Application Report
SPRACN2–September 2019

Keystonel Bootloader Resources and FAQ

Catalog Processors

ABSTRACT

This application report provides guidance for using the read-only memory (ROM) bootloader in multicore C66x devices in the Keystone family. The document consolidates software resources and provides responses to frequently asked questions. The information provided in the *DSP Bootloader for KeyStone Architecture User's Guide* and covers additional topics like boot utilities, examples to boot the device directly using the ROM bootloader, and debugging boot. It is recommend that application developers refer to the *DSP Bootloader for KeyStone Architecture User's Guide* prior to reading this document.

The examples and utilities discussed in this application report can be downloaded from TI Git. For applications that need additional customization and features that do not exist in the ROM bootloader, the recommendation is to use the two stage boot process as described in the *Boot* section of the Processor SDK Software Developer's Guide.

Contents

Trademarks

Code Composer Studio is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

1 Introduction

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

- *KeyStone Architecture DSP Bootloader User's Guide*

  This is a common user's guide across all Keystone I devices that provides the details of the BOOTROM design and its expected behavior on each of the supported boot modes across the Keystone I family. The device covers some of the details and the behavior of the device after different reset and hibernation modes. It also provides a preview of how to create boot images, flash and debug boot related issues.
- **Device-specific data sheets**: Boot Mode settings, Parameter table, DDR configuration tables and memory maps specific to the device as published in the device-specific data sheets:
  - *Multicore Fixed and Floating-Point Digital Signal Processor Data Sheet*
  - *TMS320C6670 Multicore Fixed and Floating-Point System-on-Chip Data Manual*
  - *TMS320C6655 and TMS320C6657 Fixed and Floating-Point Digital Signal Processor Data Sheet*
- **BootROM source**: Software download link provides access to boot ROM source code for reference to understand the implementation
- C6678 Boot ROM source
- C6657 Boot ROM source
- Processor SDK RTOS Bootloader: Boot software and examples validated on the TI EVM along with the flash writer and utilities are packaged as part of the device SDK.

2 Direct Boot Examples (without IBL)

The TI C66xx EVMs always implement an Intermediate bootloader 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, it has been 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 bootloader without the IBL, some examples were created 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
- C6657 EVM SPI boot example with DDR initialization
- C6657 EVM NAND boot example

NOTE: For documentation of these examples, see the user's manual and the ReadMe.txt files provided in the docs folder of these packages.

3 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.

3.1 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 the earlier MCSDK for the Keystone I device, the boot utilities were provided in the directory path mcsdk_2_xx_xx_xx\tools\boot_loader.

3.2 What Bootloader Utilities are Available for the C66xx Devices?

- Intermediate Bootloader (IBL)
- Flash Writer

3.3 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 does not leverage the intermediate bootloader (IBL). These utilities are documented are mostly are documented in the code. For typically usage of these utilities, see 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 bootloader. This utility is documented in the *Hex Conversion Utility Description* chapter of the *TMS320C6000 Assembly Language Tools User's Guide* 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 does not 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, see the E2E post here.

- 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:

- 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.
- 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
- The utility is currently designed to process boot images of the size less than 4Kb in size so, 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.
```

3.4 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 the *Hex Conversion Utility Description* chapter of the *TMS320C6000 Optimizing Compiler User's Guide*.

4 Keystone I Bootloader FAQ

4.1 General Boot Questions

4.1.1 How to Load Same Image Across Multiple Cores? Examples?

Take a look at the multi-core boot example given in the 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 initializes the SoC and loads 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.

4.1.2 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 Code Composer Studio™ 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".

4.1.3 How Do I Set the Boot Modes?

To set the desired boot modes, see the *Boot Mode DIP Switch Setting* section on the Hardware Setup Guide.

- C6657 EVM
- C6670 EVM
- C6678 EVM

4.1.4 How to Wake Up Secondary Cores From Primary Core(core0)?

During the boot process, the bootloader 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: Processor SDK RTOS at location: pdk_c665x_x_x_xx\packages\ti\boot\examples\pcie\pcieboot_helloworld\src\pcieboot_helloworld.c.

NOTE: The source has been removed from this location due to licensing issue. Locate the source code on this E2E link.

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 the 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 waits in busy loop(while(1)). Pause the debugging.
5. Go to memory browser, enter the address 0x1187FFFC .=> 0xBABEFACE.
6. This means that core0 wakes up core1 successfully. The value 0xBABEFACE has been written by core1 from function "write_boot_magic_number()".
7. 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 and 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
```

4.1.5 How to Find the Silicon Revision of the SoC Mounted on the EVM or Custom Board?

To find the silicon revision, see the *JTAG ID Register (JTAGID) Description* and *C66x DSP Device Nomenclature* sections in the device-specific data manual.

4.1.6 Where Can I Download the Keystone I EVM Resouces Like User/Startup Guide, Schematic, BoM and FPGA Bit File, and so Forth?

- C6657 EVM Design Collateral
- C6670 EVM Design Collateral
- C6678 EVM Design Collateral

4.1.7 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 found in the following 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

4.1.8 How Do I Load the EVM's With Factory Images?

Follow the instruction from the program EVM user's guide (pdf) of the MCSDK/Processor SDK.

4.1.9 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, and so forth by referring/using TI provided utilities like Clocking and DDR leveling spreadsheet, and so forth.
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.

4.1.10 Are There any Validated Big Endian Boot Examples for C66xx?

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

4.1.11 Where Can i Find the ROM Bootloader (RBL) Sources?

Find the RBL sources for PG1.0 and 2.0 from the following:

- C6678 Boot ROM source
- C6657 Boot ROM source
- C6670 Boot ROM source

4.2 IBL Boot

4.2.1 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
```

4.2.2 Can C6678 be Booted Up Directly From SPI Nor Flash, Without the Participation of Inter-Integrated Circuit (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 (Ethernet Media Access Controller (EMAC), Serial RapidIO (SRIO), PCI Express (PCIe), Hyperlink, Serial Peripheral Interface (SPI), and so forth) and I2C boot mode with bus address 0x50, DSP will initially boot from I2C EEPROM bus address 0x51(IBL) that 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.

To understand the conditions of the PLL locking problem and the workaround, see Advisory 8 in the errata document at http://www.ti.com/product/TMS320C6678. The errata specifies a workaround option for "no-boot, SPI and I2C boot modes", so you can 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.

4.2.3 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?

To solve the issue, follow the step-by-step procedure outlined in this e2e link.

4.2.4 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.

There are two methods 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. To update the configuration table in I2C, see the IBL configuration FAQ.
  - PATH: ~\tools\boot_loader\ibl\src\make\bin
- Modify the same on IBL source and program the re-built IBL binary, for 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;
```

4.2.5 How to Boot MAD Image Directly From RBL (without the EVM's IBL)?

This e2e link might help.

4.2.6 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, and so forth. If you look at the IBL code, there are few transactions made from the IBL code with the FPGA over the SPI interface. To take a look at FPGA related initialization code, see the C66xxInit.c file in the IBL source code. If your design does not 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, see the document 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/t/269182
- https://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/t/362997

4.3 NAND Boot

4.3.1 How to Boot From NAND Flash on C6670/C6678 EVM?

The NAND boot on C6670/78 EVM requires Intermediate Bootloader (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.

For IBL flashing and NAND boot setup, see the Processor SDK RTOS User's Guide.

4.3.2 How to Boot From NAND Flash on C6657 EVM?

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

For IBL booting, see the MCSDK 2.x or Processor SDK User's Guide for NAND boot setup.

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

4.3.3 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 bootloader. The ROM bootloader looks for a valid boot image in a sequential manner skipping (and marking) any bad blocks up to the first 32 blocks on the NAND before reporting a boot failure.

4.3.4 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 bootloader (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 you 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.

4.3.5 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 need to modify it to match the NAND geometry provided in the data sheet of the NAND that you are using. The geometry you have provided for your NAND does not seem to match the NAND geometry that is used on the EVM, which means that 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;
```

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

4.4 SPI Boot

4.4.1 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, 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
```

4.5 SRIO Boot

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

4.6 EMIF16/Parallel NOR Boot

4.6.1 How to Do EMIF16 NOR Flash Boot on C66xx Devices? Example?

For an example, see the EMIF16 NOR flash boot summary in the following:

- 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

4.6.2 Would NOR Code for EMIF be the Same as NAND Code? Is There Any Example for EMIF16 NOR Writer?

See the e2e thread for simple NOR writer:

- https://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/p/509244/1850271#1850271
- http://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/p/330496/1153703#1153703

5 References

- Texas Instruments: *KeyStone Architecture DSP Bootloader User's Guide*
- Texas Instruments: *Multicore Fixed and Floating-Point Digital Signal Processor Data Sheet*
- Texas Instruments: *TMS320C6670 Multicore Fixed and Floating-Point System-on-Chip Data Manual*
- Texas Instruments: *TMS320C6655 and TMS320C6657 Fixed and Floating-Point Digital Signal Processor Data Sheet*
- Keystone_Device_Architecture#Keystone_ROM_Boot_Examples_and_Reference_code
- Processor SDK RTOS Boot
- File:C6678_directROM_boot_examples
- File:C6657_directROM_Boot_example
- Processor SDK RTOS Boot Sequence
- Processor SDK RTOS Flash Writers
- Texas Instruments: *TMS320C6000 Assembly Language Tools v7.4 User's Guide*
- Texas Instruments: *TMS320C6000 Optimizing Compiler v 7.6 User's Guide*
- https://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/t/384473
- https://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/p/425551/1519565#1519565
- http://www2.advantech.com/Support/TI-EVM/6670le_of.aspx
- http://www2.advantech.com/Support/TI-EVM/6678le_of.aspx
- 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/362997

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), 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, 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 (www.ti.com/legal/termsofsale.html) 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.

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