Keystone II Boot Examples

MCSDK 3.x Boot Examples

Boot examples package download

GIT Repo for Boot Examples [1]

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

```
git clone git@gtgit01.gt.design.ti.com:git/projects/boot-examples.git
boot_examples
```

Software Dependencies

- MCSDK 3.x [2]
- MinGW for Windows [3] with msys and make tools installed. OR Linux machine with Ubuntu 12.04LTS with wine utility.
- TI ARM compiler (Typically packaged with Code composer Studio [4])

Supported Hardware

- K2H EVM rev 1.1 and later [5]
- K2E EVMs rev 1.0.2.0 and later [6]

Note: Ensure that the EVM has the latest UCD and BMC updates as described in the article Setup Hardware [7]

Software Features

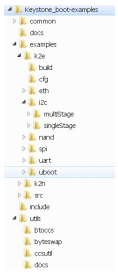
- Boot Utilities to create and format boot images
- Examples that demonstrate booting K2H and K2E devices from SPI, I2C, UART, Ethernet and NAND. Examples also demonstrate the following features of the BootROM
  - Single and Multi-Stage booting.
  - Using Boot Parameter tables to speed up booting from a boot media.
  - Using Boot Configuration tables to initialize DDR.
- Examples to demonstrate formatting uboot for above mentioned boot modes.

Directory Structure

The directory structure for the software package is shown in the image below. The examples directory is organized to follow the convention

```
    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/mulitstage 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.

Build flags: DEVICE_FLAG: Specifies the

Building the Examples

1. Setting up host environment

For Windows Environment: The top level directory in the software package contains a file setupMsysEnv which 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 which 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 don`t have this installed. Please 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.

Steps to ensure file is saved in unix format, in VI while saving the file execute following steps.

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

Description of the examples

The software packages contains 2 types of examples single stage boot examples and multistage boot examples.

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 refer to 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.

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 refer to the makefile in the path examples/<device>/<peripheral>/multiStage.

Boot Media specific details

SPI boot example

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

I2C boot examples

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

NAND examples

NAND boot is a general purpose boot mode in which the BootROM configures the PLL based on the reference clock setting provided from the boot switches. The BootROM loads the image from the first page of block 1. The example uses a DDR configuration table to initialize the DDR memory. Current package doesn`t provide a multi-stage example as the boot mode doesn`t require initialization of PLLs or DDR configuration.

UART boot examples

UART boot is a blob boot in which the BootROM configures the PLL based on the reference clock setting provided from the boot switches. The bootROM configures the UART to baudrate of 115200 bps and sends a ping character to indicate the devices is ready to accept the image. The device loads the blob boot image is loaded at the base of MSMC. On K2E devices, users 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 bootROM doesn`t 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 we also need to additionally invalidate instruction cache in the first stage. To create non-overlapping 2 stages, we use a 1K random memory segmentin the first stage.

Ethernet boot examples

Ethernet boot is a blob boot in which the BootROM configures the PLL based on the reference clock setting provided from the boot switches. The bootROM configures the PHY and booot switch and sends a bootp packet to indicate it is ready to recieve the boot image. Users need to run a dhcp server on the host to load the boot image on target K2 device.The device loads the blob boot image is loaded at the base of MSMC. On K2E devices, users 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 bootROM doesn`t 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 we also need to additionally invalidate instruction cache in the first stage. To create non-overlapping 2 stages, we use a 1K random memory segmentin the first stage.

K2E Ethernet boot errata work around

On K2E devices, there is errata associated with ethernet boot which prevents the device to directly boot of ethernet. Refer to KeyStoneII.BTS_errata_advisory.25 in K2E device errata here [8] The example included in the path examples/k2e/eth/multistage is an example of 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 an image that can be loaded onto NAND or NOR. The first stage is designed to apply the fix and after this image is loaded and finished executing, the device will transmit BOOTP packets at a regular interval which will have the MAC ID of the device that can be used 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 comment: "EVM PHY Workaround" and ends at "End of EVM PHY Workaround." The function call 'setup_Marvell_Phy(1);' in main() may also be removed if desired.

Flashing and Running boot examples

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
- Ethernet: SW1(pin1, pin2, pin3, pin4): off, on, off, on

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: 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 detailed documentation of the flash writer, refer to the ReadMe document included mcsdk_bios_3_01_00_01\tools\writer\eeprom\docs.

Flashing the I2C boot examples

- 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.
- Open the file eepromwriter_input and modify the parameters as shown below:

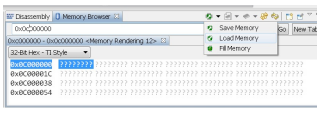
```
file_name  = app.dat
bus_addr   = 0x50
start_addr = 0
swap_data  = 1
```

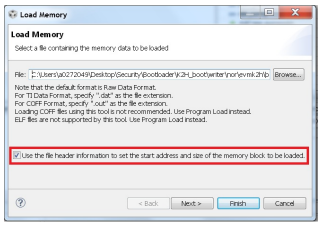
- Configure the EVM to "No Boot mode" as shown in the DIP Switch Settings [9] and connect to the EVM using and emulator and Code Composer studio.

Note details to connect to the EVM using CCS is discussed in the article: Load and running Uboot on EVM using CCS Load [10]

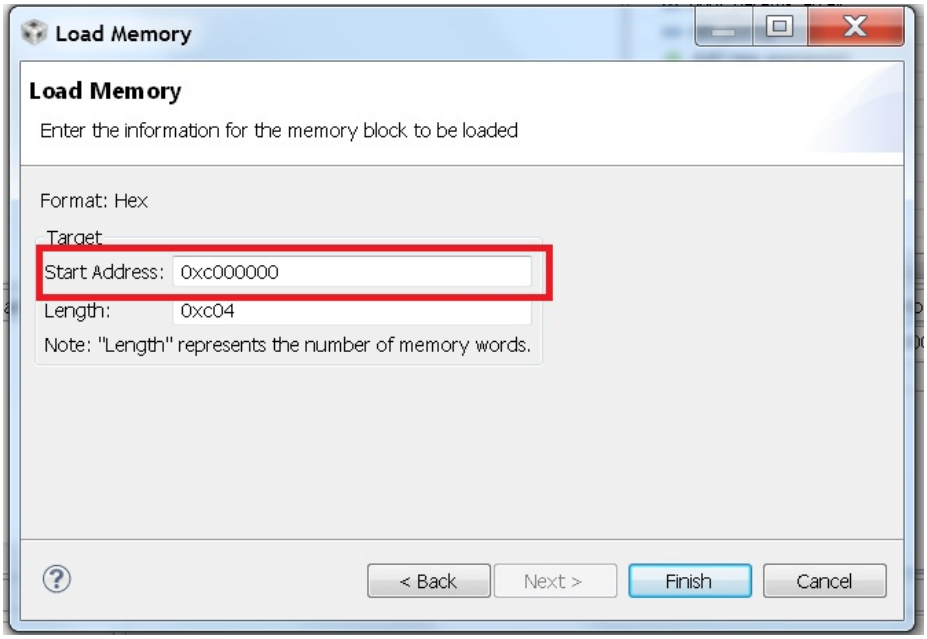
- Connect to ARM core and initialize the core using the Device GEL file found in the emualation pack found here [2]
- Connect to C66x core and load the eepromwriter_evmk2e.out on the device.

- Go to "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.





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



- 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:

- Set the EVM switches to I2C boot as shown in section DIP Switch Setting [9] and connect the serial terminal to host and configure the UART Connect to 115200 baud rate.
- 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.

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....
```

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, refer to the ReadMe document included mcsdk_bios_3_01_00_01\tools\writer\nor\docs.

Flashing the SPI NOR boot examples

- 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.
- Open the file nor_writer_input.txt and modify the parameters as shown below:

```
file_name  = app.dat
start_addr = 0
```

- Configure the EVM to "No Boot mode" as shown in section DIP Switch Setting [9] and connect to the EVM using and emulator and Code Composer studio.

Note details to connect to the EVM using CCS is discussed in the article: Load and running Uboot on EVM using CCS Load [10]

- Connect to ARM core and initialize the core using the Device GEL file found in the emulation pack found here [2]. Configure the 1Ghz and DDR configuration to 1333 Mhz
- Connect to C66x core and load the norwriter_evmk2X.out on the device.
- Go to "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. Refer to screenshot given for flashing I2C boot images for reference for loading memory.
- Ensure the start address to base of MSMC (0x80000000) and click finish.
- 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:

- Set the EVM switches to SPI boot as shown in section DIP Switch Setting [9] and connect the serial terminal to host and configure the UART Connect to 115200 baud rate.
- 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.

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....
```

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: 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, refer to the ReadMe.txt document included mcsdk_bios_3_01_00_01\tools\writer\nand\docs.

Flashing the NAND boot examples

- 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.
- Open the file nand_writer_input.txt and modify the parameters as shown below:

```
file_name  = app.dat
start_addr = 0
rbl_ecc    = 1
skip_bad   = 1
```

- Configure the EVM to "No Boot mode" as shown in section DIP Switch Setting [9] and connect to the EVM using and emulator and Code Composer studio.

Note details to connect to the EVM using CCS is discussed in the article: Load and running Uboot on EVM using CCS Load [10]

- Connect to ARM core and initialize the core using the Device GEL file found in the emulation pack found here [2]. Configure the 1Ghz and DDR configuration to 1333 Mhz
- Connect to C66x core and load the nandwriter_evmk2X.out on the device.
- Go to "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.Refer to screenshot given for flashing I2C boot images for reference for loading memory.
- Ensure the start address to base of MSMC (0x80000000) and click finish.
- 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:

- Set the EVM switches to NAND boot as shown in section DIP Switch Setting [9] and connect the serial terminal to host and configure the UART Connect to 115200 baud rate.
- 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.

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....
```

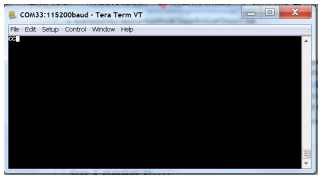
Running UART example

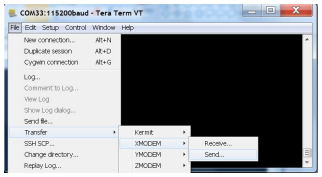
Location of Boot image binaries:

- 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:

- UART is a slave boot mode that requires the host to pass the blob boot image. connect the serial terminal to host. Users can use hyperterminal or TeraTerm on the Windows to transfer the image from 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 and select Transfer->Xmodem-Send and browse to the UART boot binaries.





- Set the EVM switches to UART boot as shown in section DIP Switch Setting [9].
- Power on the EVM. If you are running the single stage example, When the boot image transfer completes successfully, the device puts out 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.

select "File" menu and select Transfer->Xmodem-Send and 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.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....
```

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 host and your switch and connect another cable from you switch to the EVM. Make sure you don`t run this in high network traffic environment. In order to run the DHCP server on the host Windows machine we 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:

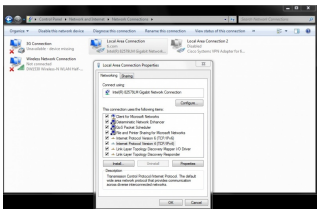
- examples/<k2 device>/eth/singleStage/bin/ethImage.dat
- examples/<k2 device>/eth/multiStage/bin/uartStage1.dat
- examples/<k2 device>/eth/multiStage/bin/ethernet2ndStage.bin

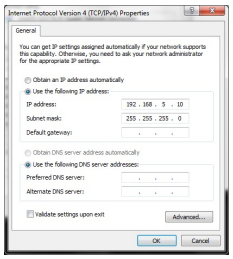
TFTP host utility download

- TFTP32 Download [11]
- Zip Installer :Media:Tftpd32.303.zip

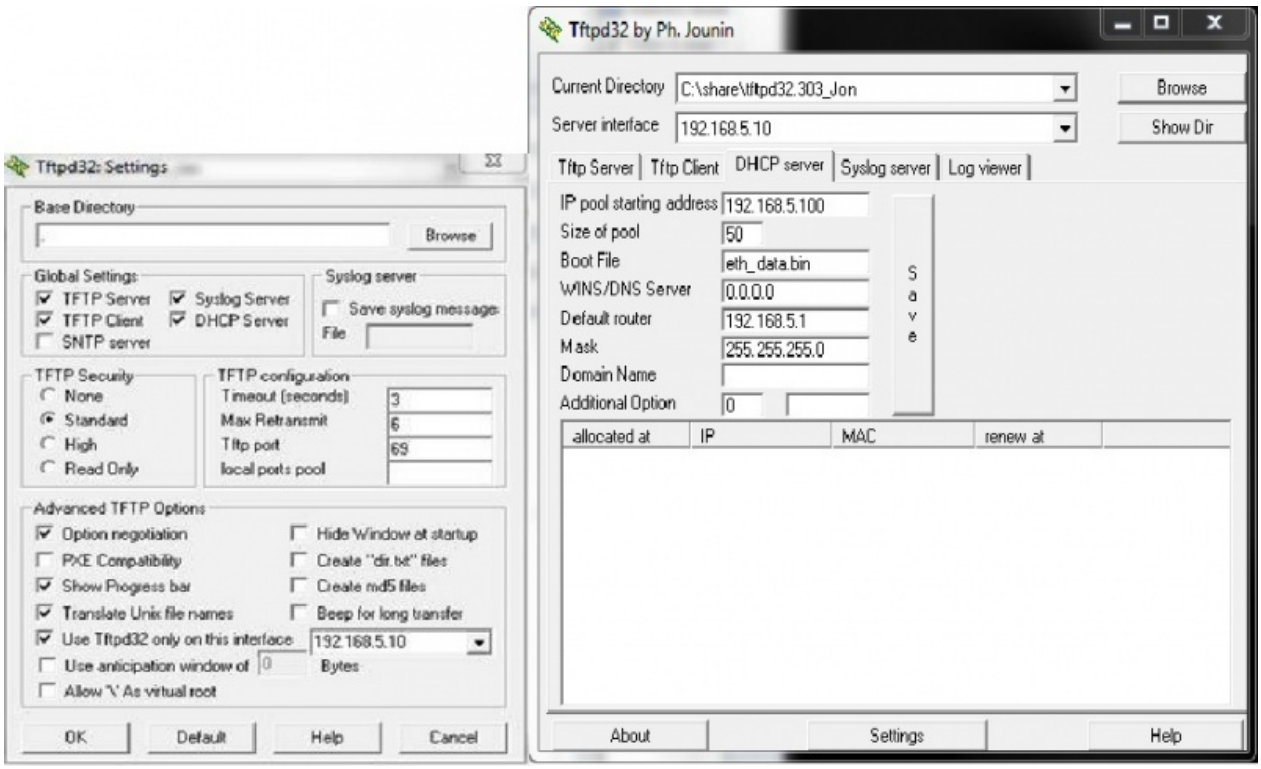
Setting up the host for ethernet boot

- 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 the image below.





- Configure the DHCP Server settings in the TFTP32 utility as shown below:



Fixing 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 mentioned below:

- COnnect your host to the Serial port and connect to both the BMC console port and the Serial port. uisng settings described on wiki section setting up the serial port [12]
- Setup the boot switches to Ethernet boot(p[1:4]=0101) using the settings described here

http:/ / processors. wiki. ti. com/ index. php/ EVMK2H_Hardware_Setup#DIP_Switch_and_Bootmode_Configurations

- Power on the EVM. You should see ARM ENET boot mode onthe LCD screen in the logs displayed along with BOOT COMPLETE.
- 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 K2H devices

- Copy the Ethernet single stage boot image to the TFTP32 folder and rename the file to eth_data.bin.
- Power on the EVM and set the ethernet boot in BMC firmware and 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. so If you don`t see uboot come up on the Serial port check if you are able to see bootp packet by using Wireshark utility.
- After the BOOTP is recieved, users 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

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

Load the first stage over UART.

- Set the boot Switches of the EVM to UART boot.
- 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.
- Select "File" menu and choose Transfer->XModem->Send and browse to uartImage1.bin in the examples/<devices>/eth/multiStage/bin.
- Run the TFTP32 utility and Power on the EVM.
- After the UART first stage boots, the device will switch to ethernet boot and loads the second stage over Ethernet.

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

- Set the boot Switches of the EVM to No boot boot.
- 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.
- Use the I2C EEPROM writer from MCSDK 3.x as described in the I2C boot example to flash the first stage to the EEPROM.
- 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.
- After the first stage boots, the device will switch to ethernet boot and 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.

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 <infile> <outfile>
 Both <infile> and <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>
```

All of the above utilities are provided in source in the software package and can be rebuilt on the host by executing "make utils" in the top level directory. Ensure that you have the host gcc compiler in the PATH variable in your environment.

Note: Under Linux environment these utilities can be used using a utility called wine, which allows users to run windows utilities in linux environment.

FAQ:

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

A The details of the ROM call tables are covered in the table given below

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**	Emif 4 configuration based in the DDR configuration table.

Note: ** This function is available only on K2E devices. K2H devices don`t provide this ROM call function.

- **How to obtain the DDR configuration table for my device.**

A. 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.

How can I run the Boot utilities in Linux environment

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

How to debug Booting from different boot modes?

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 BootROM:

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 BootROM. In the memory browser check the value in DEVSTAT register. For location of DEVSTAT register refer to device specific data manual. The interpretation of the value in DEVTAT register will be provided in section "DEVSTAT Boot Mode Pins ROM Mapping". If this value is accurate 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 device specific data manual in the section "ARM Boot RAM Memory Map". The definition for the boot parameter structure varies based on the boot mode and 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, etc. 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 so after you connect to the device you can take at the memory location where you expect you boot image to load and confirm if 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 multiStage 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 will allow you to connect to the device and then step through the code loaded by the bootROM. To view symbols, you can add symbols in CCS using Run->Load Program->Load Symbols. This doesn`t add the actual code over the emulators but just the symbols from the application.

Related Artciles and Collateral

- Keystone ARM Bootloader User Guide [13]
- Booting_uboot_on_K2H_over_ethernet [14]

References

[1] http://gtgit01.gt.design.ti.com/git/projects/?p=boot-examples.git;a=summary
[2] http://software-dl.ti.com/sdoemb/sdoemb_public_sw/mcsdk/latest/index_FDS.html
[3] http://sourceforge.net/projects/mingw/files/
[4] http://processors.wiki.ti.com/index.php/Download_CCS
[5] http://www2.advantech.com/Support/TI-EVM/EVMK2HX.aspx
[6] https://www.einfochips.com/index.php/partnerships/texas-instruments/k2e-evm.html
[7] http://processors.wiki.ti.com/index.php/MCSDK_UG_Chapter_Getting_Started#Setup_Hardware
[8] http://www.ti.com/lit/er/sprz417/sprz417.pdf
[9] http://ap-fpdsp-swapps.dal.design.ti.com/index.php/MCSDK_3.x_Boot_Examples#Dip_Switch_Settings
[10] http://processors.wiki.ti.com/index.php/MCSDK_UG_Chapter_Exploring#Loading_and_Running_U-Boot_on_EVM_through_CCS

[11] http://tftpd32.jounin.net/
[12] http://processors.wiki.ti.com/index.php/Program_EVM_UG#Serial_Port_Setup
[13] http://www.ti.com/lit/ug/spruhj3/spruhj3.pdf
[14] http://ap-fpdsp-swapps.dal.design.ti.com/index.php/Booting_uboot_on_K2H_over_ethernet

Article Sources and Contributors

Keystone II Boot Examples *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?oldid=208408 *Contributors*: A0272049

Image Sources, Licenses and Contributors

File:Boot-examples dir struct.jpg *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?title=File:Boot-examples_dir_struct.jpg *License*: unknown *Contributors*: A0272049
File:load_mem_0.jpg *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?title=File:Load_mem_0.jpg *License*: unknown *Contributors*: A0272049
File:load_memory.jpg *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?title=File:Load_memory.jpg *License*: unknown *Contributors*: A0272049
File:load_memory_2.jpg *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?title=File:Load_memory_2.jpg *License*: unknown *Contributors*: A0272049
File:UART_ping.jpg *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?title=File:UART_ping.jpg *License*: unknown *Contributors*: A0272049
File:UART_XMODEM_transfer.jpg *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?title=File:UART_XMODEM_transfer.jpg *License*: unknown *Contributors*: A0272049
File:Setting_static IP.jpg *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?title=File:Setting_static_IP.jpg *License*: unknown *Contributors*: A0272049
File:Setting_static IP_2.jpg *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?title=File:Setting_static_IP_2.jpg *License*: unknown *Contributors*: A0272049
File:TFTPd32.jpg *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?title=File:TFTPd32.jpg *License*: unknown *Contributors*: A0272049