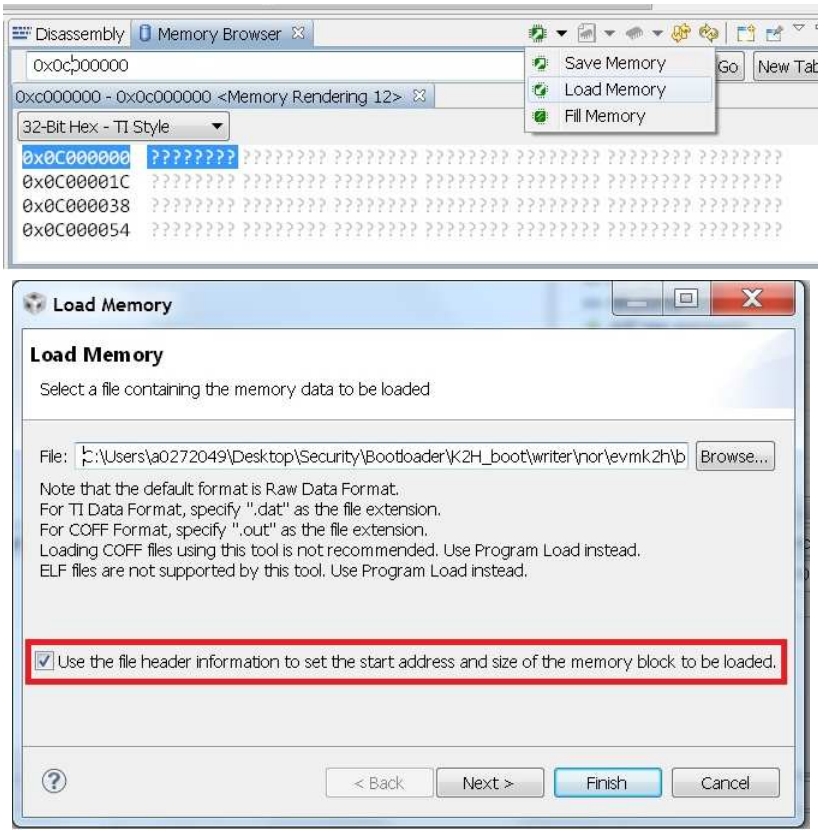

Figure 2. CCS Load Memory Option in Memory Browser

7. Ensure the start address to base of MSMC (0xc000000) and click finish.

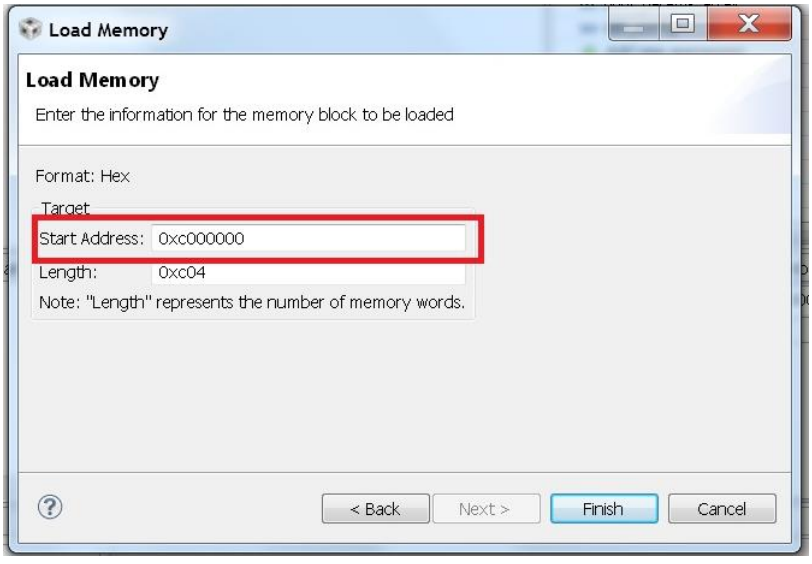

Figure 3. CCS Load Memory With I2C Boot Binary

8. Run the eepromwriter_evmk2e.out on the C66x core to flash the binary to the EEPROM on the EVM.

- Running and verifying the I2C boot:
  1. Set the EVM switches to I2C boot as shown in Section 8.4.1 and connect the serial terminal to host and configure the UART Connect to 115200 baud rate.
  2. Power on the EVM. When the boot completes successfully, the device puts out UART message indicating the cores have woken up and are standing by.

  The following messages will be displayed on the terminal window:

```
Core 0 standing by....
Core 1 standing by....
Core 2 standing by....
Core 3 standing by....
```

8.4.3 Running SPI NOR Example

- Location of Boot image binaries:
  - examples/<k2 device>/spi/singleStage/bin/spiImage.dat
  - examples/<k2 device>/spi/multiStage/bin/spiImage.dat
- SPI NOR Flash Writer software:

  SPI NOR flash writer can be located in MCSDK under the path \mcsdk_bios_3_01_00_01\tools\writer\nor\evmk2X. In order to refer to detailed documentation of the flash writer, see the ReadMe document included in mcsdk_bios_3_01_00_01\tools\writer\nor\docs.
- Flashing the SPI NOR boot examples:
  1. Copy the boot binary (spiImage.dat) to the \mcsdk_bios_3_01_00_01\tools\writer\nor\evmk2X\bin folder and rename the file to app.dat.
  2. Open the file nor_writer_input.txt and modify the parameters as shown below:

     ```
     file_name = app.dat
     start_addr = 0
     ```
  3. Configure the EVM to "No Boot mode" as shown in Section 8.4.1 and connect to the EVM using and emulator and Code Composer studio.

     NOTE: Details to connect to the EVM using CCS are discussed in the Hardware Setup Guide.
  4. Connect to and initialize the Arm core using the Device GEL file found in the emulation pack found here. Configure the 1 Ghz and DDR configuration to 1333 Mhz.
  5. Connect to C66x core and load the norwriter_evmk2X.out on the device.
  6. Go to the "View" menu; open and select "Load Memory" in the memory browser. Browse to the boot image and select "Use the file header to set the start address and size of the memory block to be loaded. For reference on loading memory, see Figure 2 and Figure 3.
  7. Ensure the start address to base of DDR (0x80000000) and click finish.
  8. Run the norwriter_evmk2X.out on the C66x core to flash the binary to the SPI NOR on the EVM.
- Running and verifying the SPI boot:
  1. Set the EVM switches to SPI boot as shown in Section 8.4.1 and connect the serial terminal to host and configure the UART Connect to 115200 baud rate.
  2. Power on the EVM. When the boot completes successfully, the device puts out the UART message indicating the cores have woken up and are standing by.

     The following messages will be displayed on the terminal window:

     ```
     Core 0 standing by....
     Core 1 standing by....
     Core 2 standing by....
     Core 3 standing by....
     ```

8.4.4 Running NAND Example

- Location of boot image binaries:
  - examples/<k2 device>/nand/singleStage/bin/nandImage.dat
  - examples/<k2 device>/nand/multiStage/bin/nandImage.dat
- NAND Flash Writer software:

  The NAND flash writer can be located in MCSDK under the path \mcsdk_bios_3_01_00_01\tools\writer\nand\evmk2X. In order to refer to detailed documentation of the flash writer, see the ReadMe.txt document included in mcsdk_bios_3_01_00_01\tools\writer\nand\docs.
- Flashing the NAND boot examples:
  1. Copy the boot binary (spiImage.dat) to the \mcsdk_bios_3_01_00_01\tools\writer\nand\evmk2X\bin folder and rename the file to app.dat.
  2. Open the nand_writer_input.txt file and modify the parameters as shown below:

     ```
     file_name = app.dat
     start_addr = 0
     rbl_ecc = 1
     skip_bad = 1
     ```
  3. Configure the EVM to "No Boot mode" as shown in Section 8.4.1 and connect to the EVM using and emulator and Code Composer studio.

     NOTE: Details to connect to the EVM using CCS are discussed in the Hardware Setup Guide.
  4. Connect to and initialize the Arm core using the Device GEL file found in the emulation pack found here. Configure the 1 Ghz and DDR configuration to 1333 Mhz.
  5. Connect to the C66x core and load the nandwriter_evmk2X.out on the device.
  6. Go to "View" menu and select "Load Memory" in the memory browser. Browse to the boot image and select "Use the file header to set the start address and size of the memory block to be loaded. For reference in loading memory, see Figure 2 and Figure 3.
  7. Ensure the start address to base of DDR (0x80000000) and click finish.
  8. Run the nandwriter_evmk2X.out on the C66x core to flash the binary to the NAND on the EVM.
- Running and verifying the NAND boot:
  1. Set the EVM switches to NAND boot as shown in Section 8.4.1 and connect the serial terminal to host and configure the UART Connect to 115200 baud rate.
  2. Power on the EVM. When the boot completes successfully, the device puts out a UART message indicating the cores have woken up and are standing by.

     The following messages will be displayed on the terminal window:

     ```
     Core 0 standing by....
     Core 1 standing by....
     Core 2 standing by....
     Core 3 standing by....
     ```

8.4.5 Running UART Example

- Location of boot image binaries:
  - examples/<k2 device>/uart/singleStage/bin/uartImage.dat
  - examples/<k2 device>/uart/multiStage/bin/uartStage1.dat
  - examples/<k2 device>/uart/multiStage/bin/uartStage2.dat
- Running and verifying the UART boot:
  1. UART is a slave boot mode that requires the host to pass the blob boot image. Connect the serial terminal to the host. You can use hyperterminal or TeraTerm on the Windows to transfer the image from the host to the target. Configure the UART port to 115200 baud rate. After you power on, you will see the ping character "C" on the host terminal. On the hyperterminal or TeraTerm select "File" menu, select Transfer->Xmodem-Send and browse to the UART boot binaries.

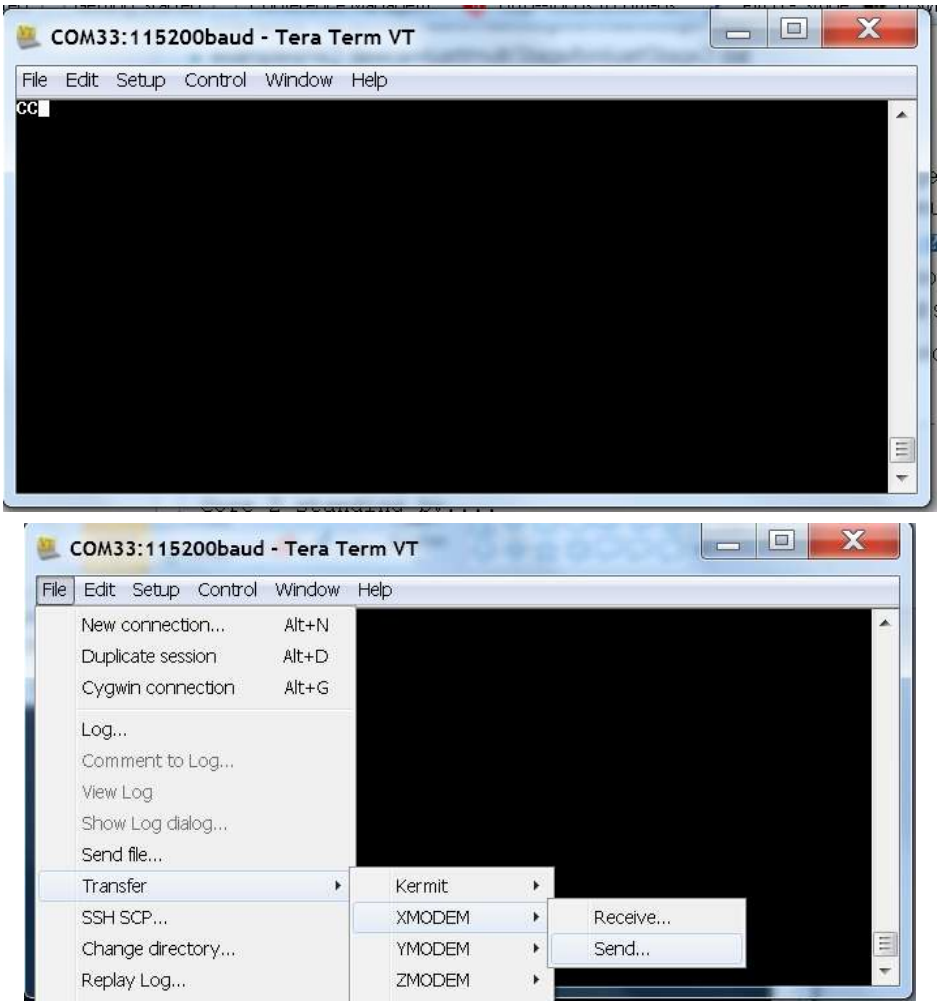

Figure 4. Teraterm Serial Terminal Usage for XMODEM Transfer of UART Boot Image

  2. Set the EVM switches to UART boot as shown in Section 8.4.1.
  3. Power on the EVM. If you are running the single stage example when the boot image transfer completes successfully, the device puts out a UART message indicating the cores have woken up and are standing by. If you are running the multi-stage after the first stage boot, you will see the ping character "C" on the host serial terminal.

4. Select the "File" menu and select Transfer->Xmodem-Send, then browse to the second stage UART boot binary. After the second stage boot completes, you should see the UART messages from the application to indicate boot is complete.

   The following messages will be displayed on the terminal window:

```
Core 0 standing by....
Core 1 standing by....
Core 2 standing by....
Core 3 standing by....
```

8.4.6 Running Ethernet Examples

Ethernet boot is a slave boot mode that requires the host to pass the blob boot image to the target running a DHCP server. Connect the ethernet cable to the host and your switch, and connect another cable from you switch to the EVM. Make sure you do not run this in a high network traffic environment. In order to run the DHCP server on the host Windows machine, use an open source TFTP32 utility in admin mode and configure our host to a static IP expected by the target.

- Location of Boot image binaries:
  - examples/<k2 device>/eth/singleStage/bin/ethImage.dat
  - examples/<k2 device>/eth/multiStage/bin/uartStage1.dat
  - examples/<k2 device>/eth/multiStage/bin/ethernetStage2.dat
- TFTP host utility download:
  - TFTP32 Download
- Setting up the host for ethernet boot:
  1. Setup Static IP on the host machine by setting the IPv4 setting of the Wired LAN settings. Set the Static IP to 192.168.5.10 as shown in Figure 5.

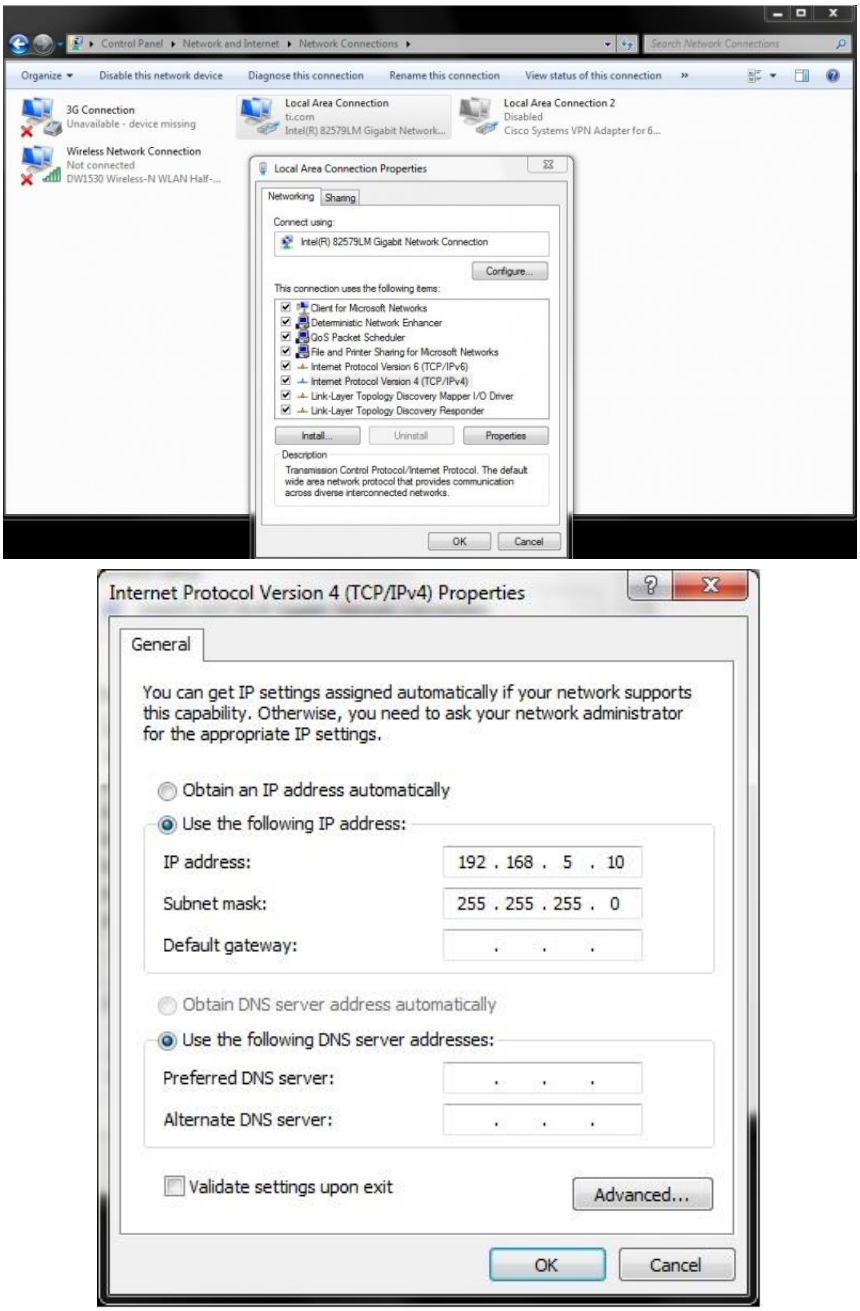

Figure 5. Static IP Configuration on Host PC

2. Configure the DHCP Server settings in the TFTP32 utility as shown in Figure 6.

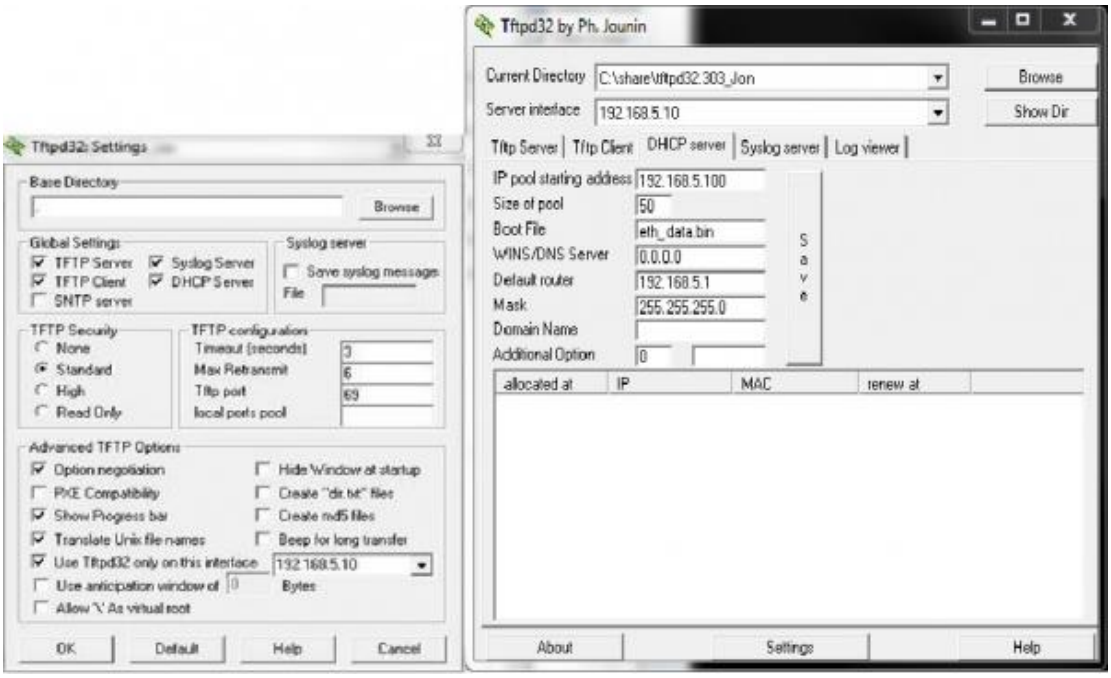

Figure 6. TFTP Transfer Configuration for Ethernet Boot

Fixing the issue with BMC firmware and boot switches to boot Arm over ethernet on K2H EVM.

Some earlier revisions of the EVM have an issue with the settings required for setting up the EVM for ethernet boot.

In order to fix the issue, follow the steps below:

1. Connect your host to the serial port and connect to both the BMC console port and the Serial port. Using the settings described in the serial port setup section of the wiki.
2. Setup the boot switches to Ethernet boot(p[1:4]=0101) using the settings described in Section 8.4.1.
3. Power on the EVM. You should see the Arm ENET boot mode on the LCD screen in the logs displayed along with BOOT COMPLETE.
4. On the BMC port, type the following:

```
>bootmode #8 0 115EEB
>bootmode #8
>reboot
```

Or

```
> setboot 115EEB
>fullrst
```

- Running and verifying the Ethernet single stage boot on the K2H devices:
  1. Copy the Ethernet single stage boot image to the TFTP32 folder and rename the file to eth_data.bin.
  2. Power on the EVM, set the ethernet boot in BMC firmware; when the device reboots, make sure the tftpd32 utitlity is running and configured as described in the previous section. When EVM boots in ethernet boot mode, it will send a bootp. If you do not see uboot come up on the Serial port, check if you are able to see bootp packet by using Wireshark utility.

3. After the BOOTP is received, you will see the transfer of the ethernet boot image and the UART messages when the boot completes:

```
Core 0 standing by....
Core 1 standing by....
Core 2 standing by....
Core 3 standing by....
```

- Running and verifying the Ethernet mutli stage boot: on K2H/K2E devices:

  The Ethernet multi-stage example runs the first stage over UART or I2C and then re-enters the boot to load the second stage over ethernet.

- Load the first stage over UART:
  1. Set the boot switches of the EVM to UART boot.
  2. Connect the serial cable from the EVM to the host and launch hyperterminal or Tera term and configure the serial terminal connection to 115200 bps.
  3. Run the TFTP32 utility and power on the EVM.
  4. After the UART first stage boots, the device switches to ethernet boot and loads the second stage over Ethernet.

  The Ethernet multi-stage example runs the first stage over UART or I2C and then re-enters boot to load the second stage over ethernet.

  NOTE: On K2E EVM, connect the ethernet cable to the top port of the ethernet switch. On K2H the device can boot first stage from either port 1 or port 2.

- Load the first stage over I2C:
  1. Set the boot switches of the EVM to No boot.
  2. Copy the second stage image to the TFTP32 utility folder and rename it to eth_data. Run the TFTP32 utility using admin privileges as described in the single stage boot section.
  3. Use the I2C EEPROM writer from MCSDK 3.x as described in the I2C boot example to flash the first stage to the EEPROM.
  4. After you have flashed the image, switch the boot switches to I2C boot. Before you power on the EVM, ensure the TFTP32 utility is running with the second stage copied over.

  NOTE: On K2E EVM, connect the ethernet cable to the top port of the ethernet switch. On K2H the device can boot first stage from either port 1 or port 2.

9 Boot Utilities

Boot Utilities are host side utilities included in the package that are used to format the boot image based on the boot mode requirements. The utilities can be located in the package under the path "utils" directory. The following utilities are include in the package.

- byteswapccs - byteswaps files in ccs format

```
Usage: byteswapccs <infile> <outfile>
Both <infile> and <outfile> are .ccs files.
```

- byteswapbin - Creates a byte swapped copy of a binary file

```
Usage: byteswapbin lt;infile> lt;outfile>
Both lt;infile> and lt;outfile> are .bin files.
```

- catccs - concatenates ccs format files

```
 Usage: catccs <infile1> <infile2> [<infile3> [<infile4> [..]]] [-out <outfile>] [-addr <address>]
<infile1>, <infile2>, <infile3>, <infile4> and <outfile> are .ccs files.
address - load address for the concatenated ccs file.
```

- ccs2bin - converts ccs format to binary

```
Usage: ccs2bin [-swap] <ccsfile> <binfile>
<ccsfile> - input .ccs files.
<binfile> - output .bin files.
```

- b2ccs - converts a hex b file into a ccs data file

```
Usage: b2ccs [-noorg] <hexfile> <ccsfile>
<hexfile> - Hexadecimal blob file.
<ccsfile> - Output .ccs file.
if -noorg is used there is only one header line
```

- ccsAddGphdr - adds a general-purpose header to a ccs format file

```
Usage: ccsAddGphdr [baseAddress] [-infile <infile>] [-outfile <outfile>] -headerEndian BE
<infile> Input CCS file with no GP header
<outfile> Output CCS file with GP header
headerEndian - Should always be forced to BE (big endian) for LE boot.
```

- ccsAddGptlr - Adds the 8 byte General Purpose Trailer

```
Usage ccsAddGptlr [-h] [-infile <infile>] [-outfile <outfile>]
<infile> Input CCS file with no GP trailer
<outfile> Output CCS file with GP trailer
```

- ccspad - Pad a ccs data file to a certain length. The length is in number of lines.

```
Usage ccspad <infile> <outfile> <pattern> <length>
<infile> Input CCS file of length
<outfile> Output CCS file padded with the <pattern>to size <length>
```

NOTE: Under Linux environment, these utilities can be used using a utility called wine, which allows you to run windows utilities in the Linux environment.

10 Frequently Asked Questions (FAQ)

Question: Where can I find the details for ROM call tables that define the ROM re-entry and other calls to ROM functions?

Answer: The details of the ROM call tables are covered in Table 1.

Table 1. ROM Function Table

Memory Address	Function Name	Details
00001000	_romtMonitorFunction	Install Monitor code
00001004	_romtBootReentry	BootROM re-entry function
00001008	_romtEnableModule	Power up a module
0000100C	_romtDisableModule	Power down a module
00001010	_romtEnterHibernation	Enter hibernation
00001014	_romtCleanupHibernation	Cleanup hibernation
00001018	_romtConfigPll	Configure PLL
0000101C	_romtDelay	Delay
00001020	_romtDeviceFreqMhz	Set device frequency specificed by device variant
00001024	_romtArmNum	Return the ARM number (will only execute as supervisor
00001028	_romtTetrisPsc	Transition the Tetris psc
0000102c	_romtCacheClean	Perform a cache clean
00001030	_romtPscSetState	System PSC set state
00001034	_romtMainEmif4Cfg (1)	Emif 4 configuration based in the DDR configuration table

(1) This function is available only on K2E devices. K2H devices do not provide this ROM call function.

Question: How to obtain the DDR configuration table for my device?

Answer: Create a GEL file for your platform that can initialize the DDR based on the clocks on you platform and the timings required by the DDR3 memory you have used on the platform. If you have the GEL file created with the stable configuration, the GEL DDR init function will contain the configuration of DDR controller settings that can be translated to DDR configuration table. For example, check, how the DDR configuration table has been created for K2E devices by comparing it with the DDR settings in the K2E EVM GEL file provided in the emulation package.

Question: How can I run the Boot utilities in Linux environment?

Answer: You can use the pre-built windows based boot utilities in Linux using a utility called "wine". There are no known issues with this usage but the package is not designed to build in the Linux environment in its current form.

Question: How to debug Booting from different boot modes?

Answer: There are multiple ways to debug booting on K2 devices. Some of these techniques are discussed below:

- Check the DEVSTAT settings and the boot parameter table loaded in by the Boot ROM:

  First, and the simplest way to check if your device is setup to boot correctly, is to connect to the device using an emulator using CCS and connect to Arm Core 0. Ensure that you remove the GEL files that are typically used to initialize the DDR as they will override the settings done by the Boot ROM. In the memory browser, check the value in DEVSTAT register. For the location of DEVSTAT register, see the device-specific data manual. The interpretation of the value in the DEVTAT register will be provided in the *DEVSTAT Boot Mode Pins ROM Mapping* section. If this value is accurately based on you boot switch settings, look at the memory location that contains the boot parameter table for the device. The location of the boot parameter table can be found in the *ARM Boot RAM Memory Map* section in the device-specific data manual. The definition for the boot parameter structure varies based on the boot mode. You can refer to the boot media specific parameter table in the data manual or in the tiboot.h file for the device. Ensure that you are specifying the correct REF clock settings for your platform and are not over clocking the device at the time of boot.
- Check for CS/CE on flash media and ping messages on slave boot modes.

  If the DEVSTAT and boot specific parameter table are loaded as expected, you can check the hook up of the boot media by ensuring that there is activity on the Chipselect or Enable of the flash media or for slave boot media check for ping messages that typically are outputted from the device. For UART boot, the device transmits a ping character "C" on the device. For ethernet, the device sends a BOOTP (ready packet) from the port, and so forth. This should indicate that the device and the boot media are configured correctly.
- Check the boot image format.

  Typically for blob boot, the K2 devices load the image at the base of MSMC and for GP boot formats the device loads the image specified by the GP header. Therefore, connect to the device at the memory location where you expect you boot image to load and confirm that the image is loaded by the ROM. To simplify this process, you can also choose to load the symbols from your application binary and see if the code in that location correlates with the data in the memory location. You can also compare your boot image with images built in this boot examples package to ensure that it is in the correct format.
- Debugging multistage boot:

  In order to debug the multi-stage boot, you can add a large delay or a while loop in each of the stages so that you can single step through the code after each stage has booted. The delay or the while loop allows you to connect to the device and then step through the code loaded by the boot ROM. To view symbols, you can add symbols in CCS using Run->Load Program->Load Symbols. This does not add the actual code over the emulators but just the symbols from the application.

11 References

- Texas Instruments: *KeyStone II Architecture ARM Bootloader User's Guide*
- GIT Repo for Boot Examples
- Processor SDK for 66AK2HX Processors - Linux and TI-RTOS support
- MCSDK 3_01_04_07 software
- https://sourceforge.net/projects/mingw/files/
- Code Composer Studio Downloads
- EVMK2HX Evaluation Modules
- https://www.einfochips.com/
- EVM Hardware Setup
- Texas Instruments: *66AK2E05/02 KeyStone SoC Silicon Errata (Silicon Rev 1.0)*
- TFTPD32 utility

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

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