Creating a bootloader for Cortex-M

Technical Note 160822

Targets:
ARM

Component:
General

Updated:
9/19/2016 2:02 PM

Introduction

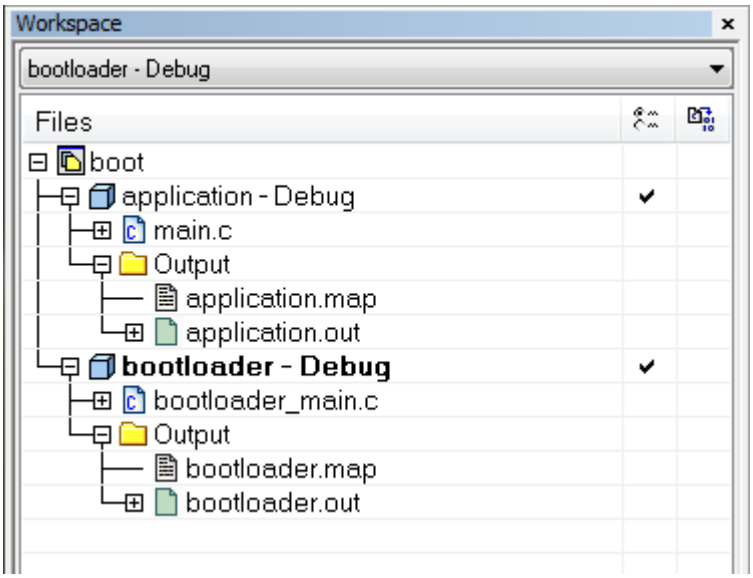
This Technical Note provides guidelines on how to create a bootloader using IAR Embedded Workbench for ARM. Most of the recommendations in the Technical Note are general, although the example project is for a Cortex-M microprocessor (specifically STMicroelectronics STM32L152VB).

Discussion

When creating a bootloader, there are some things to consider regarding project setup, and the execution handover from the bootloader to the application. See the example project for STM32L152VB (/contentassets/ba62fee31d724aa5a67a00da3cea6e3d/boot_example_ewarm_7701.zip) and follow the considerations below. Note that for clarity, this Technical Note does NOT describe how to perform a "live update" (IAP – in application programming) of the application, or how to configure communication interfaces, although this is a common feature in bootloaders.

General considerations

- Create separate projects – one for the bootloader and one for the application. The two projects can be placed in the same workspace. By keeping the projects separate, you make sure that code and library functions are not being shared between the bootloader and the application.



- Check the flash memory block size (the smallest erasable unit of the flash memory). The divide between the bootloader and application must be on a flash block boundary, otherwise you risk erasing parts of the application or bootloader when you replace one of them.

- Any hardware initialization should normally only be performed once, most often by the bootloader. Make sure that the application does not re-initialize the hardware unless that is what you want.

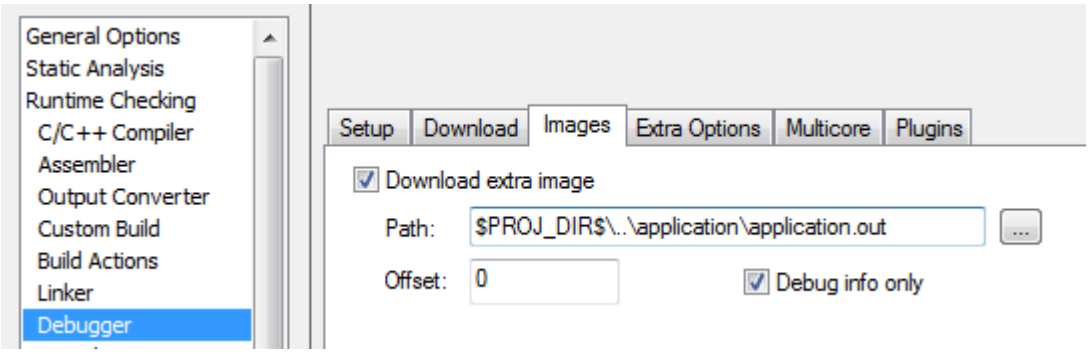
- Before the bootloader hands over execution to the application, make sure to:

  1. Disable interrupts – to avoid unexpected interrupts during application startup.
  2. Configure/reset the stack pointer – so that the application's stack memory is not already used up by the bootloader.
  3. Configure the VTOR register – so that interrupts are handled by the application – not the bootloader.

```
__disable_interrupt();              // 1. Disable interrupts
__set_SP(vector_p->stack_addr);     // 2. Configure stack pointer
SCB->VTOR = (uint32_t) &app_vector; // 3. Configure VTOR
vector_p->func_p();                 // 4. Jump to application
```

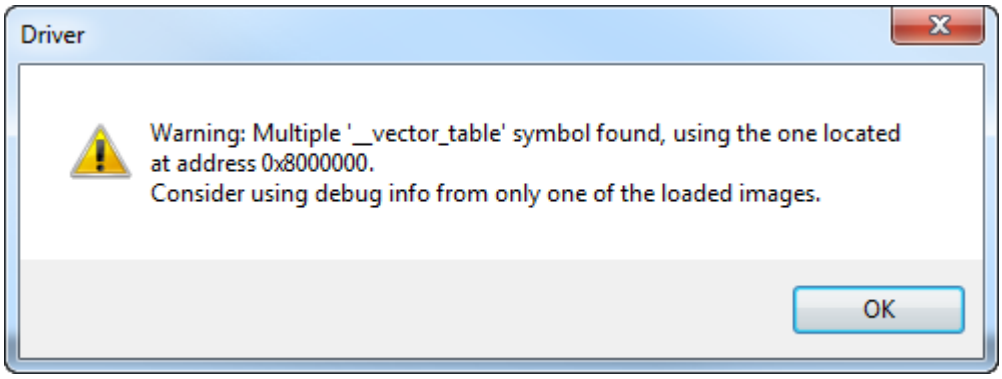
Debugging considerations

Debugging the bootloader and the application at the same time is possible, but can be tricky, for example because they both contain a main function. To make the debugger aware of the two applications, choose **Project > Options > Debugger > Images** to load debug information from the other project's ELF output file.



Make sure that the option **Run to main** is disabled on the Debugger Setup page, to avoid any confusion about which of the *main* functions to run to (the *main* function in the bootloader or the *main* function in the application).

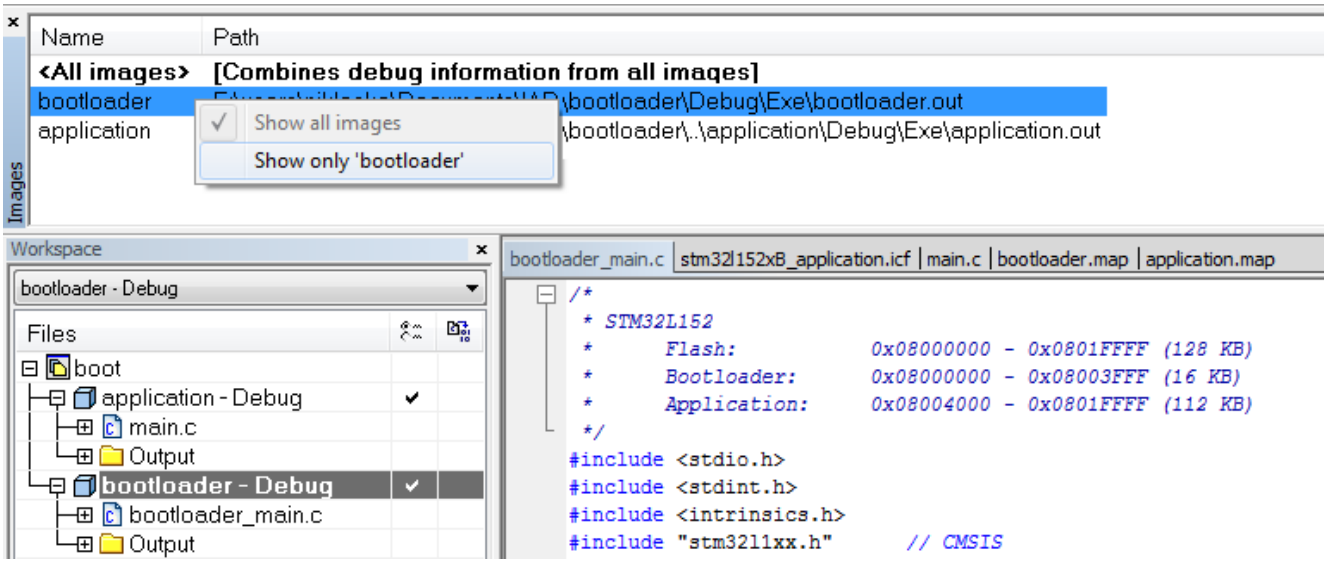
After downloading, the debugger will by default start execution at the address specified in the symbol called *__vector_table*. When you debug the bootloader and the application at the same time, the following message might be displayed:



To guide the debugger to the correct *__vector_table* symbol, you can explicitly specify where the bootloader's vector table is located. Choose **Project > Options > Debugger > Extra Options** and specify:

```
--drv_vector_table_base=0x08000000
```

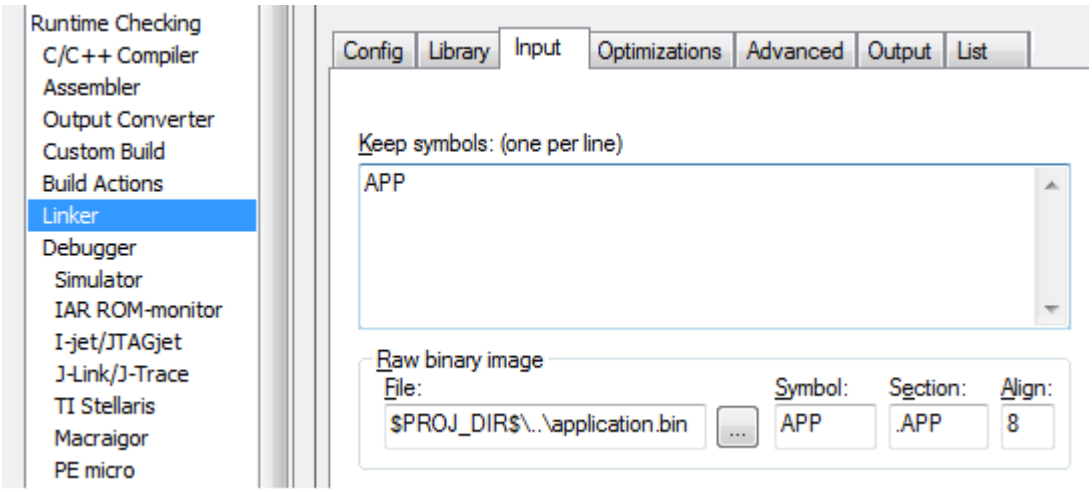
When you are debugging, use the **View > Images** window to select the application currently in scope, to avoid problems with conflicting symbol names in the two applications.



When single-stepping over the actual jump from the bootloader to the application, remember to change the currently loaded debug info (in the **Images** window).

Flash writing considerations

To download the application and bootloader to flash memory, there are many alternatives. Because the bootloader and application are placed in separate projects, they can be downloaded individually from each project. As an alternative, you can download them both to flash memory at the same time, for example by using the linker to include an application binary in the bootloader project (or vice versa). Choose **Project > Options > Linker > Input** to accomplish this.



In the linker configuration file, the application binary is placed in flash memory with:

```
place at address mem: app_vector { readonly section .APP };
```

See the example project (/contentassets/ba62fee31d724aa5a67a00da3cea6e3d/boot_example_ewarm_7701.zip) for more details. The project can be tested using the C-SPY Simulator driver or on a STM32 target device, using I-jet.

Conclusion

When you create a bootloader, you must consider a few things concerning project structure and the actual jump from the bootloader into the application. This Technical Note aims to serve as a general guideline for creating a bootloader using IAR Embedded Workbench for ARM. To get more information about debugger windows, options and the *__vector_table* symbol, choose **Help > C-SPY Debugging Guide**. For examples on how to write to flash memory from an application, choose **Help > Information Center > Example Projects** for your specific device.

RELATED TECH NOTES

Creating a bootloader for Cortex-M
Technical Note 160822

(/support/tech-notes/general/creating-a-bootloader-for-cortex-m/)

Creating a bootloader for MSP430
Technical Note 13285

(/support/tech-notes/general/creating-a-bootloader-for-msp430/)

Saving the ROM content size in a variable
Technical Note 52791

(/support/tech-notes/linker/how-can-i-save-the-size-of-the-rom-content-in-a-variable-located-in-flash/)

CMSIS-DAP - Probe not found
Technical Note 69741

(/support/tech-notes/debugger/cmsis-dap---probe-not-found/)

Debug two application images using macro __loadModule
Technical Note 73861

(/support/tech-notes/debugger/debugging-two-application-images-using-macro-__loadmodule-ewarm-5.30-or-__loadimage-ewarm-6.21/)

Splitting hex or srec files
Technical Note 35923

(/support/tech-notes/linker/splitting-hex-or-srec-files/)

Unlock LPC-1768-SK
Technical Note 79394

(/support/tech-notes/general/unlock-lpc-1768-sk/)

AT91SAM9261 copy from dataflash to SDRAM at boot-time
Technical Note 44002

(/support/tech-notes/debugger/example-for-at91sam9261-how-to-copy-from-dataflash-to-sdram-at-boot-time/)

(https://www.youtube.com/channel/UCxxoUOKedI8_s3ZdJFuu6Yg)