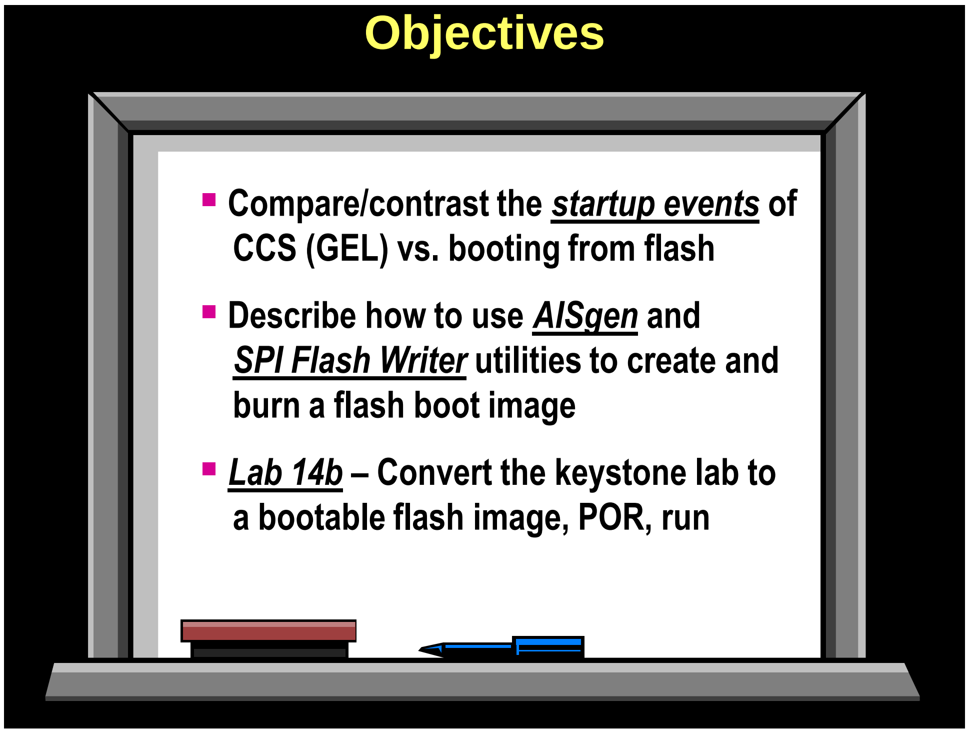
Booting From Flash

Introduction

In this chapter the steps required to migrate code from being loaded and run via CCS to running autonomously in flash will be considered. Given the AISgen and SPIWriter tools, this is a simple process that is desired toward the end of the design cycle.

Objectives



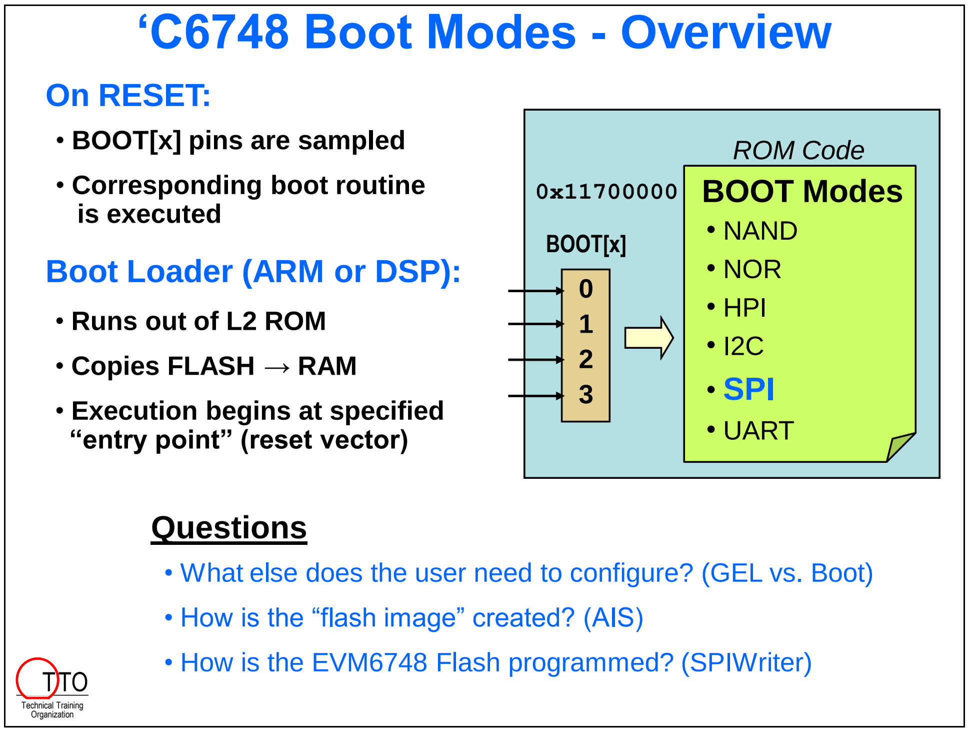
Objectives

- Compare/contrast the ***startup events*** of CCS (GEL) vs. booting from flash
- Describe how to use ***AISgen*** and ***SPI Flash Writer*** utilities to create and burn a flash boot image
- ***Lab 14b*** – Convert the keystone lab to a bootable flash image, POR, run

Module Topics

Booting From Flash

Boot Modes – Overview



‘C6748 Boot Modes - Overview

On RESET:

- **BOOT[x] pins are sampled**
- **Corresponding boot routine is executed**

Boot Loader (ARM or DSP):

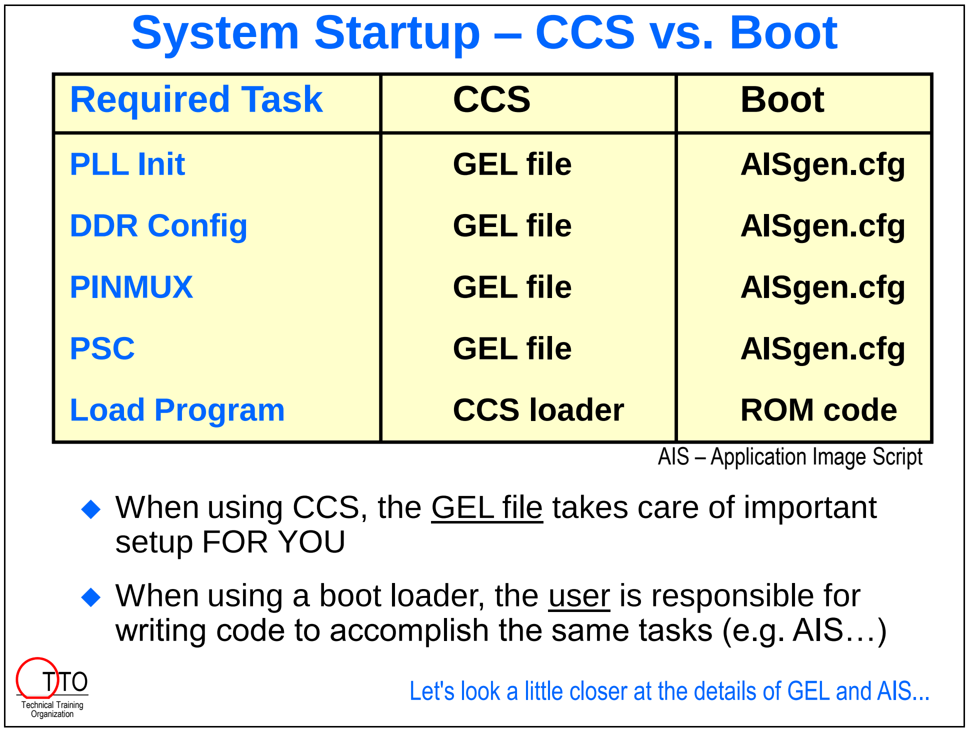
- **Runs out of L2 ROM**
- **Copies FLASH → RAM**
- **Execution begins at specified “entry point” (reset vector)**

ROM Code
0x11700000
BOOT[x]
0
1
2
3
BOOT Modes
• NAND
• NOR
• HPI
• I2C
• SPI
• UART
TTO
Technical Training Organization

Questions

- What else does the user need to configure? (GEL vs. Boot)
- How is the “flash image” created? (AIS)
- How is the EVM6748 Flash programmed? (SPIWriter)

System Startup



System Startup – CCS vs. Boot

Required Task	CCS	Boot
PLL Init	GEL file	AISgen.cfg
DDR Config	GEL file	AISgen.cfg
PINMUX	GEL file	AISgen.cfg
PSC	GEL file	AISgen.cfg
Load Program	CCS loader	ROM code

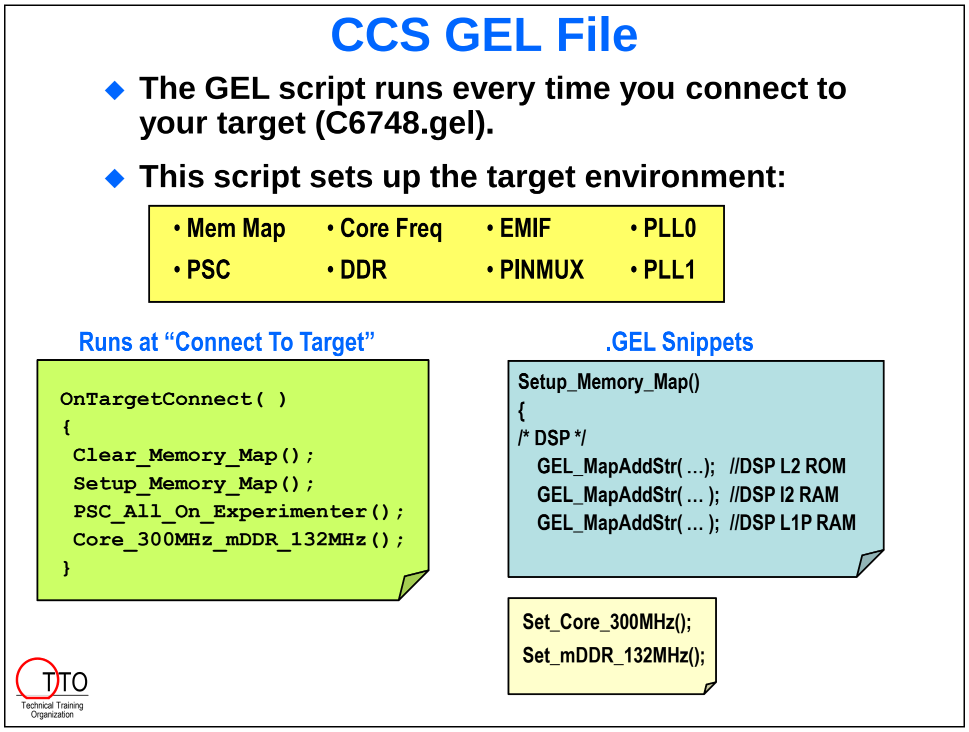
AIS – Application Image Script

- When using CCS, the GEL file takes care of important setup FOR YOU
- When using a boot loader, the user is responsible for writing code to accomplish the same tasks (e.g. AIS…)

TTO Technical Training Organization

Let's look a little closer at the details of GEL and AIS...

Init Files



CCS GEL File

- **The GEL script runs every time you connect to your target (C6748.gel).**
- **This script sets up the target environment:**

• Mem Map	• Core Freq	• EMIF	• PLL0
• PSC	• DDR	• PINMUX	• PLL1

Runs at "Connect To Target"

```
OnTargetConnect( )
{
 Clear_Memory_Map();
 Setup_Memory_Map();
 PSC_All_On_Experimenter();
 Core_300MHz_mDDR_132MHz();
}
```

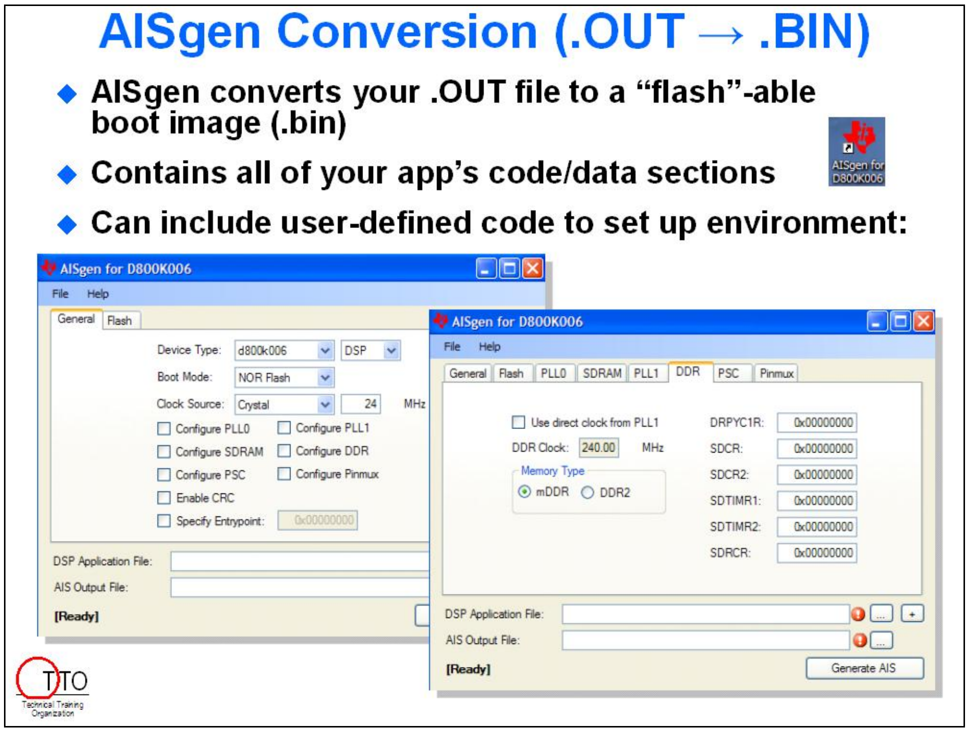
.GEL Snippets

```
Setup_Memory_Map()
{
/* DSP */
   GEL_MapAddStr( ...);  //DSP L2 ROM
   GEL_MapAddStr( ... );  //DSP l2 RAM
   GEL_MapAddStr( ... );  //DSP L1P RAM
```

```
Set_Core_300MHz();
Set_mDDR_132MHz();
```

TTO Technical Training Organization

AISgen Conversion



AISgen Conversion (.OUT → .BIN)

- AISgen converts your .OUT file to a "flash"-able boot image (.bin)
- Contains all of your app's code/data sections
- Can include user-defined code to set up environment:

AISgen for D800K006

File Help

General | Flash

Device Type: d800k006 DSP
Boot Mode: NOR Flash
Clock Source: Crystal 24 MHz

- ☐ Configure PLL0 ☐ Configure PLL1
- ☐ Configure SDRAM ☐ Configure DDR
- ☐ Configure PSC ☐ Configure Pinmux
- ☐ Enable CRC
- ☐ Specify Entrypoint: 0x00000000

DSP Application File:
AIS Output File:
[Ready]

AISgen for D800K006

File Help

General | Flash | PLL0 | SDRAM | PLL1 | DDR | PSC | Pinmux

☐ Use direct clock from PLL1

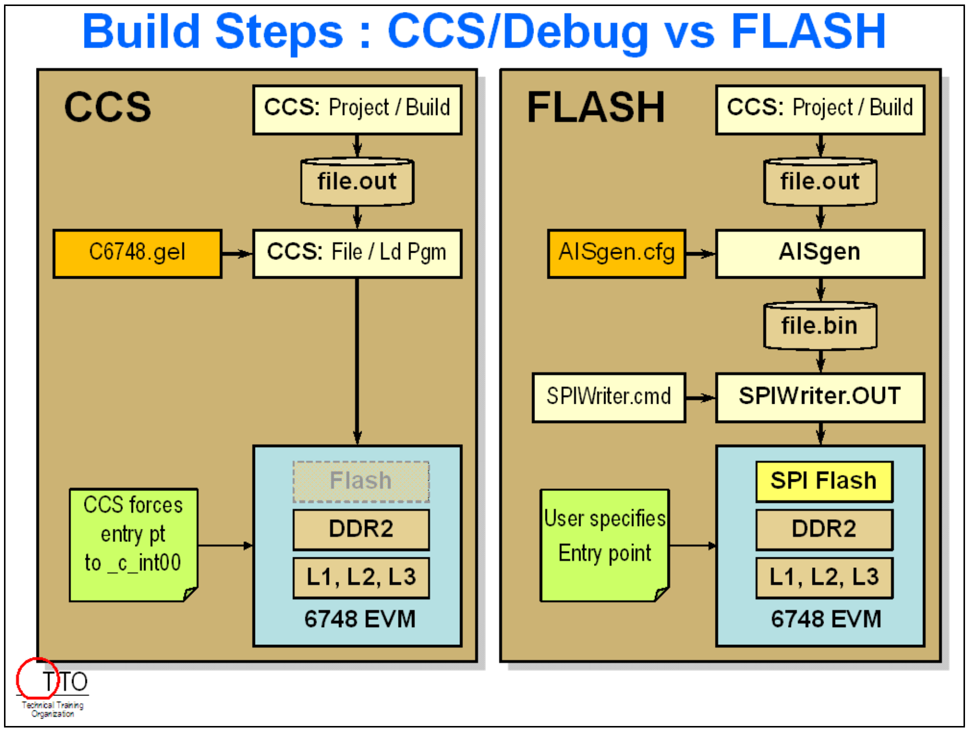
DDR Clock: 240.00 MHz

Memory Type: (● mDDR) ○ DDR2

DRPYC1R:	0x00000000
SDCR:	0x00000000
SDCR2:	0x00000000
SDTIMR1:	0x00000000
SDTIMR2:	0x00000000
SDRCR:	0x00000000

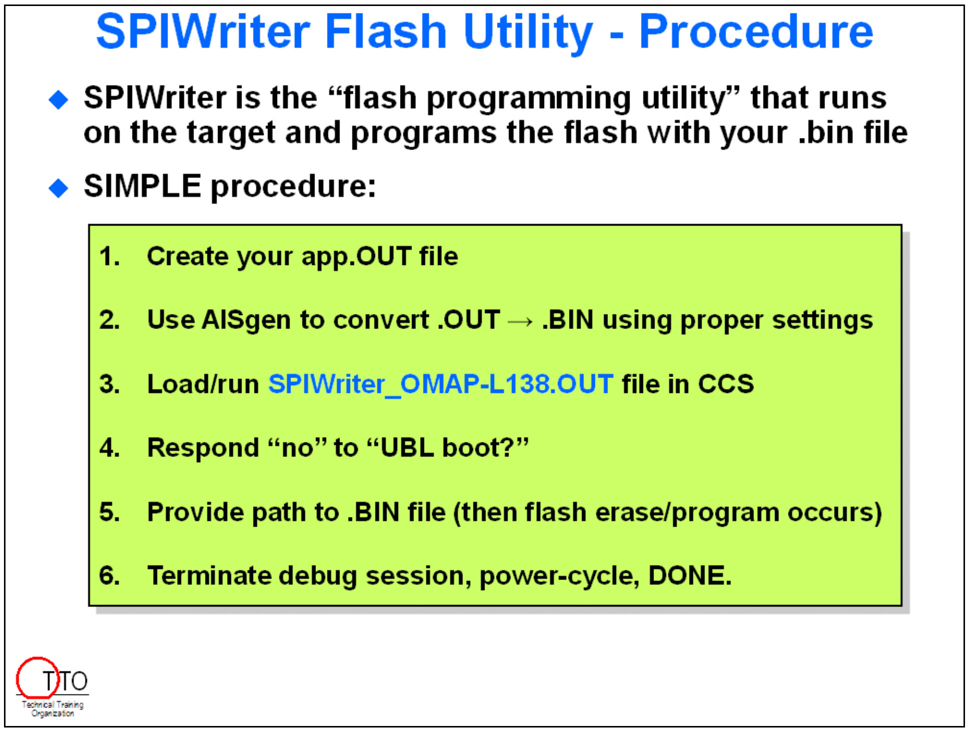
DSP Application File:
AIS Output File:
[Ready]
Generate AIS

Build Process



Build Steps : CCS/Debug vs FLASH

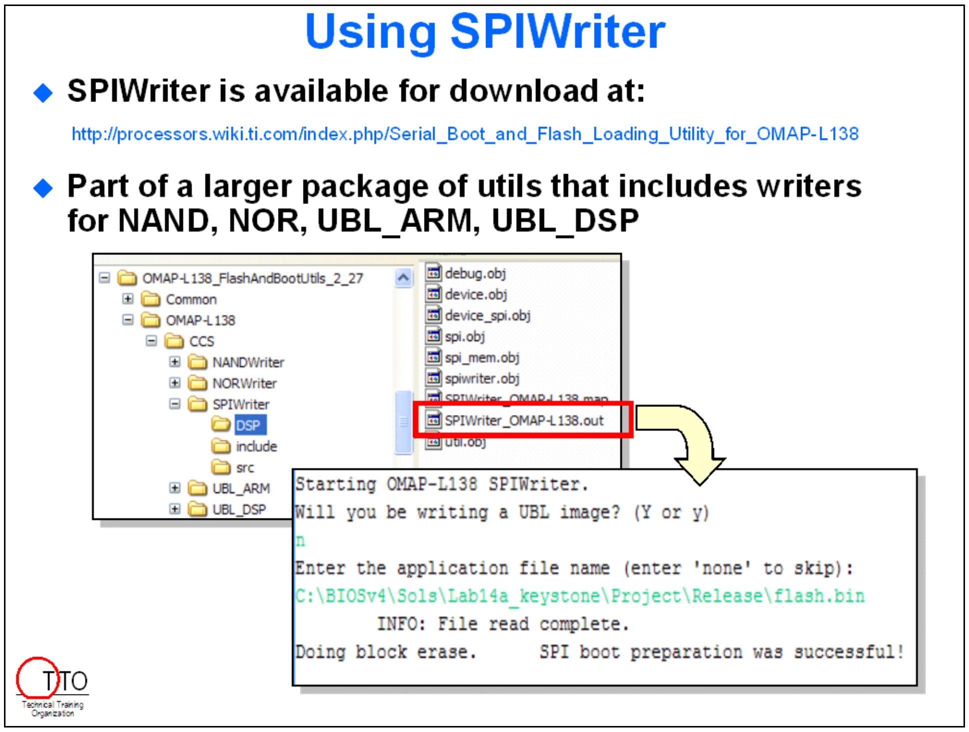
SPIWriter Utility (Flash Programmer)



SPIWriter Flash Utility - Procedure

- **SPIWriter is the "flash programming utility" that runs on the target and programs the flash with your .bin file**
- **SIMPLE procedure:**

1. **Create your app.OUT file**
2. **Use AISgen to convert .OUT → .BIN using proper settings**
3. **Load/run SPIWriter_OMAP-L138.OUT file in CCS**
4. **Respond "no" to "UBL boot?"**
5. **Provide path to .BIN file (then flash erase/program occurs)**
6. **Terminate debug session, power-cycle, DONE.**

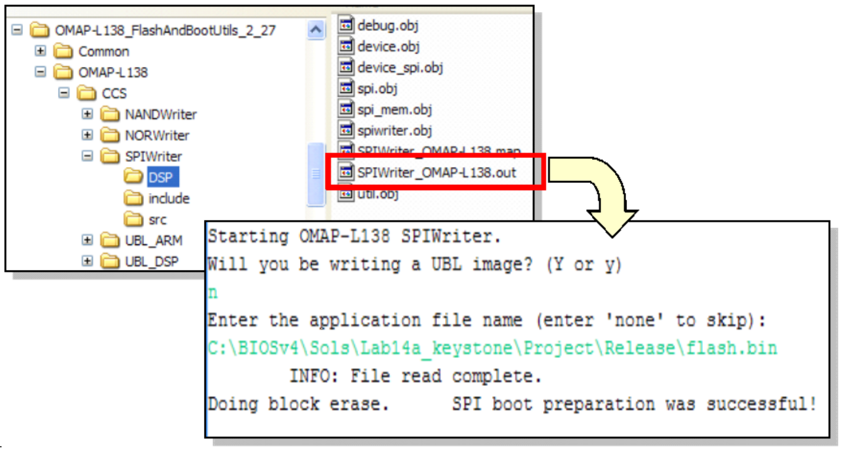


Using SPIWriter

- **SPIWriter is available for download at:**

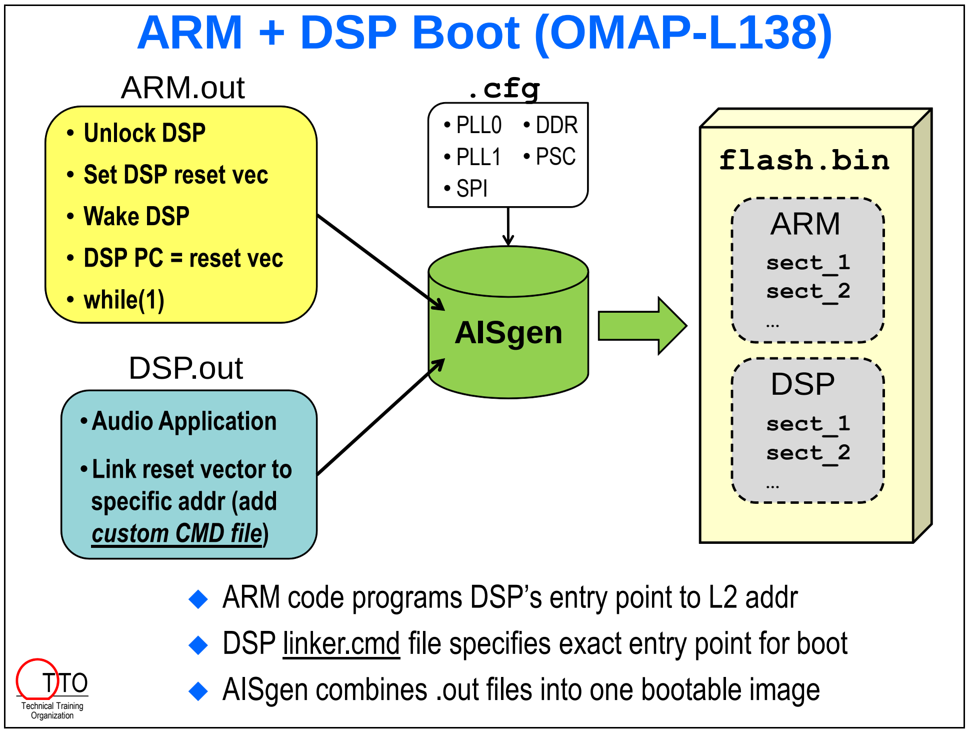
  http://processors.wiki.ti.com/index.php/Serial_Boot_and_Flash_Loading_Utility_for_OMAP-L138

- **Part of a larger package of utils that includes writers for NAND, NOR, UBL_ARM, UBL_DSP**



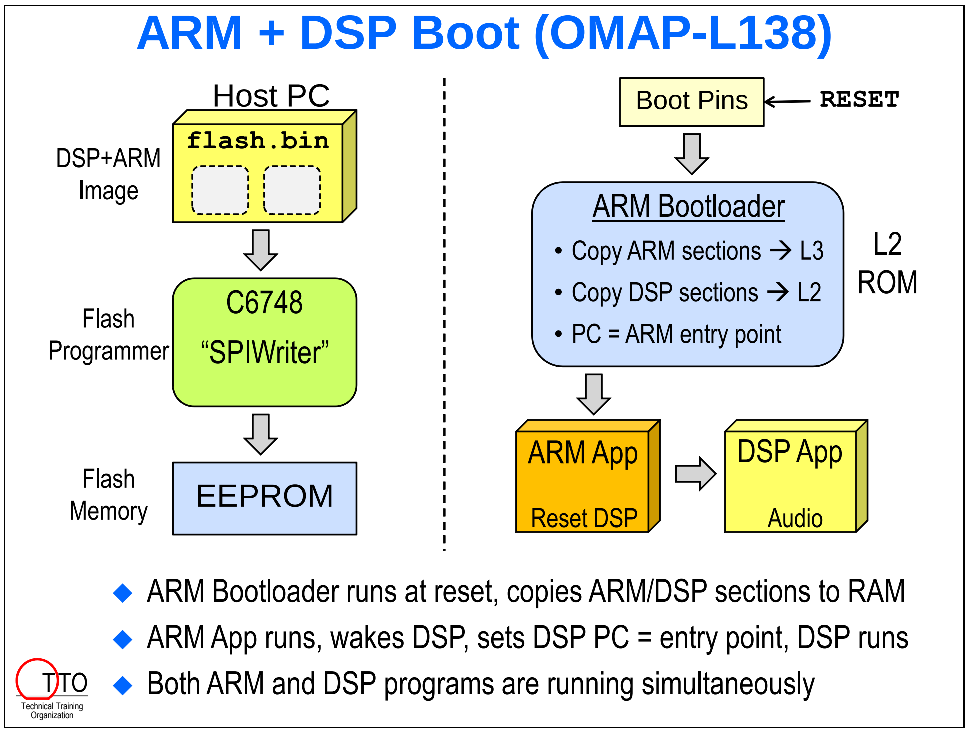
```
Starting OMAP-L138 SPIWriter.
Will you be writing a UBL image? (Y or y)
n
Enter the application file name (enter 'none' to skip):
C:\BIOSv4\Sols\Lab14a_keystone\Project\Release\flash.bin
        INFO: File read complete.
Doing block erase.      SPI boot preparation was successful!
```

ARM + DSP Boot



ARM + DSP Boot (OMAP-L138)

- ARM code programs DSP's entry point to L2 addr
- DSP linker.cmd file specifies exact entry point for boot
- AISgen combines .out files into one bootable image

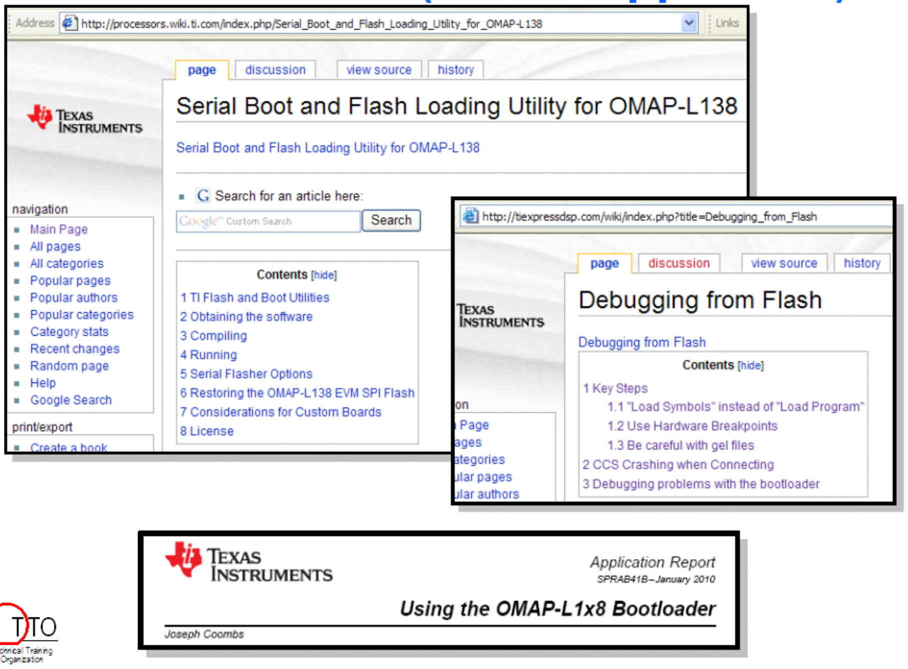


ARM + DSP Boot (OMAP-L138)

- ARM Bootloader runs at reset, copies ARM/DSP sections to RAM
- ARM App runs, wakes DSP, sets DSP PC = entry point, DSP runs
- Both ARM and DSP programs are running simultaneously

Additonal Info…

For Add'l Info…(Wiki & App Notes)



Address http://processors.wiki.ti.com/index.php/Serial_Boot_and_Flash_Loading_Utility_for_OMAP-L138

page | discussion | view source | history

Serial Boot and Flash Loading Utility for OMAP-L138

Serial Boot and Flash Loading Utility for OMAP-L138

Search for an article here:

navigation
- Main Page
- All pages
- All categories
- Popular pages
- Popular authors
- Popular categories
- Category stats
- Recent changes
- Random page
- Help
- Google Search

print/export
- Create a book

Contents [hide]
1 TI Flash and Boot Utilities
2 Obtaining the software
3 Compiling
4 Running
5 Serial Flasher Options
6 Restoring the OMAP-L138 EVM SPI Flash
7 Considerations for Custom Boards
8 License

http://tiexpressdsp.com/wiki/index.php?title=Debugging_from_Flash

page | discussion | view source | history

Debugging from Flash

Debugging from Flash

Contents [hide]
1 Key Steps
 1.1 "Load Symbols" instead of "Load Program"
 1.2 Use Hardware Breakpoints
 1.3 Be careful with gel files
2 CCS Crashing when Connecting
3 Debugging problems with the bootloader

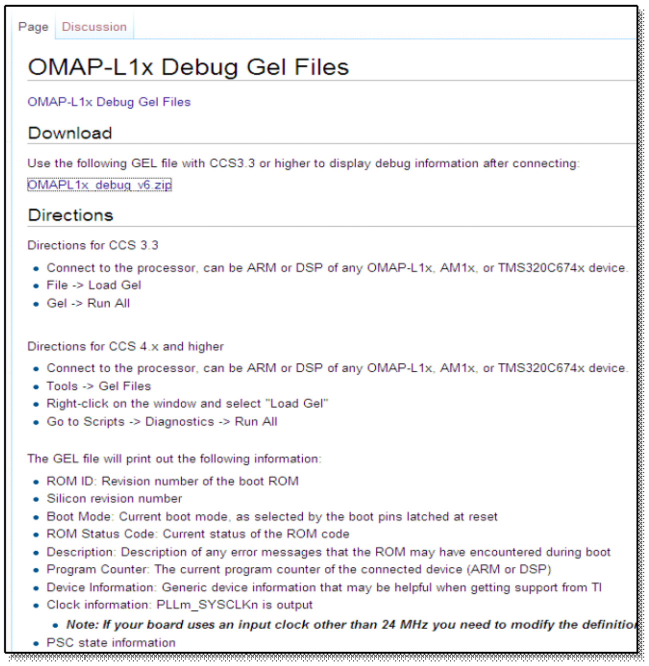
Texas Instruments — Application Report SPRAB41B–January 2010

Using the OMAP-L1x8 Bootloader

Joseph Coombs

TTO Technical Training Organization

OMAP-L1x Debug GEL Files



Page | Discussion

OMAP-L1x Debug Gel Files

OMAP-L1x Debug Gel Files

Download

Use the following GEL file with CCS3.3 or higher to display debug information after connecting:
OMAPL1x_debug_v6.zip

Directions

Directions for CCS 3.3
- Connect to the processor, can be ARM or DSP of any OMAP-L1x, AM1x, or TMS320C674x device.
- File -> Load Gel
- Gel -> Run All

Directions for CCS 4.x and higher
- Connect to the processor, can be ARM or DSP of any OMAP-L1x, AM1x, or TMS320C674x device.
- Tools -> Gel Files
- Right-click on the window and select "Load Gel"
- Go to Scripts -> Diagnostics -> Run All

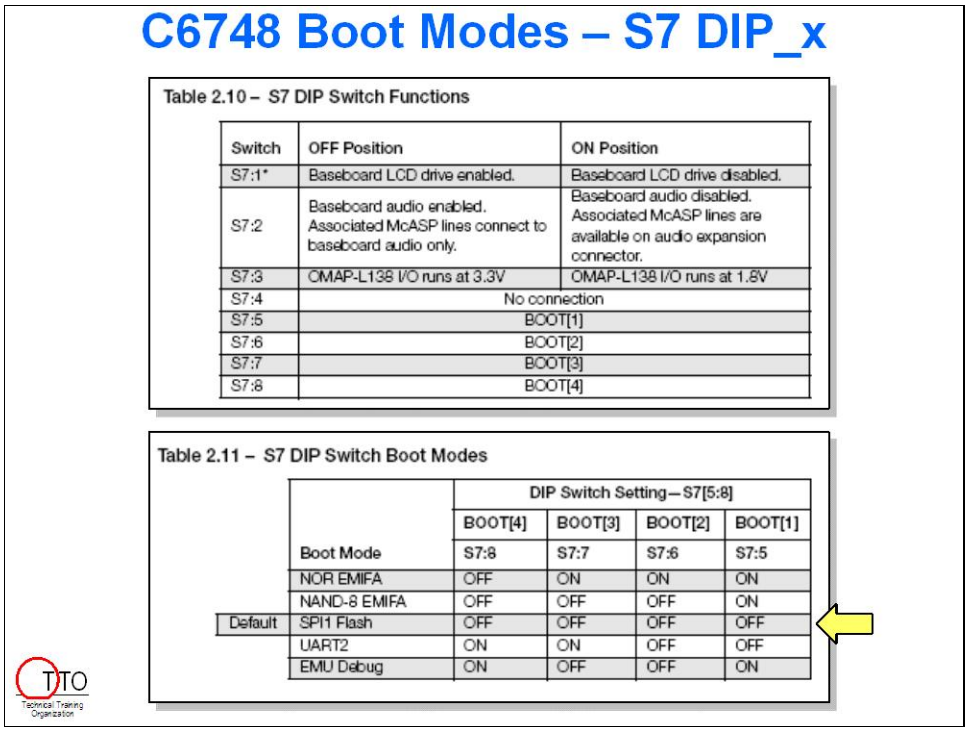
The GEL file will print out the following information:
- ROM ID: Revision number of the boot ROM
- Silicon revision number
- Boot Mode: Current boot mode, as selected by the boot pins latched at reset
- ROM Status Code: Current status of the ROM code
- Description: Description of any error messages that the ROM may have encountered during boot
- Program Counter: The current program counter of the connected device (ARM 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 definitio*
- PSC state information

Debug THIS !

- ROM ID
- Si Revision
- Boot Mode
- ROM Status Code
- Boot ROM Errors
- Current PC
- Device Info
- Clock Info
- PSC States

Outputs results to the Console Window

C6748 Boot Modes (S7, DIP_x)



C6748 Boot Modes – S7 DIP_x

Table 2.10 – S7 DIP Switch Functions

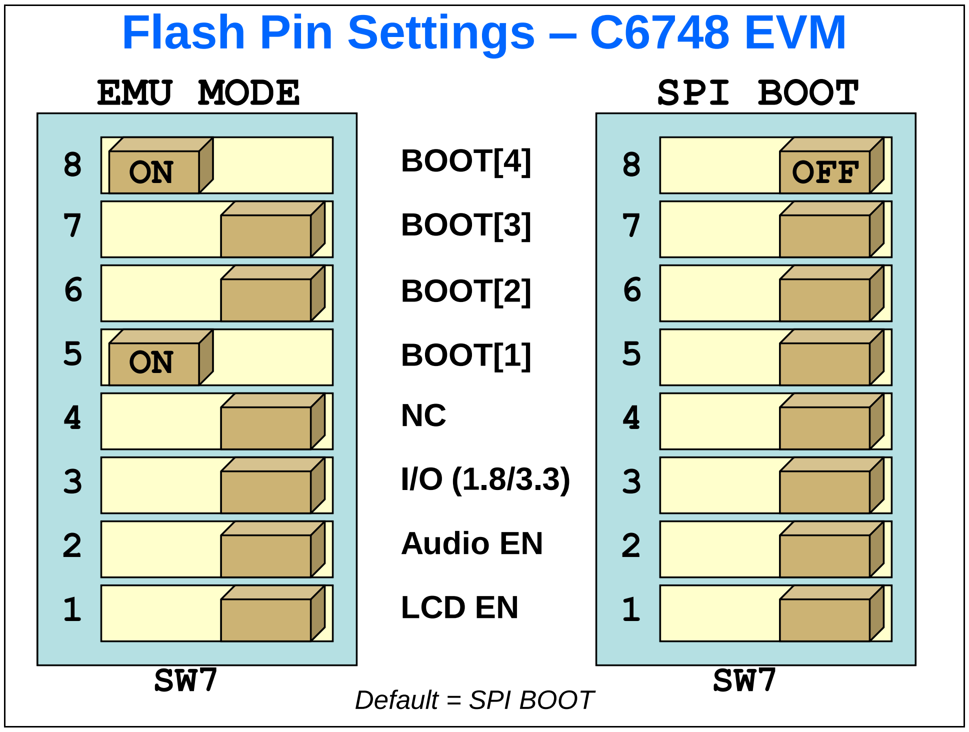
Switch	OFF Position	ON Position
S7:1*	Baseboard LCD drive enabled.	Baseboard LCD drive disabled.
S7:2	Baseboard audio enabled. Associated McASP lines connect to baseboard audio only.	Baseboard audio disabled. Associated McASP lines are available on audio expansion connector.
S7:3	OMAP-L138 I/O runs at 3.3V	OMAP-L138 I/O runs at 1.8V
S7:4	No connection	
S7:5	BOOT[1]	
S7:6	BOOT[2]	
S7:7	BOOT[3]	
S7:8	BOOT[4]	

Table 2.11 – S7 DIP Switch Boot Modes

		DIP Switch Setting – S7[5:8]			
		BOOT[4]	BOOT[3]	BOOT[2]	BOOT[1]
	Boot Mode	S7:8	S7:7	S7:6	S7:5
	NOR EMIFA	OFF	ON	ON	ON
	NAND-8 EMIFA	OFF	OFF	OFF	ON
Default	SPI1 Flash	OFF	OFF	OFF	OFF
	UART2	ON	ON	OFF	OFF
	EMU Debug	ON	OFF	OFF	ON

TTO Technical Training Organization

Flash Pin Settings – C6748 EVM



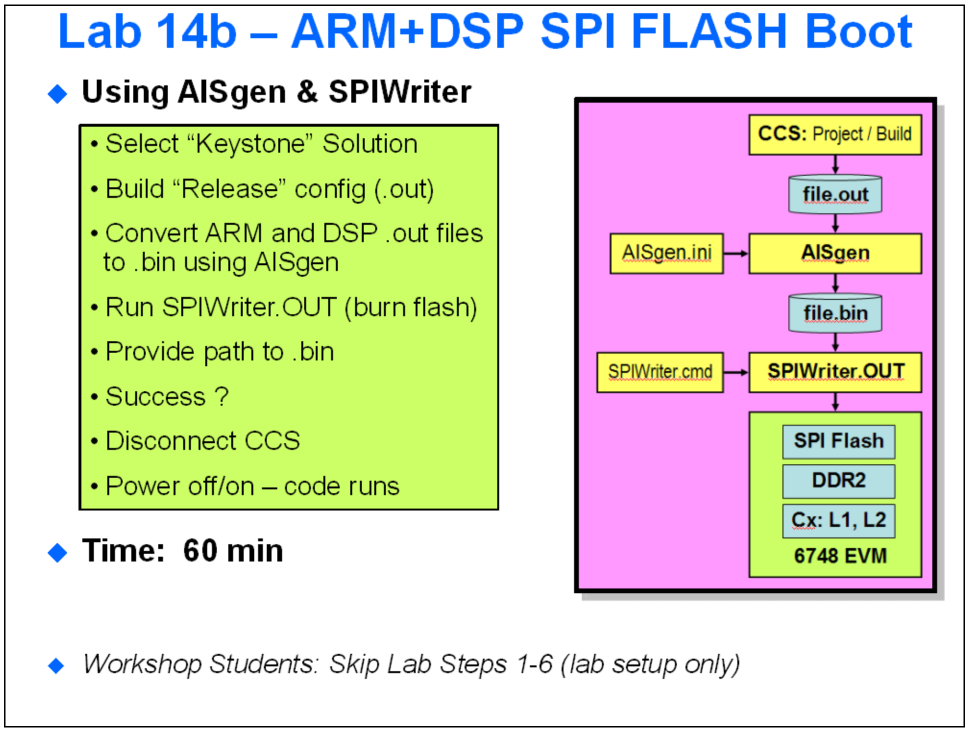
Default = SPI BOOT

*** this page was accidentally created by a virus – please ignore ***

Lab 14b: Booting From Flash

In this lab, a .out file will be loaded to the on-board flash memory so that the program may be run when the board is powered up, with no connection to CCS.

Any lab solution would work for this lab, but again we'll standardize on the "keystone" lab so that we ensure a known quantity.



Lab 14b – ARM+DSP SPI FLASH Boot

- Using AISgen & SPIWriter
  - Select "Keystone" Solution
  - Build "Release" config (.out)
  - Convert ARM and DSP .out files to .bin using AISgen
  - Run SPIWriter.OUT (burn flash)
  - Provide path to .bin
  - Success ?
  - Disconnect CCS
  - Power off/on – code runs
- Time: 60 min
- *Workshop Students: Skip Lab Steps 1-6 (lab setup only)*

CCS: Project / Build
file.out
AISgen.ini
AISgen
file.bin
SPIWriter.cmd
SPIWriter.OUT
SPI Flash
DDR2
Cx: L1, L2
6748 EVM

Lab14b – Booting From Flash - Procedure

Hint: This lab procedure will work with either the C6748 SOM or OMAP-L138 SOM. The basic procedure is the same but a few steps are VERY different. These will be noted clearly in this document. So, please pay attention to the HINTS and grey boxes like this one along the way.

Tools Download and Setup (Students: SKIP STEPS 1-6 !!)

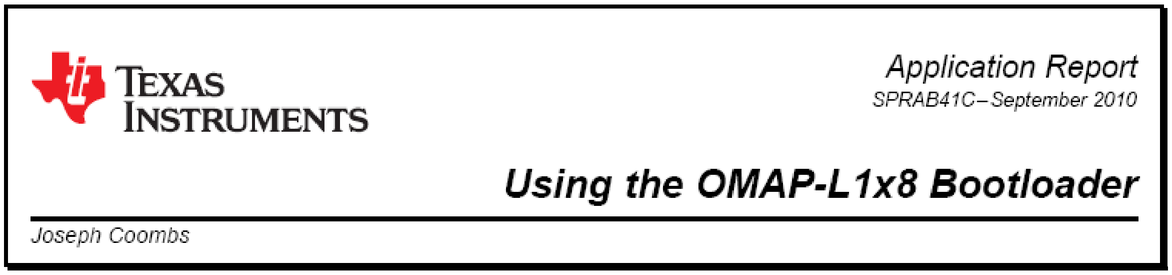
The following steps in THIS SECTION ONLY have already been performed. So, workshop attendees can skip to the next section. These steps are provided in order to show exactly where and how the flash/boot environment was set up (for future reference).

1. **Download AISgen utility – SPRAB41c.**

   Download the pdf file from here:

   http://focus.ti.com/dsp/docs/litabsmultiplefilelist.tsp?docCategoryId=1&familyId=1621&literatureNumber=sprab41c&sectionId=3&tabId=409

   A screen cap of the pdf file is here:



Texas Instruments

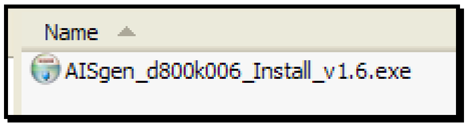
Application Report

SPRAB41C–September 2010

Using the OMAP-L1x8 Bootloader

Joseph Coombs

The contents of this zip are shown here:



Name

AISgen_d800k006_Install_v1.6.exe

2. **Create directories to hold tools and projects.**

   Three directories need to be created:

   - `C:\BIOSv4\Labs\Lab14b_keystone` – will contain the audio project (keystone) to build into a `.OUT` file.
   - `C:\BIOSv4\Labs\Lab14b_ARM_Boot – will contain the ARM boot code required to start up the DSP after booting.`
   - `C:\BIOSv4\Labs\Lab14b_SPIWriter` – will contain the `SPIWriter.out` file used to program the flash on the EVM.
   - `C:\BIOSv4\Labs\Lab14b_AIS` – contains the `AISgen.exe` file (shown above) and is where the resulting AIS script (bin) will be located after running the utility (`.OUT → .BIN`)

   Place the "*keystone*" files into the `\Lab14b_keystone\Files` directory. Users will build a new project to get their `.OUT` file.

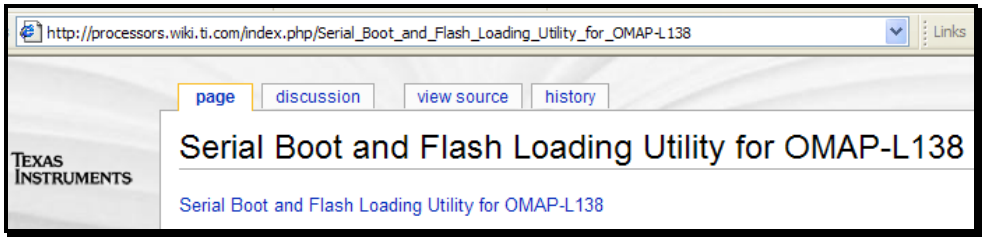
   Place the recently downloaded `AISgen.exe` file into `\Lab14a_AIS` directory.

3. **Download SPI Flash Utilities.**

   You can find the SPI Flash Utility here:

   http://processors.wiki.ti.com/index.php/Serial_Boot_and_Flash_Loading_Utility_for_OMAP-L138

   This is actually a TI wiki page:

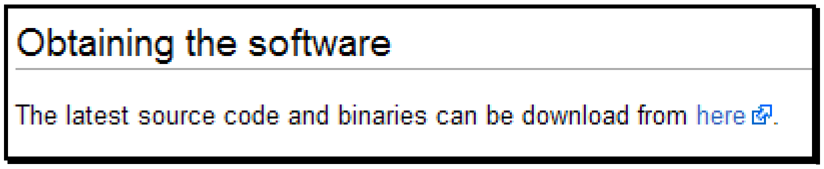


http://processors.wiki.ti.com/index.php/Serial_Boot_and_Flash_Loading_Utility_for_OMAP-L138

Serial Boot and Flash Loading Utility for OMAP-L138

Serial Boot and Flash Loading Utility for OMAP-L138

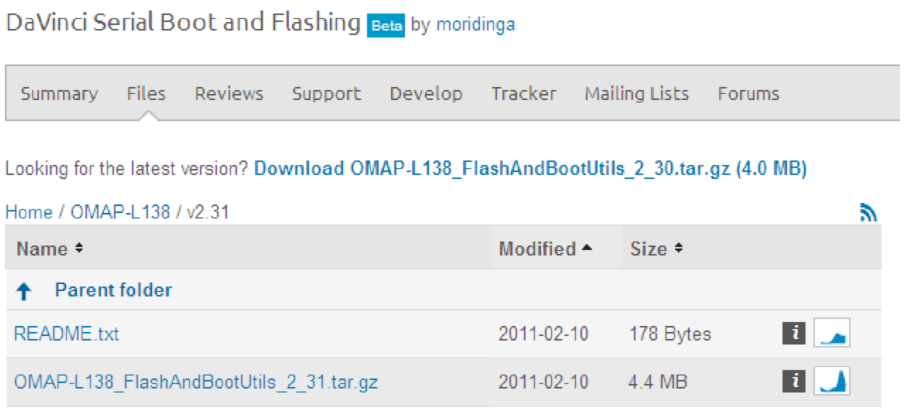
   From here, locate the following and click "here" to go to the download page:



Obtaining the software

The latest source code and binaries can be download from here.

   This will take you to a SourceForge site that will contain the tools you need to download.



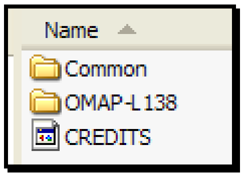
DaVinci Serial Boot and Flashing Beta by moridinga

Looking for the latest version? Download OMAP-L138_FlashAndBootUtils_2_30.tar.gz (4.0 MB)

Home / OMAP-L138 / v2.31

Name	Modified	Size
Parent folder		
README.txt	2011-02-10	178 Bytes
OMAP-L138_FlashAndBootUtils_2_31.tar.gz	2011-02-10	4.4 MB

   Click on the latest version under OMAP-L138 and download the tar.gz file. UnTAR the contents and you'll see this:

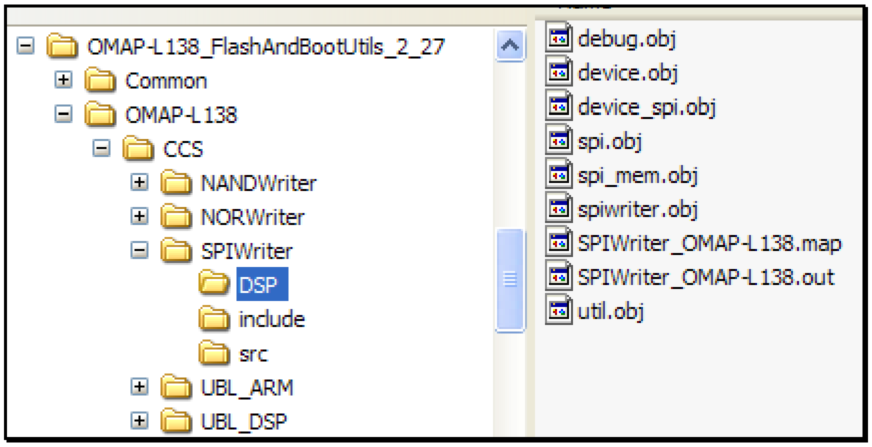


Name
Common
OMAP-L138
CREDITS

   The path we need is `\OMAP-L138`. If we dive down a bit, we will find the `SPIWriter.out` file that is used to program the flash with our boot image (`.bin`).

4. **Copy the SPIWriter.out file to** `\Lab14b_SPIWriter\` **directory.**

   Shown below is the initial contents of the Flash Utility download:



   Copy the following file to the `\Lab14b_SPIWriter\` directory:

   `SPIWriter_OMAP-L138.out`

5. **Install AISgen.**

   Find the download of the AISgen.exe file and double-click it to install. After installation, copy a shortcut to the desktop for this program:

6. **Create the keystone project.**

   Create a new CCSv4 BIOS project with the source files listed in `C:\BIOSv4\Lab14b_keystone\Files`. Create this project in the neighboring `\Project` folder. Also, don't forget to add the BSL library and BSL includes (as normal) Make sure you use the RELEASE configuration only.

Hint: [workshop students: START HERE]

Build Keystone Project: [Src → .OUT File]

7. **Import keystone audio project and make a few changes.**

   Import "*keystone_flash*" project from the following directory:

   ```
   C:\BIOSv4\Labs\Lab14b_keystone\Project
   ```

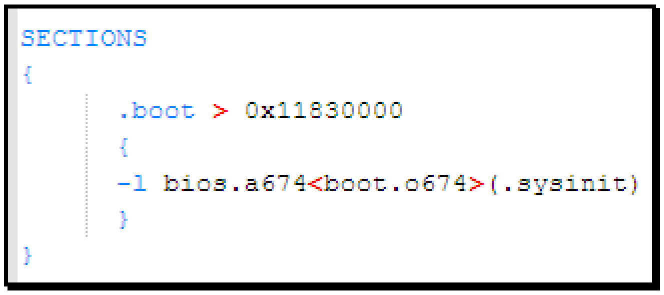
   This project was built for emulation with CCSv4 – i.e there is a GEL file that sets up our PLL, DDR2, etc. In creating a boot image, as discussed in the chapter, we have to perform these actions in code vs. the GEL creating this nice environment for us.

   So, we have a choice here – write code that runs in main to set up PLL0, PLL1, DDR, etc. OR have the bootloader do it FOR US. Having the bootloader perform these actions offers several advantages – fewer mistakes by human programmers AND, these settings are done at bootload time vs waiting all the way until main() for the settings to take effect.

Hint: The following step is for OMAP-L138 SOM Users ONLY !!

8. **View the c_int00_locater_cmd file (OMAP-L138 ARM+DSP only).**

   Here is one of the "tricks" that must be employed when using both the ARM and DSP. The ARM code has to know the entry point (reset vector, c_int00) of the DSP. Well, if you just compile and link, it could go anywhere in L2. So, this little command file specifies EXACTLY where the .boot section should go for a BIOS project (this is not necessary for a non-BIOS program).



```
SECTIONS
{
        .boot > 0x11830000
        {
        -l bios.a674<boot.o674>(.sysinit)
        }
}
```

9. **Build the keystone project.**

   Using the RELEASE build configuration, build the project. This should create the .OUT file. Go check the \Release directory and locate the .OUT file:

   ```
   keystone_flash.out
   ```

   Load the .OUT file and make sure it executes properly. We don't want to flash something that isn't working. ☺

   Do not close the Debug session yet.

10. **Determine silicon rev of the device you are currently using.**

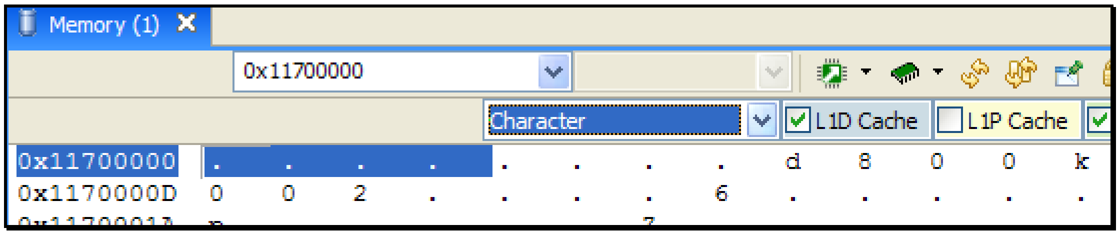
    AISgen will want to know which silicon rev you are using. Well, you can either attempt to read it off the device itself (which is nearly impossible) or you can visit a convenient place in memory to see it.

    Now that you have the Debug perspective open, this should be relatively straightforward. Open a memory view window and type in the following address:

    ```
    0x11700000
    ```

    Can you see it? No? Shame on you. Ok. Try changing the style view to "*Character*" instead. See something different?

    Like this?



```
0x11700000  .  .  .  .  .  .  .  .  d  8  0  0  k
0x1170000D  0  0  2  .  .  .  .  6  .  .  .  .  .
```

That says "d800k002" which means rev2 of the silicon. That's an older rev…but whatever yours is…write it down below:

```
Silicon REV:  ________________________
```

FYI – for OMAP-L138 (and C6748), note the following:

- d800k002 = Rev 1.0 silicon (common, but old)
- d800k004 = Rev 1.5 silicon (not found very often)
- d800k006 = Rev 2.0 silicon (if you have a newer board, this is the latest)

There ARE some differences between Rev1 and Rev2 silicon that we'll mention later in this lab – very important in terms of how the ARM code is written.

You will probably NEVER need to change the memory view to "Character" ever again – so enjoy the moment. ☺

Next, we need to convert this .out file and combine it with the ARM .out file and create a single flash image for both using the AIS script via AISgen…

11. **Use the Debug GEL script to locate the Silicon Rev.**

    This script can be run at any time to debug the state of your silicon and all of the important registers and frequencies your device is running at. This file works for both OMAP-L137/8 and C6747/8 devices. It is a great script to provide feedback for your hardware engineer.

    It goes kind of like this: we want a certain frequency for PLL1. We read the documentation and determine that these registers need to be programmed to a, b and c. You write the code, program them and then build/run. Well, is PLL1 set to the frequency you thought it should be? Run the debug script and find out what the processor is "reporting" the setting is. Nice.

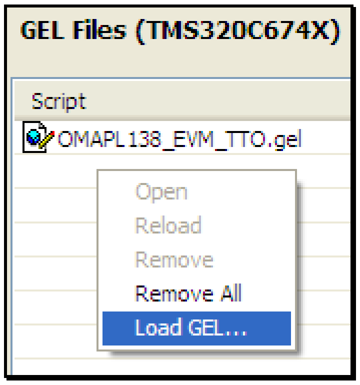
    This script outputs its results to the Console window.

    Let's use the debug script to determine the silicon rev as in the previous step.

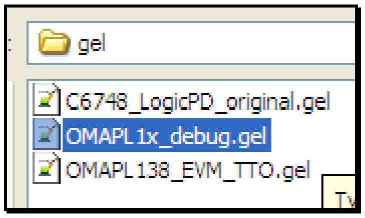
    First, we need to LOAD the gel file. This file can be downloaded from the wiki shown in the chapter. We have already done that for you and placed that GEL file in the \gel directory next to the GEL file you've been using for CCS.

    Select Tools → GEL Files.

    Right-click in the empty area under the currently loaded GEL file and select: Load Gel.



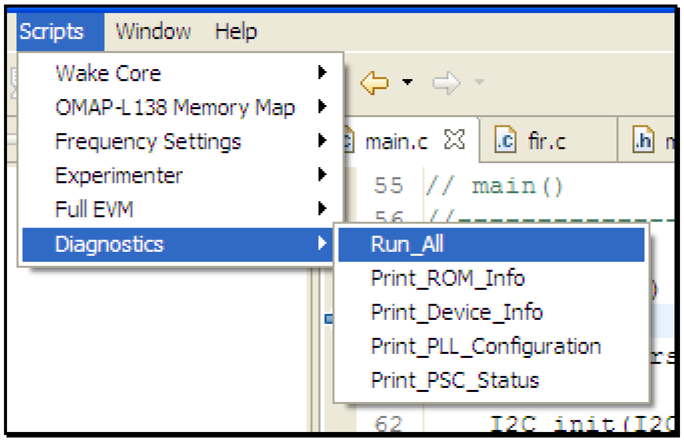
    The \gel directory should show up and the file `OMAPL1x_debug.gel` should be listed. If not, browse to `C:\BIOSv4\Labs\evmc6748_v1-1\gel`.



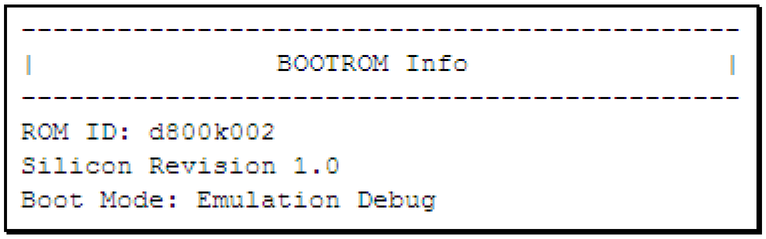
    Click Open.

This will load the new GEL file and place the scripts under the “Scripts” menu.

Select “Scripts” → Diagnostics → Run All:



You can choose to run only a specific script or “All” of them. Notice the output in the Console window. Scroll up and find the silicon revision. Also make note of all of the registers and settings this GEL file reports. Quite extensive.



```
---------------------------------------------
|               BOOTROM Info                |
---------------------------------------------
ROM ID: d800k002
Silicon Revision 1.0
Boot Mode: Emulation Debug
```

Does your report show the same rev as you found in the previous step? Let’s hope so…

Write down the Si Rev again here:

Silicon Rev (again): ____________________

Use AISgen To Convert [.OUT → .BIN]

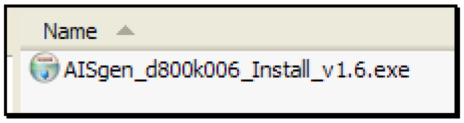
AISgen (*Application Image Script Generator*) is a free downloadable tool from TI – check out the beginning of this lab for the links to get this tool.

12. Locate AISgen.exe (only if requiring installation…if not, see next step).

The installation file has already been downloaded for you and is sitting in the following directory:

```
C:\BIOSv4\Labs\Lab14b_AIS
```

Here, you will find the following install file:



Name

AISgen_d800k006_Install_v1.6.exe

This is the INSTALL file (fyi). You don't need to use this if the tool is already installed on your computer…

13. Run AISgen.

There should be an icon on your desktop that looks like this:

If not, you will need to install the tool by double-clicking on the install file, installing it and then creating a shortcut to it on the desktop (you'll find it in *Programs* → *Texas Instruments* → *AISgen*).

Double-click on the icon to launch AISgen and fill out the dialogue box as shown on the next page…there are several settings you need…so be careful and go SLOWLY here…

It is usually BEST to place all of your PLL and DDR settings in the flash image and have the bootloader set these up vs. running code on the DSP to do it. Why? Because the DSP then comes out of reset READY to go at the top speeds vs. running "slow" until your code in main() is run. So, that's what we plan to do….

Note: Each dialogue has its own section below. It is quite a bit of setup…but hey, you are enabling the bootloader to set up your entire system. This is good stuff…but it takes some work…

Hint: When you actually use the DSP to burn the flash in a later step, the location you store your .bin file too (name of the .bin file AND the directory path you place the .bin file in) CANNOT have ANY SPACES IN THE PATH OR FILENAME.

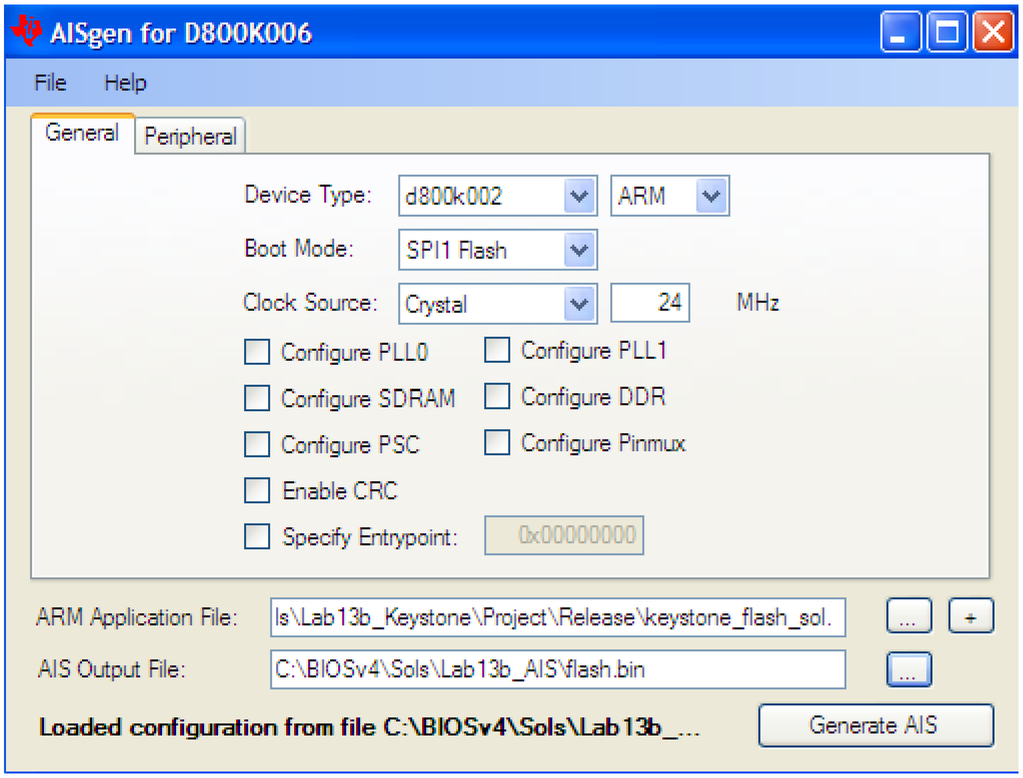
Main dialogue – basic settings.

Fill out the following on this page:

- Device Type (match it up with what you determined before)
- For OMAP-L138 SOM (ARM + DSP), choose “ARM”. If you’re using the 6748 SOM, choose “DSP”.
- Boot Mode: SPI1 Flash. On the OMAP-L138, the SPI1 port and UART2 ports are connected to the flash.
- For now, wait on filling in the Application and Output files.

Hint: For C6748 SOM, choose “DSP” as the Device type

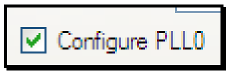
Hint: For OMAP-L138 SOM, choose “ARM” as the Device type



Device Type: d800k002 ARM
Boot Mode: SPI1 Flash
Clock Source: Crystal 24 MHz
☐ Configure PLL0 ☐ Configure PLL1
☐ Configure SDRAM ☐ Configure DDR
☐ Configure PSC ☐ Configure Pinmux
☐ Enable CRC
☐ Specify Entrypoint: 0x00000000
ARM Application File: ls\Lab13b_Keystone\Project\Release\keystone_flash_sol.
AIS Output File: C:\BIOSv4\Sols\Lab13b_AIS\flash.bin
Loaded configuration from file C:\BIOSv4\Sols\Lab13b_...

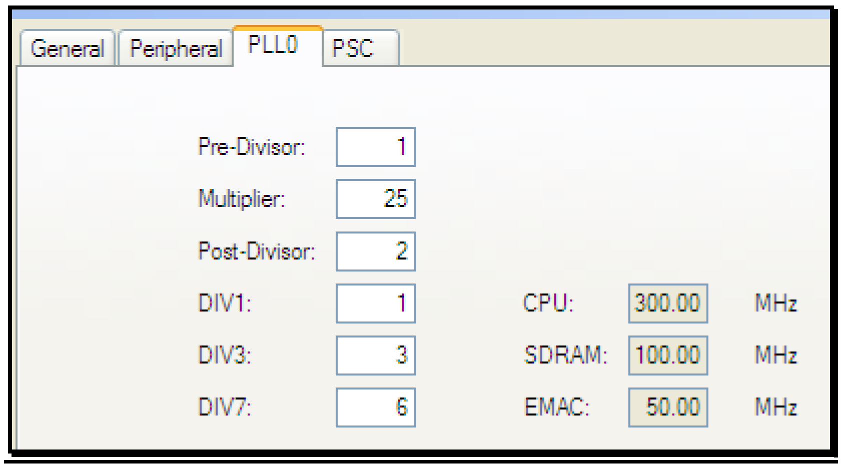
Configure PLL0, PLL0 Tab

On the "General" tab, check the box for "Configure PLL0" as shown:


✓ Configure PLL0

Then click on the PLL0 tab and view these settings. You will see the defaults show up. Make the following modifications as shown below.

Change the multiplier value from 20 to 25 and notice the values in the bottom RH corner change.

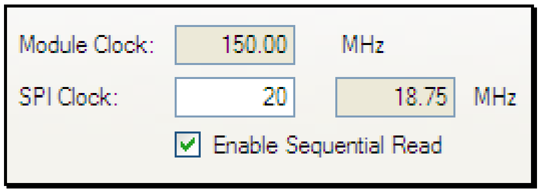


Pre-Divisor:	1			
Multiplier:	25			
Post-Divisor:	2			
DIV1:	1	CPU:	300.00	MHz
DIV3:	3	SDRAM:	100.00	MHz
DIV7:	6	EMAC:	50.00	MHz

Peripheral Tab

Next, click on the Peripheral tab. This is where you will set the SPI Clock. It is a function (divide down) from the CPU clock. If you leave it at 1MHz, well, it will work, but the bootload will take WAY longer. So, this is a "speed up" enhancement.

Type "20" into the SPI Clock field as shown:



Module Clock:	150.00	MHz	
SPI Clock:	20	18.75	MHz

✓ Enable Sequential Read

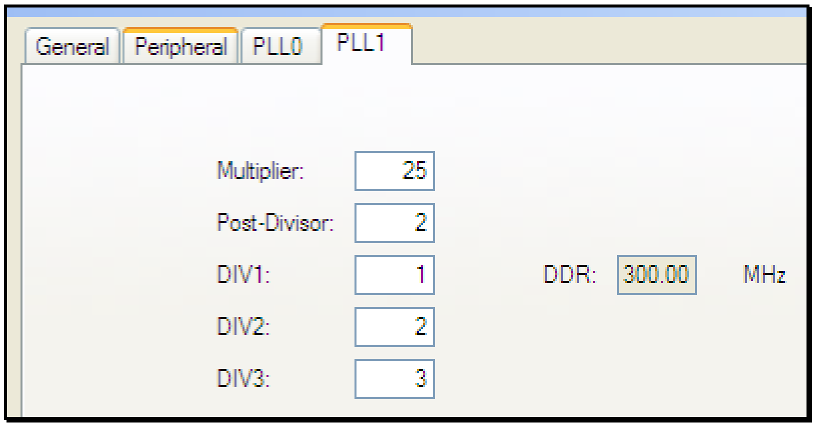
Also check the "Enable Sequential Read" checkbox. Why is this important? Speed of the boot load. If this box is unchecked, the ROM code will send out a read command (0x03) plus a 24-bit address before every single BYTE. That is a TON of read commands.

However, if we CHECK this box, the ROM code will send out a single 24-bit address (0x000000) and then proceed to read out the ENTIRE boot image. WAY WAY faster.

Configure PLL1

Just in case you EVER want to put code or data into the DDR, PLL1 needs to be set in the flash image and therefore configured by the bootloader.

So, click the checkbox next to "Configure PLL1", click on that tab, and use the following settings:

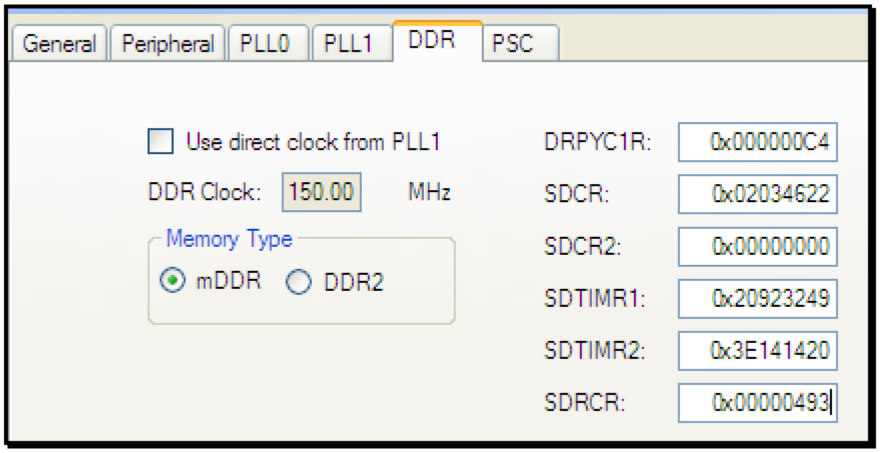


Multiplier: 25
Post-Divisor: 2
DIV1: 1
DIV2: 2
DIV3: 3
DDR: 300.00 MHz

This will clock the DDR at 300MHz. This is equivalent to what our GEL file sets the DDR frequency to. We don't have any code in DDR at the moment – but now we have it setup just in case we ever do later on. Now, we need to write values to the DDR config registers…

Configure DDR

You know the drill. Click the proper checkbox on the main dialogue page and click on the DDR tab. Fill in the following values as shown. If you want to know what each of the values are on the right, look it up in the datasheet. ☺



☐ Use direct clock from PLL1
DDR Clock: 150.00 MHz
Memory Type: [✓ mDDR] DDR2

DRPYC1R: 0x000000C4
SDCR: 0x02034622
SDCR2: 0x00000000
SDTIMR1: 0x20923249
SDTIMR2: 0x3E141420
SDRCR: 0x00000493

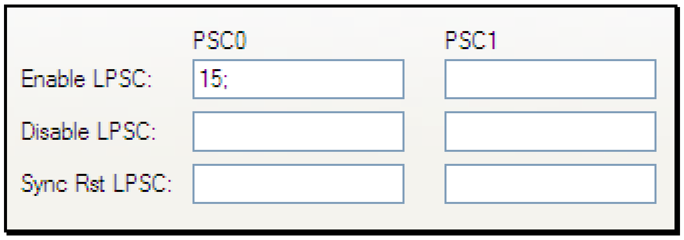
Configure PSC0, PSC0 Tab

Next, we need to configure the Low Power Sleep Controller (LPSC) to allow the ARM to write to the DSP's L2 memory. If both the ARM and DSP code resided in L3, well, the ARM bootloader could then easily write to L3. But, with a BIOS program, BIOS wants to live in L2 DSP memory (around 0x11800000). In order for the ARM bootloader code to write to this address, we need to have the DSP clocks powered up. Enabling PSC0 does this for us.

On the main page, "check" the box next to "Configure PSC" and go to the PSC tab.

In the GEL file we've been using in the workshop, a function named `PSC_All_On_Full_EVM()` runs to set all the PSC values. We could cheat and just type in "15" as shown below:

Minimum Setting (don't use this for the lab):

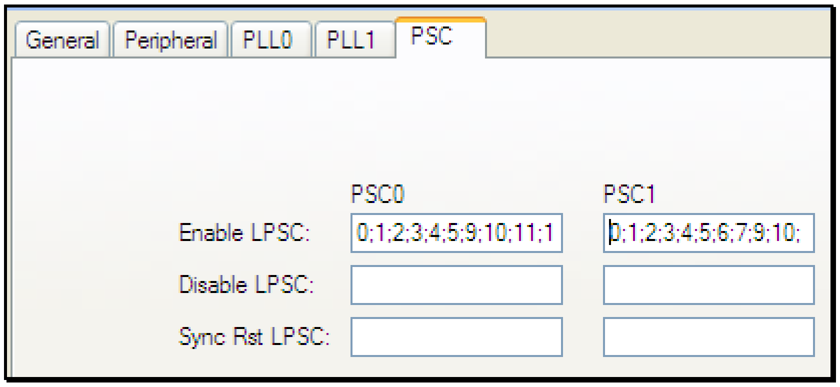


	PSC0	PSC1
Enable LPSC:	15;	
Disable LPSC:		
Sync Rst LPSC:		

This would Enable module 15 of the PSC which says "de-assert the reset on the DSP megamodule" and enable the clocks so that the ARM can write to the DSP memory located in L2. However, this setting does NOT match what the GEL file did for us. So, we need to enable MORE of the PSC modules so that we match the GEL file.

Note: When doing this for your own system, you'll need to pick and choose the PSC modules that are important to your specific system.

Better Setting (USE THIS ONE for the lab – or as a starting point for your own system)



	PSC0	PSC1
Enable LPSC:	0;1;2;3;4;5;9;10;11;1	0;1;2;3;4;5;6;7;9;10;
Disable LPSC:		
Sync Rst LPSC:		

The numbers scroll out of sight, so here are the values:

PSC0: 0;1;2;3;4;5;9;10;11;12;13;15

PSC1: 0;1;2;3;4;5;6;7;9;10;11;12;13;14;15;16;17;18;19;20;21;24;25;26;27;28;29;30;31

Note: Note: PSC1 is MISSING modules 8, 22-23 (see datasheet for more details on these).

Notice for SATA users:

PSC1 Module 8 (SATA) is specifically NOT being enabled. There is a note in the System Reference Guide saying that you need to set the FORCE bit in MDCTL when enabling SATA. That's not an option in the GUI/bootROM so we simply cannot enable it. If you ignore the author's advice and enable module 8 in PSC1, you'll find the boot ROM gets stuck in a spin loop waiting for SATA to transition and so ultimately your boot fails as a result.

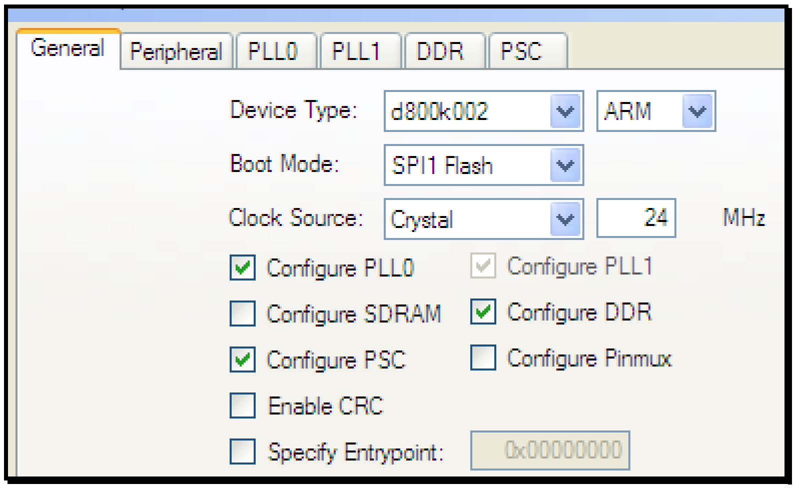
So, there are really two pieces to this puzzle if using SATA:

A. Make sure you do NOT try to enable PSC1 Module 8 through AISgen

B. If you need SATA, make sure you enable this through your application code and be sure to set the FORCE bit in MDCTL when doing so.

Configure PSC0, PSC0 Tab

So, your final main dialogue should look like this with all of these tabs showing. Please double-check you didn't forget something:



Device Type: d800k002 ARM
Boot Mode: SPI1 Flash
Clock Source: Crystal 24 MHz

- [x] Configure PLL0
- [x] Configure PLL1
- [ ] Configure SDRAM
- [x] Configure DDR
- [x] Configure PSC
- [ ] Configure Pinmux
- [ ] Enable CRC
- [ ] Specify Entrypoint: 0x00000000

Save your `.cfg` file in the `\Lab14b_AIS` folder for potential use later on – you don't want to have to re-create all of these steps again if you can avoid it. If you look in that folder, it already contains this .cfg file done for you. Ok, so we could have told you that earlier, but then the learning would have been crippled.

The author named the solution's config file:

```
OMAP-L138-ARM-DSP-LAB14B_TTO.cfg
```

Hint: C6748 Users: You will only specify ONE output file (DSP.out)

Hint: OMAP-L138 Users: You will specify TWO files (an ARM.out and a DSP.out).

ARM/DSP Application & Output Files

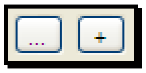
Ok, we're almost done with the AISgen settings.

Hint: 6748 SOM Users – follow THESE directions (OMAP Users can skip this part)

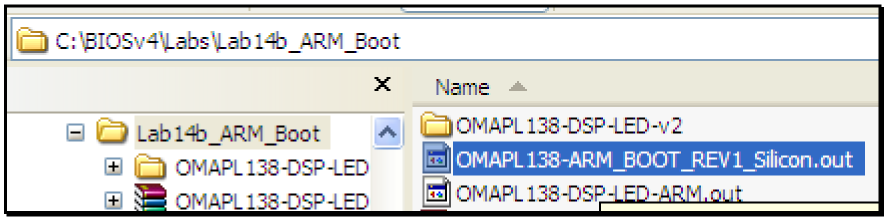
For the "DSP Application File", browse to the .OUT file that was created when you built your keystone project: `keystone_flash.out`

Hint: OMAP-L138 SOM Users – follow THESE directions:

For OMAP-L138 users: you will enter the paths to *both* files and AISgen will combine them into ONE image (.bin) to burn into the flash. You must FIRST specify the ARM.out file followed by the DSP.out file – this order MATTERS.



Click the "…" and browse to the ARM code – located at:


C:\BIOSv4\Labs\Lab14b_ARM_Boot

This ARM code is for rev1 silicon. It should also work on Rev2 silicon – but not tested.

Next, click on the "+" sign and browse to your `keystone_flash.out` file you built earlier. You should now have two .out files listed under "ARM Application File" – first the ARM.out, then the DSP.out files separated by a semicolon. Double-check this is the case.

Hint: ALL SOM Users – Follow THIS STEP…

For the Output file, name it "flash.bin" and use the following path:

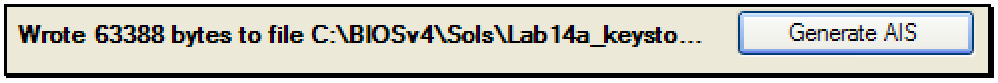
```
C:\BIOSv4\Labs\Lab14b_AIS\flash.bin
```

Hint: Again, the path and filename CANNOT contain any spaces. When you run the flash writer later on, that program will barf on the file if there are any spaces in the path or filename.

Before you click the "*Generate AIS*" button, notice the other configuration options you have here. If you wanted AIS to write the code to configure any of these options, simply check them and fill out the info on the proper tab. This is a WAY cool interface. And, the bootloader does "system" setup for you instead of writing code to do it – and making mistakes and debugging those mistakes…and getting frustrated…like getting tired of reading this rambling text from the author….

14. **Generate AIS script (flash.bin).**

    Click the "Generate AIS" button. When complete, it will provide a little feedback as to how many bytes were written. Like this:



Wrote 63388 bytes to file C:\BIOSv4\Sols\Lab14a_keysto…

So, what did you just do?

For OMAP-L138 (ARM+DSP) users, you just combined the ARM.out and DSP.out files into one flash image – flash.bin. For C6748 Users, you simply converted your .out file to a flash image.

The next step is to burn the flash with this image and then let the bootloader do its thing…

Program the Flash: [.BIN → SPI1 Flash]

15. **Check target config and pin settings.**

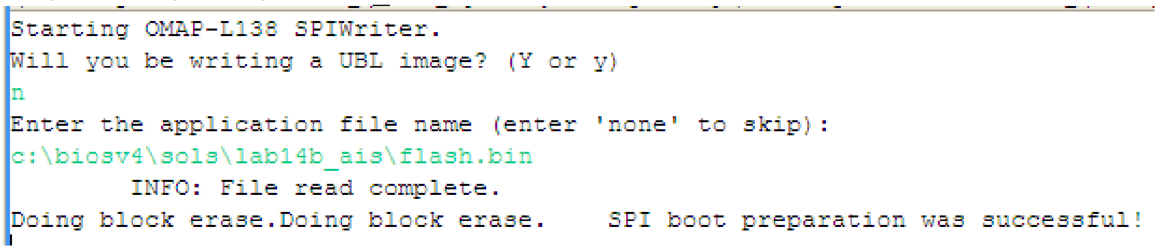
    Use the standard XDS510 Target Config file that uses one GEL file (like all the other labs in this workshop). Make sure it is the default.

    Also, make sure pins 5 and 8 on the EVM (S7 – switch 7) are ON/UP – so that we are in EMU mode – NOT flash boot mode.

16. **Load SPIWriter.out into CCS.**

    The SPIWriter.out file should already be copied into a convenient place:

    `C:\BIOSv4\Labs\Lab14b_SPIWriter`



```
Starting OMAP-L138 SPIWriter.
Will you be writing a UBL image? (Y or y)
n
Enter the application file name (enter 'none' to skip):
c:\biosv4\sols\lab14b_ais\flash.bin
        INFO: File read complete.
Doing block erase.Doing block erase.    SPI boot preparation was successful!
```

In CCS,

- Launch TI Debugger
- Connect to target
- Select "Load program" and browse to this location:

  `C:\BIOSv4\Labs\Lab14b_SPIWriter\SPIWriter_OMAP-L138.out`

17. PLAY !

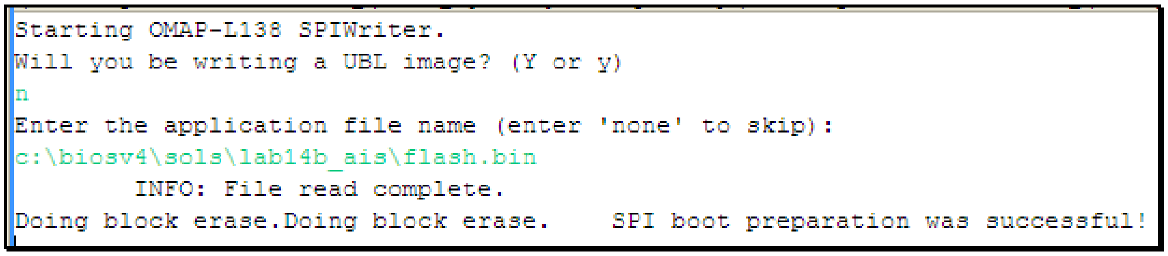
Click Play. The console window will pop up and ask you a question about whether this is a UBL image. The answer is NO. Only if you were using a TI UBL which would then boot Uboot, the answer is no. This assumes that Linux is running. Our ARM code has no O/S.

Type a smallcase "n" and hit [*ENTER*]. To respond to the next question, provide the path name for your `.BIN` file (`flash.bin`) created in a previous step, i.e.:

```
C:\BIOSv4\Labs\Lab14b_AIS\flash.bin
```

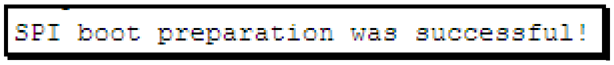
Hint: Do NOT have any spaces in this path name for SPIWriter – it NO WORK that way.

Here's a screen capture from the author (although, you are using the \Labs dir, not \Sols:



```
Starting OMAP-L138 SPIWriter.
Will you be writing a UBL image? (Y or y)
n
Enter the application file name (enter 'none' to skip):
c:\biosv4\sols\lab14b_ais\flash.bin
        INFO: File read complete.
Doing block erase.Doing block erase.    SPI boot preparation was successful!
```

Let it run – shouldn't take too long. 15-20 seconds (with an XDS510 emulator). You will see some progress msgs and then see "success" – like this:



```
SPI boot preparation was successful!
```

18. Terminate the Debug session, close CCS.

19. Ensure DIP switches are set correctly and get music playing, then power-cycle!

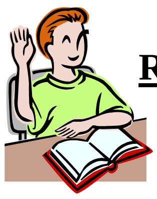
Make sure ALL DIP switches on S7 are DOWN [OFF]. This will place the EVM into the SPI-1 boot mode. Get some music playing. Power cycle the board and THERE IT GOES…

No need to re-flash anything like a POST – just leave your neat little program in there for some unsuspecting person to stumble on one day when they forget to set the DIP switches back to EMU mode and they automagically hear audio coming out of the speakers when the turn on the power. Freaky. You should see the LED blinking as well…great work !!

Hint: DO NOT SKIP THE FOLLOWING STEP.

20. Change the boot mode pins on the EVM back to their original state.

Please ensure DIP_5 and DIP_8 of S7 (the one on the right) are UP [ON].

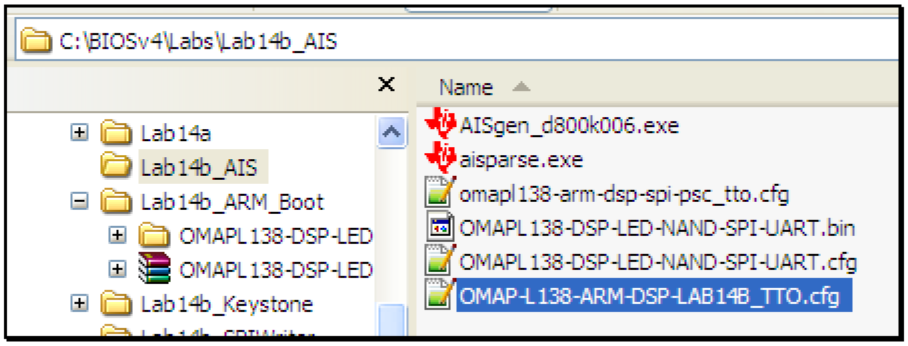


<u>RAISE YOUR HAND</u> and get the instructor's attention when you have completed this lab. If time permits, move on to the next OPTIONAL part…

Optional – DDR Usage

Go back to your keystone project and link the .text section into DDR memory. Re-compile and generate a new .out file. Then, use AISgen to create a new flash.bin file and flash it with SPIWriter. Then reset the board and see if it worked. Did it?

FYI – to make things go quicker, we have a .cfg file pre-loaded for AISgen. It is located at:

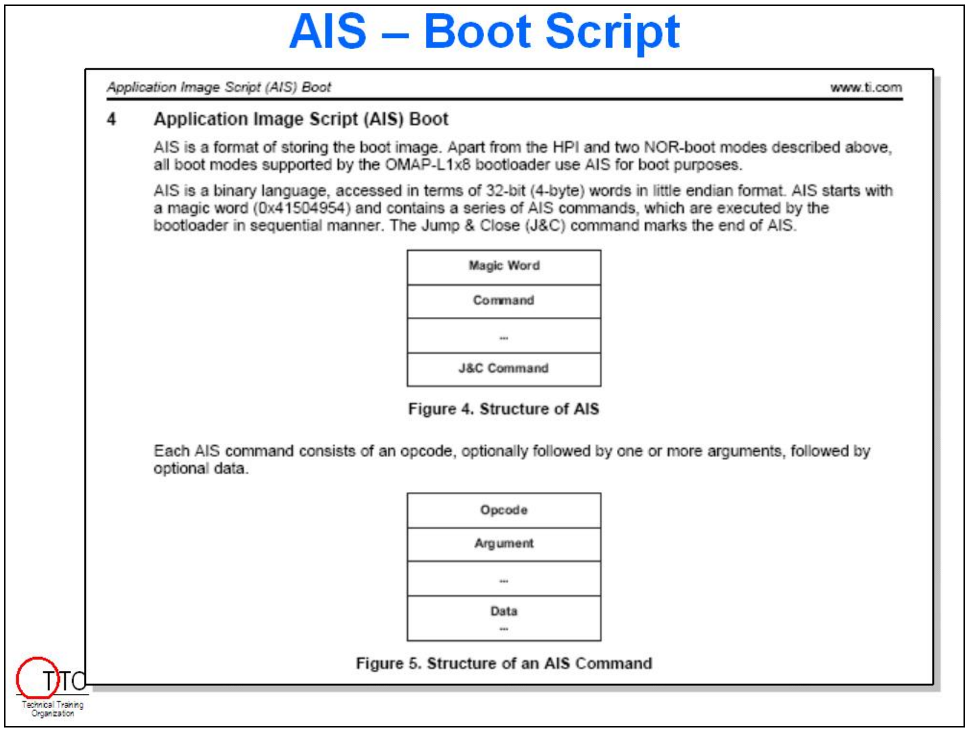

C:\BIOSv4\Labs\Lab14b_AIS

Lab14a
Lab14b_AIS
Lab14b_ARM_Boot
OMAPL138-DSP-LED
OMAPL138-DSP-LED
Lab14b_Keystone

Name
AISgen_d800k006.exe
aisparse.exe
omapl138-arm-dsp-spi-psc_tto.cfg
OMAPL138-DSP-LED-NAND-SPI-UART.bin
OMAPL138-DSP-LED-NAND-SPI-UART.cfg
OMAP-L138-ARM-DSP-LAB14B_TTO.cfg

When running AISgen, you can simply load this config file and it contains ALL of the settings from this lab. Edit, recompile, load this cfg, generate .bin, burn, reset. Quick.

Additional Information



AIS – Boot Script

Application Image Script (AIS) Boot www.ti.com

4 Application Image Script (AIS) Boot

AIS is a format of storing the boot image. Apart from the HPI and two NOR-boot modes described above, all boot modes supported by the OMAP-L1x8 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.

Magic Word
Command
...
J&C Command

Figure 4. Structure of AIS

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

Opcode
Argument
...
Data ...

Figure 5. Structure of an AIS Command

TTO Technical Training Organization