Application Report
SPRAB41F–January 2014–Revised January 2019

Using the OMAP-L132/L138 Bootloader

Joseph Coombs

ABSTRACT

This application report describes various boot mechanisms supported by the OMAP-L132/L138 bootloader read-only memory (ROM) image. Topics covered include the Application Image Script (AIS) boot process, an AISgen tool used to generate boot scripts, protocol for booting the device from an external master device, a UART Boot Host GUI for booting the device from a host PC, and any limitations, default settings, and assumptions made by the bootloader.

Project collateral discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/sprab41.

Contents

List of Figures

List of Tables

Trademarks

1 Introduction

The OMAP-L132/L138 bootloader resides in the ROM of the device. This document describes the boot protocol used by the bootloader, discusses tools required to generate boot script, and talks about limitations/assumptions for the bootloader.

The OMAP-L132/L138 bootloader has undergone multiple revisions. To check the version of your device, perform the following steps.

1. Connect to the device in the Code Composer Studio™ software.
2. Select View → Memory.
3. Enter the address of the beginning of the ROM, 0xFFFD0000, at the top of the memory window.
4. Select *Character* mode at the bottom.

The text d800k008 should appear in the memory window at offset 0x08. For earlier ROM revisions, the text could also appear as d800k002, d800k004, or d800k006. It's important to know your ROM revision when generating boot images. If you don't see any of these values in the memory window, this document is not applicable to your device. An abbreviated summary of the ROM boot loader revision history can be found in Appendix E.

2 Boot Modes

The bootloader supports booting from various memory devices (master mode) as well as from an external master (slave mode). A complete list of the supported boot modes and the configuration of the boot pins to select each one of them can be found in Appendix A.

All boot modes, except the host port interface (HPI) and two out of the three NOR-boot modes, make use of the AIS for boot purpose. AIS is a Texas Instruments, Inc. proprietary boot script format widely used in TI devices. All boot modes supporting AIS present a unified interface to you. AIS and AISgen, the tool used to generate AIS, will be discussed in detail later in this document.

There are few boot modes that do not make use of AIS and have a special boot interface. For instance,

- The HPI boot method requires the HPI host to load the application image to the device memory and does not use AIS.
- There are three methods to boot from a NOR Flash, only one of which uses AIS.

Each of these will be discussed later in the document.

3 Non-AIS Boot Modes

3.1 NOR Boot

NOR (or parallel Flash) boot happens from a NOR Flash device connected to the external memory interface (EMIFA) peripheral on EMA_CS[2]. For this boot mode, the bootloader configures EMIFA for 8-bit access and reads the first word from the NOR Flash. This first word indicates if the NOR Flash should be accessed in 16-bit or 8-bit mode, as well as which boot method to be used. This word is interpreted as shown in Figure 1

The NOR boot configuration word register is shown in Figure 1 and described in Table 1.

Figure 1. NOR Boot Configuration Word

31	12	11	8
Reserved		COPY	

7	6	5	4	3	1	0
Reserved		METHOD		Reserved		ACCESS

Table 1. NOR Boot Configuration Word Field Descriptions

Bit	Field	Value	Description
31-12	Reserved	0	Reserved
11-8	COPY		Length of data to copy from the base of the NOR Flash to the base of the Shared RAM. This value is used only for the Legacy NOR boot method.
		0x00	1 KB
		0x01	2 KB
		...	
		0x0E	15 KB
		0x0F	16 KB
7-6	Reserved	0	Reserved
5-4	METHOD		Boot method
		0x0	Legacy NOR boot
		0x1	Direct NOR boot
		0x2	AIS NOR boot
3-1	Reserved	0	Reserved
0	ACCESS		EMIFA access mode
		0x0	8-bit access
		0x1	16-bit access

If ACCESS == 0x1, bootloader reconfigures EMIFA for 16-bit access before using specified boot METHOD to boot from NOR. The default configuration of the bootloader is for an 8-bit access.

3.1.1 Legacy NOR Boot

When METHOD==0x0, the Legacy NOR boot option will be executed. For Legacy NOR boot, the bootloader copies a block of data, whose size is indicated by the COPY field, from the start of NOR Flash (address 0x60000000) to the start of Shared RAM (0x80000000). This block of data should hold a secondary bootloader, as shown in Figure 2.

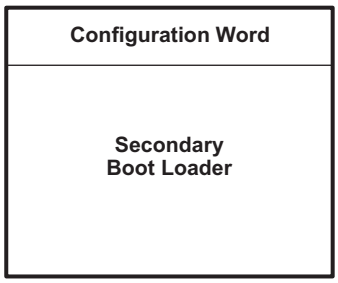


Figure 2. Structure of Secondary Bootloader for NOR Boot

After copying the required data to Shared RAM, the bootloader transfers control to the secondary bootloader by branching to address 0x80000004.

3.1.2 Direct NOR Boot

When METHOD==0x1, the Direct NOR boot option will be executed. For Direct NOR boot, the bootloader transfers control directly to the secondary bootloader present in NOR Flash by branching to address 0x60000004. The secondary bootloader is directly executed from there.

3.1.3 AIS NOR Boot

When METHOD==0x2, the AIS NOR boot option will be executed. When booting with this method, the bootloader expects the AIS image to start from address 0x60000004, which is mapped to NOR Flash.

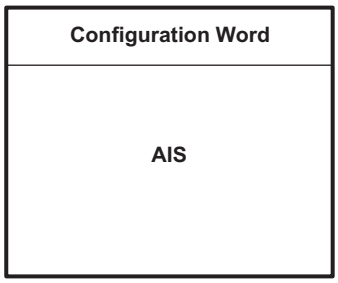


Figure 3. Placement of AIS for NOR Boot

The AIS boot method is described in detail later in this document.

3.2 Host Port Interface (HPI) Boot

HPI boot happens from the HPI0 peripheral in 16-bit mode. The sequence to boot from HPI is listed below:

- Bootloader interrupts the host by setting the HINT bit, in the HPIC register, to inform that it is ready and that the host can start loading the application image to device memory.
- Host acknowledges this interrupt by clearing the HINT bit.
- Host loads the application image to device memory and writes application entry point to location 0x80000000 in device memory.
- Host reads back the final word it wrote to device memory to make sure all HPI writes have completed successfully.
- Host interrupts bootloader by setting DSPINT bit in HPIC register to inform that loading of application image is complete.
- Bootloader acknowledges host by clearing DSPINT bit.
- Bootloader reads application entry point (written by host) from address 0x80000000 and branches to it.

3.3 Emulation Debug Boot

The emulation debug boot powers on the device, but does not load or execute any application code. Instead, the ARM falls into an idle loop immediately after the device powers on. An emulation connection (JTAG) must then be established to load program data or perform any other operations.

The bootloader powers the DSP on before idling the ARM. This allows emulators to connect to the DSP core immediately after boot as well.

4 Application Image Script (AIS) Boot

AIS is a format of storing the boot image. Apart from the HPI and two NOR-boot modes described above, all boot modes supported by the OMAP-L132/L138 bootloader use AIS for boot purposes.

AIS is a binary language, accessed in terms of 32-bit (4-byte) words in little endian format. AIS starts with a magic word (0x41504954) and contains a series of AIS commands, which are executed by the bootloader in sequential manner. The Jump & Close (J&C) command marks the end of AIS.

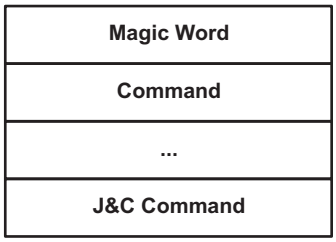


Figure 4. Structure of AIS

Each AIS command consists of an opcode, optionally followed by one or more arguments, followed by optional data.

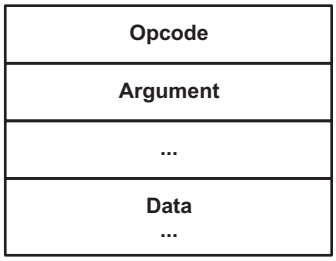


Figure 5. Structure of an AIS Command

The opcode and its arguments are each one word (4 bytes) wide. If the length of data is not a multiple of 4 bytes, it is padded with zeros to make it so.

Knowledge of AIS commands is not required to use the bootloader, but will be discussed in the following sub-sections for completeness. You can skip to the next section if knowledge of AIS commands is not desired.

4.1 Section Load Command (0x58535901)

The user application consists of a number of initialized sections and an application entry point. The Section Load command is used to load each initialized section of the application to device memory.

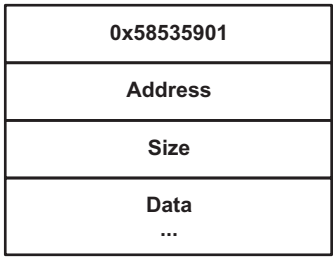


Figure 6. Section Load Command

This command takes two arguments: address and size of the section to be loaded, followed by contents of the section (data). If the length of the section content is not a multiple of 4 bytes, appropriate zero padding is added to make it so; zero padding is not reflected in the SIZE argument.

When CRC checking is enabled, the address and size arguments are fed into the CRC update function in that order. After the section is fully loaded to memory, the section data is fed into the CRC update function beginning at the specified address.

4.2 Section Fill Command (0x5853590A)

The Section Fill command is an optimized version of the Section Load command, which is used when a section is completely filled with a pattern (for example, 0x00 or 0xFF).

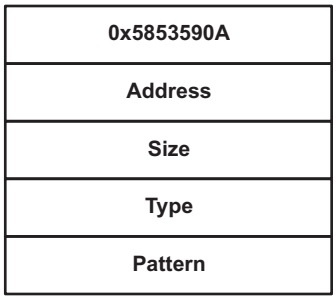


Figure 7. Section Fill Command

This command takes four arguments: address and size of the section to be filled, the type of memory access, and the pattern that will fill the memory.

Table 2. Type Word Values for Section Fill Opcode

Value	Length of Data Pattern
0	8-bit
1	16-bit
2	32-bit

When CRC checking is enabled, the address, size, type, and pattern arguments are fed into the CRC update function in that order. After the section is completely filled with the specified pattern, the section data is fed into the CRC update function beginning at the specified address.

4.3 Enable CRC Command (0x58535903)

This command enables calculation of the cyclic redundancy check (CRC) over the user-application data loaded using the Section Load/Section Fill commands.

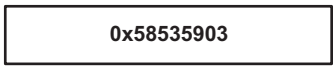


Figure 8. Enable CRC Command

This command does not take any arguments or data.

4.4 Disable CRC Command (0x58535904)

This command disables the calculation of the CRC.

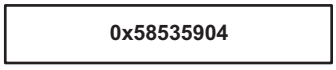


Figure 9. Disable CRC Command

This command does not take any arguments or data.

4.5 Validate CRC Command (0x58535902)

This command is used to validate the CRC calculated by the bootloader.

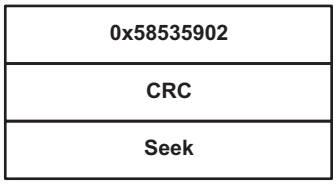


Figure 10. Validate CRC Command

This command takes two arguments: the CRC and the seek value. The CRC is the expected value of the CRC with which the calculated CRC should be compared. In the case of a CRC match, the seek value is ignored and the next command is executed. However, in the case of a CRC mismatch, the seek value can be added to the current position in AIS to locate the last Section Load/Section Fill command so that the command can be executed again.

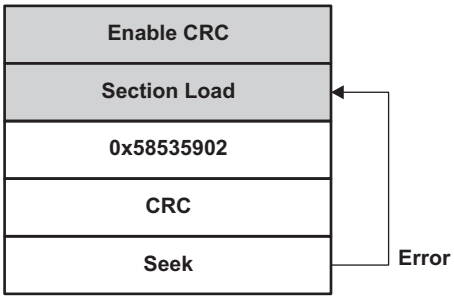


Figure 11. Handling CRC Error

This command behaves differently for master- and slave-boot modes. In master-boot mode, the bootloader reads the expected CRC from the boot device and compares it with the calculated CRC. In case of an error, the bootloader adds the seek value to the current read position in AIS and starts executing commands from that position in AIS.

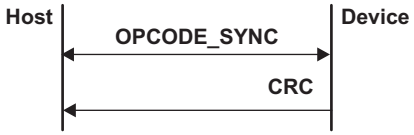


Figure 12. Validate CRC Flow for Slave Mode

In slave-boot mode, on receiving the Validate CRC command, the bootloader provides the calculated CRC to the host. The host then compares this value with the one from AIS and updates its AIS read position, depending on the result of the CRC comparison. In the case of an error, the host sends the Start-Over command to the device so that the bootloader can re-initialize the calculated CRC and be ready to receive the next command.

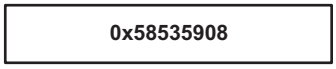


Figure 13. Start-Over Command

The Start-Over command (0x58535908) takes no arguments or data. This command does not appear in AIS and is only used in slave-boot mode by the host to recover from a CRC error.

4.6 Jump & Close Command (0x58535906)

The Jump & Close command is used to mark the end of AIS. On receiving this command, the bootloader closes the boot peripheral, restores the selected configurations of the device to its default state, and then transfers control to the user application.

Figure 14. Jump and Close Command

This command takes one argument: the entry point of the ARM application. This is the address the bootloader transfers control to after closing the boot peripheral. The application starts execution after this jump and the bootloader loses its control over the device.

4.7 Jump Command (0x58535905)

This command is similar to the Jump & Close command, except that the bootloader does not close the boot peripheral and does not change any device state. This command is not used to transfer control to the application. Rather, it is used to execute a temporary code, which may tweak the bootloader or device state. This command is used to add post-ROM features to the bootloader.

Figure 15. Jump Command

This command takes one argument: the address of a temporary function to be called. This function should be loaded to the device memory before the Jump command, and it should also return control to the bootloader after performing its intended task.

4.8 Sequential Read Enable Command (0x58535963)

When booting from a serial peripheral interface (SPI) or inter-integrated circuit (I2C) slave device, the bootloader uses a *random* read method to read the boot image from the slave memory device. This method requires sending a read command and a read address to read each byte. Many recent memory devices support reading in *sequential* method, where data can be read in sequential order without sending a read command/address for each byte. Using this method to read data greatly reduces boot time. The Sequential Read Enable command is used to enable the bootloader to use the sequential read method.

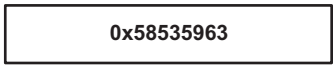


Figure 16. Sequential Read Enable Command

This command takes no arguments or data.

4.9 Function Execute Command (0x5853590D)

The Function Execute command is a generic interface for device-specific initialization functions such as phase-locked loop (PLL) and external memory interface (EMIF) configuration. A set of pre-defined functions are part of the ROM and a function table is maintained by the bootloader with pointers to each of them. The Function Execute command can be used to execute any of them. Details of pre-defined ROM functions that can be called using this command are given in Appendix D.

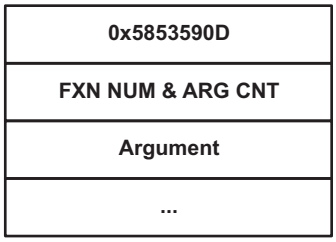


Figure 17. Function Execute Command

The number of arguments in this command is variable. The first argument specifies the function ID (index of function in the function table) in the lower 16 bits and the number of arguments that the function takes in the upper 16 bits. The number of arguments following the first argument matches the number specified in its upper 16 bits.

4.10 Boot Table Command (0x58535907)

The Boot Table (or SET) command writes 8-, 16-, or 32-bit data to any address in device memory. Additionally, it instructs the device to wait for a fixed number of cycles after the memory write occurs. This can allow memory-mapped register writes to take effect before the bootloader moves on to the next opcode.

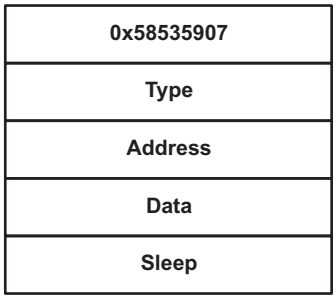


Figure 18. Boot Table Command

This command takes four arguments. First is the type (size and format) of the memory location to be written; the contents of this word are described in the table below. The address comes next, followed by the data. Note that the data is given as 32 bits in the AIS regardless of how many bits will actually be written. The last parameter is the number of cycles to delay execution of the next opcode.

Figure 19. Type Word for Boot Table Opcode

31 24	23 16	15 8	7 0
Reserved	STOP	START	LENGTH

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 3. Type Word for Boot Table Opcode Field Descriptions

Bit	Field	Value	Description
31-24	Reserved	0	Reserved
23-16	STOP	0-31	The highest (or most significant) bit of the custom data field. Only used when LENGTH = 3 or 4.
15-8	START	0-31	The lowest (or least significant) bit of the custom data field. Only used when LENGTH = 3 or 4.
7-0	LENGTH		Size of data word
		0	8-bit
		1	16-bit
		2	32-bit
		3-4	Custom field defined by STOP, START. Data outside this field at the target address will be preserved.
		5-FFhC	Reserved

5 AISgen: Tool to Generate Boot Script (AIS Image)

AISgen is a Windows®-based tool that is used to generate the boot image in AIS format. This tool requires Microsoft® .NET Framework for its operation.

5.1 Installation

Before starting, make sure that you have Microsoft .NET Framework Version 2.0 or later installed on your system. You can download it from the Microsoft website at http://www.microsoft.com.

To install AISgen, download and execute the latest installer from the following URL: http://www.ti.com/lit/zip/sprab41.

It is recommended to install this tool at its default location.

5.2 Getting Started

On successful installation, a link to start AISgen is created in the following folder: Start Menu → Program Files → Texas Instruments → AISgen for D800K008.

Click on this link to start AISgen. It may take some time for the main window to appear.

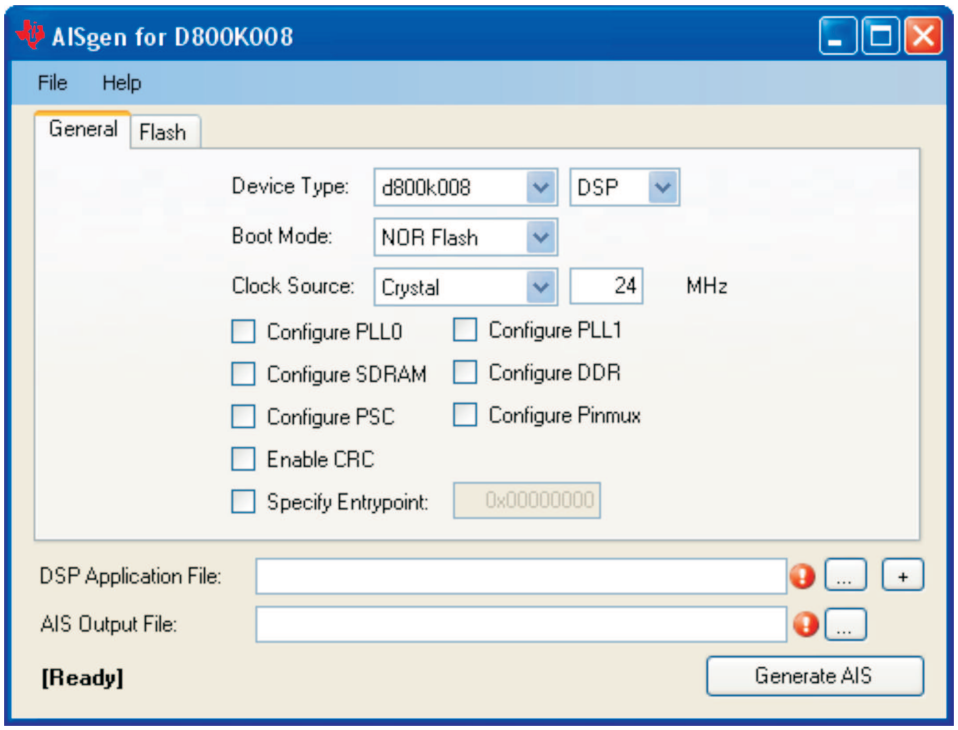


Figure 20. AISgen Main Window

The AISgen window groups controls within tabs to reduce interface clutter. The General tab is always enabled and controls general configuration settings, such as the boot mode selection. Table 4 lists additional tabs that can appear depending on selections made in the General tab.

Table 4. Optional Tabs in the AISgen Window

Tab Name	Enabled By	Description
Flash	Boot mode selection (1)	Configures data width (NOR only) and EMIF timing for Flash interface
Peripheral	Boot mode selection (2)	Configures peripheral speed and enables sequential read
PLL0	Configure PLL0 checkbox	Configures PLL0 clock multiplication and division
SDRAM	Configure SDRAM checkbox	Configures SDRAM interface register settings
PLL1	Configure PLL1 checkbox	Configures PLL1 clock multiplication and division
DDR	Configure DDR2 checkbox	Configures DDR interface register settings
PSC	Configure PSC checkbox	Allows LPSCs to be turned on, turned off, or placed in sync. reset
Pinmux	Configure Pinmux checkbox	Configures Pinmux registers

(1) Enabled for NOR Flash and NAND Flash-boot modes only
(2) Enabled for SPI master, I2C master, and UART-boot modes only

You may notice that some controls are disabled. A control may be disabled for multiple reasons:

- You may not have selected an associated option. For example, when the Specify Entrypoint checkbox is not selected, the adjacent numeric entry field is disabled.
- The control displays a calculated result and cannot be set directly. For example, CPU frequency on the PLL0 tab is a calculated result.

The File menu can be used to remember and re-apply previous settings. File → Save Configuration saves the current settings from all tabs to a file for later use, and File → Load Configuration re-applies settings from a file. Select File → Restore Defaults at any time to restore all fields to the default configuration shown in Figure 20.

You may also notice an icon next to one or more controls (such as the AIS File text box near the bottom of the main window) of a red circle and exclamation point. This icon indicates an improper value in the control left of the icon.

To see error details, hover the mouse over this icon for 2 seconds and an error message tool tip will appear. An icon next to the AIS file specification box indicates that a file name is required to generate AIS.

The area below the AIS file specification box is used to display status messages.

For example, if you click on the Generate AIS button when one or more error indicators are visible, a message appears stating: *One or more parameter error must be corrected*.

When an AIS file is successfully generated, this area will display the AIS file's size. It will also report when the AISgen configuration settings are saved to or loaded from a file.

5.3 Generating AIS

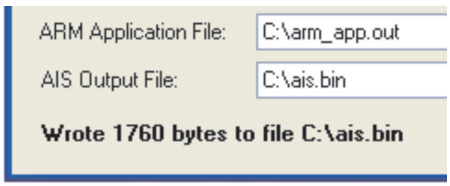
As an example, create a simple AIS that boots an application on the ARM using the default configuration.

Click the first *Device Type* dialog box and select the appropriate ROM revision ID for your device. If you're not sure which option to pick, please refer to the introduction of this document for instructions on how to check the ROM ID of your device. The default selection in the AISgen tool is d800k008, which corresponds to the most recent ROM revision.

Click the second *Device Type* dialog box and select *ARM*. You have to manually specify the ARM application and output AIS files. To specify the AIS file, type `C:\ais.bin` in the AIS file text box (left of error indicator) or use the browser button (right of the error indicator) to do so. Notice that the error indicator vanishes as the text is entered in the AIS file specification box.

Specify an ARM application file (*.out) in the appropriate box by either typing its name in the appropriate box or using the browse button to locate it. A valid ARM application can be created by building a project in Code Composer Studio.

Now, clicking on the Generate AIS button will successfully generate an AIS file (as `C:\ais.bin`). The size of this file depends on the ARM application.

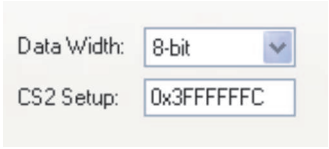


Now you are ready to examine some of the additional options available.

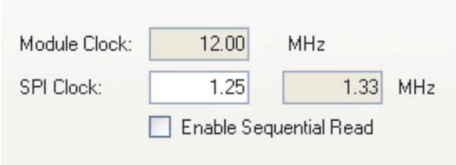
5.3.1 Boot Mode and Boot Peripheral Setup

The General tab allows you to specify the boot mode that you plan to use. Depending on your selection, a Flash or Peripheral tab may appear. These tabs contain additional controls that allow you to optimize boot time (in master mode).

The Flash tab appears when either NOR Flash or NAND Flash-boot mode is selected. This tab contains controls that specify 8- or 16-bit data width (NAND is fixed at 8-bit data width) and interface timing settings for the EMIF flash interface. For more information on the register fields, see *TMS320C674x/OMAP-L1x Processor External Memory Interface A (EMIFA) User's Guide* (SPRUFL6).



The Peripheral tab appears when one of the SPI or I2C master-boot modes is selected or when universal asynchronous receiver/transmitter (UART) boot mode is selected. This tab configures the peripheral speed and defaults to conservative speeds for broader compatibility. These are documented in Table 6. If the boot peripheral supports faster speeds, you can specify a speed so that the bootloader can re-configure clocks and boot faster. The Module Clock field displays the clock input to the boot peripheral from the PLL. This value may change based on PLL configuration.



For SPI and I2C master modes, you can enter a desired speed. Depending on the PLL settings and granularity in peripheral clock division, AISgen calculates the actual speed nearest to the desired speed and shows it in an adjacent box. Recent I2C devices support speeds up to 400 kHz, and SPI devices support speeds up to 33 MHz.

SPI and I2C master modes also support sequential read mode. This option speeds up the boot process by allowing the bootloader to repeatedly read sequential data words from a slave memory device after sending only one read request and address to the device. Make sure that your slave memory device supports sequential read mode before selecting the Enable Sequential Read checkbox.

For UART-boot modes, baud rate is fixed at 115.2 kbps for boot purposes and cannot be changed. However, AISgen will still calculate an actual baud rate based on your PLL configuration and show it in the Peripheral tab.

5.3.2 Phase-Locked Loop (PLL) Setup

When the device is taken out of reset, the PLL modules are set in bypass mode by default. During application development, the PLL configuration is typically handled by a GEL file. In a production environment, the bootloader can configure the PLL. The AISgen General tab specifies the clock source and its frequency, and the PLL0 and PLL1 tabs can configure the multipliers and dividers to reach the desired CPU frequency.

The PLL0 tab appears when the Configure PLL0 checkbox is selected. This tab configures the multiplier and dividers for PLL0. When changing any of these values, calculated frequencies will update to reflect the changes. The multiplier and divider values that you enter are the actual multiplication and division factors (x), not the values that get programmed into the corresponding PLL registers (x - 1). The default settings of the PLL dividers that are not configurable are shown in Table 5.

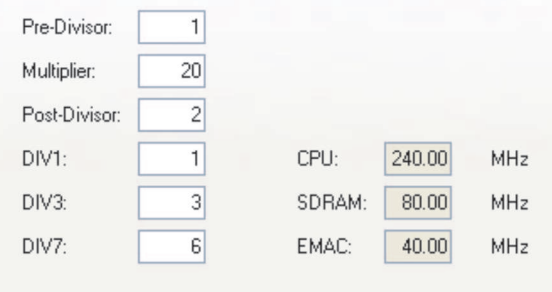
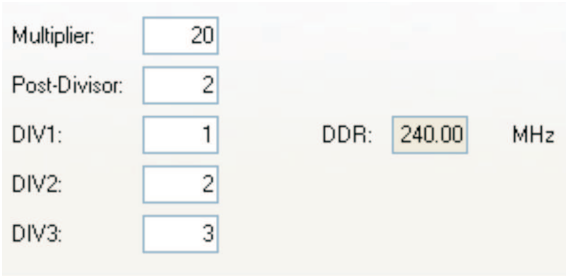


Table 5. Values of Non-User Configurable PLL0 Dividers (relative to DIV1)

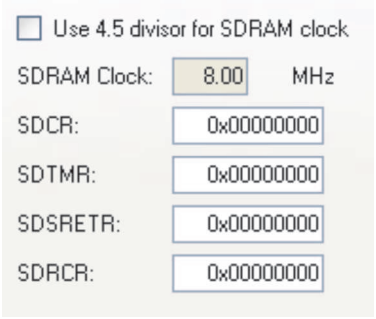
Divider	Configured To
PLL0 DIV2	Divide by 2
PLL0 DIV4	Divide by 4
PLL0 DIV6	Divide by 1

The PLL1 tab appears when the Configure PLL1 checkbox is selected. Note that PLL1 must be configured if DDR is configured. Selecting the Configure DDR checkbox will automatically select Configure PLL1 as well. This tab is similar to the PLL0 tab, but its settings are applied to PLL1 instead of PLL0.



5.3.3 Synchronous Dynamic Random Access Memory (SDRAM) Setup

The SDRAM tab appears when the Configure SDRAM checkbox is selected. This tab configures external SDRAM access through EMIFA. The EMIFA registers need to be configured before any access to SDRAM is made.



Values entered for each the EMIF register are directly programmed to the corresponding register by the bootloader. The Use 4.5 divisor for the SDRAM checkbox enables the use of an alternate clock divider. The SDRAM clock field displays the calculated clock speed, and is the same as the identical field in the PLL0 tab.

5.3.4 DDR Setup

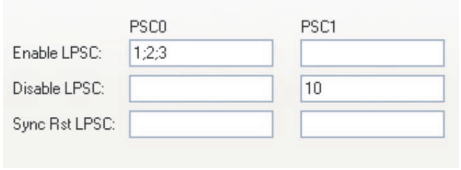
The DDR tab appears when the Configure DDR checkbox is selected. This tab configures external DDR memory access through the DDR Controller. The DDR Controller registers need to be configured before any access to DDR is made. The DDR controller can be configured to access DDR2 or mDDR memory. Certain registers are only used for mDDR.

Values entered for each DDR register are directly programmed to the corresponding register by the bootloader. The Use direct clock from PLL1 checkbox allows the DDR Controller to use the output of PLL1 before the post divider is applied. The DDR clock field displays the calculated clock speed for DDR access, and is the same as the identical field in the PLL1 tab.

5.3.5 PSC Setup

The PSC tab appears when the Configure PSC checkbox is selected. This tab instructs the bootloader to explicitly set the state of specified LPSCs within power and sleep controller (PSC). LPSCs can be enabled, disabled, or placed in synchronous reset. Each text box can contain an arbitrary number of LPSC numbers separated by semicolons (;). The bootloader automatically enables the LPSC for the active boot peripheral.

Care must be exercised when specifying the state of LPSC modules. The AISgen tool does not comprehend the function of each LPSC, and it will not warn you if inappropriate choices are made. Turning on extra LPSCs can cause unnecessary power consumption, and turning off critical LPSCs can disable the boot peripheral or even shut down the CPU. For more information on the use of each LPSC, see the device-specific data sheet .



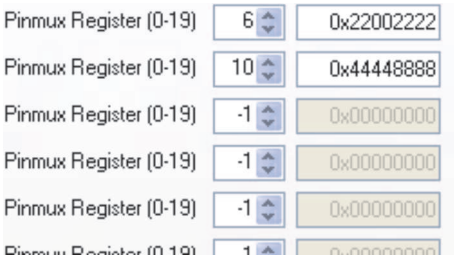
NOTE: Enabling PSC1 module 8 requires a forced transition that is not supported by the bootloader. The AISgen application will display an error message if you attempt to enable this module.

5.3.6 Pin Multiplexing Setup

The Pinmux tab appears when the Configure Pinmux checkbox is selected. This tab instructs the bootloader to apply specified pin multiplexing (pinmux) settings to one or more pinmux register. The bootloader automatically applies pinmux necessary to operate the active boot peripheral.

To set a pinmux register during boot, set a numeric up/down field to the desired pinmux register (0-19). The adjacent entry field will then specify the value for that pinmux register. The AISgen application only allows one bit per hexadecimal digit to equal 1; the only valid digits are 0, 1, 2, 4, and 8. Additional restrictions may apply for a particular register. For more information on the pinmux registers, see your device-specific *System Reference Guide*.

Take care not to undo the pinmux setting for the active boot peripheral. Doing so can cause the boot process to fail.



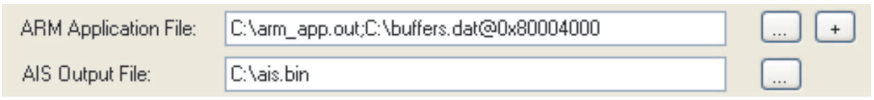
5.3.7 Application File Selection

The bootloader requires a valid application in order to boot the ARM. The full path to an application file must be specified in the ARM Application text field.

NOTE: As of version 1.3, AISgen recognizes both COFF and ELF input files as application files. Files in any other format will be treated as binary files.

Non-application files may also be used. AISgen automatically parses input files to determine whether they are valid application files. Files that don't match the expected format are treated as binary files. Binary files require a target address, which is specified as "@<32-bit hex address>" immediately following the file name. Note that SDRAM or DDR addresses will only load successfully if the appropriate configuration is applied; the AISgen utility will not automatically configure external memory access. For more information, see Section 5.3.3 and Section 5.3.4.

The AISgen utility allows multiple application and/or binary files to be combined into a single AIS file. Each file name must be separated by a semicolon (;). If an entrypoint is specified in the General tab, that entrypoint always takes precedence. Otherwise, the entrypoint of the first application file is used. If no application file is specified (all files are binary files), an entrypoint must be specified. For more information specifying an application entrypoint, see Section 5.3.10.2.



Files can be entered directly into the ARM Application File text box or selected graphically using the browse button (…). To add another application or binary file without deleting the current contents of the text box, use the add button (+).

5.3.8 AIS File Selection

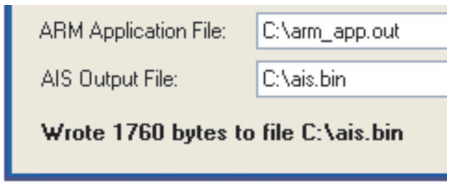
Finally, you must specify the name of the AIS file that you want to generate. You can manually enter the full path of the desired AIS file or click on the browse button (…) to select it graphically.

If the specified AIS file has *.h or *.c as an extension, it is generated in C header format that can be inserted into the source of another application. This format is plain text and can be read to check information about embedded AIS commands, their arguments, and optional data. This can be useful for diagnostic purposes.

When any other extension is specified, the file is generated in AIS Binary format. A file in this format is typically written to a Flash memory device.

5.3.9 Status and Messages

After specifying all the boot parameters, generate the AIS by clicking on the Generate AIS button at the lower right corner of the AISgen main window. If AIS generation succeeds, a message will appear reporting the AIS file size.



If AIS generation fails, a message reporting the error encountered will appear. Possible error conditions include specifying only binary input files without specifying an entrypoint, invalid AIS file names, or file access errors.

The AISgen utility also stores more detailed information in the file AISgen_log.txt. Before viewing this file, close the AISgen utility to ensure all log data has been written. Also, be aware that this file is reset (overwritten) every time the AISgen utility is run.

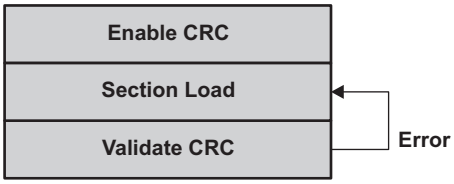
5.3.10 Additional AIS Options

In addition to the basic options, the bootloader also supports error checking (CRC) and the ability to specify a custom entry point for the application following boot.

5.3.10.1 CRC

When the Enable CRC checkbox is selected, AISgen adds extra commands in the AIS to check for errors while transferring/loading the application data.

For master boot modes, the bootloader calculates the CRC over each section of the application data and checks it against the expected value from AIS. In case of an error, the bootloader loads the section again, re-calculates the CRC and checks again with the expected value. If a CRC error is found in three successive attempts, the boot process is aborted.



For slave-boot modes, the external master reads the calculated CRC from the device and validates it against the expected value from AIS. In case of an error, it is up to the external master to decide how many times it wants to retry loading the section before reporting failure.

5.3.10.2 Specifying the Application Entrypoint

When the Specify Entrypoint checkbox is selected, AISgen allows a custom entrypoint to be specified in the adjacent text box. This is the address to which the bootloader will exit after boot completes. It is often unnecessary to specify this value; AISgen will automatically read the entrypoint from an ARM application file (in COFF or ELF format).

If an entrypoint is specified, that entrypoint always takes precedence. Otherwise, the entrypoint of the application file is used. If more than one application file is specified, the entrypoint from the first file is used, but a manually specified entrypoint supersedes any number of application files.

If no application file is specified (all files are binary files), an entrypoint must be specified manually.

5.3.11 Command Line Usage

The AISgen tool supports command line usage to facilitate the automation. To run AISgen from a terminal window, follow this procedure:

1. Create a configuration file (*.cfg) using the GUI. This file will specify all parameters needed to generate an AIS file, including the input and output file names.
2. Open a terminal window and navigate to the AISgen directory.
3. Run AISgen with the following usage:

```
AISgen_d800k008.exe –cfg=″<CFG file
                    name>″
```

This procedure allows quick generation of an updated AIS image after modifying the underlying application or binary input files.

6 Master Boot – Booting From a Slave Memory Device

6.1 I2C EEPROM Boot

To boot from a slave memory device connected to an I2C peripheral, the AIS contents (in binary format) can be flashed directly to the memory device. The boot image must be placed at address 0 of the memory device. The memory device must respond to I2C slave address 0x50.

6.2 SPI EEPROM or Flash Boot

To boot from a slave memory device connected to an SPI peripheral, the AIS contents (in binary format) can be flashed directly to the memory device. The boot image must be placed at address 0 of the memory device. SPI EEPROMs must use 16-bit addressing, while SPI flash chips must use 24-bit addressing. The read command opcode for the SPI flash device must be 0x03.

6.3 NOR Flash Boot

To boot from a NOR Flash, a configuration word is required before AIS as shown in Figure 3. NOR Flash should be connected to EMA_CS[2] of the EMIFA peripheral.

6.4 NAND Flash Boot

To boot from NAND Flash, the AIS should be written to NAND block 1 (NAND block 0 is not used by default) in a sequential manner, skipping (and marking) any bad blocks. The bootloader detects a bad block by examining the spare bytes in the first and second pages of the current block. For NAND devices that comply with the ONFI standard, the first and last pages are used instead. Figure 21 illustrates the structure of a NAND data page. Each page includes N segments of spare bytes, where N is the number of data bytes per page divided by 512. Each segment of spare bytes contains 6 test bytes and 10 ECC bytes. For those pages that are checked during bad block detection, all the test bytes in each segment must equal FFh; any other value indicates that the page (and its entire block) is bad and should not be used.

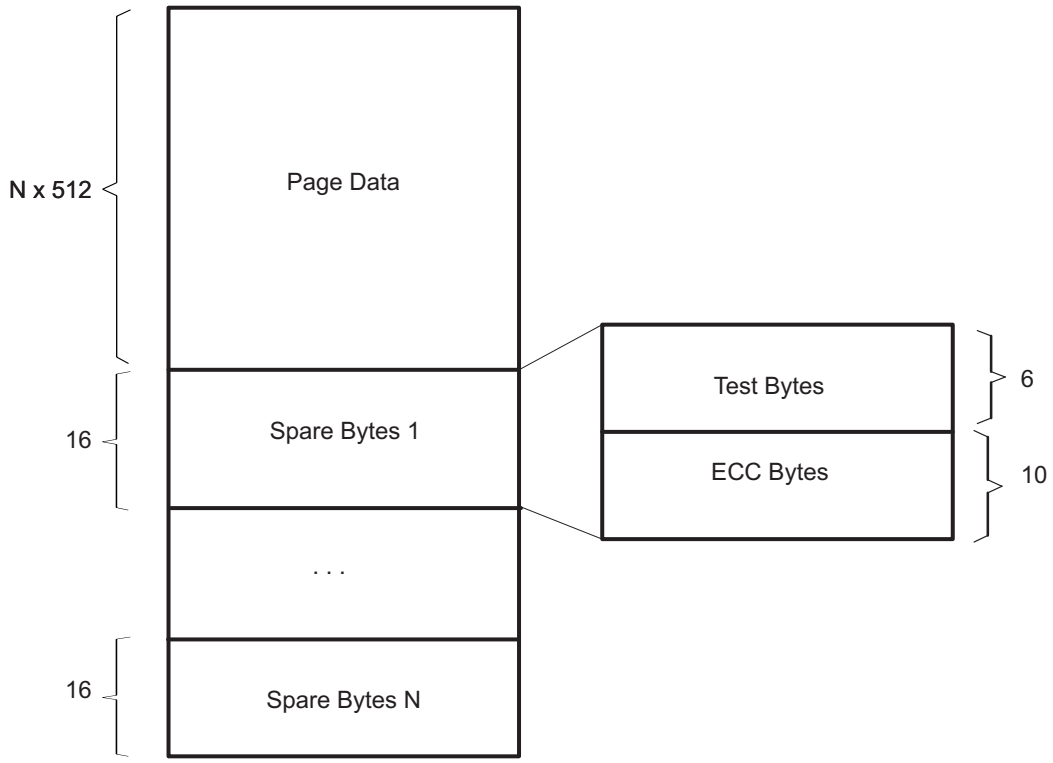


Figure 21. Structure of NAND Page and Spare Bytes

NAND Flash should be connected to EMA_CS[3] of the EMIFA peripheral. The ALE and CLE pins of the NAND device should be connected to the EMA_A[1] and EMA_A[2] pins of the EMIFA peripheral, respectively. Complete details of supported NAND devices are available in Appendix B.

6.5 MMC/SD Boot

The MultiMedia Card/Secure Data (MMC/SD) boot mode is compliant with version 2.0 of the SD specification and version 4.2 of the enhanced MultiMedia Card (eMMC) specification. The boot loader and MMC/SD peripheral does not support features of the eMMC 4.3 or 4.4 specifications related to boot (no boot operation mode and no support for boot partitions). The AIS boot image is expected to be in the user data area of the memory device, written to address 0 of that region. The bootloader detects the AIS image by checking for the magic word (0x41504954). If the magic word is not found, the bootloader increments the starting address by 0x200 and tries again. The bootloader fails if the magic word is not found within the first 2 MB of the memory card.

Typically, the ROM boot loader will first try to detect an SD card. If that fails (a timeout occurs), the boot loader will then attempt to find an MMC or eMMC device. In cases where it is known that an SD card will never be present (for example, a memory card slot is not used and an eMMC device is placed on the PCB), the BOOT[5] pin of the SoC device should be pulled high to force the bootloader to skip the SD detection. This will guarantee the quickest boot in these circumstances. For further details, see Section 9.

7 Slave Boot – Booting From an External Master Host

When booting from an external host processor, the host processor acts as the communication master, and the bootloader acts as the slave. Since the bootloader doesn't have direct access to the AIS, the host processor must transfer it to the bootloader through a well-defined protocol explained in the following sections. An AIS interpreter is required on the host processor to control this transfer. A reference implementation of the host-side AIS parsing process is provided with the software collateral for this application report. For more information, see Section 8.3.

7.1 About the AIS Interpreter on the Host

The AIS interpreter on the host processor is responsible for transferring the AIS to the bootloader. Therefore, the host has to understand the transfer protocol and implement the required handshake mechanism.

NOTE: For the sake of simplicity, the AIS interpreter on the host processor will simply be referred to as *host* in these sections.

It is important to establish a reliable link between the host and the bootloader before starting a serial boot load using the AIS. Once this link is established, the AIS can be transferred to the bootloader. This process is divided into three states: start-word synchronization (SWS), ping Op-code synchronization (POS) and op-code synchronization (OS).

7.2 Start-Word Synchronization (SWS)

After power ON reset (POR), the bootloader takes some time to initialize and configure the boot peripheral. Similarly, the host also takes its own time to initialize and prepare to boot the device. A SWS mechanism is used to synchronize the device and host after POR.

To achieve SWS, the host repeatedly sends transmit-start-word (XMT_START) to the device until it receives the proper response (receive-start-word or RECV_START) from the device.

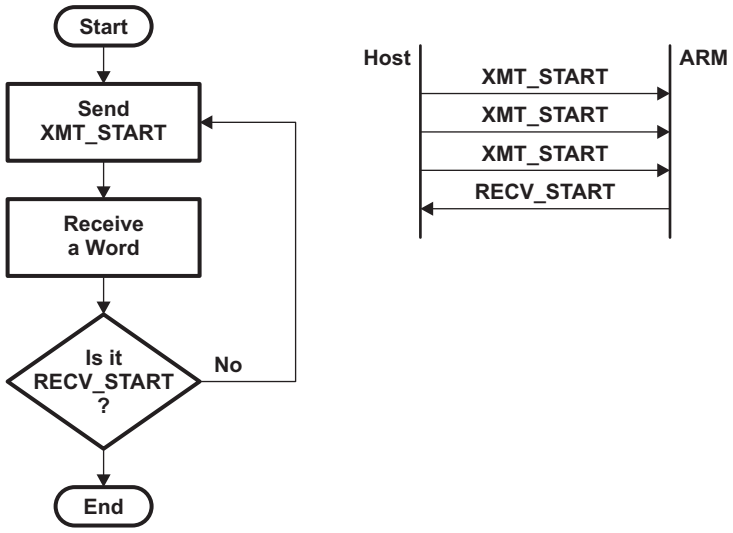


Figure 22. Flowchart: Start-Word Synchronization

For the SPI and I2C slave modes, the bootloader operates the SPI/I2C peripheral in 16-bit mode, so that both start words are 16-bit (0x5853 & 0x5253). For the UART-boot mode, the bootloader operates the UART peripheral in 8-bit mode, so that both start words are 8-bit (0x58 & 0x52).

NOTE: While start-words are 8-/16-bit, all other data including op-codes and CRC are 32-bit, and all 32 bits need to be transmitted to the bootloader regardless of the boot mode.

For UART boot mode, the bootloader transmits the ASCII string *BOOTME* to the host before it is ready to begin SWS. This transmission occurs only once immediately following reset, and the host should not initiate SWS until after receiving this string. SWS and all subsequent steps proceed as normal using binary data transmission; no other data should be sent or received in ASCII format.

NOTE: When the bootloader begins running, the device's PLL is configured in bypass mode. If the external host device is fast enough, it will need to insert a delay after each transmission to allow the bootloader time to finish processing the previous data. The need for such delays can be alleviated later in the boot process by configuring the PLL via the Function Execute Command.

7.3 Ping Op-Code Synchronization (POS)

The POS is used to further ensure that the serial link between the host and the device is reliable for exchanging boot information.

After successful SWS:

- The host sends the PING_DEVICE (0x5853590B) op-code to the bootloader and receives the RECV_PING_DEVICE (0x5253590B) as acknowledgment from the bootloader.
- The host sends an arbitrary 32-bit number (N) to the bootloader and gets back the same number (N) from the bootloader.
- The host counts from 1 to N, sending each number to the bootloader and receiving the same number in response. Each number sent or received is a 32-bit value.

NOTE: All multi-word values transmitted by the host should be sent in little-endian order. For instance, the Ping opcode (0x5853590B) is sent as 0x0B, 0x59, 0x53, 0x58 in 8-bit mode or 0x590B, 0x5853 in 16-bit mode. The same applies to responses received from the bootloader.

The count (N) is selected by the host (OEM) and should be appropriate to assure successful communication between the device and host. The recommended value of N is 2.

Figure 23 shows the flowchart of how the POS should be implemented on the host:

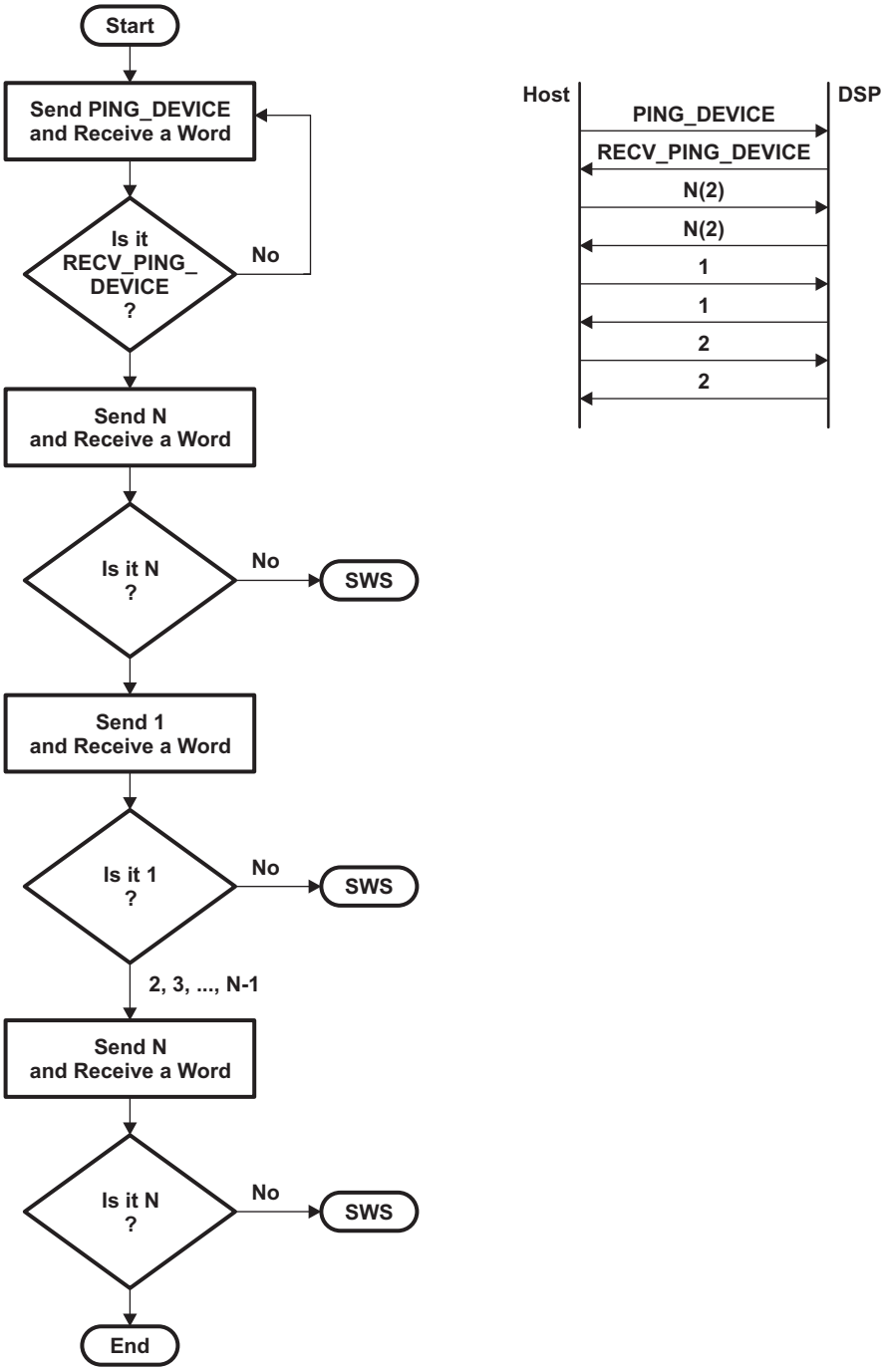


Figure 23. Flowchart: Ping Op-Code Synchronization

After the successful execution of POS, the host starts reading AIS commands (magic number is checked and ignored by the host) from its source and starts booting the device.

7.4 Opcode Synchronization (OS)

As the bootloader may take an indeterminate amount of time to execute an AIS command, a handshake mechanism is needed between the host and the bootloader before the host can send any command to the bootloader. The opcode synchronization method is used for this purpose.

All opcodes, including PING_DEVICE, that are transmitted by the host to the bootloader are of the form 0x585359##, where ## varies for individual opcodes. The bootloader acknowledges each opcode with a corresponding RECV opcode. The RECV opcodes are generated from the original opcodes by changing the most significant byte to 0x52. Thus, they are of the form 0x525359##.

Not getting a correct response (RECV opcode) from the bootloader indicates that the bootloader is busy executing the previous command. The host should continue sending the opcode until it is successfully acknowledged by the bootloader. Figure 24 shows the flowchart of how the OS should be implemented on the host:

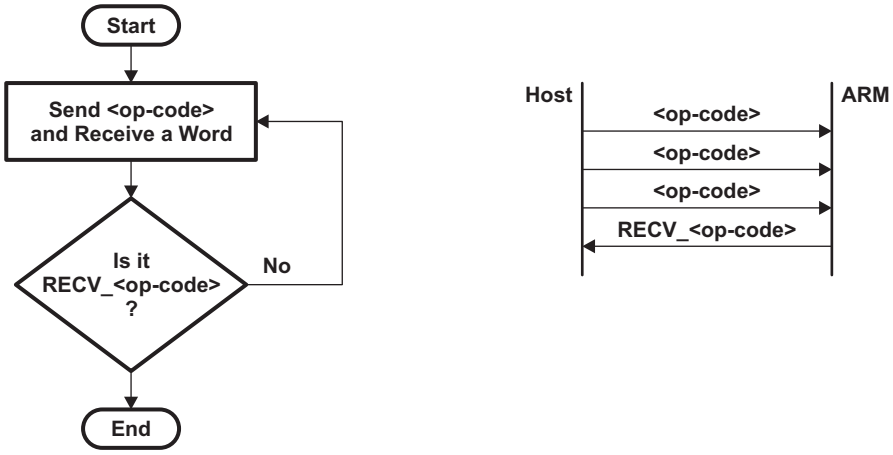


Figure 24. Flowchart: Op-Code Synchronization

The host is required to understand each command and supply the required arguments and data to the bootloader.

8 UART Boot Host - Using Your PC as a UART Boot Master

UART Boot Host is a Windows™ based tool that is serves as an external boot master for UART boot mode. It uses a binary AIS file generated by AISgen to execute the entire UART boot process using a COM port on the host PC. This tool requires Microsoft .NET Framework Version 2.0 to run.

8.1 Getting Started

The UART Boot Host utility is installed alongside the AISgen utility. For installation instructions, see Section 5.2. After installing AISgen, look for the UART Boot Host in the UartHost subfolder. Run UartHost.exe to begin.

The UART Boot Host utility consists of a single window. This window handles all configuration, allows you to start and stop the boot process, and reports status and error messages as the device boots (see Figure 25).

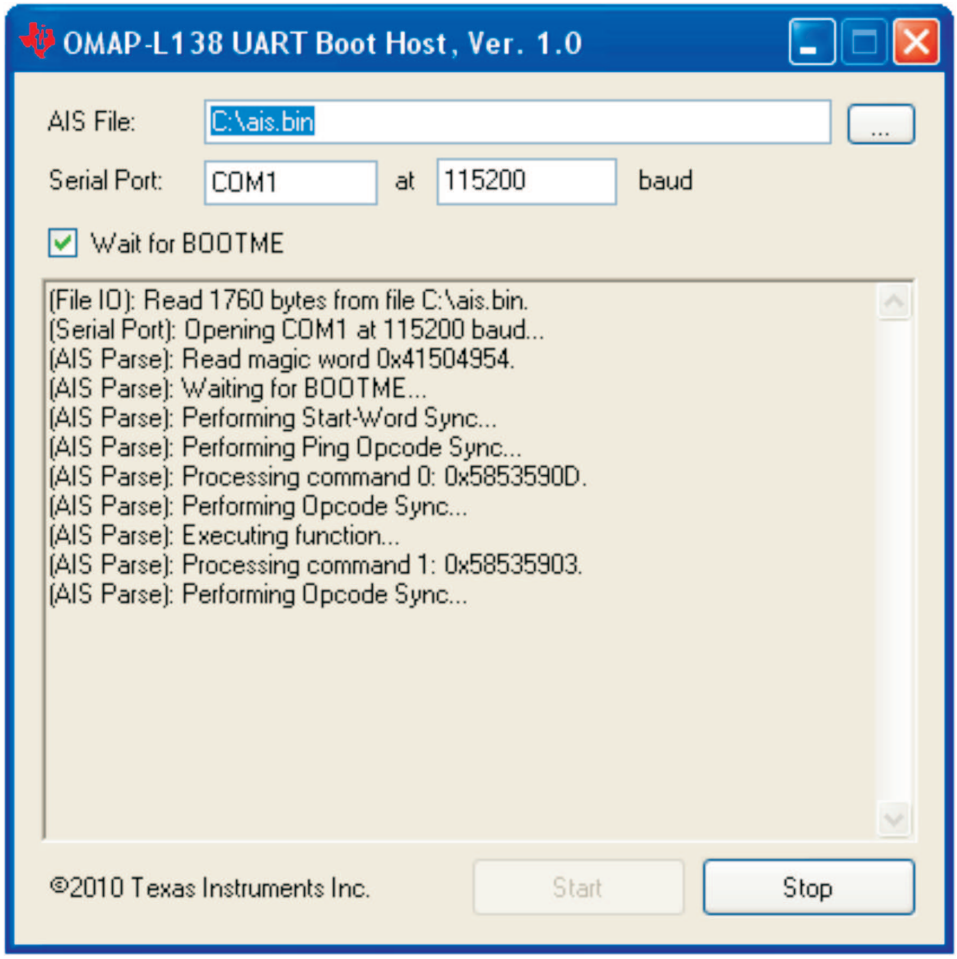


Figure 25. UART Boot Host utility

The AIS File text box specifies the AIS file to be used during boot. Note that only binary AIS files should be used; AIS files in C header format are invalid. The file can be typed manually or found using the browse button ("…").

The Serial Port text box specifies the name of the serial port that will be used to boot the device. For most PCs, the serial port has a name like *COM1* or *COM2*. The default baud rate used by the bootloader is 115.2 kbps. For more information on using different baud rates, see Section 9.

The Wait for BOOTME checkbox allows you to specify whether the PC waits to receive *BOOTME* on its serial port before beginning boot. This should be checked by default.

The large read-only text box displays status and error message during the boot process. It will initially be blank and should automatically clear itself at the beginning of each boot process.

The Start and Stop buttons, respectively, initiate and abort the boot process. The Start button is only active when no boot is currently in progress and the Stop button is only active when boot is currently in progress.

8.2 Booting the Device

If *Wait for BOOTME* is checked:

1. Connect the PC serial port to the boot device.
2. Choose a binary AIS file and serial port in the UART Boot Host GUI.
3. Click Start
4. Turn on (or reset) the boot device

If *Wait for BOOTME* is not checked:

1. Connect the PC serial port to the boot device.
2. Choose a binary AIS file and serial port in the UART Boot Host GUI.
3. Turn on (or reset) the boot device
4. Click Start

The boot process runs until completing successfully or encountering an error. A series of messages will appear in the large text box as the device boots. Messages are prefixed by category names:

- **File I/O** – PC system messages related to opening and reading the specified AIS file.
- **Serial Port** – PC system messages related to sending and receiving data with the UART device.
- **System** – Miscellaneous PC system messages.
- **AIS Parse** – Messages related to the boot master process as outlined in Section 7.

When a message stating *boot completed successfully* appears, the boot device is already executing the application contained within the AIS file.

A boot in progress can be cancelled at any time using the Stop button.

8.3 The AIS_Util.cs Source Code

The UART Boot Host includes a single C# source file named AIS_Util.cs, which serves as a reference implementation of the boot master requirements laid out in Section 7. This source file is used by the UART Boot Host application itself, and is provided as-is with no direct support.

9 Boot Requirements, Constraints and Default Settings

9.1 General Comments

- Non-NAND Memory Usage: The bootloader uses 2 KB of ARM local RAM starting from 0xFFFF0000. This memory should not be used by any initialized section of the user application.
- Peripheral Configuration: The bootloader automatically configures the PSC and pinmux for the boot peripheral. It is not necessary to manually configure the PSC and pinmux for the boot peripheral.
- DDR Configuration: If the DDR configuration ROM function is called, then the bootloader automatically configures the PSC and pinmux for the DDR interface. It is not necessary to manually configure the PSC and pinmux for the DDR interface if the ROM function is called.
- Peripheral Configuration: Peripheral configuration settings applied during boot are not necessarily preserved when boot completes. If your application code uses the same peripheral that boots the device, your application should perform the normal procedure to initialize and configure that peripheral after boot. The exceptions to this rule are LPSC and pinmux settings, which are preserved after boot completes.

- Default clocks: For I2C and SPI master-boot modes and UART-boot modes, the bootloader applies default peripheral clocks settings as shown in Table 6.

Table 6. Default Clock Configurations for Various Boot Modes

Boot Mode	Input Clock	
	25 MHz	24 MHz
I2C0 Master	104.2 kHz	100.0 kHz
I2C1 Master	130.2 kHz	125.0 kHz
SPI Master	833 kHz	800 kHz
UART	-	115200 baud

9.2 UART-Boot Modes

- By default, UART boot operates at 115.2 kbps. The external UART device must be configured as follows: 115200 baud, 8 data bits, no parity, 1 stop bit, and no flow control.
- To obtain a standard baud rate of 115.2 kbps with default settings, the device input clock must be 24.000 MHz. Optionally, boot pins BOOT[7:5] may be used to enable alternative input clock sources and baud rates. You must specify the appropriate input clock speed, PLL0 multiplier, and UART baud rate in the AISgen tool to ensure that the UART peripheral operates at constant speed throughout the boot process.

Table 7. UART Baud Rate Selection Using Boot Pins

BOOT[7..5]	PLL0 Multiplier	UART Divider	UART Oversampling	Input Clock (MHz)	Resulting Baud Rate	Usable Standard Baud
000	x1	8	x13	24	115384	115200
000	x1	8	x13	12	57692	57600
001	x1	9	x13	27	115384	115200
001	x1	9	x13	13.5	57692	57600
010	x1	10	x13	30	115384	115200
010	x1	10	x13	15	57692	57600
011	x25	70	x13	16.8	115384	115200
100	x25	80	x13	19.2	115384	115200
101	x20	82	x13	24.576	115272	115200
101	x20	82	x13	12.288	57636	57600
110	x23	96	x13	25	115184	115200
111	x21	91	x13	26	115384	115200
111	x21	91	x13	13	57692	57600

NOTE: ROM revision d800k002 ignores these pins and always uses the 000 configuration. To check your ROM revision, follow the instructions in Section 1.

9.3 I2C-Boot Modes

- Default clocks: For I2C and SPI master-boot modes and UART-boot modes, the bootloader applies default peripheral clocks settings as shown in Table 6:
- All I2C Modes: The I2C bus must use 7-bit addressing.
- When booting from I2C0 in slave mode, the input clock source must be in the 21.0 MHz – 36.0 MHz range.
- The bootloader must be addressed with the I2C slave address 0x28.

- When booting from I2C1 in slave mode, the input clock source must be in the 16.8 MHz – 30.0 MHz range. In this boot mode, the bootloader initializes the PLL0 with the following parameters and takes it out of the default bypass mode. This is required to set the I2C1 module clock to a valid range.
- The bootloader must be addressed with the I2C slave address 0x29.
- The EEPROM must respond to the I2C slave address 0x50. The bootloader will look for an AIS image at offset 0x00000000.

Table 8. Default PLL Configuration in I2C1 Slave-Boot Mode

Div/Mul	Value	Clock	Freq Assuming 24 MHz Input	Freq Assuming 25 MHz Input
PREDIV	/1			
PLLM	*20			
POSTDIV	/2			
PLLDIV1	/1	SYSCLK1	240	250
PLLDIV2	/2	SYSCLK2	120	125
PLLDIV3	/4	SYSCLK3	60	62.5
PLLDIV4	/4	SYSCLK4	60	62.5
PLLDIV6	/1	SYSCLK6	240	250
PLLDIV7	/5	SYSCLK7	48	50

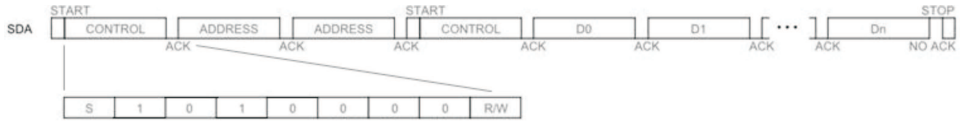


Figure 26. I2C SDA Signal Diagram for I2C EEPROM Boot (with sequential read enabled)

9.4 SPI-Boot Modes

- All SPI boot modes use the chip select 0 signal. The appropriate pin (SPI0_SCS[0] or SPI1_SCS[0]) must be connected to the external SPI device.
- The SPI EEPROM device must use 16-bit addressing, and its read command must equal 0x03. The bootloader will look for an AIS image at offset 0x00000000.
- The SPI flash device must use 24-bit addressing, and its read command must equal 0x03. The bootloader will look for an AIS image at offset 0x00000000.
- In the SPI-boot modes, the received data is sampled at the rising edge of the clock and the data to be transmitted on the falling edge of the clock as shown in Figure 27:

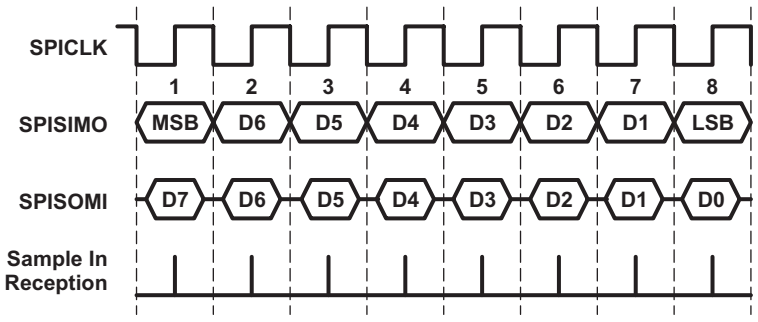


Figure 27. SPI Mode for Communication

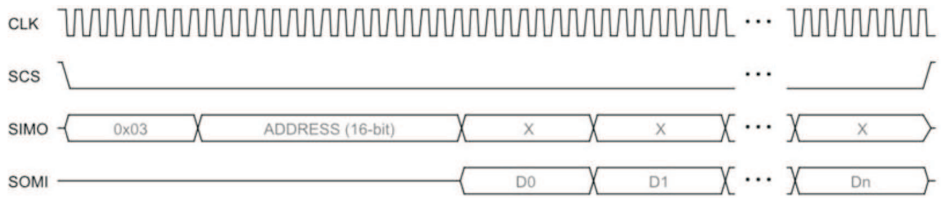


Figure 28. SPI Signal Diagram for SPI EEPROM Boot (with sequential read enabled)

9.5 NOR-Boot Modes

- For NOR legacy boot mode, the bootloader uses between 1 KB and 16 KB of Shared RAM starting from 0x80000000. The exact amount used depends on the NOR configuration word. For more details, see Section 3.1. This memory should not be used by any initialized section of the user application.
- See Section 6 for details.

9.6 NAND-Boot Modes

- The bootloader uses 8 KB of ARM local RAM starting from 0xFFFF0000. This memory should not be used by any initialized section of the user application.
- The bootloader normally begins with NAND block 1 and reads an entire block of NAND memory before any ROM functions, including PLL configuration, can be run. Optionally, boot pins BOOT[6:5] can be used to start at NAND block 0 and apply a standard PLL configuration before the initial NAND read operation. This PLL configuration is superseded by any configuration that is later applied using the PLL0 or PLL1 Configuration ROM functions.
- See Section 6 for details.

Table 9. NAND Configuration Selection Using Boot Pins

BOOT[6:5]	Starting NAND Block	PLL0 Configuration Applied
00	1	None
01	1	PLLM = 24; POSTDIV = 3; SYSCLK3 = 2
10	0	None
11	0	PLLM = 24; POSTDIV = 3; SYSCLK3 = 2

NOTE: ROM revisions d800k002, d800k004, and d800k006 ignore these pins and always use the 00 configuration. To check your ROM revision, follow the instructions in Section 1.

9.7 MMC/SD-Boot Modes

- The bootloader determines which type of memory card to read (MMC or SD) based on boot pin BOOT[5].

Table 10. MMC/SD Memory Card Selection Using Boot Pins

BOOT[5]	Memory Card Type
0	SD or MMC
1	MMC only

- See Section 6 for details.

9.8 HPI-Boot Modes

- The bootloader uses the first 4 B of Shared RAM starting at 0x80000000. This memory should not be used by any initialized section of the user application.

10 References

- *TMS320C674x/OMAP-L1x Processor External Memory Interface A (EMIFA) User's Guide* (SPRUFL6)

Boot Mode Selection Table

A.1 Boot Mode Selection Table

Table 11 lists various boot modes supported by the bootloader and a configuration of boot pins required to select a boot mode. The boot pins are latched by the bootloader when the device exits reset (on the rising edge of reset).

Table 11. Boot Mode Selection

BOOT[7:0] (1)	Boot Mode	AIS
0000 0010	NOR	Yes (2)
0xx0 1110 (3)	NAND 8	Yes
0xx1 0000 (3)(4)	NAND 16	Yes
00x1 1100 (5)(6)	MMC/SD0	Yes
0000 0000	I2C0 EEPROM	Yes
0000 0110	I2C1 EEPROM	Yes
0000 0001	I2C0 Slave	Yes
0000 0111	I2C1 Slave	Yes
0000 1000	SPI0 EEPROM	Yes
0000 1001	SPI1 EEPROM	Yes
0000 1010	SPI0 Flash	Yes
0000 1100	SPI1 Flash	Yes
0001 0010	SPI0 Slave	Yes
0001 0011	SPI1 Slave	Yes
xxx1 0110 (7)	UART0	Yes
xxx1 0111 (7)	UART1	Yes
xxx1 0100 (7)	UART2	Yes
0000 0100	HPI	No
0001 1110	Emulation Debug	No

(1) The boot pins, BOOT[7:0], are multiplexed with the following signals: VPIF_DOUT[15:8], LCD_DATA[15:8], UPP_XDATA[7:0], and GPIO_7[7:0].
(2) There are three methods to boot from NOR flash; only one of them uses AIS.
(3) In NAND boot mode, BOOT[6:5] apply additional configuration at the start of the boot process. For BOOT[6:5] = 00, NAND boot begins in block 1 and does not apply any initial PLL configuration before the first block read. For more information, see Section 9.
(4) NAND 16 boot mode is not supported for ROM revision d800k002. To check your ROM revision, follow the instructions in Section 1.
(5) In MMC/SD boot mode, BOOT[5] determines which type of memory card is used: MMC or SD. For more information, see Section 9.
(6) MMC/SD boot mode is only supported by ROM revision d800k008. To check your ROM revision, follow the instructions in Section 1.
(7) In UART boot mode, BOOT[7:5] configure the UART peripheral to run at standard baud rates for different input clocks. For BOOT[7:5] = 000, the UART peripheral runs at 115.2 kbps with a 24 MHz input clock. For more information, see Section 9.

Details of Supported NAND Devices

B.1 Details of Supported NAND Devices

The OMAP-L1x8 bootloader supports NAND devices that comply with the ONFI standard. If a NAND device is ONFI-compliant, the bootloader reads device information from the NAND *parameters page*.

If the device is not ONFI-compliant or if the bootloader fails to read valid ONFI parameters (with correct CRC), the bootloader reads the NAND device ID and attempts to use Table 12 to determine the device parameters. If the table reports that the NAND is larger than 128 MB, or if the device ID is not found in the table, the bootloader attempts to read parameters from the NAND device's fourth ID byte. Otherwise, the table parameters are used. The bootloader expects the contents of this ID byte to match Table 13. Fields marked *unused* are not checked by the bootloader.

Table 12. Parameters for Supported NAND Devices

Device ID	Number of Blocks	Pages Per Block	Bytes Per Page	Size	Interface
0x33	1024	32	512+16	16 MB	8 bit
0x35	2048	32	512+16	32 MB	8 bit
0x36	4096	32	512+16	64 MB	8 bit
0x39	1024	16	512+16	8 MB	8 bit
0x43	1024	32	512+16	16 MB	16 bit
0x45	2048	32	512+16	32 MB	16 bit
0x46	4096	32	512+16	64 MB	16 bit
0x49	1024	16	512+16	8 MB	16 bit
0x53	1024	32	512+16	16 MB	16 bit
0x55	2048	32	512+16	32 MB	16 bit
0x56	4096	32	512+16	64 MB	16 bit
0x59	1024	16	512+16	8 MB	16 bit
0x6B	1024	16	512+16	8 MB	8 bit
0x71	16384	32	512+16	256 MB	8 bit
0x72	8192	32	512+16	128 MB	16 bit
0x73	1024	32	512+16	16 MB	8 bit
0x74	8192	32	512+16	128 MB	16 bit
0x75	2048	32	512+16	32 MB	8 bit
0x76	4096	32	512+16	64 MB	8 bit
0x78	8192	32	512+16	128 MB	8 bit
0x79	8192	32	512+16	128 MB	8 bit
0xA1	1024	64	2048+64	128 MB	8 bit
0xB1	1024	64	2048+64	128 MB	16 bit
0xC1	1024	64	2048+64	128 MB	16 bit
0xE3	512	16	512+16	4 MB	8 bit
0xE5	512	16	512+16	4 MB	8 bit
0xE6	1024	16	512+16	8 MB	8 bit
0xF1	1024	64	2048+64	128 MB	8 bit
0xF5	2048	32	512+16	32 MB	8 bit

Table 13. Expected Contents of Fourth ID Byte for NAND Devices Listed in Table 12 With Sizes Greater Than 128 MB

Bit	Field	Value	Description
7	-	-	Unused
6	BUS	0	Data bus width (8-bit)
5:4	BLOCK		Block size (without spare bytes)
		0x0	64 KB
		0x1	128 KB
		0x2	256 KB
		0x3	512 KB
3	-	-	Unused
2	SPARE	1	Spare area size (16 B)
1:0	PAGE		Page size (without spare bytes)
		0x0	1 KB
		0x1	2 KB
		0x2	4 KB
		0x3	8 KB (not supported)

Table 14 provides a list of verified-supported NAND flash devices.

Table 14. Supported NAND Devices

Device	Manufacturer
MT29F4G08AACWC:C	Micron
MT29F4G08ABADAWP:D	Micron
MT29F2G08AAC	Micron
NAND01GR3B2CZ	STMicroelectronics
MT29F8G08ADBDAH4	Micron
MT29F4G16ABADAH4-IT:D	Micron

CRC Computation Algorithm

C.1 CRC Computation Algorithm

The following code demonstrates calculation of the CRC as performed by the AIS generation script and the bootloader. This code should be used as a reference to implement CRC calculation on other platforms.

```
static Uint32 LOCAL_updateCRC (Uint8 *data_ptr, Uint32
     section_size, Uint32 crc)
{
   Uint32 i;

   // Prepare input to get back into calculation state (this means
   // initial input when starting fresh should be 0x00000000 )
   crc = crc ^ 0xFFFFFFFF;

   // Perform the algorithm on each byte
   for (i = 0; i < section_size; i++)
   {
     crc = (crc >> 8) ^ CRC_Lut[(crc & 0xFF) ^ data_ptr[i]];
   }

   // Exclusive OR the result with the specified value
   crc = (crc ^ 0xFFFFFFFF);

   return crc;
}
```

This algorithm takes a lookup table (LUT) approach. The table consists of 256 32-bit values, which are given in Table 15.

Table 15. Lookup Table for CRC Algorithm

Index	Word	Index	Word	Index	Word	Index	Word
1	0x00000000	65	0x76DC4190	129	0xEDB88320	193	0x9B64C2B0
2	0x77073096	66	0x01DB7106	130	0x9ABFB3B6	194	0xEC63F226
3	0xEE0E612C	67	0x98D220BC	131	0x03B6E20C	195	0x756AA39C
4	0x990951BA	68	0xEFD5102A	132	0x74B1D29A	196	0x026D930A
5	0x076DC419	69	0x71B18589	133	0xEAD54739	197	0x9C0906A9
6	0x706AF48F	70	0x06B6B51F	134	0x9DD277AF	198	0xEB0E363F
7	0xE963A535	71	0x9FBFE4A5	135	0x04DB2615	199	0x72076785
8	0x9E6495A3	72	0xE8B8D433	136	0x73DC1683	200	0x05005713
9	0x0EDB8832	73	0x7807C9A2	137	0xE3630B12	201	0x95BF4A82
10	0x79DCB8A4	74	0x0F00F934	138	0x94643B84	202	0xE2B87A14
11	0xE0D5E91E	75	0x9609A88E	139	0x0D6D6A3E	203	0x7BB12BAE
12	0x97D2D988	76	0xE10E9818	140	0x7A6A5AA8	204	0x0CB61B38
13	0x09B64C2B	77	0x7F6A0DBB	141	0xE40ECF0B	205	0x92D28E9B
14	0x7EB17CBD	78	0x086D3D2D	142	0x9309FF9D	206	0xE5D5BE0D
15	0xE7B82D07	79	0x91646C97	143	0x0A00AE27	207	0x7CDCEFB7
16	0x90BF1D91	80	0xE6635C01	144	0x7D079EB1	208	0x0BDBDF21

Table 15. Lookup Table for CRC Algorithm (continued)

Index	Word	Index	Word	Index	Word	Index	Word
17	0x1DB71064	81	0x6B6B51F4	145	0xF00F9344	209	0x86D3D2D4
18	0x6AB020F2	82	0x1C6C6162	146	0x8708A3D2	210	0xF1D4E242
19	0xF3B97148	83	0x856530D8	147	0x1E01F268	211	0x68DDB3F8
20	0x84BE41DE	84	0xF262004E	148	0x6906C2FE	212	0x1FDA836E
21	0x1ADAD47D	85	0x6C0695ED	149	0xF762575D	213	0x81BE16CD
22	0x6DDDE4EB	86	0x1B01A57B	150	0x806567CB	214	0xF6B9265B
23	0xF4D4B551	87	0x8208F4C1	151	0x196C3671	215	0x6FB077E1
24	0x83D385C7	88	0xF50FC457	152	0x6E6B06E7	216	0x18B74777
25	0x136C9856	89	0x65B0D9C6	153	0xFED41B76	217	0x88085AE6
26	0x646BA8C0	90	0x12B7E950	154	0x89D32BE0	218	0xFF0F6A70
27	0xFD62F97A	91	0x8BBEB8EA	155	0x10DA7A5A	219	0x66063BCA
28	0x8A65C9EC	92	0xFCB9887C	156	0x67DD4ACC	220	0x11010B5C
29	0x14015C4F	93	0x62DD1DDF	157	0xF9B9DF6F	221	0x8F659EFF
30	0x63066CD9	94	0x15DA2D49	158	0x8EBEEFF9	222	0xF862AE69
31	0xFA0F3D63	95	0x8CD37CF3	159	0x17B7BE43	223	0x616BFFD3
32	0x8D080DF5	96	0xFBD44C65	160	0x60B08ED5	224	0x166CCF45
33	0x3B6E20C8	97	0x4DB26158	161	0xD6D6A3E8	225	0xA00AE278
34	0x4C69105E	98	0x3AB551CE	162	0xA1D1937E	226	0xD70DD2EE
35	0xD56041E4	99	0xA3BC0074	163	0x38D8C2C4	227	0x4E048354
36	0xA2677172	100	0xD4BB30E2	164	0x4FDFF252	228	0x3903B3C2
37	0x3C03E4D1	101	0x4ADFA541	165	0xD1BB67F1	229	0xA7672661
38	0x4B04D447	102	0x3DD895D7	166	0xA6BC5767	230	0xD06016F7
39	0xD20D85FD	103	0xA4D1C46D	167	0x3FB506DD	231	0x4969474D
40	0xA50AB56B	104	0xD3D6F4FB	168	0x48B2364B	232	0x3E6E77DB
41	0x35B5A8FA	105	0x4369E96A	169	0xD80D2BDA	233	0xAED16A4A
42	0x42B2986C	106	0x346ED9FC	170	0xAF0A1B4C	234	0xD9D65ADC
43	0xDBBBC9D6	107	0xAD678846	171	0x36034AF6	235	0x40DF0B66
44	0xACBCF940	108	0xDA60B8D0	172	0x41047A60	236	0x37D83BF0
45	0x32D86CE3	109	0x44042D73	173	0xDF60EFC3	237	0xA9BCAE53
46	0x45DF5C75	110	0x33031DE5	174	0xA867DF55	238	0xDEBB9EC5
47	0xDCD60DCF	111	0xAA0A4C5F	175	0x316E8EEF	239	0x47B2CF7F
48	0xABD13D59	112	0xDD0D7CC9	176	0x4669BE79	240	0x30B5FFE9
49	0x26D930AC	113	0x5005713C	177	0xCB61B38C	241	0xBDBDF21C
50	0x51DE003A	114	0x270241AA	178	0xBC66831A	242	0xCABAC28A
51	0xC8D75180	115	0xBE0B1010	179	0x256FD2A0	243	0x53B39330
52	0xBFD06116	116	0xC90C2086	180	0x5268E236	244	0x24B4A3A6
53	0x21B4F4B5	117	0x5768B525	181	0xCC0C7795	245	0xBAD03605
54	0x56B3C423	118	0x206F85B3	182	0xBB0B4703	246	0xCDD70693
55	0xCFBA9599	119	0xB966D409	183	0x220216B9	247	0x54DE5729
56	0xB8BDA50F	120	0xCE61E49F	184	0x5505262F	248	0x23D967BF
57	0x2802B89E	121	0x5EDEF90E	185	0xC5BA3BBE	249	0xB3667A2E
58	0x5F058808	122	0x29D9C998	186	0xB2BD0B28	250	0xC4614AB8
59	0xC60CD9B2	123	0xB0D09822	187	0x2BB45A92	251	0x5D681B02
60	0xB10BE924	124	0xC7D7A8B4	188	0x5CB36A04	252	0x2A6F2B94
61	0x2F6F7C87	125	0x59B33D17	189	0xC2D7FFA7	253	0xB40BBE37
62	0x58684C11	126	0x2EB40D81	190	0xB5D0CF31	254	0xC30C8EA1
63	0xC1611DAB	127	0xB7BD5C3B	191	0x2CD99E8B	255	0x5A05DF1B

Table 15. Lookup Table for CRC Algorithm (continued)

Index	Word	Index	Word	Index	Word	Index	Word
64	0xB6662D3D	128	0xC0BA6CAD	192	0x5BDEAE1D	256	0x2D02EF8D

Details of Pre-Defined ROM Functions

The OMAP-L132/L138 bootloader can call several ROM functions using the AIS Function Execute command. This appendix describes the available ROM functions and the arguments required to call them.

Table 16. List of Pre-Defined ROM Functions

Index	Function	Section
0	PLL0 Configuration	Section D.1
1	PLL1 Configuration	Section D.2
2	Clock Configuration	Section D.3
3	mDDR/DDR2 Controller Configuration	Section D.4
4	EMIFA SDRAM Configuration	Section D.5
5	EMIFA ASYNC Configuration	Section D.6
6	PLL and Clock Configuration	Section D.7
7	Power and Sleep Controller Configuration	Section D.8
8	Pinmux Configuration	Section D.9

D.1 PLL0 Configuration (Index = 0, Argument Count = 2)

The PLL0 Configuration function configures the PLL0 registers. This function takes two arguments, as shown below.

The PLL0 configuration register is shown in Figure 29 and described in Table 17.

Figure 29. PLL Configuration Register

	31 24	23 16	15 8	7 0
Arg1	CLKMODE	PLLM	PREDIV	POSTDIV
Arg2	Reserved	PLLDIV1	PLLDIV3	PLLDIV7

Table 17. PLL Configuration Register Field Descriptions

	Bit	Field	Value	Description
Arg1	31-24	CLKMODE		Value to be programmed to the PLL clock source.
			0	Crystal
			1	Oscillator
	23-16	PLLM		Value to be programmed to the PLL multiplier register.
	15-8	PLLDIV		Value to be programmed to the PLL PREDIV register, used to divide the input clock before the PLL multiplier.
	7-0	POSTDIV		Value to be programmed to the PLL POSTDIV register, used to divide the output of the the PLL multiplier.
Arg2	31-24	Reserved	0	Reserved
	23-0	PLLDIV1 PLLDIV3 PLLDIV7		Values to be programmed to the PLLDIV1, PLLDIV3 and PLLDIV7 registers, used to generate SYSCLK1, SYSCLK2, SYSCLK4, SYSCLK6, SYSCLK3, and SYSCLK7. SYSCLK6 = SYSCLK1, SYSCLK2 = SYSCLK1/2, SYSCLK4 = SYSCLK1/4

D.2 PLL1 Configuration (Index = 1, Argument Count = 2)

The PLL1 Configuration function configures the PLL1 registers. This function takes two arguments, as shown below.

The PLL1 configuration register is shown in Figure 30 and described in Table 18.

Figure 30. PLL1 Configuration Register

	31 24	23 16	15 8	7 0
Arg1	PLLM	POSTDIV	PLLDIV1	PLLDIV2
Arg2	Reserved			PLLDIV3

Table 18. PLL Configuration Register Field Descriptions

	Bit	Field	Value	Description
Arg1	31-24	PLLM		Value to be programmed to the PLL Multiplier register.
	23-18	POSTDIV		Value to be programmed to PLL POSTDIV register, used to divide the output of the PLL multiplier.
	15-8	PLLDIV1		Values to be programmed to the PLLDIV1, PLLDIV2, and PLLDIV3 registers, used to generate SYSCLK1, SYSCLK2, and SYSCLK3.
	7-0	PLLDIV2		Values to be programmed to the PLLDIV1, PLLDIV2, and PLLDIV3 registers, used to generate SYSCLK1, SYSCLK2, and SYSCLK3.
Arg2	31-8	Reserved	0	Reserved
	7-0	PLLDIV3		Values to be programmed to the PLLDIV1, PLLDIV2, and PLLDIV3 registers, used to generate SYSCLK1, SYSCLK2, and SYSCLK3.

D.3 Clock Configuration (Index = 2, Argument Count = 1)

The clock configuration function configures the clocks for the active boot peripheral. It programs the peripheral clocks in SPI/I2C master boot modes, UART boot mode, or MMC/SD boot mode. In all other boot modes, this function has no effect. This function takes only one argument, but the contents of that argument vary depending on boot mode.

D.3.1 SPI Master Register

The SPI master register is shown in Figure 31 and described in Table 19.

Figure 31. SPI Master Register

	31 8	7 0
Arg1	Reserved	PRESCALE

Table 19. SPI Master Register Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7-0	PRESCALE		Value to be programmed to the PRESCALE field of the SPIFMT register

D.3.2 I2C Master Register

The I2C master register is shown in Figure 32 and described in Table 20.

Figure 32. I2C Master Register

31	24	23	16	15	8	7	0
Arg1	Reserved		IPSC		ICCL		ICCH

Table 20. I2C Master Register Field Descriptions

Bit	Field	Value	Description
31-24	Reserved	0	Reserved
23-16	IPSC		Value to be programmed to I2C ICPSC register
15-8	ICCL		Value to be programmed to I2C ICCLKL register
7-0	ICCH		Value to be programmed to I2C ICCLKH register

D.3.3 UART Slave Register

The UART master register is shown in Figure 33 and described in Table 21.

Figure 33. I2C Master Register

31	24	23	16	15	8	7	0
Arg1	Reserved		OSR		DLH		DLL

Table 21. I2C Master Register Field Descriptions

Bit	Field	Value	Description
31-24	Reserved	0	Reserved
23-16	OSR		Value to be programmed to OSR field of UART MDR register
15-8	DLH		Value to be programmed to UART DLH register
7-0	DLL		Value to be programmed to UART DLL register

D.3.4 MMC/SD Register

The MMC/SD register is shown in Figure 34 and described in Table 22.

Figure 34. MMC/SD Register

31	16	15	8	7	0
Arg1	Reserved		DIV4		CLKRT

Table 22. MMC/SD Register Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-8	DIV4		Value to be programmed to DIV4 field of MMCCLK register
7-0	CLKRT		Value to be programmed to CLKRT field of MMCCLK register

D.4 mDDR/DDR2 Controller Configuration (Index = 3, Argument Count = 8)

The mDDR/DDR2 controller configuration function configures the mDDR/DDR2 registers responsible for DDR timing and configuration. Since the mDDR/DDR2 controller setup requires that the PLL1 is configured, the PLL1 configuration function is called before the mDDR/DDR2 controller itself is initialized. Therefore, the first two arguments are the same as required for the PLL1 configuration function.

The next five arguments are required timing parameters for the mDDR/DDR2 Controller registers. They are written directly to mDDR/DDR2 controller registers with the same names.

The last argument contains three fields: PASR, ROWSIZE, and CLK2XSRC. The first two fields are copied to fields with the same names in the SDCR2 register if and only if MSDRAMEN of SDCR (bit 25 of Arg4) = 1. In other words, these fields only apply when using mDDR, not DDR2. The third field applies in all cases and allows selection of the clock source for the mDDR/DDR2 controller. A value of zero uses the normal clock source, while a value of one selects an un-divided clock that is typically twice as fast. (More precisely, it ignores the PLL1 post divider.)

	31 19	18 16	15 11	10 8	7 0
Arg1	PLL1 Configuration Arg1				
Arg2	PLL1 Configuration Arg2				
Arg3	DRPYC1R				
Arg4	SDCR				
Arg5	SDTIMR1				
Arg6	SDTIMR2				
Arg7	SDRCR				
Arg8	Reserved	PASR	Reserved	ROWSIZE	CLK2XSRC

The mDDR/DDR2 Controller Configuration function wakes up the mDDR/DDR2 peripheral from its default reset state and correctly applies the register configurations as required.

NOTE: For devices with ROM revision d800k002, this function supports only DDR2, not mDDR. When creating AIS files for this version of the bootloader, the AISgen tool uses a software patch to configure mDDR as required. Subsequent ROM revisions (e.g., d800k004 and later) apply mDDR configuration using the up-to-date ROM function. For this reason, it's important to specify the correct ROM ID in the AISgen tool when configuring mDDR/DDR2.

AIS files generated for the d800k002 ROM revision will still work with later revisions of the ROM, but AIS files created for later revisions may not work with d800k002 devices.

D.5 EMIFA SDRAM Configuration (Index = 4, Argument Count = 5)

The EMIFA SDRAM configuration function configures the EMIFA registers responsible for SDRAM timing and configuration.

	31 0
Arg1	SDCR
Arg2	SDTIMR
Arg3	SDSRETR
Arg4	SDRCR
Arg5	DIV4p5_CLK_EN

This first four function arguments are written directly to the EMIFA registers with the same names.

The DIV4p5_CLK_EN is a Boolean value to enable the use of the 4.5 divider of the PLL0 multiplier output as the input clock to the EMIFA peripheral.

Before programming EMIFA registers, this function applies the necessary PINMUX for 16- or 32-bit SDRAM access (based on the value of the SDCR register) and wakes up the EMIFA peripheral from its default reset state.

D.6 EMIFA Async Configuration (Index = 5, Argument Count = 5)

The EMIFA CE Space Configuration function configures the EMIFA CExCFG registers. This function takes five arguments and writes them to the EMIFA registers with the same names.

	31 … 0
Arg1	CE2CFG
Arg2	CE3CFG
Arg3	CE4CFG
Arg4	CE5CFG
Arg5	NANDFCR

Before programming the EMIFA registers, this function wakes up the EMIFA peripheral from its default reset state.

D.7 PLL and Clock Configuration (Index = 6, Argument Count = 3)

The PLL and clock configuration function combines the PLL configuration and clock configuration functions into a single function call. This function takes three arguments. The first two arguments match the arguments for the PLL configuration function (Section D.1), and the third argument matches the argument for the clock configuration function (Section D.3).

	31 … 0
Arg1	PLL0 Configuration Arg1
Arg2	PLL0 Configuration Arg2
Arg3	Clock Configuration Arg1

If the device is booting from the boot modes listed in the clock configuration description, it is required to call this function rather than calling the PLL0 Configuration and Clock Configuration functions separately. Failure to do so may result in the boot failing since the peripheral functionality is tied to the PLL0 settings, and setting the PLL0 first may cause the peripheral to operate at an unintended frequency.

D.8 Power and Sleep Configuration (PSC) (Index = 7, Argument Count = 1)

The power and sleep configuration (PSC) function can be used multiple times to set the power domains for various LPSC modules of the system's two PSCs; 0x1-0x3 are the only valid values (all others are reserved).

	31 … 24	23 … 16	15 … 8	7 … 0
Arg1	PSCNUM	MODULE	PD	STATE

For more details on the power domains and the Power and Sleep Controller, see the device-specific datasheet or system guide.

Table 23. Power and Sleep Configuration (PSC) Register Field Descriptions

Bit	Field	Value	Description
31-24	PSCNUM		Selected PSC
		0	PSC0
		1	PSC1
23-16	MODULE		The module for which the LPSC state is being changed.
15-8	PD		The power domain to which the module belongs (typically 0).
7-0	STATE		The state to which transition

D.9 Pinmux Configuration (Index = 8, Argument Count = 3)

The pinmux configuration function can be used multiple times to set the state of the 20 system pinmux registers during the boot process.

	31	0
Arg1	REGNUM	
Arg2	MASK	
Arg3	VALUE	

Table 24. Pinmux Configuration Register Field Descriptions

	Bit	Field	Value	Description
Arg1	31-0	REGNUM		Pinmux register number (0-19)
Arg2	31-0	MASK		Register mask to select which bits of the register are modified
Arg3	31-0	VALUE		The value (filtered by the mask) to apply to the register

The function applies to the pinmux register value as follows:

```
Pinmux[REGNUM] = (Pinmux[REGNUM] & ~MASK) |
          (MASK & VALUE)
```

ROM Revision History

E.1 ROMID: D800K002, Silicon Revision 1.0

Initial ROM boot loader revision.

E.2 ROMID: D800K004, Silicon Revision 1.1

- Added 16-bit raw NAND boot modes
- Added check for SPI TXINTFLAG when doing SPI master boot modes
- Added new default UART and PLL clock settings for UART boot modes (based on BOOT[7..5] pins)
- Removed the locking of the KICK registers that happened at the end of boot
- Updated pinmux control Function Execute to mask and update with one write to the PINMUX register (instead of two)
- Added support for configuring mDDR (in addition to DDR2) in the DDR Configure Function Execute (added ability to specify contents of SDCR2 register)
- In NAND ECC check function, added a dummy read to clear the ECC data registers when the ECC failed
- Changed NAND wait-for-ready logic to check if R/Bn line of the NAND goes low before going high (so we don't think it is ready before it actually is)

E.3 ROMID: D800K006, Silicon Revision 2.0

No substantive changes to the ROM boot loader. Note that the KICK registers were disabled in hardware with this silicon revision.

E.4 ROMID: D800K008, Silicon Revision 2.1

- Fix potential boot hang (corrects Silicon Advisory 2.0.20)
- All peripheral read functions updated to read in chunks of required size, rather than one 32-bit word at a time (boot time optimization)
- Added MMC/SD0 boot mode (boot from MMC cards, SD card, and eMMC devices)
- Updated NAND boot mode to offer boot from block 0 or block 1, and to select default PLL settings for faster boot
- Update NAND boot modes so that page data is read from the EMIF as 32-bit words instead of 8-bit bytes (boot time optimization)
- Update to NAND boot modes so that if no valid boot image is found at in current block, next block is tried until image is found (within first 32 blocks).
- Adjustment to NAND timeout value (improves boot time in cases where the ROM misses the NAND R/Bn low to high transition)
- Make boot abort function callable via pointer (makes the ROM abort functionality patchable)
- Update DDR Configure Function Execute to skip VTP calibration if it is already done

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from E Revision (January 2014) to F Revision	Page
• Updates were made in Section B.1	32

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