KeystoneI Bootloader Resources and FAQ

Contents

Keystone I Bootloader

Bootloader documentation

The documentation for Keystone I bootloader is distributed across 3 resources:

- Keystone I Bootloader User guide (https://training.ti.com/system/files/docs/keystone-DSP-bootloader-user-guide.pdf?keyMatch=Keystone%20DSP%20bootloader%20user%20guide&tisearch=Search-EN-Everything)

Common User guide across all Keystone I devices which will provide the details of the BOOTROM design and it`s expected behaviour on each of the supported boot modes across the Keystone I family. The device covers-in some of the details and the behavior of the device after different reset & hibernation modes. It will also provide a preview of how

- Device datasheet : Boot Mode settings, Parameter table, DDR configuration tables and memory maps specific to the device as published in the device datasheet
  - TMS320C6678 (http://www.ti.com/lit/ds/symlink/tms320c6678.pdf)
  - C6670 (http://www.ti.com/lit/ds/symlink/tms320c6670.pdf)
  - C6657 (http://www.ti.com/lit/ds/symlink/tms320c6657.pdf)
- BootROM source (http://processors.wiki.ti.com/index.php/Keystone_Device_Architecture#Keystone_ROM_Boot_Examples_and_Reference_code): Wiki section that provides access to boot ROM source code for reference to understand the implementation
- Processor SDK RTOS Bootloader wiki (http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_BOOT_C66x): Boot software and examples validated on the TI EVM along with the flash writer and utilities are packaged as part of the device SDK.

Direct Boot Examples (without IBL)

The TI C66xx EVMs always implement an Intermediate boot loader after power on reset. The FPGA firmware on the EVM redirects the core to the IBL flashed on the EEPROM and then directs it to the desired boot mode. However, in custom designs, we have observed that adding an additional EEPROM for implementing the IBL may not be practical due to cost and size reasons. The C667x PG 2.0 silicon has the PLL lock up issue fixed and no longer requires an implementation of an IBL on custom boards; the C665x PG 1.0 silicon does not have the PLL lock up issue, so it also does not require an implementation of an IBL on custom boards.

In order to demonstrate that the hardware can be booted directly from the ROM boot loader without the IBL, we have created some examples that can be used to validate this flow on the EVM hardware. The two examples described below are for SPI boot on C6678 and NAND boot on C6657. Each of the examples have a ReadMe.txt or document that walks users through the processing of creating these boot images.

- C6678 EVM SPI boot example (http://processors.wiki.ti.com/index.php/File:C6678_directROM_boot_examples.zip)
- C6657 EVM SPI boot example with DDR initialization (http://processors.wiki.ti.com/index.php/File:C6657_directROM_Boot_example.zip)
- C6657 EVM NAND boot example (http://processors.wiki.ti.com/index.php/File:C6657_directROM_Boot_example.zip)

Note: For documentation of these example refer to the user Manual and the ReadMe.txt files provided in the docs folder of these packages.

Bootloader Utilities

This section describes the boot utilities that are provided by TI in order to create, format, flash the boot image so that it can be loaded using the ROM Bootloader on the device.

Where can I find the boot utilities ?

The boot utilities for Keystone I devices are packaged as part of the bootloader software package in the Processor SDK RTOS under the folder path pdk_<deviceName>_x_x_x\packages\ti\boot

Note: In earlier MCSDK for Keystone I device, the boot utilities, were provided in the directory path mcsdk_2_xx_xx_xx\tools\boot_loader

What bootloader utilities are available for the C66xx devices?

- Intermediate Bootloader(IBL) (http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_BOOT_C66x#Bootloader_Execution_Sequence)
- Flash Writer (http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_BOOT_C66x#Flash_Writers)
- Multicore Application development(MAD) Utility (http://processors.wiki.ti.com/index.php/MAD_Utils_User_Guide)

What utilities should I use if I am not using a secondary bootloader or IBL ?

The boot utilities provided in utls folder of the IBL package and in the bin folder of the C6000 compiler can also be used for creating boot image if your boot design doesn`t leverage the intermediate boot loader (IBL) . These utilities are documented are mostly are documented in the code. For typically usage of these utilities refer to the SDK examples or the Direct boot example in the previous section.

Here is brief descriptions for these boot utilities :

- **Hex6x utility in C6000 Compiler:** The hex conversion utility converts an object file (.out) into the boot table format required by the DSP boot loader. This utility is documented Chapter 11 of the C6000 Assembly Tools user guide (http://www.ti.com/lit/ug/spru186w/spru186w.pdf) included as part of the C6000 compiler.

> **NOTE**
>
> There is a known issue with hex6x that can cause issues with C66x ROM bootloader that you need to be aware of. The hex6x utility rounds of code sections to 4 byte boundary. This can cause problems with the C66x bootloader as the bootloader doesn`t have a way to interpret those padded bytes. Hence it is recommended to make sure that the size parameter in boot table format reflects the size with the padded bytes. For details refer E2E post here (https://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/t/384473)

- **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
```

- **bfmerge** - Combine two boot tables

```
Usage:   usage bfmerge file1.btbl file2.btbl [file3.btbl [file4.btbl [...]]] > output.btbl
fileX.btbl     : Input boot table files.  Always start with core0 boot table file.
output.btbl    : Output boot table file
```

- **Romparse** Utility is used to append a boot parameter table to the boot table

```
Usage:   romparse [-noorg] <hexfile> <ccsfile>
 <hexfile> - Hexadecimal blob file.
 <ccsfile> - Output .ccs file.
```

Note 1: This utility in the SDK is only a reference implementation that addresses usecase of the SDK. It may need to be modified if there are some boot parameter table fields missing or if lesser than 8 boot parameter table need to be attached to the boot table data.The utility is provided in source to allow for customization.

Note2: Also note the utility hard code the i2c addresses of EEPROM used on the EVM in the boot parameter table. If you are using SPI boot then the address needs to be set to 00

Note3: The utility is currently designed to process boot images of the size less than 4Kb in size so please modify the size parameter in the utility if your boot image is greater than 4Kb in size

- **b2i2c**

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

- **bconvert64x** Utility is used to format the boot table data that is not multiple of 4 bytes to the correct endian format expected by the boot loader.

```
Usage:   bconvert -be|-le [input_file] [output_file]
 <input_file> - Input Boot table data file
 <output_file> - Output Boot table data file.
```

What is the syntax that needs to be used in an .rmd file that is used to create the boot image?

The .rmd file is intended to be used along with the hex6x tool provided with the C6000 compiler. The syntax to create a binary image using hex6x and .rmd file has been described in Section "Hex Conversion Utility Description" of the C6000 compiler documentation

Keystone I boot loader FAQ

General Boot Questions

How to load same image across multiple cores? Examples?

Please take a look at the multi-core boot example given in SDK under the directory ~\boot_loader\examples\srio\srioboot_helloworld\src. This example shows how you can wake up secondary cores using IPC interrupts and populate the magic address for the cores to start executing code. This structure is useful, if you want to create a single application that boots on all cores and then has control code to partition itself across secondary cores by checking the core number on which it is running.

The boot ROM will initialize the SOC and load a single image into memory. All cores will run the same image but the code will have control code to check which core is running the image and partition the code. Core0 boots first so it will need to populate the entry point of the image in the magic address for all the secondary cores and issue them an IPC interrupt to wake them up.

Which gel files can be used to connect C66xx DSP cores and where can I find them?

GEL stands for General Extension Language (previously GODSP Extension Language). GEL is the expression language that is used by the CCS debugger.

Target configurations in CCS often specify a startup script. These scripts are typically used to setup the memory map for the debugger, set any initial target state (via memory or register writes) that is necessary in order to connect the debugger. These scripts are usually written in GEL. For example, it is the function OnStartup() in the GEL file that is run when the debugger is launched and a function OnTargetConnect() is called when the target is connected. The startup(gel) scripts define these functions.

These gel files packaged as part of CCS and older MCSDK has it packaged in its tools directory.

GEL Files:

In MCSDK 2.x, find the appropriate gel files for Keystone-I EVMs under the path "~\ti\mcsdk_2_0x_0x_0x\tools\program_evm\gel".

In CCSvX, find the gel files under the path "~\ti\ccsvX\ccs_base\emulation\boards\evmc66xxl".

Target Configuration Files:(Default is available here and you can create as many as required)

In MCSDK 2.x, find the *.ccxml files under the path "~\ti\mcsdk_2_0x_0x_0x\tools\program_evm\configs".

In Processor SDK, find the *.ccxml files under the path "~\ti\processor_sdk_rtos_c66xx_x_xx_xx_xx\bin\configs\evm66xxl".

How do I set the boot modes ?

Please refer the Boot mode DIP switch setting section on Hardware Setup Guide to set the desired boot modes.

C6657 EVM	processors.wiki.ti.com/index.php/TMDSEVM6657L_EVM_Hardware_Setup (http://processors.wiki.ti.com/index.php/TMDSEVM6657L_EVM_Hardware_Setup)
C6670 EVM	processors.wiki.ti.com/index.php/TMDXEVM6670L_EVM_Hardware_Setup (http://processors.wiki.ti.com/index.php/TMDXEVM6670L_EVM_Hardware_Setup)
C6678 EVM	processors.wiki.ti.com/index.php/TMDXEVM6678L_EVM_Hardware_Setup (http://processors.wiki.ti.com/index.php/TMDXEVM6678L_EVM_Hardware_Setup)

How to wake up secondary cores from primary core(core0)?

During the boot process, the boot loader executes an IDLE command on the secondary CorePacs and keeps the secondary CorePacs waiting for an interrupt. After loading the secondary CorePacs application, the BOOT_MAGIC_ADDRESS in individual corePacs are populated, the application code in the corePac0 can trigger the IPC interrupt to wake up the secondary cores and branch up to the address specified in the BOOT_MAGIC_ADDRESS. Please try below procedure in EVM,

Source: <a _fcknotitle="true" href="Helloworld.zip">Helloworld.zip **Note:** The source has been removed from this location due to licensing issue. Please locate the source code on E2E link (https://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/p/425551/1519565#1519565)

1. Set the EVM in "No Boot" mode, connect to core0 and use gel file to initialize the core0. (Also, You can run Global_Default_Setup - to initialize the PLL and DDR).

2. Go to memory browser, read the address **0x1187FFFC** and make a note of it.

3. Build the attached source and load and run on core0.

4. After completion of execution, core0 will wait in busy loop(while(1)). Pause the debugging.

5. Go to memory browser, enter the address **0x1187FFFC** .=> 0xBABEFACE

This means that core0 wakes up core1 successfully. The value 0xBABEFACE has been written by core1 from function "write_boot_magic_number()".

You can follow this for all secondary cores by updating **NUMBER_OF_CORES**.

Note:

Make sure to build the application with appropriate macros for EVM and magic address on step 2 & 5. The above steps are shown for C6678.

```
  /* Magic address RBL is polling */
#ifdef _EVMC6657L_
#define MAGIC_ADDR 0x8ffffc
#endif
#ifdef _EVMC6678L_
#define MAGIC_ADDR 0x87fffc
#endif
#ifdef _EVMC6670L_
#define MAGIC_ADDR 0x8ffffc
#endif
```

How to find the silicon revision of the SoC mounted on the EVM or Custom board?

Refer section "JTAG ID Register (JTAGID) Description" and "C66x DSP Device Nomenclature" on the data manual to find the silicon revision.

Where can I download the keystone I EVM resouces like User/Startup Guide, Schematic, BoM and FPGA bit file etc.?

C6657 EVM	www.einfochips.com/partnerships/silicon-partnerships/texas-instruments/tms320c6657-evm.html (https://www.einfochips.com/partnerships/silicon-partnerships/texas-instruments/tms320c6657-evm.html)
C6670 EVM	www2.advantech.com/Support/TI-EVM/6670le_of.aspx (http://www2.advantech.com/Support/TI-EVM/6670le_of.aspx)
C6678 EVM	www2.advantech.com/Support/TI-EVM/6678le_of.aspx (http://www2.advantech.com/Support/TI-EVM/6678le_of.aspx)

Where can I find build and program instructions for Keystone I Boot Examples and utilities?

The keystone I boot resources are documented as Readme.txt files as MCSDK SDK installation directories. Please refer at them from below installation paths,

Writers: ~\tools\writer\<eeprom/nand/nor>\docs

POST: ~\tools\post\docs

IBL: ~\tools\boot_loader\ibl\doc

Boot Examples: ~\tools\boot_loader\examples\<ethernet/i2c/mad/pcie/srio>\docs

How do I load the EVM's with factory images?

Please follow the instruction from program evm user guide(pdf) of MCSDK/Processor SDK.

PATH: ~\ti\mcsdk_2_0x_0x_0x\tools\program_evm

How do I test/validate/bring-up the custom boards with C66xx devices?

1. Port the gel file for custom board with respect to the changes like clock, speed, memory configurations etc. by referring/using TI provided utilities like Clocking and DDR levelling spredsheet etc.

2. Port the platform_test for custom board and test it.

Note: There is a simple DDR test that is part of the GEL, it is called ddr3_memory_test () and you can control the start and end address of the space.

Are there any validated big endian boot examples for C66xx ?

No. By default, all our boot examples are validated on little endian mode only.

Where can i find the ROM bootloader (RBL) sources?

Please find the RBL sources for PG1.0 and 2.0 from below wiki link. http://processors.wiki.ti.com/index.php/Keystone_Device_Architecture#Keystone_ROM_Boot_Examples_and_Reference_code

IBL boot

I have programmed IBL and want to boot application from NOR flash. How to convert my .out to .bin to flash it?

You can re-name the .out file to .bin to program on NOR flash. If the file size is very huge then you can remove symbols from the binary using the strip6x utility.

Use strip6x.exe to convert .out file to .bin format. "strip6x.exe -p -o <Output file name .bin> <Input file .out>"

```
Example :

  C:\ti\C6000 Code Generation Tools 7.4.0\bin>strip6x.exe -p -o ..\..\..\Bin\nor.bin ..\..\..\Bin\hua_evmc6657l.out
```

Can C6678 be booted up directly from SPI Nor Flash, without the participation of I2C? Is it necessary to program IBL and IBL configuration on I2C EEPROM at bus address 0x51 when I test SPI boot mode on TMDXEVM6678L?

Yes. It is possible. With the silicon revision 1.0, the PLL fix is required. With the silicon revision 2.0, the PLL fix is not required(the IBL workaround is not required).

On the Keystone I EVM, for ROM boot modes (EMAC,SRIO,PCIe,Hyperlink,SPI etc) and I2C boot mode with bus address 0x50, DSP will initially boot from I2C EEPROM bus address 0x51(IBL) which does the PLL reset workaround, updates the DEVSTAT for appropriate values based on the DIP switch settings (SW3 through SW6 settings) and then re enters the ROM to accomplish the desired boot mode.

Please review the errata document to understand the conditions of the PLL locking problem and the workaround. This is Advisory 8. The errata specifies a workaround option for "no-boot, SPI and I2C boot modes", so you could do the same thing with a SPI Flash instead of I2C EEPROM.

On the EVM, this is implemented using an FPGA and I2C EEPROM 0x51 to allow you to have as much flexibility as possible with your EVM boot selection. If the EVM is used, it is necessary to program the IBL and its configurations in address 0x51.

After flashing IBL, I am trying to program the IBL configuration. After loading i2cConfig.gel, I cannot "Run the GEL script "EVM c6678 IBL"->setConfig_c6678_main ". I have checked under menu "Scripts", which has no drop down menu shown. I am stuck at "EVM c6678 IBL"->setConfig_c6678_main". How to solve this issue?

Please follow the step by step procedure outlined in below post to solve the issue,

https://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/p/451825/1626076#1626076

How do I modify the IBL Configuration Table contents? For example, the IBL on the TMDSEVM66xx is not configured out of the box to be able to boot an ELF image from NAND, and in order to make it do so I must change the IBL Configuration Table.

Two method to modify the IBL Configuration Table contents.

Using i2cConfig.gel GEL file - Update ibl_BOOT_FORMAT_BBLOB to ibl_BOOT_FORMAT_ELF in i2cConfig.gel GEL file. Refer IBL configuration FAQ for updating configuration table in I2C.

PATH: ~\tools\boot_loader\ibl\src\make\bin

(OR)

Modify the same on IBL source and program the re-built IBL binary.

Example:

~\tools\boot_loader\ibl\src\util\iblConfig\src\device.c, edit this file by changing ibl_t c6678_ibl_config(void) ====> if you are using 6678.

```
ibl.bootModes[1].u.nandBoot.bootFormat        = ibl_BOOT_FORMAT_ELF;
```

How to boot MAD image directly from RBL (without the EVM's IBL)?

Please refer below post from e2e that might help,

https://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/p/430837/1614992#1614992

Can I use the MCSDK IBL for custom boards? Please describe the changes required for custom IBL, If any

Yes. You can use the IBL for custom boards however it needs to be ported for custom board configurations. The IBL provided in the MCSDK/Processor SDK package is provided for the EVMs that have an FPGA on it.

You may need to modify the IBL if the custom board has different hook ups to boot media for example CS on which NAND is connected is different, NAND flash device geometry is different, change the clocking function based on your input clock etc. If you look at the IBL code there are few transactions made from the IBL code with the FPGA over the SPI interface. Please refer to C66xxInit.c file in the IBL source code to take a look at FPGA related initialization code. If your design doesn`t contain an FPGA, you can clean up the code that sets up the FPGA configuration and should able to boot the IBL from I2C EEPROM.

Building IBL:

For build instructions and C66xx specific documentation refer to doc folder under ~\tools\boot_loader\ibl\doc folder.

Few e2e posts for reference(Build issues and solutions):

e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/p/361368/1277497#1277497 (https://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/p/361368/1277497#1277497)

https://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/t/269182 (https://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/t/269182)

https://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/t/362997 (https://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/t/362997)

NAND boot

How to boot from NAND flash on C6670/C6678 EVM?

The NAND boot on C6670/78 EVM requires Intermediate Boot Loader (IBL) however the custom boards with Revision 2.0 SoC's does not require IBL. The primary motivation for the IBL is due to a PLL workaround, which is not present in the basic ROM boot. You can find references on this in the ibl/src/device directory.

Please refer the MCSDK user guide for IBL flashing and NAND boot setup.

How to boot from NAND flash on C6657 EVM?

The direct and IBL based NAND booting is possible on C6657.

For IBL booting, please refer MCSDK 2.x or Processor SDK user guide for NAND boot setup.

For direct NAND booting, please find the direct NAND boot example which is validated on C6657 EVM(on the Direct Boot Examples Section).

When there are bad blocks in nand flash, is the program in C66xx can boot from nand flash normally?

Yes. there is a bad block check implemented in the ROM boot loader. The ROM boot loader looks for a valid boot image in a sequential manner skipping (and marking) any bad blocks upto the first 32 blocks on the NAND before reporting a boot failure.

Why updated FPGA bit file is needed for direct NAND boot on C6657? Where can I get the FPGA bit file?

There is an advisory note that is being added to the Errata associated with this boot mode that I would like to make you aware of. Here is the summary of the issue,

Booting from a NAND flash device will fail when the C6657/C6655 at full speed. The root cause of the problem is that insufficient time is allowed for the NAND device to complete the initial reset command sent to it by the DSP ROM code. After sending the reset command, the DSP executes a wait loop(for time out). This wait loop allows a maximum number of iterations before halting and declaring that the NAND is not responding correctly. The wait loop does not allow enough time for the NAND to complete its initial reset sequence.

Proposed work around:

- Use the FPGA firmware on the system to first apply reset to the NAND flash device and then apply reset to C6657 device after the NAND device is ready. After a POR of the device, apply CPU reset.
- The CPU reset only restarts the RBL and allows the NAND flash device to complete its initial transition out of reset. This will work because NAND devices complete the second and subsequent reset sequences faster when compared to the initial reset sequence (5 micro-seconds vs. 300 micro-seconds) NAND boot can be made to work by simply taking C6657/C6655 back into reset and then releasing it from reset. This extra reset sequence must be done without cycling power to the NAND device. Resetting C6655/C6657 after an initial NAND boot attempt works since the wait loop is long enough for success on a second boot attempt)

The C6657 EVM comes with a secondary boot loader (IBL) that is written on the I2C EEPROM that is on the EVM. The FPGA firmware v02 forces the device to execute the IBL before it can execute any other code. This is a requirement for most of the Out of box demos on the EVM.

Firmware upgrade to v03 allows user to bypass the IBL and native boot from the boot media using the Rom code on the device.

The FPGA bit files can be downloaded from EVM manufacturers resources download link.

I have used different NAND chip in custom board. Is nandwrite.c needed to modify? how to modify it ?

The NANDWrite.c code provided with the example was built for the NAND flash used on the EVM and was created only to validate the boot during EVM bring up. You will need to modify it to match the NAND geometry provided in the datasheet of the NAND that you are using. The geormetry you have provided for your NAND doesn`t seem to match the NAND geometry that is used on the EVM which is you may be facing this issue.

Try to change the following parameters in nandwrite.c to match the geometry of your NAND.

```
 unsigned int busWidth = 8;
unsigned int pageSizeBytes = 2048;
unsigned int pagesPerBlock = 64;
unsigned int spareBytesPerSegment = 16;
unsigned int spareBytesPerPage = 0;
```

We also provide a NANDWriter under ~\tools\writer\nand\evmc66xxl\bin. Instructions to flash the NAND using the NANDWriter are provided in the ReadMe under ~\tools\writer\nand\docs.

SPI boot

I am able to boot direct SPI NOR example however it fails to boot with complex .out(big .out file) file. How to solve this issue?

If you are using the b2i2c utilities in the MCSDK, please look at the source for the B2i2c function in the ~/tools/ibl/src/util folder. The default max size configured in the tool is 0x20000 .

You can change the macro SIZE to be able to process your image.

```
  #define SIZE 0x20000
```

SRIO boot

How to run SRIO boot example using C6670 and C6678 EVMs?

The SRIO boot example requires two EVMs and a break out card to execute the example. The detailed procedure of setup and break out card configuration are documented in below thread.

https://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/t/456577

EMIF16/Parallel NOR boot

How to do EMIF16 NOR Flash boot on C66xx devices? Example?

Please refer EMIF16 NOR flash boot summary from below thread.

https://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/t/367102

https://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/t/421142

Would NOR code for EMIF be the same as NAND code? Is there any example for EMIF16 NOR writer?

Please refer below e2e thread for simple NOR writer, http://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/p/330496/1153703#1153703
https://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/p/509244/1850271#1850271

{{ 1. switchcategory:MultiCore= ▪ For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum ▪ For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum Please post only comments related to the article **Keystone1 Bootloader Resources and FAQ** here.	Keystone= ▪ For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum ▪ For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum Please post only comments related to the article **Keystone2 Bootloader Resources and FAQ** here.	C2000=*For technical support on the C2000 please post your questions on The C2000 Forum. Please post only comments about the article **Keystone1 Bootloader Resources and FAQ** here.*	DaVinci=*For technical support on DaVincoplease post your questions on The DaVinci Forum. Please post only comments about the article **Keystone1 Bootloader Resources and FAQ** here.*	MSP430=*For technical support on MSP430 please post your questions on The MSP430 Forum. Please post only comments about the article **Keystone1 Bootloader Resources and FAQ** here.*	OMAP35x=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **Keystone1 Bootloader Resources and FAQ** here.*	OMAPL1=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **Keystone1 Bootloader Resources and FAQ** here.*	MAVRK=*For technical support on MAVRK please post your questions on The MAVRK Toolbox Forum. Please post only comments about the article **Keystone1 Bootloader Resources and FAQ** here.*	*For technical s please post yo questions at http://e2e.ti.co Please post or comments abo article **Keysto Bootloader Resources ar** here.* }}

Links

Amplifiers & Linear	DLP & MEMS	Processors	Switches & Multiplexers
Audio	High-Reliability	▪ ARM Processors	Temperature Sensors & Control ICs
Broadband RF/IF & Digital Radio	Interface	▪ Digital Signal Processors (DSP)	Wireless Connectivity
Clocks & Timers	Logic	▪ Microcontrollers (MCU)	
Data Converters	Power Management	▪ OMAP Applications Processors	