Application Report
SPRABA4C– June 2012

Using the AM17xx Bootloader

Urmil Parikh and Joseph Coombs

ABSTRACT

This application report describes various boot mechanisms supported by the AM17xx 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/SPRABA4.

Contents

List of Figures

List of Tables

1 Introduction

The AM17xx 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 and assumptions for the bootloader.

The AM17xx 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 d800k005 should appear in the memory window at offset 0x08. For earlier ROM revisions, the text could also appear as d800k003, d800k001. It's important to know your ROM revision when generating boot images. If you don't see either of these values in the memory window, this document is not applicable to your device.

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

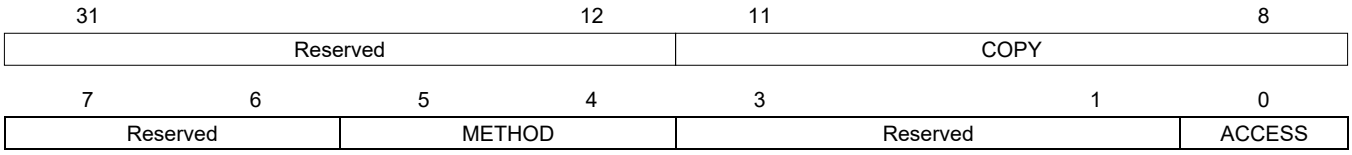


Table 1. NOR Boot Configuration Word Field Descriptions

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 On-chip 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

Table 1. NOR Boot Configuration Word Field Descriptions (continued)

Bit	Field	Value	Description
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.

With a total of 15 address lines (EMA_A[12:0] and EMA_BA[1:0]) of EMIFA, only 32 KB of NOR Flash can be accessed by the bootloader. If boot image is longer than this, a secondary bootloader is required that can access more data from NOR Flash by managing higher address lines of NOR Flash using general-purpose input/output (GPIO) pins or some other mechanism.

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 On-chip RAM (0x80000000). This block of data should hold a secondary bootloader, as shown in Figure 2.

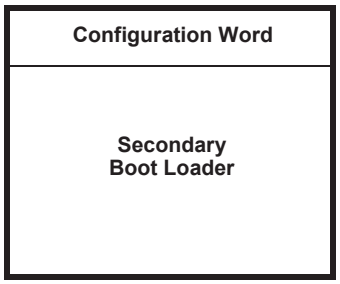


Figure 2. Structure of Secondary Bootloader for NOR Boot

After copying the required data to On-chip 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.

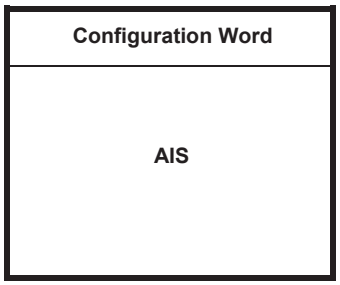


Figure 3. Placement of AIS for NOR Boot

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

4.1 Host Port Interface (HPI) Boot

HPI boot happens from the HPI0 peripheral in 16-bit mode. The boot 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 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.

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 AM17xx 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.

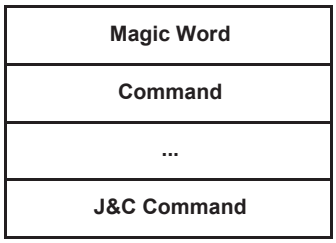


Figure 4. Structure of AIS

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

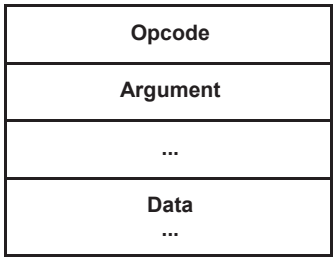


Figure 5. Structure of an AIS Command

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.

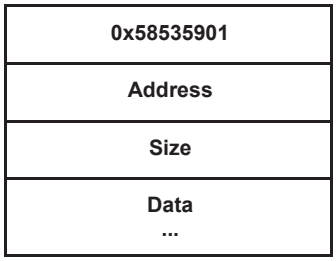


Figure 6. Section Load Command

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.

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).

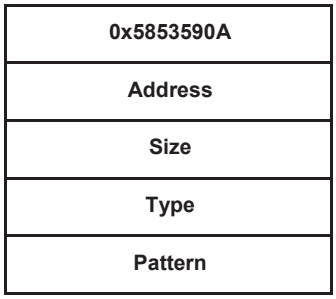


Figure 7. Section Fill Command

This command takes four arguments: address and size of the section to be filled, the type of memory access (8, 16, and 32-bit), and the pattern that will fill the memory.

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 and Section Fill commands.

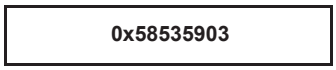


Figure 8. Enable CRC Command

This command does not take any arguments or data.

4.4 Disable CRC Command (0x58535904)

This command disables the calculation of the CRC.

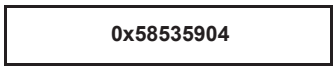


Figure 9. Disable CRC Command

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.

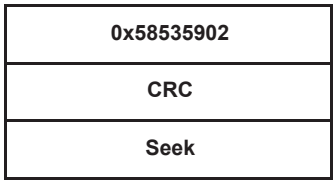


Figure 10. Validate CRC Command

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 and Section Fill command so that the command can be executed again.

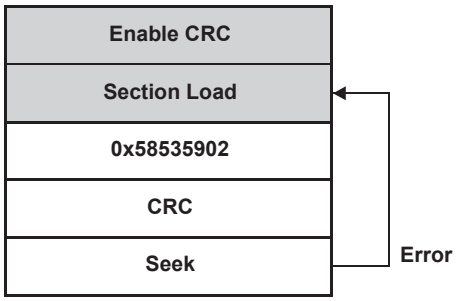


Figure 11. Handling CRC Error

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.

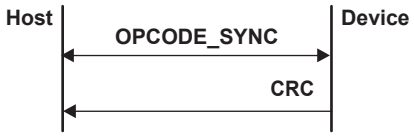


Figure 12. Validate CRC Flow for Slave Mode

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.

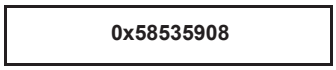


Figure 13. Start-Over Command

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.

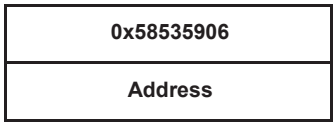


Figure 14. Jump and Close Command

This command takes one argument: the entry point of the DSP 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.

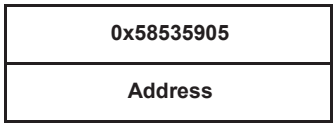


Figure 15. Jump Command

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 or 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.

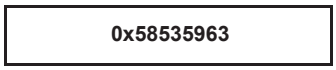


Figure 16. Sequential Read Enable Command

This command takes no arguments or data.

4.9 Compressed Section Load Command (0x58535909)

When compression of the user-application data is enabled, the Compressed Section Load commands are used in AIS instead of the Section Load commands. Compressed sections are split into multiple Compressed Section Load commands to limit the size of temporary buffers required by the decompression algorithm.

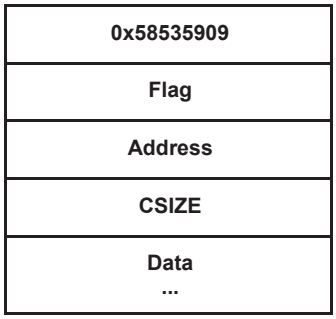


Figure 17. Compressed Section Load Command

This command takes three arguments. The first argument is a flag that indicates whether the section is new or a continuation of the previous section. The second argument is the address of the section; this argument is optional and is present only if the section is new. The third argument is the size of the compressed data present in this command.

4.10 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.

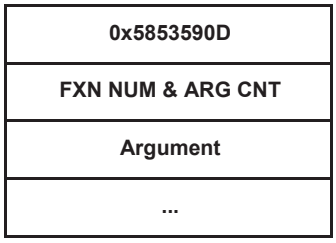


Figure 18. Function Execute Command

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.

4.11 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.

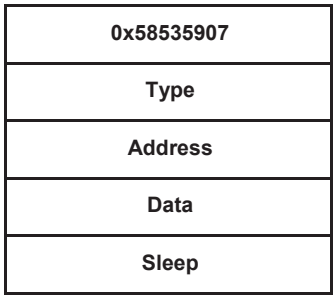


Figure 19. Boot Table Command

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 shown in Figure 20 and described in the Table 2. 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 20. 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 2. 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-FFh	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 Version 2.0 for its operation.

5.1 Installation

Before starting, please 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 (http://www.microsoft.com).

To install AISgen, download and execute the latest installer from the following URL: http://www-s.ti.com/sc/techlit/sprab04.zip.

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

5.2 Getting Started

On successful installation, a link to start AISgen will be created in the following folder: Start Menu → Program Files → Texas Instruments → AISgen for D800K005.

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

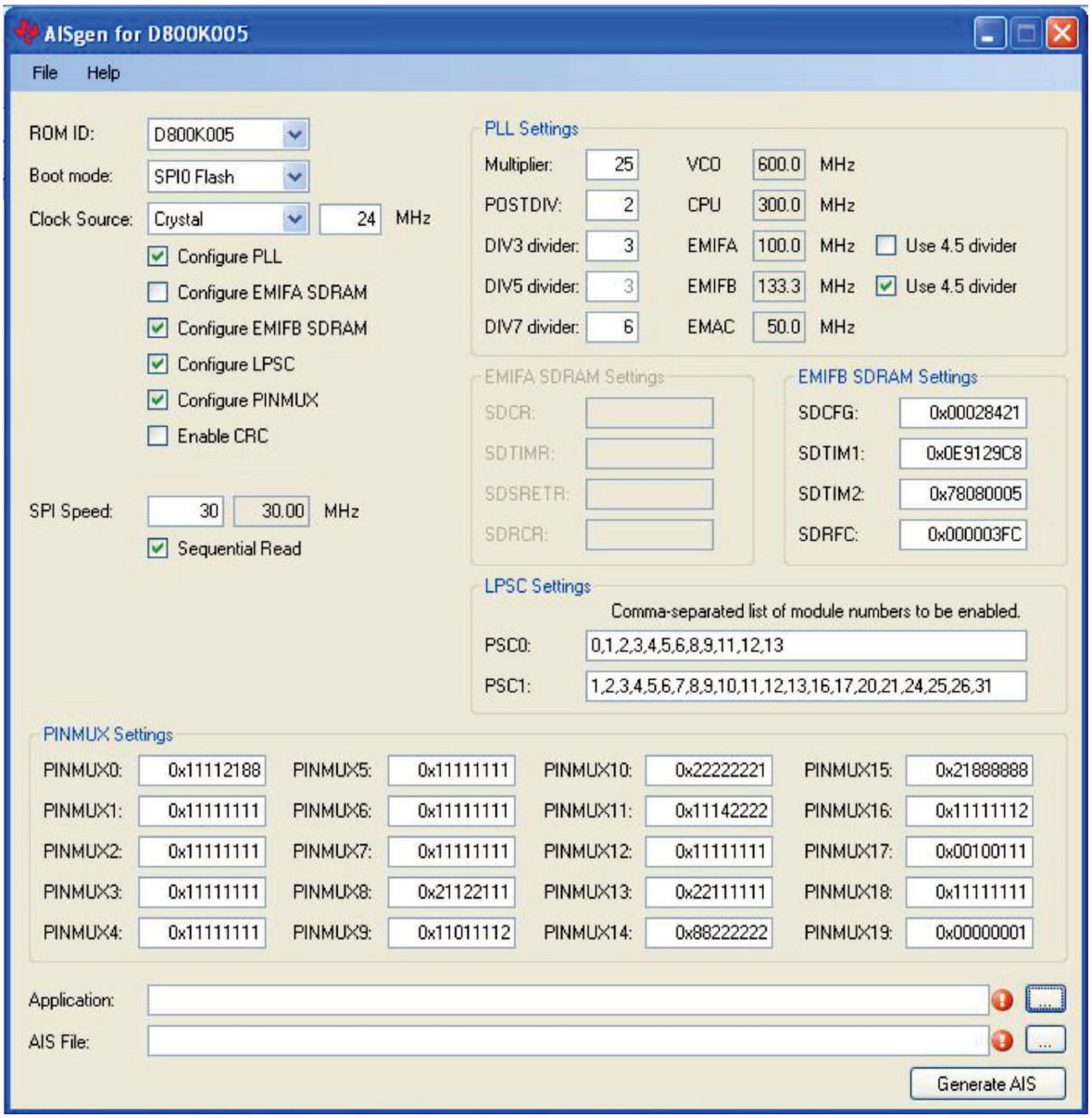


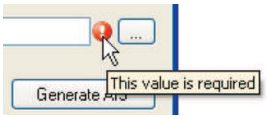
Figure 21. AISgen Main Window

You may notice that many controls on this window are initially disabled. A control may be disabled for multiple reasons:

- You may not have selected an associated option. For example, when the *Configure PLL* checkbox is not selected, the controls inside the *PLL Settings* box will be disabled.
- The control displays a calculated result and cannot be set directly. For example, CPU frequency is a calculated result.

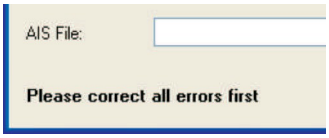
To get started quickly, select File → Configure for the evaluation module (EVM). This sets all fields to the default configuration shown in Figure 21.

You may also notice an icon next to one or more controls (such as *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 the AIS file is required to generate AIS.

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



For example, if you click on the *Generate AIS* button when one or more error indicators are visible, a message saying *Please correct all errors first* will appear.

When an AIS file is successfully generated, this area will display the file's size.

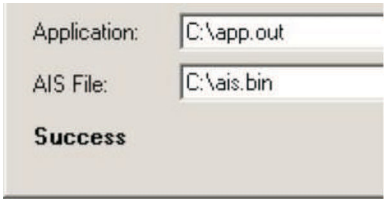
5.3 Generating AIS

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

Select File → Configure for the EVM to apply default settings to the PLL and EMIF controls. 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. You will notice that the error indicator vanishes as the text is entered in the AIS file specification box.

Specify a 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 AIS (as `C:\ais.bin`). The size of this file depends on the ARM application.

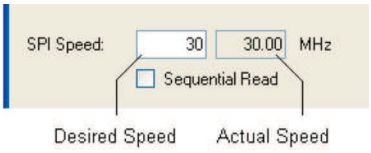


Now you are ready to examine some of the options in greater detail.

5.3.1 Boot Mode and Boot Peripheral Setup

Next, you have to specify the boot mode that you plan to use. This information is used by AISgen to control a few parameters of the AIS generation and also to optimize boot time (in master mode) by letting you specify the maximum clock speed of the boot peripheral, where applicable.

For master-boot modes (booting from EEPROM or Flash), the bootloader configures the peripherals to standard speeds by default for broader compatibility. These are documented in Table 4. If the boot peripheral being used supports faster speeds, you can specify a speed so that the bootloader can re-configure clocks and boot faster.

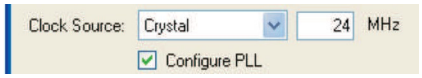


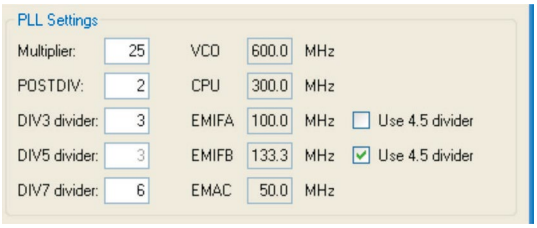
For I2C- and SPI-master modes, you can enter the desired speed. Depending on the PLL settings and granularity supported by the I2C and SPI peripherals, AISgen calculates the actual speed and shows it in an adjacent box. Recent I2C devices support speeds up to 400 kHz, and SPI devices support speeds up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

For universal asynchronous receiver/transmitter (UART) boot modes, baud rate is fixed (for boot purpose) and cannot be changed; however, AISgen will still calculate actual baud rate and show it.

5.3.2 PLL Setup

When the device is taken out of reset, the PLL is set in bypass mode by default. During application development, PLL configuration is typically handled by a GEL file. For production environment, the bootloader is required to configure the PLL. AISgen gives the option to specify the clock source, its frequency, and to configure the selected multipliers and dividers.





When changing any of these values, calculated frequencies will update to reflect the changes. The multiplier and divider values that you have to 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 3.

Table 3. Values of Non-User Configurable PLL Dividers

Divider	Configured to
PREDIV	Divide by 1
PLLDIV1	Divide by 1
PLLDIV2	Divide by 2
PLLDIV4	Divide by 4
PLLDIV6	Divide by 1

5.3.3 SDRAM Setup

EMIF registers need to be configured before any access to external synchronous dynamic random access memory (SDRAM) is made. The bootloader supports configuring EMIFB and EMIFA SDRAM registers for this purpose.

Values entered for each EMIF register are directly programmed to the corresponding register by the bootloader.

5.3.4 Application File Selection

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

You can manually enter the full path to the application executable in the box or click on the browser button on the right to select it graphically.

5.3.5 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 browser button to select it graphically.

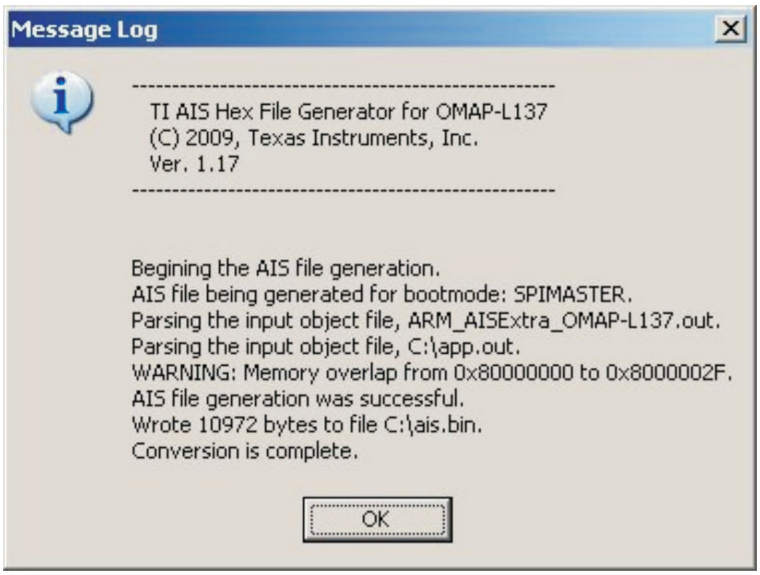
If the specified AIS file has .h as an extension, it is generated in C Header format that can be embedded in the source of another application. This format is human readable and provides information about embedded AIS commands, their arguments, and optional data. This can be useful for diagnostic purposes.

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.

5.3.6 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. AISgen displays a message *Generating…* in the status area near the left bottom of main window. This text changes to *Success* or *AIS generation failed* when AIS generation completes.

If AIS generation fails, a message box will pop up indicating errors reported by underlying AIS generation tools and a detailed log will be copied to the clipboard.

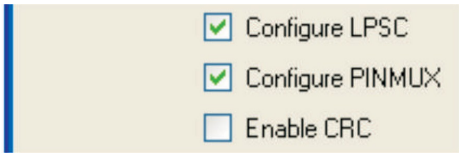


For example, the above error message indicates that the specified ARM application executable file is not in TI COFF format and AISgen failed to parse it.

For non-trivial errors, a detailed log (copied to the clipboard) may be sent to a TI support person for diagnosis.

5.3.7 Additional AIS Options

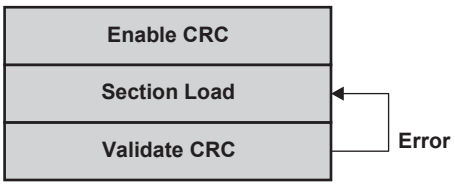
Apart from the basic options, the bootloader gives additional options for error checking and system configuration.



5.3.7.1 CRC

When the CRC option is selected, AISgen adds extra commands in the AIS to check for errors in transferring and loading of 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.

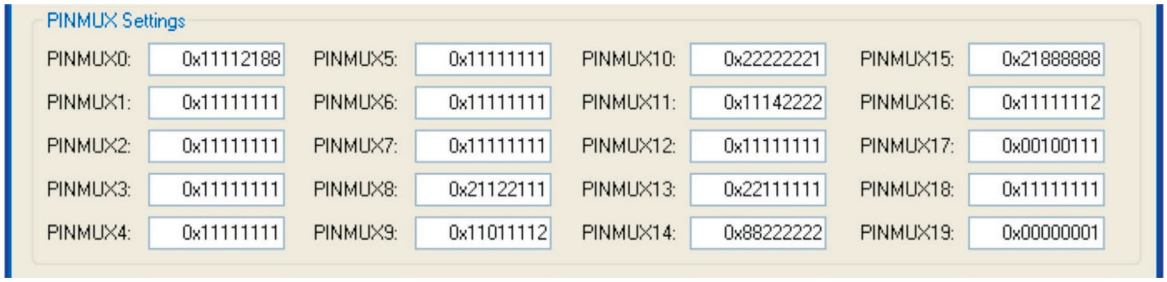


For slave-boot modes, the external master reads the calculated CRC from the device and validates it against the expected value from AIS. In case of an error, it is up to the external master to decide how many times it wants to retry loading the section before reporting failure.

5.3.7.2 PINMUX Settings

When *Configure PINMUX* is selected, the PINMUX Configuration panel activates. This allows you to specify pin multiplexing settings for the target device. This is a useful feature for applications that relied on a GEL file to apply PINMUX settings during development.

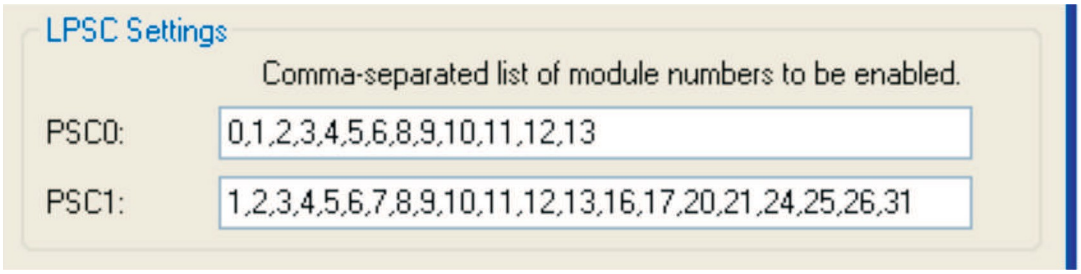
The AISgen tool can import *.pin files generated by the Pin Multiplexing Utility using the File → Import PINMUX Settings… command.



NOTE: This feature is not available for devices with the d800k001 ROM revision. See the introduction for instructions to check your device's ROM revision.

5.3.7.3 LPSC Settings

When *Configure LPSC* is selected, the LPSC Configuration panel activates. This allows you to specify power modules to enable in LPSC0 and LPSC1. Both are specified as simple, comma-separated lists.



NOTE: This feature is not available for devices with the d800k001 ROM revision. See the introduction for instructions to check your device's ROM revision.

6 Master Boot – Booting From a Slave Memory Device

To boot from a slave memory device connected to SPI or I2C peripherals, AIS (in binary format) can be directly Flashed to the memory device.

For booting 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.

For booting from NAND Flash, the AIS should be written to NAND block 1 (NAND block 0 is not used by bootloader) 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 22 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.

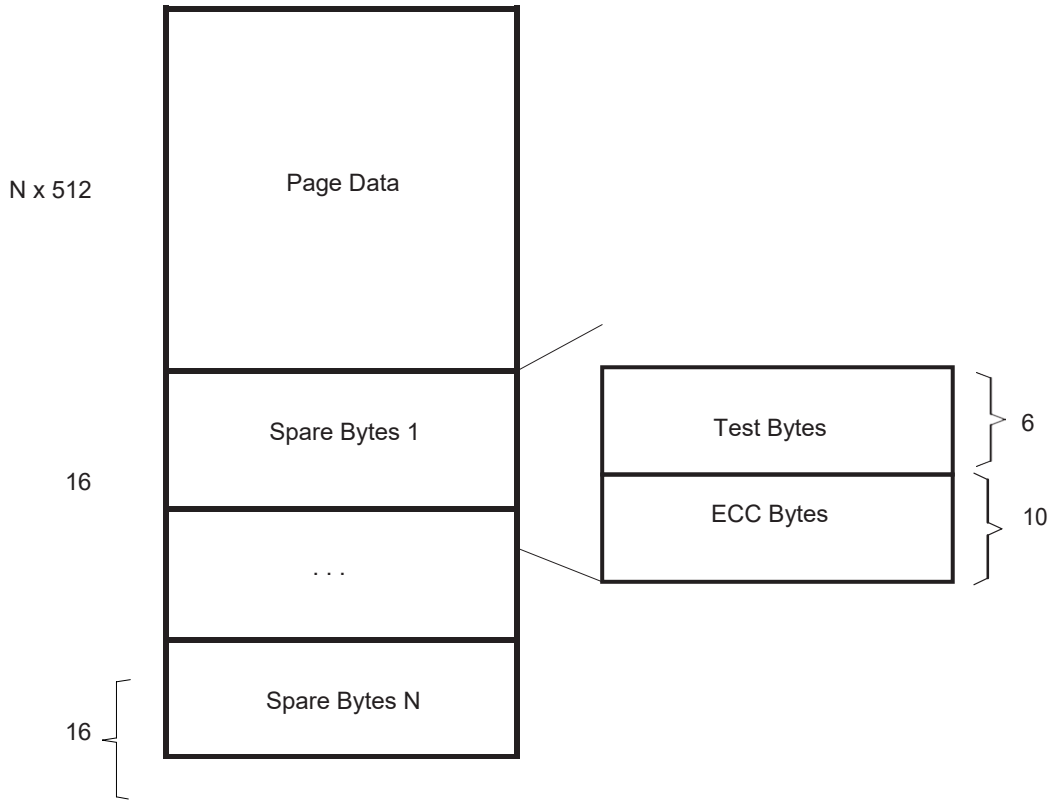


Figure 22. Structure of NAND Page and Spare Bytes

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.

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.

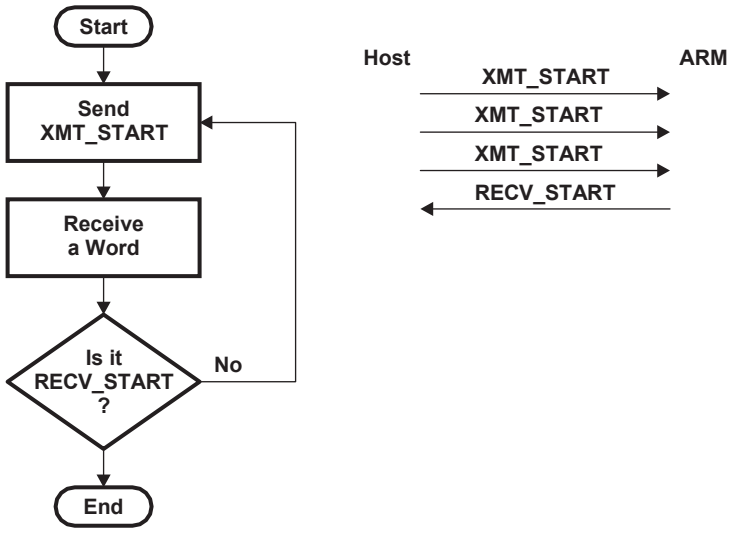


Figure 23. Flowchart: Start-Word Synchronization

For the SPI and I2C slave modes, the bootloader operates the SPI and I2C peripherals in 16-bit mode, so that both start words are 16-bit (0x5853 and 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 and 0x52).

NOTE: While start-words are 8- and 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 24 shows the flowchart of how the POS should be implemented on the host:

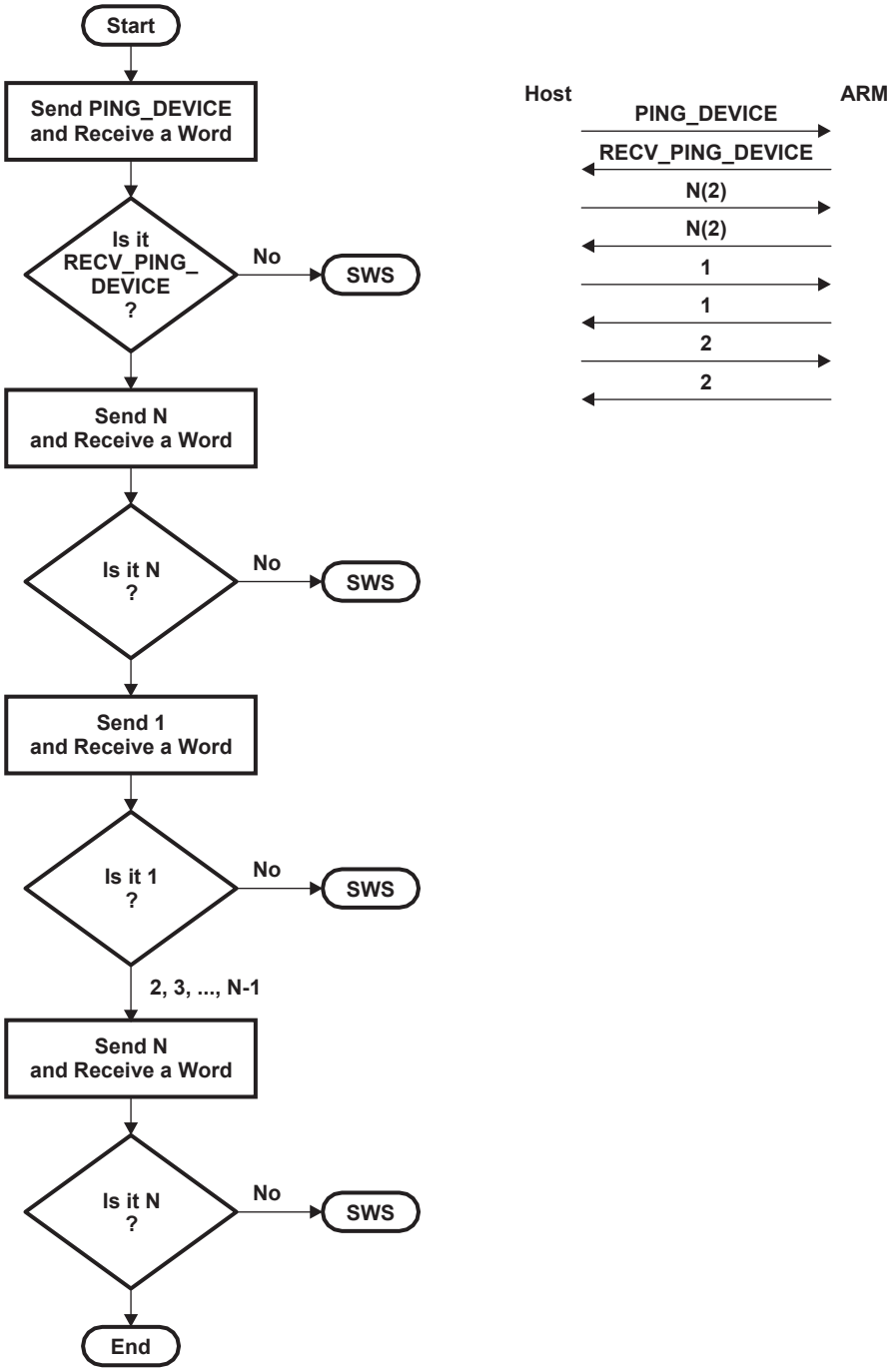


Figure 24. Flowchart: Ping Op-Code Synchronization

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 25 shows the flowchart of how the OS should be implemented on the host:

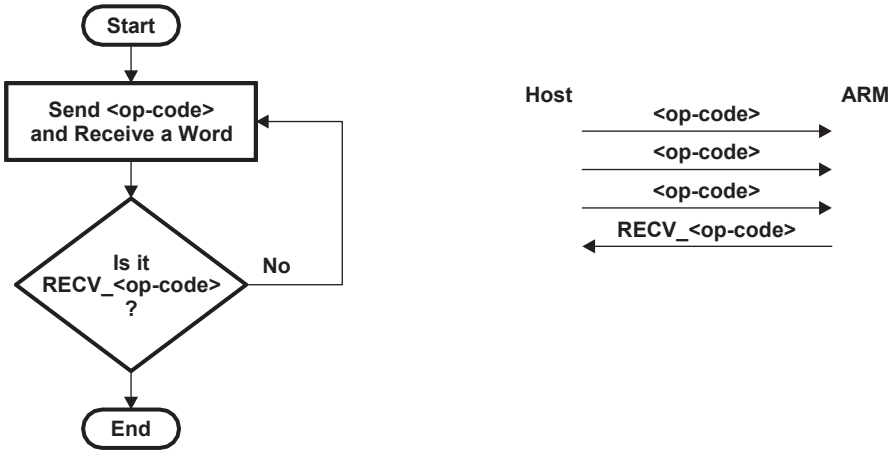


Figure 25. Flowchart: Op-Code Synchronization

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

NOTE: SPI slave boot modes suffer from a bug during Opcode Synchronization. The bootloader will hang if an opcode is sent by the host before the bootloader is ready to receive it (for example., if the bootloader is still processing the previous opcode). Therefore, in SPI slave boot modes, the host should insert sufficient delay between opcodes that the bootloader is ready to receive the next opcode. This bug does not affect other slave boot modes (I2C, UART, etc.).

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 detailed installation instructions, see Section 5.1. 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 26).

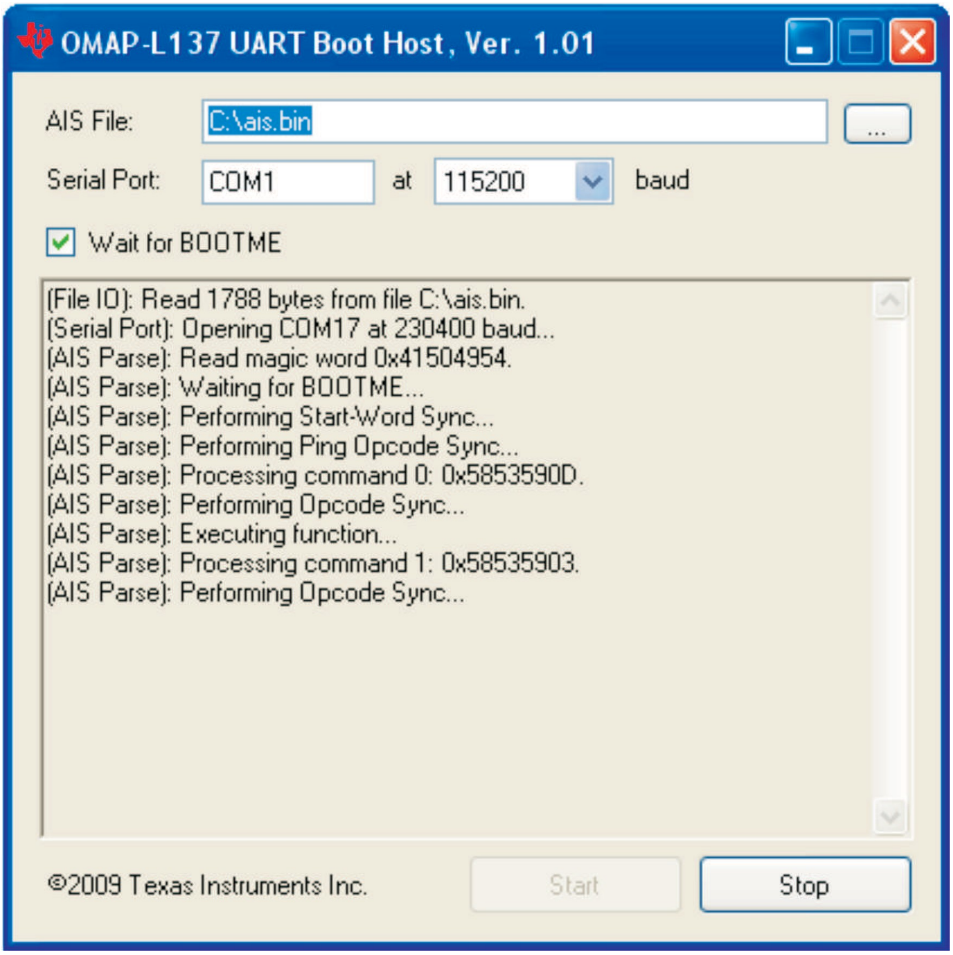


Figure 26. UART Boot Host utility

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. See Section 9 for more information.

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 messages 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.

NOTE: For the d800k001 ROM revision, UART boot runs at 230.4 kpbs instead of the standard 115.2 kbps. To boot this device using a PC as the UART boot master, your PC must be equipped with a serial port that is capable of running at this speed. An add-on serial port (e.g., attached via PCI or USB) may be necessary if the PC lacks a viable built-in port.

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

- Memory Usage: The bootloader uses 16 KB of On-chip RAM starting from 0x80000000 for multiple purposes. This memory should not be used by any initialized section of the user application.
- 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.
- NAND Memory Usage: 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.
- UART: For all UART-boot modes, an input clock source of 24.000 MHz is required. The UART clock cannot be changed from its default speed, listed below.
- UART: Configure the external UART device as follows: 115200 baud, 8 data bits, no parity, 1 stop bit, and no flow control.
- Default clocks: For I2C and SPI master-boot modes and UART-boot modes, the bootloader configures the peripheral clocks by default as shown in Table 4:

Table 4. Default Clock Configurations for Various Boot Modes

Boot Mode	25 MHz	24 MHz
I2C0 Master	49.6 kHz	47.6 kHz
I2C1 Master	97.7 kHz	93.8 kHz
SPI Master	962 kHz	923 kHz
UART (d800k001 ROM)	-	230400 baud
UART (d800k003+ ROM)	-	115200 baud

- I2C0 Slave mode: When booting from I2C0 in slave mode, the input clock source must be in the 13.4 MHz – 26.6 MHz range.
- I2C1 Slave mode: When booting from I2C1 in slave mode, the input clock source must be in the 16.1 MHz – 31.9 MHz range. In this boot mode, the bootloader initializes the PLL with the following parameters and takes it out of the default bypass mode. This is required in order to set the I2C1 module clock to a valid range.
- I2C0 Slave: The bootloader must be addressed with the I2C slave address 0x28.
- I2C1 Slave: The bootloader must be addressed with the I2C slave address 0x29.
- I2C EEPROM: The EEPROM must respond to the I2C slave address 0x50. The bootloader will look for an AIS image at offset 0x00000000.
- All I2C Modes: The I2C bus must use 7-bit addressing.

Table 5. Default PLL Configuration in I2C1 Slave-Boot Mode

Div and 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	/2	SYSCLK3	120	125
PLLDIV4	/4	SYSCLK4	60	62.5
PLLDIV5	/2	SYSCLK5	120	125
PLLDIV6	/1	SYSCLK6	240	250
PLLDIV7	/5	SYSCLK7	48	50

- I2C diagram: Detail: control packet containing I2C slave address (0x50). Note: Assumes sequential read is enabled.

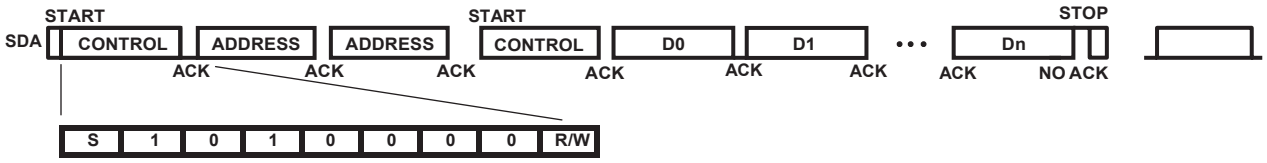


Figure 27. I2C SDA Signal Diagram for I2C EEPROM Boot

- SPI mode for communication: 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 28:

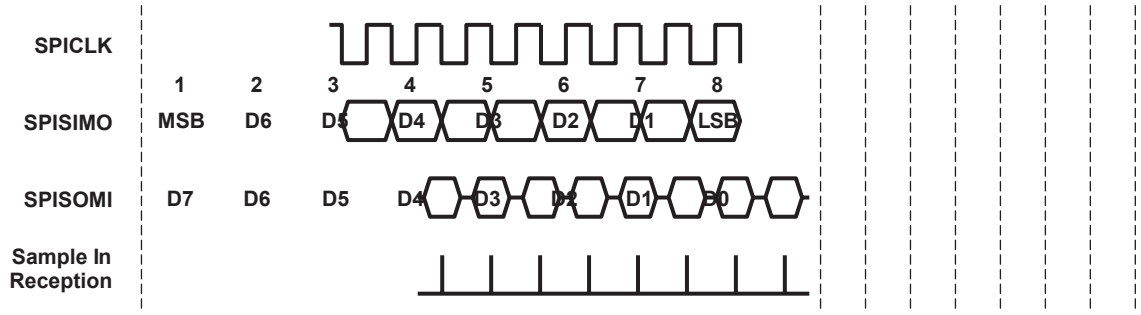


Figure 28. SPI Mode for Communication

- SPI mode: 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.
- SPI EEPROM: 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.

- SPI EEPROM signal diagram: Detail: control packet containing I2C slave address (0x50). Note: assumes sequential read is enabled.

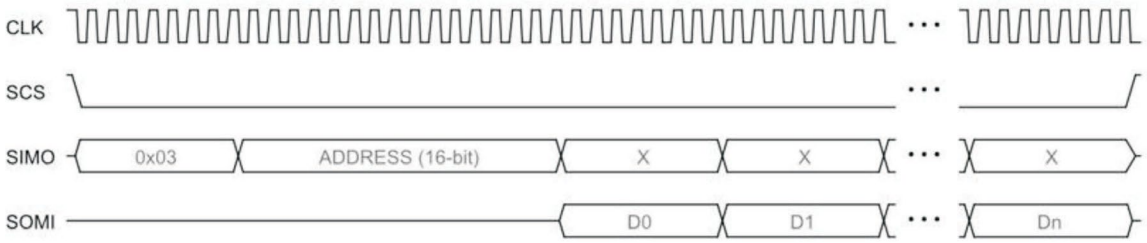


Figure 29. SPI Signal Diagram for SPI EEPROM Boot

- SPI Flash: 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.
- NOR and NAND Boot: See Section 6 for details.

Appendix A Boot Mode Selection Table

Table 6 lists various boot modes supported by the bootloader and configuration of boot pins required to select a boot mode. The first three rows in each column show the signals that are multiplexed with that boot pin. Only the boot pins listed in Table 6 are used by the bootloader; unlisted boot pins are unused. The boot pins are latched by the bootloader when the device exits reset (for example, on the rising edge of reset).

Table 6. Boot Mode Selection

SPI1_CLK EQEP1S BOOT[7]	SPI0_CLK EQEP1I BOOT[2]	SPI0_SIMO[0] EQEP0S BOOT[1]	SPI0_SOMI[0] EQEP0I BOOT[0]	nSPI0_ENA nUART0_CTS EQEP0A BOOT[3]	Multiplexed Pins Boot Mode	 AIS
0	0	0	1	X	NOR	Yes [1]
0	0	1	0	X	HPI	No
0	1	0	1	X	SPI0 Flash	Yes
0	1	1	0	X	SPI1 Flash	Yes
0	1	1	1	X	NAND 8	Yes
1	0	0	0	0	NAND 16 [2]	Yes
0	0	0	0	0	I2C0 Master	Yes
0	0	0	0	1	I2C0 Slave	Yes
0	0	1	1	0	I2C1 Master	Yes
0	0	1	1	1	I2C1 Slave	Yes
0	1	0	0	0	SPI0 EEPROM	Yes
0	1	0	0	1	SPI1 EEPROM	Yes
1	0	0	1	0	SPI0 Slave	Yes
1	0	0	1	1	SPI1 Slave	Yes
1	0	1	1	0	UART0	Yes
1	0	1	1	1	UART1	Yes
1	0	1	0	0	UART2	Yes
1	1	1	1	0	Emulation Debug	N/A

(1) There are three methods to boot from NOR Flash, only one of them uses AIS.
(2) NAND 16 boot mode is not supported for ROM revision d800k001. To check your ROM revision, follow the instructions in Section 1.

Appendix B Details of Supported NAND Devices

The AM17xx 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 parameters (with correct CRC), the bootloader reads the NAND device ID and uses Table 7 to determine the device parameters. If the table reports that the NAND is larger than 128 MB, the bootloader attempts to read parameters from the NAND device's fourth ID byte. The bootloader expects the contents of this ID byte to match Table 8. Fields marked *unused* are not checked by the bootloader.

NOTE: The d800k001 ROM revision does not support 16-bit NAND devices. For instructions on how to check which ROM revision is used by your device, see the introduction to this document.

The AM17xx bootloader does not support NAND devices with page size greater than 4 KB. This applies to both ONFI and non-ONFI NAND devices. If a NAND device is not ONFI-compliant and does not appear in Table 7, it is not supported by the AM17xx bootloader. The boot process will abort when the device ID is not recognized.

Table 7. List of Supported NAND Devices

Device ID	Number of Blocks	Pages Per Block	Bytes Per Page	Size (MB)	Interface
0x33	1024	32	512+16	16	8 bit
0x35	2048	32	512+16	32	8 bit
0x36	4096	32	512+16	64	8 bit
0x39	1024	16	512+16	8	8 bit
0x43	1024	32	512+16	16	16 bit
0x46	4096	32	512+16	64	16 bit
0x49	1024	16	512+16	8	16 bit
0x53	1024	32	512+16	16	16 bit
0x56	4096	32	512+16	64	16 bit
0x59	1024	16	512+16	8	16 bit
0x6B	1024	16	512+16	8	8 bit
0x71	16384	32	512+16	256	8 bit
0x72	8192	32	512+16	128	16 bit
0x73	1024	32	512+16	16	8 bit
0x74	8192	32	512+16	128	16 bit
0x75	2048	32	512+16	32	8 bit
0x76	4096	32	512+16	64	8 bit
0x78	8192	32	512+16	128	8 bit
0x79	8192	32	512+16	128	8 bit
0xA1	1024	64	2048+64	128	8 bit
0xA3	8192	64	2048+64	1024	8 bit
0xA5	16384	64	2048+64	2048	8 bit
0xAA	2048	64	2048+64	256	8 bit
0xAC	4096	64	2048+64	512	8 bit
0xB1	1024	64	2048+64	128	16 bit
0xB3	8192	64	2048+64	1024	16 bit
0xB5	16384	64	2048+64	2048	16 bit
0xBA	2048	64	2048+64	256	16 bit
0xBC	4096	64	2048+64	512	16 bit
0xC1	1024	64	2048+64	128	16 bit
0xC3	8192	64	2048+64	1024	16 bit

Table 7. List of Supported NAND Devices (continued)

Device ID	Number of Blocks	Pages Per Block	Bytes Per Page	Size (MB)	Interface
0xC5	16384	64	2048+64	2048	16 bit
0xCA	2048	64	2048+64	256	16 bit
0xCC	4096	64	2048+64	512	16 bit
0xD3	8192	64	2048+64	1024	8 bit
0xD5	16384	64	2046+64	2048	8 bit
0xDA	2048	64	2048+64	256	8 bit
0xDC	4096	64	2048+64	512	8 bit
0xE3	512	16	512+16	4	8 bit
0xE5	512	16	512+16	4	8 bit
0xE6	1024	16	512+16	8	8 bit
0xF1	1024	64	2048+64	128	8 bit
0xF5	2048	32	512+16	32	8 bit

Table 8. Expected Contents of Fourth ID Byte for NAND Devices Listed in Table 7 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)

Appendix C 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.

```
// data_ptr : Start address of current section (must be 4-byte aligned)
// size     : Size of current section (in bytes)
// crc      : Old crc (from earlier sections) or zero (for first section)

// Process complete 32-bit words
for (size / 4)
{
    // Load a word from memory
    word = *dataPtr++;

    // Update CRC
    bit_no = 31;
    while (bit_no >= 0)
    {
        bit_no--;
        msb_bit = crc & 0x80000000;
        crc = ((word >> bit_no) & 0x1) ^ (crc << 1);
        if (msb_bit)
            crc ^= 0x04C11DB7;                              // CRC-32 polynomial
    }
}

// Process incomplete last word if present
if (remain = size % 4)
{
    word = *dataPtr;

    // Pad incomplete word with zeros to make it complete
    word = (word << remain * 8) >> remain * 8;

    // Update CRC
    bit_no = 31;
    while (bit_no >= 0)
    {
        bit_no--;
        msb_bit = crc & 0x80000000;
        crc = ((word >> bit_no) & 0x1) ^ (crc << 1);
        if (msb_bit)
            crc ^= 0x04C11DB7;                              // CRC-32 polynomial
    }
}
```

Appendix D Details of Pre-Defined ROM Functions

The AM17xx ROM 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 9. List of Pre-Defined ROM Functions

Index	Function
0	PLL Configuration
1	Clock Configuration
2	EMIFB SDRAM Configuration
3	EMIFA SDRAM Configuration
4	EMIFA CE Space Configuration
5	PLL and Clock Configuration
6	Power and Sleep Controller Configuration
7	Pinmux Configuration

D.1 PLL Configuration

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

The PLL configuration register is shown in Figure 30 and described in Table 10.

Figure 30. PLL Configuration Register

31 24	23 16	15 8	7 0
PLLM	POSTDIV	PLLDIV3	PLLDIV5
CLKMODE	PLLDIV7	PLL_LOCK_TIME_CNT	

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

Table 10. PLL Configuration Register Field Descriptions

Bit	Field	Value	Description
31-24	PLLM		Value to be programmed to PLL Multiplier register.
23-16	POSTDIV		Value to be programmed to PLL POSTDIV register, used to generate SYSCLK1 and SYSCLK6. SYSCLK2 = SYSCLK1 / 2 SYSCLK4 = SYSCLK1 / 4
15-8	PLLDIV3		Values to be programmed to PLLDIV3, PLLDIV5 and PLLDIV7 registers, used to generate SYSCLK3, SYSCLK5 and SYSCLK7.
7-0	PLLDIV5		Values to be programmed to PLLDIV3, PLLDIV5 and PLLDIV7 registers, used to generate SYSCLK3, SYSCLK5 and SYSCLK7.
31-24	CLKMODE		Value to be programmed to select PLL clock source. CLKMODE = 0 Crystal CLKMODE = 1 Oscillator
23-16	PLLDIV7		Values to be programmed to PLLDIV3, PLLDIV5 and PLLDIV7 registers, used to generate SYSCLK3, SYSCLK5 and SYSCLK7.
15-0	PLL_LOCK_TIME_CNT		Number of clocks to wait for PLL to lock.

D.2 Clock Configuration

The Clock Configuration function configures the clocks for the active boot peripheral. It programs the SPI and I2C clocks in SPI and I2C master-boot modes or the UART clock in UART-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.2.1 SPI Master Register

The SPI master register is shown in Figure 31 and described in Table 11.

Figure 31. SPI Master Register

31	8	7	0
Reserved		PRESCALE	

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

Table 11. SPI Master Register Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
15-8	PRESCALE		Value to be programmed to the PRESCALE field of the SPIFMT register

D.2.2 I2C Master Register

The I2C master register is shown in Figure 32 and described in Table 12.

Figure 32. I2C Master Register

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

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

Table 12. 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.2.3 UART Slave Register

The UART master register is shown in Figure 33 and described in Table 13.

Figure 33. I2C Master Register

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

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

Table 13. 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 EMIFB SDRAM Configuration

The EMIFB SDRAM configuration function configures the EMIFB registers responsible for SDRAM timing and configuration. This function takes four arguments and writes them to the EMIFB registers with the same names:

- SDCR
- SDTIM1
- SDTIM2
- SDRFC

Before programming EMIFB 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 EMIFB peripheral from its default reset state.

D.4 EMIFA SDRAM Configuration

The EMIFA SDRAM configuration function configures the EMIFA registers responsible for SDRAM timing and configuration. This function takes four arguments and writes them to the EMIFB registers with the same names:

- SDCR
- SDTIMR
- SDSRETR
- SDRCR

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.5 EMIFA CE Space Configuration

The EMIFA CE space configuration function configures the EMIFA CExCFG registers. This function takes four arguments and writes them to the EMIFA registers with the same names:

- CE2CFG
- CE3CFG
- CE4CFG
- CE5CFG

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

This function is not used by the AISgen tool.

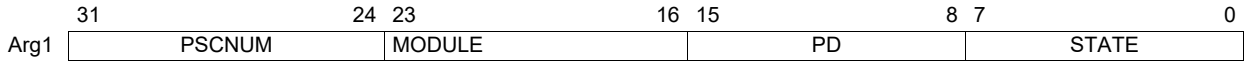
D.6 PLL and Clock Configuration

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.2).

It is preferable to call this function rather than calling the PLL configuration and clock configuration functions separately.

D.7 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).



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

Table 14. 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

NOTE: This feature is not available for devices with the d800k001 ROM revision. See the introduction for instructions to check your device's ROM revision.

D.8 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 15. 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)
```

NOTE: This feature is not available for devices with the d800k001 ROM revision. See the introduction for instructions to check your device's ROM revision.

Appendix E ROM Revision History

E.1 ROMID: D800K001, Silicon Revision 1.0, 1.1

Initial ROM boot loader revision.

E.2 ROMID: D800K003, Silicon Revision 2.0

- Added ARM boot loader code (applies only to ARM-only or ARM-boot devices)
- Added new default UART and PLL clock settings for UART boot modes. Fixed UART divider to correct baud rate to be 115200 (from 230400).
- Added support for LPSC and Pinmux configuration via Function Execute commands.
- Removed the locking of the KICK registers that happened at the end of boot. Note that the KICK registers were disabled in hardware with this silicon revision.

E.3 ROMID: D800K005, Silicon Revision 2.1

Fixed intermittent boot failure mentioned in *OMAP-L137 C6-Integra DSP+ARM Processor Errata (Silicon Revs 2.1, 2.0, 1.1 and 1.0)* (SPRZ291) by fixing the bug in the PRU code.

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