Application Report
SPRACM8–June 2019

OMAPL138/C6748 ROM Bootloader Resources and FAQ

Catalog Processors

ABSTRACT

This application report provides guidance for using the OMAP-L132/L138 on chip read-only memory (ROM) bootloader, consolidates software resources, and answers frequently asked questions. This document supplements the information provided in *Using the OMAP-L132/L138 Bootloader* and covers additional topics like boot utilities, examples to boot the device directly using the ROM bootloader, and debugging boot. It also provides boot time estimates. It is recommended that developers refer to *Using the OMAP-L132/L138 Bootloader* prior to reading this document.

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

Contents

List of Tables

Trademarks

1 OMAP-L138 Boot Process

This application report describes the boot process of the OMAP-L138 Arm® +DSP SoC. The content here also applies to the AM1808 and C6748 devices, unless otherwise specified.

1.1 During Reset

There are two types of reset:

- Power-On Reset (POR) - POR occurs when both RESET and TRST are low. Internal memory is cleared during this state. The boot pins will be latched when RESET goes high.
- Warm Reset - Warm Reset occurs when RESET is low and TRST is high. Internal memory is retained during this state. The boot pins will be not be latched when RESET goes high, but instead retain the same values from the last POR.

It is important for the device to experience a POR when initially powered up. This will clear the emulation and PLL logic, as well as latch the boot pins correctly. Therefore, make sure TRST is externally pulled down on your board. Not doing so can be the cause of many boot related problems.

When the device is held in reset, all device I/Os are internally pulled down.

For the boot and configuration pins, if they are both routed out and 3-stated (not driven), it is strongly recommended that an external pullup/pulldown resistor be implemented. Although, internal pullup/pulldown resistors exist on these pins and they may match the desired configuration value, providing external connectivity can help ensure that valid logic levels are latched on these device boot and configuration pins. In addition, applying external pullup/pulldown resistors on the boot and configuration pins adds convenience to the user in debugging and flexibility in switching operating modes.

For details on how to choose values to oppose the internal pullup/pulldown resistors, see the *Pullup/Pulldown Resistors* section of the *OMAP-L138 C6000™ DSP+ ARM® Processor Data Manual*.

1.2 After Reset

On the rising edge of RESETn, the following steps will occur:

1. The boot pins will be latched.

NOTE: This only occurs when coming out of a POR, as the boot pins will not be relatched after a warm reset.

2. The boot master core [1] begins execution of the ROM bootloader.
   a. The ROM bootloader reads from the boot medium and begins sequentially executing the AIS commands in the boot image [2].
   b. The last AIS command will jump to the entry point of the executable.

2 Boot Utilities

2.1 AISgen

AISgen is a Windows-based tool that is used to generate the boot image in AIS format. Section 3 shows how to take an executable file and convert it to a bootable image using AISgen.

- AISgen Download Link

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

[1] For OMAPL138 and AM1808, ARM9 is is the boot master core. For C6748, C674x core is the boot master core.
[2] For non-AIS boot modes such as NOR legacy, or NOR direct, the process is different. For details on these boot modes, see *Using the OMAP-L132/L138 Bootloader*.

2.2 Serial Boot and Flash Loading Utility

This package contains a set of utilities running from the command-line on Windows for flashing the NAND, NOR, and serial peripheral interface (SPI) Flash of the OMAP-L138 EVM via the serial port.

The latest source code and binaries can be downloaded from TI Git.

The Serial Boot and Flash Loading Utility (which offers the same functionality as a previous program called DVFlasher) executable is called sfh_OMAP-L138.exe.

These programs each encapsulate a distinct binary UBL that is transferred via the universal asynchronous receiver/transmitter (UART). This implies that the chip must be operating in the UART boot mode, showing the BOOTME prompt.

NOTE: The assumption is made that the UART of the device operates at 115200, 8N1. If the oscillator used with the device does not match the one used on the EVM (on a custom platform), the baud rate assumption may be incorrect.

2.2.1 Compiling

A makefile is included for compiling the host and target parts of each utility. The target portions are built first since they are embedded into the host executables. These target portions are loaded to the OMAP-L138 device via the UART boot mode.

Note that the pre-built executables have been tested to work on the EVMs. It is not necessary to rebuild them unless changes need to be made for a custom board.

2.2.1.1 Under Windows

For a particular platform, the detailed instructions for rebuilding the utilities are included in this package, see Section 2.4.4.

2.2.1.2 Under Linux

The Mono Framework must be installed and in the path. RPMs are available at the Mono website. The below instructions assume the GNU cross-compiler tools (arm-arago-linux-gnueabi-gcc, and so forth) are in the current PATH.

1. The C6x Compiler Tools are also required to build some components. These are available free of charge on the TI website.
2. You need to add the bin directory to your PATH and to set the environment variables as directed by the installer. Edit the build.mak file under the Common directory and add the paths to TI Armand C6x compilers.
3. Then go to the GNU directory of the package and run:

```
make
```

2.2.2 Running

2.2.2.1 Under Windows

This utility can be run from the command line under Windows with the .Net Framework 2.0 or later installed through the following steps:

1. Set the boot pins to UART2 boot mode.

   For LogicPD OMAP-L138 EVM:

Pin Number	1	2	3	4	5	6	7	8
Position	OFF	OFF	OFF	OFF	OFF	OFF	ON	ON

   For Spectrum Digital EVM:

Pin Number	1	2	3	4	5	6	7	8
Position	OFF	OFF	OFF	OFF	OFF	ON	OFF	ON

2. Connect a serial cable from the host computer to the OMAP-L138 EVM.
3. Open a command prompt on the host computer and change the directory to where the SFH executable is located.
4. Run the flashing utility with this command:
   a. ..\OMAP-L138\gnu\sfh_OMAP-L138.exe [options]
5. Turn on the board.

2.2.2.2 Under Linux

The serial flasher can be run on a Linux machine with the latest open-source Mono Framework installed. The steps are identical to the Windows environment except the command:

- mono ./sfh_OMAP-L138.exe [options]

2.2.3 Serial Flasher Options

There are three modes for using the serial flasher:

- Erase the target flash type - This erases the entire contents of the flash.
  – ..\sfh_OMAP-L138.exe -erase
- Flash the memory with a single application image - This places an application image at address 0x0 of the flash. This must be an AIS format binary that can be created with the AISgen utilities bundled with the *Using the OMAP-L132/L138 Bootloader* and associated files.
  – ..\sfh_OMAP-L138.exe -flash_noubl <binary application file>
- Custom usage of secondary bootloader: A third option is supported in a tool to allow users to use a user defined secondary bootloader (SBL) that uses a two step boot process. In this setup the secondary bootloader is programmed at address 0x0 of the flash and the application is flashed at offset 0x10000.
  Example usage:
  – ..\\sfh_OMAP-L138.exe -flash <UBL binary file> <binary application file>

    For use of secondary boot approach for these devices, see the Processor SDK RTOS.

For C6748 devices, use the following option: "-targetType C6748".

For AM1808 devices, use the following option: "-targetType AM1808".

NOTE:

- Only the UBL found in the OMAP-L138/GNU/UBL directory can be used with the serial flasher. Other UBLs are not compatible with the serial flasher format.
- The 'flash_noubl' and 'flash' options automatically erases the necessary amount of memory in order to fit the UBL or application image.

Currently, the only supported flash types are NAND, NOR, and SPI.

Additional options are shown below:

```
-targetType            : Specifies exact target type within OMAP-L138 family (default OMAP-L138)
-flashType             : Specifies exact flash type (default SPI_MEM)
-p <COM PORT NAME>     : Allows specifying com port other than default 'COM1' or '/dev/ttyS0'.
-h                     : Show help text.
-v                     : See verbose output from target device
-baud <BAUD RATE>      : Allows specifying baud rate other than default (115200)
-APPStartAddr          : Changes entry point of application (default 0xC1080000)
-APPLoadAddr           : Changes load address of application (default 0xC1080000)
-APPFlashBlock         : Changes the block to flash the image into (only for no_ubl mode)
```

Once any command is run, the "Waiting for BOOTME..." prompt shows. Power cycle the board or press the reset button to continue.

2.3 Modifications for Custom Boards

The default settings used by the tools apply only to the EVMs. For custom boards, changes will most likely be required and the tools must be rebuilt. The custom changes should be made to the files in the OMAP-L138/Common/src and OMAP-L138/Common/include directories. Common changes include:

- DDR Configuration
  - OMAP-L138/Common/src/device.c: Modify the parameters passed to the function DEVICE_ExternalMemInit to match the DDR timing requirements for the custom board.
- UART Settings
  - OMAP-L138/Common/include/device_uart.h: Modify the #define DEVICE_UART_PERIPHNUM to set which UART is connected to the host PC.
  - Note that flow control is not used, so only the RX and TX lines need to be connected for boot and flashing purposes.
- SPI Settings
  - OMAP-L138/Common/include/device_spi.h: Modify the global macros to select the appropriate peripheral and chip select numbers.
  - OMAP-L138/Common/src/device_spi.c: Modify the flash organization in DEVICE_SPI_MEM_params.
  - The SPI flash on the EVM does not need to be unlocked in order to erase or write. Some SPI flashes may need to set the BL bits before writing. To see if this step needs to be added to the initialization, see the device-specific data sheet.
- NAND Settings
  - OMAP-L138/Common/include/device_async_mem.h: Modify the #define DEVICE_ASYNC_MEM_NANDBOOT_BUSWIDTH to match the bus width of your NAND.
- PLL Settings (for non 24 MHz input clocks)
  - OMAP-L138/Common/src/device.c: Modify the parameters passed into the function DEVICE_PLL0Init and DEVICE_PLL1Init to set the proper PLL output frequencies.
  - OMAP-L138/Common/src/device_uart.c: Modify the 'divider' field in DEVICE_UART_config to achieve 115200 baud rate.

2.4 Rebuilding and Customization of Boot Utilities

2.4.1 Download the Flash and Boot Utilities

Download from TI Git.

NOTE: For those that are using version 2.36, it is strongly recommended to upgrade to version 2.40 for a better setup and build experience.

2.4.2 Install and Configure the Required Software

2.4.2.1 Cygwin

1. Cygwin can be downloaded from the Cygwin website.
2. When installing, add the following packages that are not selected by default:
   a. Devel-->make: The GNU version of the 'make' utility
   b. Devel-->subversion: A version control subsystem
   c. Editor-->vim (or similar)
3. After installing, verify that the variables TMP and TEMP both contain /tmp.
   `echo $TMP echo $TEMP`
4. Both commands should return "/tmp". If for some reason they do not, you need to edit the cygwin\home\[user]\.bashrc file to create them:
   a. Export TMP=/tmp
   b. Export TEMP=/tmp
5. Restart Cygwin after this step.

2.4.2.2 Microsoft® .NET Framework

- Download the latest version of the .NET Framework (4.0 or higher).
- Add the location of the C# compiler (csc.exe) to the system path environment variable.
  – Typically this is C:\WINDOWS\Microsoft.NET\Framework\v4.0

2.4.3 Compiler Tools

Note that both and C6x compiler tools are required to build all components of the serial flasher, even for ARM-only parts.

2.4.3.1 Compiler Tools (CODESOURCERY G++ LITE)

- Download the compiler tools from the Mentor Graphics website. Note, Code Sourcery is now available only to registered users and no longer available free of charge.
  – If using v2.40 or later: Edit the Common/build.mak file to have the correct ARM_TOOLS_PATH and ARM_TOOLS_PREFIX variables.
  – If using v2.36 or earlier: Add the bin directory of the cross-compiler tools to the system path environment variable.

2.4.3.2 C6X Compiler Tools

- These can be downloaded from the TI website.
  – If using v2.40 or later: Edit the Common/build.mak file to have the correct DSP_TOOLS_PATH variable
  – If using v2.36 or earlier: Add the bin directory of the TI C6000 Code generation tools to the system path environment variable (C:\Program Files\Texas Instruments\C6000 Code Generation Tools 6.1.11\bin)

Since cl6x program (C6000 compiler) is a native Windows app, the build environment expects users to set Windows include and library search paths, therefore, set DSP_TOOLS_PATH using Cygwin path but DSP_LIB_PATH using Windows conventions. The Cygwin path will help your make setup find the compiler and the Windows library search path will enable the compiler find the RTS libraries.

For example:

```
DSP_TOOLS_PATH?=/cygdrive/c/ccsvx/tools/compiler/c6000/
DSP_LIB_PATH?="C:\ccsvx\tools\compiler\c6000\lib"
```

2.4.3.3 Newer CCS

The CCSv3 projects files can be imported into the newer versions of Code Composer Studio™. It is recommended to use the default import options when doing so. Do not copy the project files into your workspace. They should be left in place.

2.4.4 Rebuilding the Serial Flash and Boot Utils Package for a Particular Platform

1. For a particular platform, the extracted package consists of a 'Common' directory and a <PlatformName>' directory.
2. Open a Cygwin prompt, which is like a Unix/Linux prompt under Windows.
   a. If using v2.36 or earlier, add necessary components to the path:
      `export PATH=<arm-compiler-root>/bin:<C6000-Compiler-Root>/bin:$PATH`
   b. If using v2.40 or later, edit the ARM_TOOLS_PATH, ARM_TOOLS_PREFIX, and DSP_TOOLS_PATH variables in Common/build.mak as needed for your system.
3. Enter into the <PlatformName> directory.
   `cd <PlatformName>`
4. Edit device.mak to include only the part number and flash type required, in order to speed up the build process.
   `vim device.mak`
5. [Optional] To rebuild only the command-line tools (not the CCS projects), cd into the GNU directory.
   `cd GNU`
6. [Optional] To rebuild only the CCS projects, cd into the CCS directory
   `cd CCS`
7. If you are rebuilding everything, stay in the <Platform> directory.
8. Run 'make clean' and 'make'
   ```
   make clean
   make
   ```
9. If you wish to clean-up already built components, run 'make clean' from the path you wish to clean.

2.4.5 Rebuilding the HexAIS Utility for OMAPL13x

HexAIS utility is a boot image generation command line utility provided in the Serial Flash and Boot Utilities package. The source for the package can be found in the Common directory in the root directory. The source files are distributed under the AIS and UtilLib directories. The host utility has been written in C#. To rebuild the utility, set the environment variables as explained in Section 2.4.2. Change the directory to OMAP-L13x/GNU/AISUtils/HexAIS and run 'make clean' and 'make'.

NOTE: You need to build this utility before trying to build the SFH and SFT packages.

3 Boot Examples

3.1 Booting Binaries

The OMAP-L138 contains an Arm9 core as well as a C674x DSP core. Out of reset, the DSP is held in reset and begins executing the bootloader. The following instructions shows how to boot code that blinks the LEDs on the OMAP-L138 EVM. There is a section for booting code to run on Arm and a section for booting code to run on the DSP.

3.1.1 Description

This section provides information on taking an executable file and using the AISgen tool to convert it to a bootable image.

This section does not apply to C6748 devices.

A simple test program is used that repeatedly blinks the LEDs on the OMAP-L138 EVM after booting.

The SOC design requires the Arm core to be in supervisor mode to configure pin multiplexing (PINMUX) registers. The TI Arm toolchain configures the core in user mode during initialization so the Arm application needs to include a boot.asm source file to change the Arm execution state to supervisor mode. This file should be included in all projects that need to access certain SYSCFG registers and require supervisor mode.

3.1.2 Obtaining the Software

The following software is required:

- Sample code and bootable images can be downloaded from TI Git.
  - Contents include:
    - Precompiled AIS boot images (SPI, NAND, NOR, UART) for blinking LEDs on the OMAP-L138 EVM.
    - CCS version 3.3 project and source code for blinking LEDs on the OMAP-L138 EVM.
    - Config files for the AISgen tool used for generating AIS boot images.
- AISgen Tool. For more information, see Section 2.1.
- Serial Boot and Flash Loading Utility. For more information, see Section 2.2.

3.1.3 Running

1. Compile the source code (optional).
   1. Open the project file in CCS and click 'Build.' A precompiled .out file is located in the Debug directory of the software package.

      Windows 7 Users

      Run the AISGen and the HexAISGen utilities under administrative privileges. On the command line utility of the AISGen tool, right click and open as administrator, then type the following:

      ```
      C:> runas /user:Administrator HexAIS_OMAP-L138 <options>
      ```
2. Generate AIS image (optional):
   a. Open the AISgen tool and select File -> Load Configuration.
   b. Choose the .cfg file (located in the "Debug" directory of this software package) corresponding to the desired boot medium.
   c. Choose the correct Device Type for the silicon revision you are using. If you are unsure, use the GEL file provided in Section 4 to determine this.
   d. Provide the path to the OMAP-L138-ARM-LED.out file for the " Application File" box.
   e. Provide the path where the destination AIS file will be created.
   f. Click Generate AIS.

3. Flash the AIS image:
   a. For NAND, NOR, and SPI boot modes, use the serial flasher to flash the image to the desired boot medium using the "-flash_noubl" option. For more information, see Section 2.2.3.
   b. For UART boot mode, use the serial loader to boot from the UART host as follows:

```
./slh_OMAP-L138.exe -waitForDevice OMAPL138-ARM-LED-uart.bin
```

3.2 Booting DSP Binaries on AM1808/OMAPL138

3.2.1 Description

This section provides information on how to boot a DSP executable on Arm-boot devices. A user boot loader (UBL) will run first and is required to wake up the DSP, which will begin execution of the DSP application. The Arm and the DSP programs are combined into a single AIS image that can be flashed and booted directly.

NOTE: This section does not apply to AM1808 devices.

A simple test program is used, in which the DSP repeatedly blinks the LEDs on the OMAP-L138 EVM after booting.

This is a very straightforward process, with a few key things to remember:

- The Arm must set the HOST1CFG register to change the DSP reset vector. This should point to the entry point of the DSP executable, but it can only be set to 1KB boundaries (the bottom 10 bits are reserved and read as 0). Therefore, the linker command file for the DSP application should specifically force the entry point to also be aligned to a 1KB boundary. For this example, a memory region named "entry_point" was created at address 0x80010000, and the section ".text:_c_int00" was assigned to that memory region.

  NOTE: If you are using SYS/BIOS, see Appendix A for setting c_int00.

- The and DSP .map files cannot overlap. If both programs use the same memory address, they will overwrite each other and most likely crash. The linker command files should be written to prevent this.
- For silicon revision 1.0 and 1.1, the KICK registers must be unlocked before writing to the HOST1CFG register. Additionally, for all revisions, the Arm must be in supervisor mode. The SOC design requires the Arm core to be in supervisor mode to configure pin multiplexing (PINMUX) registers. The TI Arm toolchain configures the core in user mode during initialization so the Arm application needs to include a boot.asm source file to change the Arm execution state to supervisor mode . This file should be included in all projects that need to access certain SYSCFG registers and require supervisor mode.
- If any sections of the DSP memory map are located in DSP L2 RAM (0x11800000), be aware of two things:
  – Make sure all L2 RAM addresses in the DSP linker command file are referenced in the 0x118xxxxx range and not 0x008xxxxx, as they cannot write to the 0x008xxxxx address range
  – Use the "Configure PSC" function of AISGen to enable the DSP LPSC (PSC0,#15). If the DSP megamodule is in reset, the L2 RAM will not be accessible and the section loads will fail. The AISGen CFG file included with this project enables PSC0,#15 by default.

3.2.2 Obtaining the Software

The following software is required:

- Sample code and bootable images: here
  - Contents include:
    - Precompiled AIS boot images (SPI, NAND, NOR, UART) for blinking LEDs on the OMAP-L138 and C6748 EVM.
    - CCS version 3.3 and DSP projects and source code for blinking LEDs with the DSP on the OMAP-L138 and C6748 EVM.
    - Config files for the AISgen tool used for generating AIS boot images.
- AISgen Tool. For more information, see Section 2.1.
- Serial Boot and Flash Loading Utility. For more information, see Section 2.2.

3.2.3 Running

3.2.3.1 OMAP-L138 EVM

1. Compile the source code:
   a. Two projects are provided: one for the UBL and one for the DSP LED blinking program. Open each project file in CCS and click'Build' in the respective processor window. Precompiled .out files are located in the Debug directories of the software package.
2. Generate the AIS image.
3. Open the AISgen tool and select File -> Load Configuration.
4. Choose the .cfg file (located in the base directory of the OMAPL138-DSP-LED folder) corresponding to the desired boot medium.
5. Choose the correct Device Type for the silicon revision you are using. If you are unsure, use the GEL file provided in Section 4 to determine this.
6. Fill in the " Application File" field with both the UBL and the DSP program. When given two source application files, the AISgen tool combines them into a single image in the order that they are listed.
   a. First, click the "..." button and provide the path to the OMAPL138-DSP-LED-ARM.out file.
   b. Next, click the "+" button and choose the OMAPL138-DSP-LED-DSP.out file.
7. Fill in the "AIS Output File" field by giving the path where the destination AIS file will be created.
8. Click Generate AIS.

Flash the AIS image:

a. For NAND, NOR, and SPI bootmodes, use the serial flasher to flash the image to the desired boot medium using the "-flash_noubl" option. For more information, see Section 2.2.3.
b. For UART bootmode, use the serial loader to boot from the UART host as follows:
   i. ./slh_OMAP-L138.exe -waitForDevice OMAPL138-ARM-LED-uart.bin

3.2.3.2 C6748 EVM

In this case, only the DSP program is required. The same instructions above apply with a few exceptions:

- Use the C6748-LED-x.cfg file to create C6748-LED-x.bin. Note that only the DSP application is needed as an input file.
- When flashing, use the -targetType C6748 option.

4 Debugging Bootloader

The OMAP-L1x Debug GEL file can be used to debug boot issues, determine ROM revisions, and display other information such as PLL settings. This GEL file was created to work with all OMAP-L13x, AM1x, and C674x devices.

The Debug GEL file comes bundled with the CCS installation and can be found at CCS_INSTALL_PATH\ccs_base\emulation\boards\evmomapl138\gel.

Directions for CCS:

1. Connect to the processor, can be Arm or DSP of any OMAP-L1x, AM1x, or TMS320C674x device.
2. Tools -> Gel Files.
3. Right-click on the window and select "Load Gel".
4. Go to Scripts -> Diagnostics -> Run All.

NOTE: Make sure the MMU is disabled when you run the scripts in order to properly read the registers. You can disable the 's MMU in CCS by going to Tools -> Advanced Features.

The GEL file will print out the following information:

- ROM ID: Revision number of the boot ROM
- Silicon revision number
- Boot Mode: Current boot mode, as selected by the boot pins latched at reset
- ROM Status Code: Current status of the ROM code
- Description: Description of any error messages that the ROM may have encountered during boot
- Program Counter: The current program counter of the connected device (or DSP)
- Device Information: Generic device information that may be helpful when getting support from TI
- Clock information: PLLm_SYSCLKn is output

NOTE: If your board uses an input clock other than 24 MHz you need to modify the definition at the start of the gel file accordingly.

- PSC state information

5 OMAP-L138 Boot Benchmarks

This section gives measurements of the boot times for various boot modes supported by OMAP-L138/AM1808/C6748 and derivatives. Although the number can be considered typical, they only apply to the exact environment used for the benchmarking.

5.1 Host Boot Performance

Table 1. Host Boot Performance

Device	Boot Mode	Boot Time (ms)	
		1KB	124KB
C6748	NAND8	60	80
	NAND16	36	52
	NOR	3	15
	SPI Flash	4	96
	SD	34	195
	SDHC	42	180
	MMC	68	84
OMAP-L138 AM1808	NAND8	200	244
	NAND16	144	203
	NOR16	32	56
	SPI Flash	36	312
	SD	320	592
	SDHC	336	544
	MMC	264	460

5.2 Test Details

5.2.1 Methodology

The test case that was used immediately toggles a GPIO at the start of the main function. Boot times were measured from the rising edge of RESETn to the falling edge of the GPIO.

The board begins in a powered state to remove variability of power-up times. For example, some MMC/SD cards can take up to 1 second after power is applied to be accessible.

Two images sizes were used:

- The 1KB image size boot time basically represents the initialization time to read any data from the device.
- The 124KB image size shows how long it takes to boot a larger image.

The following formula can be used to estimate boot times (time_x) for other image sizes (size_x):

time_x = time_1kb + size_x * (time_124kb - time_1kb) / 123

5.2.2 Software

The test case is loaded completely into L3 shared memory, so no external memory is used. The bootloader enables the PLLs and the host CPU runs at 456 MHz.

5.2.3 Hardware

Testing was performed on Revision 2.1 of OMAP-L138/AM1808/C6748 devices. Boot times may differ greatly from previous revisions for some boot modes.

Table 2. Hardware Table

Hardware	Part Number	Frequency
Board	LogicPD OMAP-L138 EVM	-
SoC	OMAP-L138/AM1808/C6748 Rev. 2.1	456 MHz
NAND8	Micron MT29F4G08AAC 8-bit NAND Flash	-
NOR	Intel PC28F640P30T85 16-bit NOR Flash	-
SPI Flash	Winbond W25X64VSFIG 64M Serial Flash	45.6 MHz
SDHC	Transcend 4GB Micro SD HC	38 MHz
SD	Patriot 2GB SD	38 MHz
MMC	palmOne 256MB MultiMediaCard	38 MHz

5.2.4 Discussion

The results show that the OMAP-L138/AM1808 boot times (boot devices) are generally much longer than for the C6748 (DSP boot device). There are a few explanations for this:

- The cache is not enabled by default, while the DSP cache is. This means that the code, especially delay loops, will execute much faster on the DSP.
- The OMAP-L138/AM1808 require additional time before the bootloader begins executing.
- The DSP compiler generates fewer instructions for delay loops than the compiler, resulting in faster execution.

For larger boot images on OMAP-L138/AM1808, it would be beneficial to enable the cache through AIS commands. Future versions of AISgen will provide this option.

6 OMAP-L138 Bootloader FAQ

Question: How do I boot an executable?

Answer: The executable in COFF/ELF must be converted to AIS format using the AISgen tool. For more details and examples on this process, see Section 3.1.

Question: How do I boot a DSP executable?

Answer: Since OMAP-L138 is an ARM-boot device, an executable must be booted first to wake up the DSP. For C6748, the DSP AIS binary can be booted directly. For more details and examples on this process, see Section 3.2.

Question: Are there any sample AISgen config files that work with the EVM?

Answer: The following file contains AISgen config files that set up the core/mDDR frequencies to 456/150 MHz and 300/132 MHz.

Download Sample AISgen Config Files. These are available with the AISgen package at the location: AISgen for D800K008\cfg_files.

Note that the core voltage must be scaled appropriately when choosing an OPP.

Question: Why does my program work in CCS but does not boot from flash?

Answer: Gel File Reliance

The most common problem is that some configuration is done in the GEL file that the code needs, such as external memory configuration, pinmux configuration, or PLL configuration. Most of these functions can also be performed by the bootloader using the AISgen tool. Compare the PLL, pinmux, and DDR registers after booting and verify they match what the GEL file sets them to.

Answer: Incorrect External Memory Configuration

If your code has sections in external memory, the bootloader must configure the controller settings before the code is copied. Verify that the DDR configuration settings used in AISgen match those used in the GEL file. You should be able to poke the DDR memory region in CCS (0xC0000000) if the DDR is configured correctly.

Answer: Supervisor vs User Mode

The SOC design requires the Arm core to be in supervisor mode to configure pin multiplexing (PINMUX) registers. The TI Arm toolchain configures the core in user mode during initialization so the Arm application needs to include a boot.asm source file to change the Arm execution state to supervisor mode. This file can be found in the zip file in Section 3.2.2. In CCS, the "Search libraries in priority order (--priority, -priority)" box should be checked in the project linker options to avoid linking errors.

Answer: Kick Register Unlocking

On previous revision devices (1.0 and 1.1), the KICK registers must be unlocked before accessing certain protected registers. While your GEL file in CCS may do this automatically, it should be done once at the beginning of your code to make it portable. For more information on this process, see the *OMAP-L138 C6000 DSP+ARM Processor Technical Reference Manual*.

Answer: Incorrect Boot Mode

Double check that the boot pins are at proper voltages by the rising edge of RESET. Use the Debug GEL file in Section 4 to determine what boot mode the device latched and see any other ROM error messages. Also, verify that TRST is externally pulled down.

Answer: Using the Bootloader Shared Memory

The ROM bootloader itself uses 16 KB of Shared RAM starting from 0x80000000 for multiple purposes. This memory should not be used by any initialized section of the user application.

Answer: RAM vs ROM Autoinitialization Model

When booting code through the ROM, make sure you are using the ROM autoinitialization model (-c) in your linker options. If the RAM autoinitialization model is used (-cr), some memory locations will never get loaded even though they did under CCS. You can find the build option here:

- CCS3.3: Build Options -> Linker -> Autoinit Model
- CCS4+: Build Options -> Runtime Environment

Question: Do I need a secondary bootloader (UBL)?

Answer: A secondary bootloader, aka, User Bootloader (UBL), was required on older devices where the bootloader could not parse AIS files. By using the AISgen tool with the OMAP-L138 bootloader, most of the functions previously performed by the UBL can be done instead by the bootloader.

For a typical Linux application, the old flow looked something like this:

- UBL (sets up DDR, PSC, and copies U-Boot to memory)
- U-Boot (loads Linux and file system)
- Linux

The flow for the OMAP-L138 would look like this:

- AIS-signed U-boot (sets up DDR, PSC, and loads Linux and file system)
- Linux

So in general, there is not a need for a separate UBL, as the AIS functions can perform most of the same tasks.

Question: How do I prepare an image for nor direct or nor legacy boot modes?

Answer: For instructions on creating the boot image, see the *Boot* section of the Processor SDK Software Developer's Guide.

Question: How should an SD/MMC card be formatted in order to boot?

Answer: For details on preparing the SD/MMC card, see the *Boot* section of the Processor SDK Software Developer's Guide..

7 References

Texas Instruments: *OMAP-L138 C6000™ DSP+ Arm® Processor Data Manual*

Texas Instruments: *OMAP-L138 C6000 DSP+Arm Processor Technical Reference Manual*

Texas Instruments: *Using the OMAP-L132/L138 Bootloader*

Appendix A

A.1 Setting c_int00 Using SYS/BIOS

If you are using SYS/BIOS, system start up and initialization is handled by XDCtools. For some targets, the c_int00 code is a C function. The C code is compiled with the "-mo" flag, which places the symbol _c_int00 in a subsection of .text (.text:_c_int00).

To place the .text symbol at an explicit address, add a custom linker command (.cmd) file to your project with a directive similar to:

```
SECTIONS {
   .text:_c_int00 > 0xc3000000
}
```

On other targets, the _c_int00 code is written in assembly language. As of XDC 3.24.02, this assembly code is placed in .text and not .text:_c_int00. This will be fixed in a future release of XDC and BIOS. But, until then, the following linker workaround can be applied to place the _c_init00 function. You will have to update your path and library name accordingly. Check your .map file for the name of the boot library and the generated linker.cmd filed for the full path.

```
 boot : > 0x3D8000 PAGE = 0
{-
l"C:\ti\ccsv5_3_0_00042\xdctools_3_24_02_30\packages\ti\targets\rts2800\lib\boot.a28FP"
<boot_cg.o28FP> (.text)
}
```

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