VORAGO VA108x0 Boot Memory Robustness Improvement Application Note

July 2019 Version 01.0

VA10800/VA10820

Abstract

For systems relying on an external memory from which to boot, there exists a single point failure mechanism that could shut down the system. This application note provides demonstrated methods for improving the response to memory corruption and Single Event Functional Interruption (SEFI) errors.

Memory corruption in instruction code space can be detected via software and an alternative code space used. This effectively halves the available code space but can provide a straightforward method to address a small number of errors.

SEFI errors may place the memory in a non-functional state such that neither reads nor writes are possible but no actual memory corruption occurs. This normally requires the power to be cycled to resume normal operation. SEFI is a special case of Single Event Upset (SEU) changing an internal control signal. Adding a few components to the board can allow the memory to be powered off when not in use thereby greatly minimizing the exposure to SEFI errors.

Table of Contents

1 Memory corruption detection and response

The likelihood of a space grade memory becoming corrupt is very remote, but still investors of multi-million-dollar spacecraft want the most reliable systems possible. One potential error in NVM components is a single bit error where the transistors or storage element for a single bit become damaged or weaken over time. Some memories have built-in Error Code

Correction (ECC) to address any single bit errors. From an MCU perspective, detecting this type of error is commonly done with software running a cyclical-redundancy-check (CRC) on a portion of the memory. The CRC can detect only that an error occurred; it does not have the data to correct the error.

Sections 1 & 2 detail a bootloader to conduct the memory check and use an alternative code image if an error is found.

1.1 VA108x0 Boot process

The VA108x0 MCU will always boot from an external SPI memory for resets caused by power-on reset (POR) and pin reset events. Memory is read 32 bytes at a time. Two reads of the 32-byte block are made and compared against each other. If the blocks are not equal, the whole process begins again starting at address 0x00. This check is mainly for electrical noise corrupting the SPI data transfer but can also capture NVM bits that are partially programmed. If the two reads of the 32-byte block are equal, the internal SRAM of the device is written with this information and the process continues until all 128 Kbytes are loaded. The address map of the NVM and the MCU are identical.

The VA108x0 MCU can be configured in software to react to a software initiated reset differently than a POR or pin reset. The alternate boot sequence does not load memory from the SPI NVM and immediately starts to execute code from on-chip SRAM . We will use this functionality with the bootloader explained in the next section.

The Cortex-M0 does not support vector redirection but because the instruction space and vector information are located in SRAM, rewriting vector information is possible. This functionality is used in the CRC bootloader.

1.2 Bootloader with CRC

Having a separately calculated CRC for different portions of memory allows for a redundant image to be used if the primary image is corrupt. This application note introduces a method of having two user code images that are checked with a CRC algorithm located in a common “bootloader” program. This method is a variant of the bootloader detailed in AN1216. This approach is still vulnerable to a single corrupt bit in the bootloader section causing issues that prevent normal code execution. However, the vulnerable memory space of a single byte causing problems is reduced from 128 Kbyte to less than 4 Kbyte.

The basic flow of the bootloader project is shown in Figure 1. Note that the VA108x0 can be configured via software to not reload memory from the SPI NVM for software-initiated resets. This allows the vector table to be overwritten by software and a different set of

actions to take place from a POR or external pin reset. The provided project will use image A if the CRC of the bootloader fails. This may not be the preferred way of handling an error. Figure 2 shows a slightly modified flow that a user can implement with an error message being reported if the bootloader CRC fails.

Figure 1 - Flow chart of memory coherency check and associated actions based on the CRC results (this is the example project implementation)

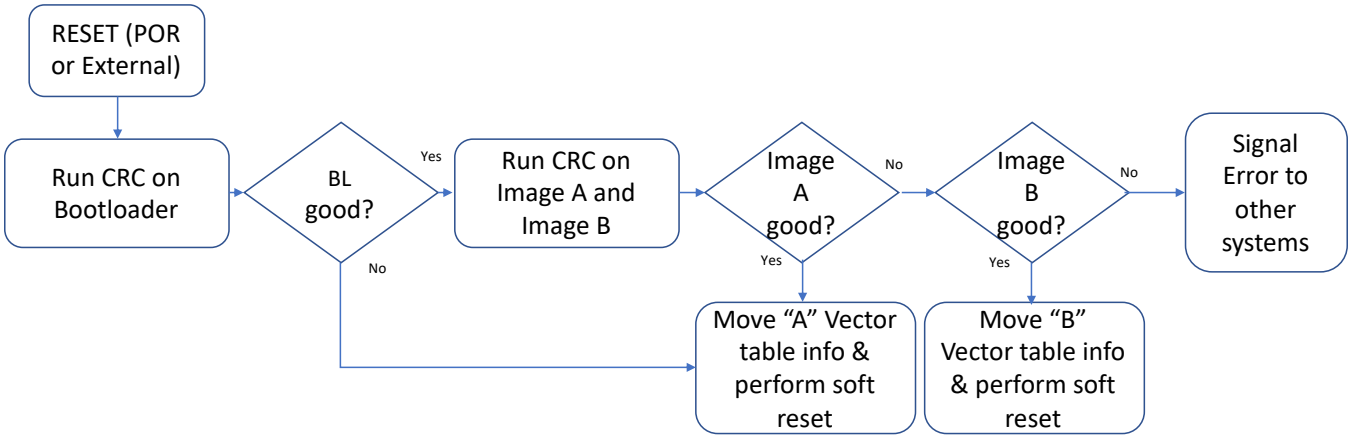
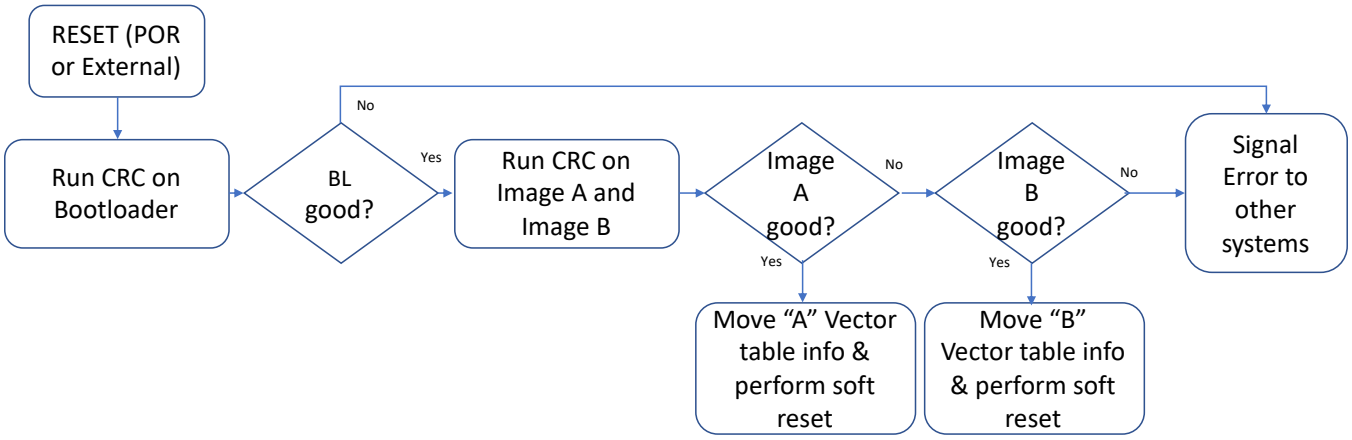


Figure 2 - Flow chart with conceptual implementation of signaling error if any CRC fails



For the above flowcharts to work, three separate images must be created and programmed into the device. Two images will be created from the same user code project, but the linker target memory must be uniquely set as shown in Figure 3 and

shows the memory map of the VA108x0 and SPI NVM during the process. The NVM is programmed with three images and three 16-bit CRC characters will be stored during the programming steps.

Figure 3 - Keil MDK Option page showing IROM1 space information

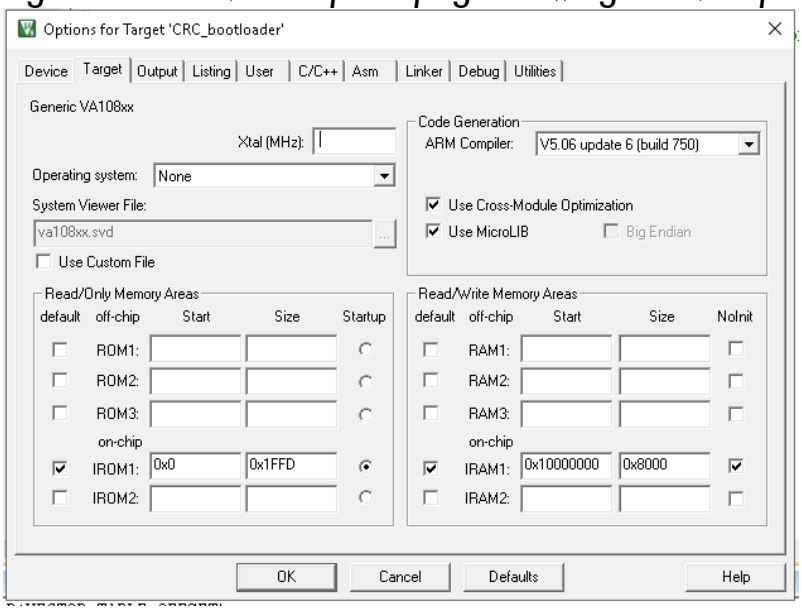
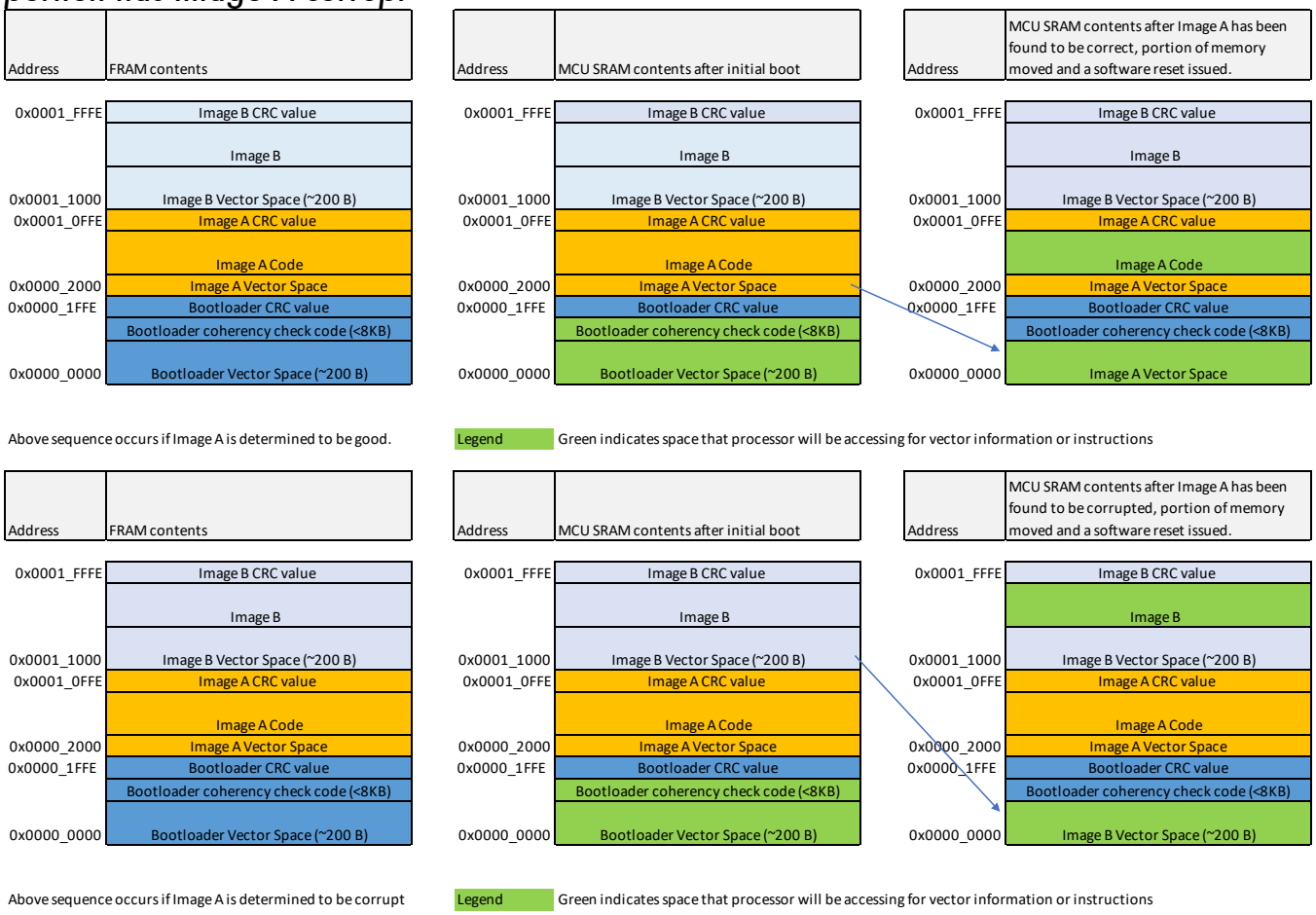


Figure 4 - Table with IROM1 start and stop values for each image

	Start Address	Size	Size	CRC location
		(Hex)	(Decimal)	
Bootloader	0x0000_0000	0x1FFD	8189	0x0000_1FFE
Image A	0x0000_2000	0xEFFD	61437	0x0001_0FFE
Image B	0x0001_1000	0xEFFD	61437	0x0001_FFFE

Figure 5 - Memory map with three images shown. Top portion has Image A valid. Bottom portion has Image A corrupt



1.3 Key Features of the VA108x0 CRC Bootloader

A Keil MDK project accompanies this application note with code to implement a CRC bootloader with the following features:

a. Small code size < 8 Kbytes (Customer can reduce this to under 1 Kbyte by eliminating reprogramming and terminal window support)
b. UART interface for easy connection to a PC or other MCU
c. CRC checking on three separate memory ranges
d. Reprogramming support via terminal window
e. Open source
f. No security implementation

1.4 Installing the programming file

A unique programming algorithm needs to be used that automatically calculates the CRC for 3 specific ranges and stores that information in NVM memory. That programming algorithm is provided with this application note as VA108_FM25V20A_FRAM_128K_CRC.FLM. This will only work with the Cypress SPI FRAM devices: FM25V20A or CYRS15B102.

In the root directory of the AN1217_SW.zip file, is a file titled: VA108_FM25V20A_FRAM_128K_CRC.FLM. This file needs to be moved onto the PC with the Keil MDK installed. Copy the file to the C:Keil_v5/ARM/Flash directory. From that location, the IDE can locate the programming file.

2 Steps to install the Bootloader with CRC

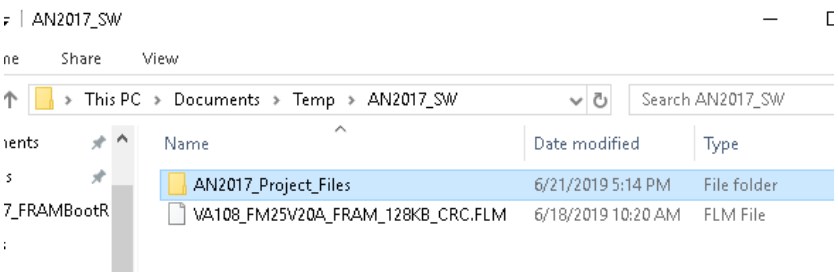
This section contains step by step instructions on how to install and use the provided bootloader.

2.1 Create and program image 1 – bootloader with CRC

The bootloader is programmed via the JTAG interface in the same way that other code images are loaded. For the REB1 development board, the bootloader can be programmed via USB through the J-Link OB interface. For other applications, use of an external JTAG adapter will be necessary.

The AN1217_SW.zip file can be decompressed to reveal a folder structure like what is shown here: RH-SIP -> CRC_bootloader

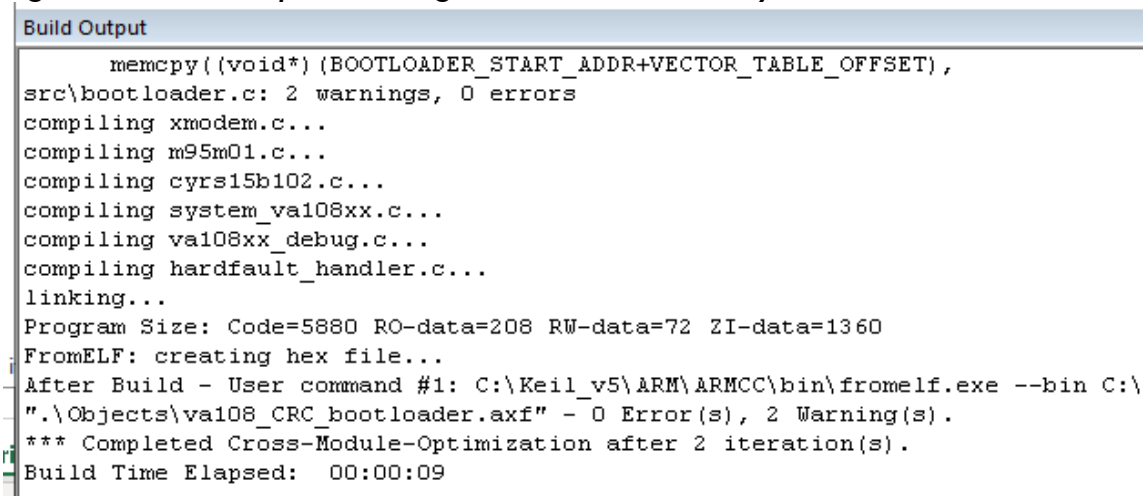
Figure 6 - Folder structure of AN2017_SW.zip



Navigate to the CRC_bootloader folder and double click on the "CRC_bootloader" file with file type = uVision5 Project. This should open the Keil IDE. It may be necessary to open the options menu and change the debugger to match the one being used on your bench.

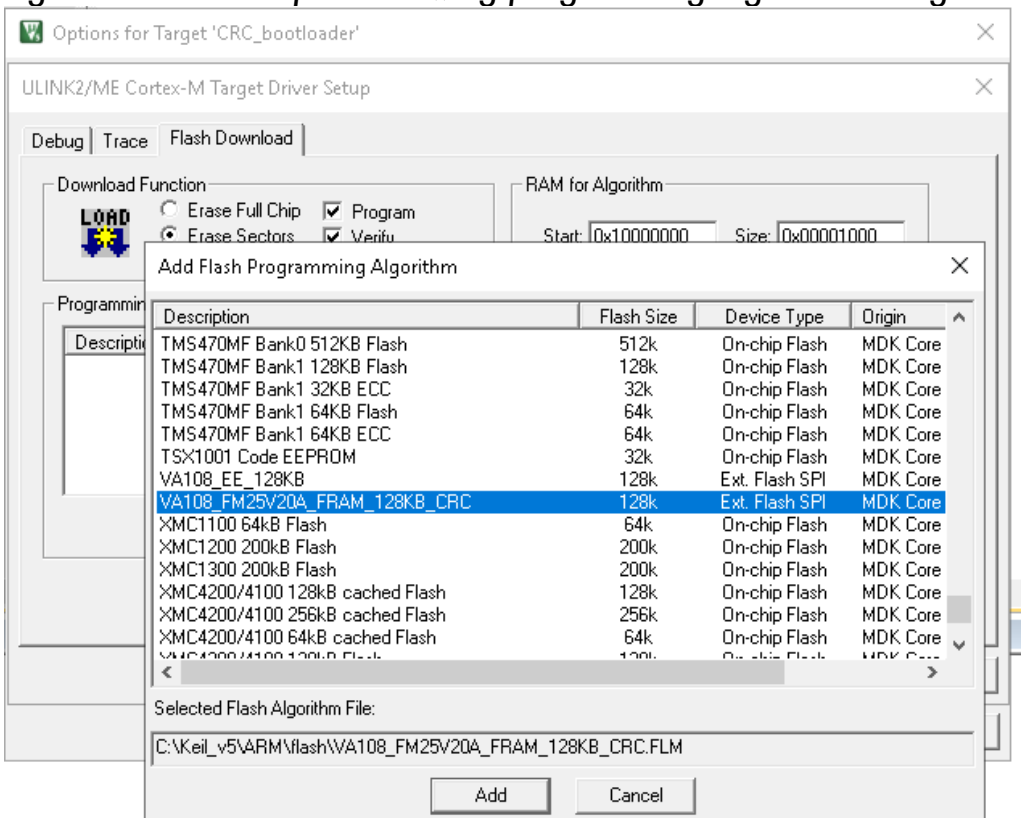
Build the project by selecting "Rebuild all target files" under the "Project" menu. The build output window should show the code = 5880 bytes.

Figure 7 - Build output showing code size of 5880 bytes



Before programming the NVM, the programming algorithm must be selected. Under the Project -> Options menu, select Utilities as shown here. Hit the "add" button and scroll to find "VA108_FM25V20A_FRAM_128K_CRC.FLM". Exit this menu by hitting OK and OK.

Figure 8 - Screen capture showing programming algorithm being selected

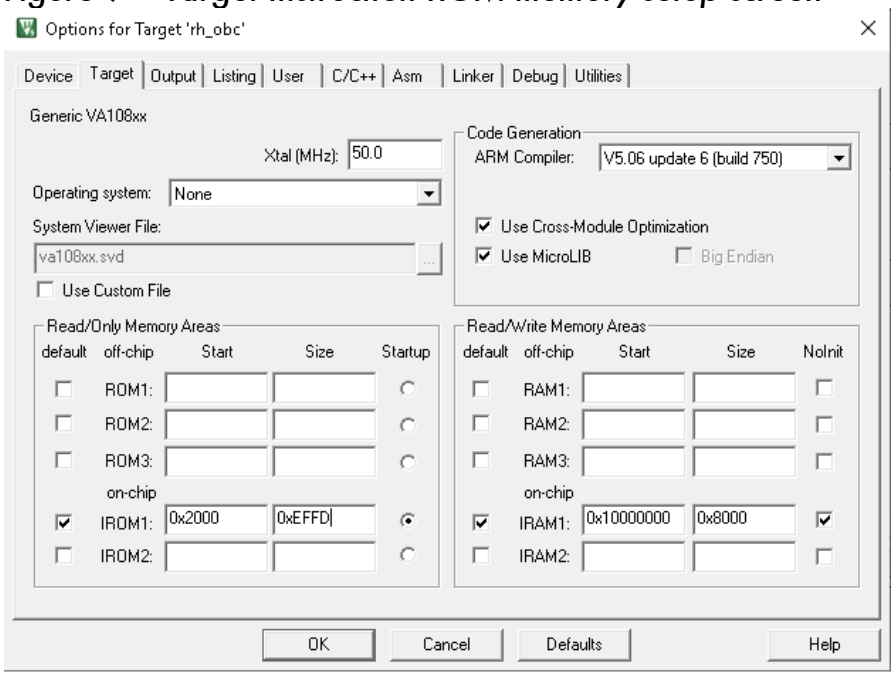


Under the Flash menu, select the Download option. The console window at the bottom of the IDE should show the results of the program operation after a few seconds. At this point, the bootloader image has been programmed in the NVM.

2.2 Create and program image 2 – User code: Image A

Once a program (<61,437 bytes) has been created and debugged using native Arm M0 vector locations, it is ready for incorporating with the bootloader CRC. No change to the code is necessary. However, the target memory location must be altered as shown in Figure 9.

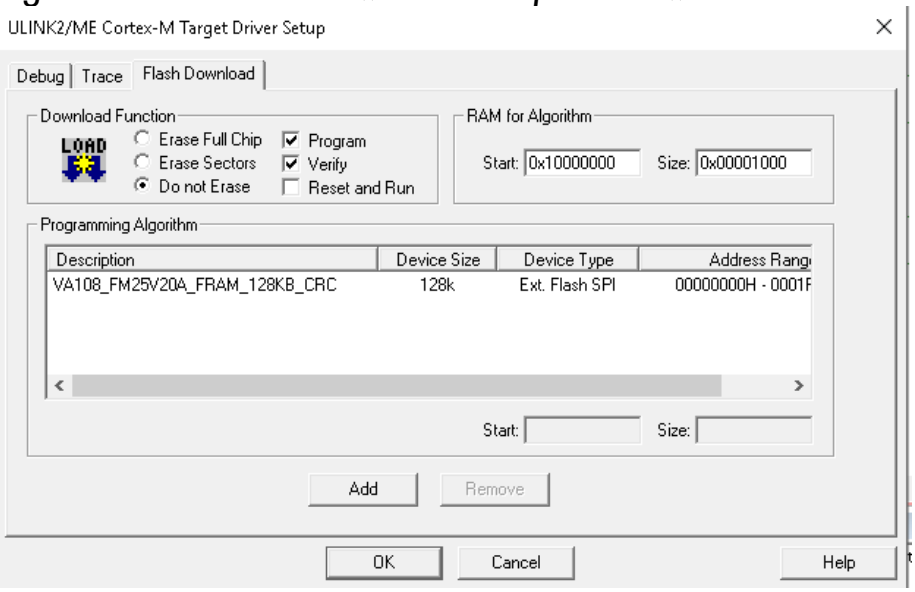
Figure 9 – Target Instruction ROM memory setup screen



Once this memory configuration information is entered, rebuild the project.

The program can be loaded via the bootloader as explained in the following sections of this document or it may be done via the Keil IDE. When using the Keil IDE, a minor change to the programming algorithm setup is recommended to avoid erasing the bootloader. As shown in the below figure, click the "Do not erase" button. The FRAM device is not required to be erased before programming and this will keep the bootloader and any other images intact.

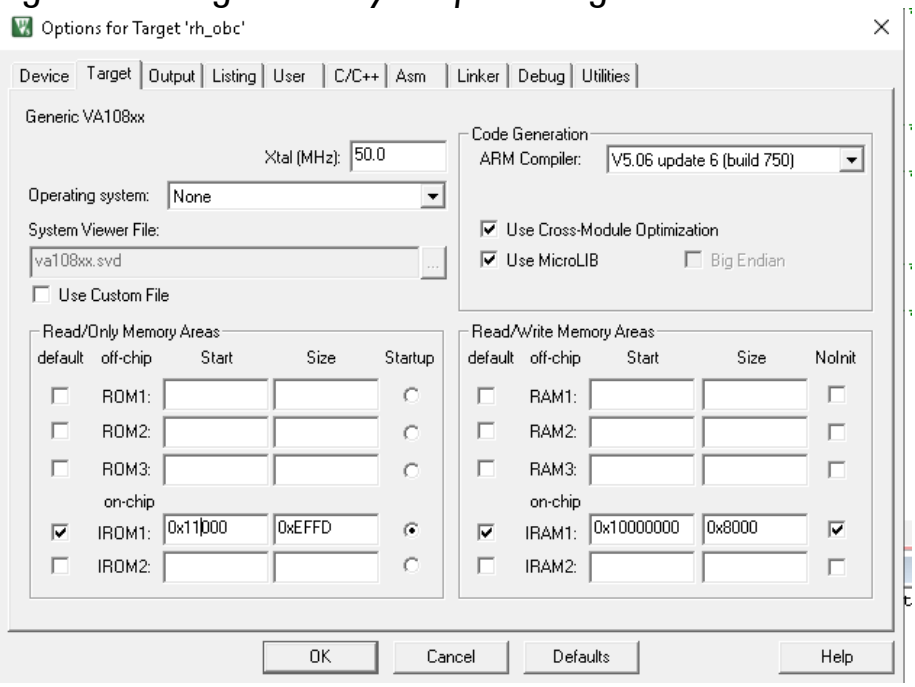
Figure 10 - Keil Flash Download setup screen with "Do not Erase" selected



2.3 Create and program image 3 - User code: Image B

Creating and programming image B is very similar to image A with the sole difference being the specified address for the programming to reside. The target memory setup should match the following screen.

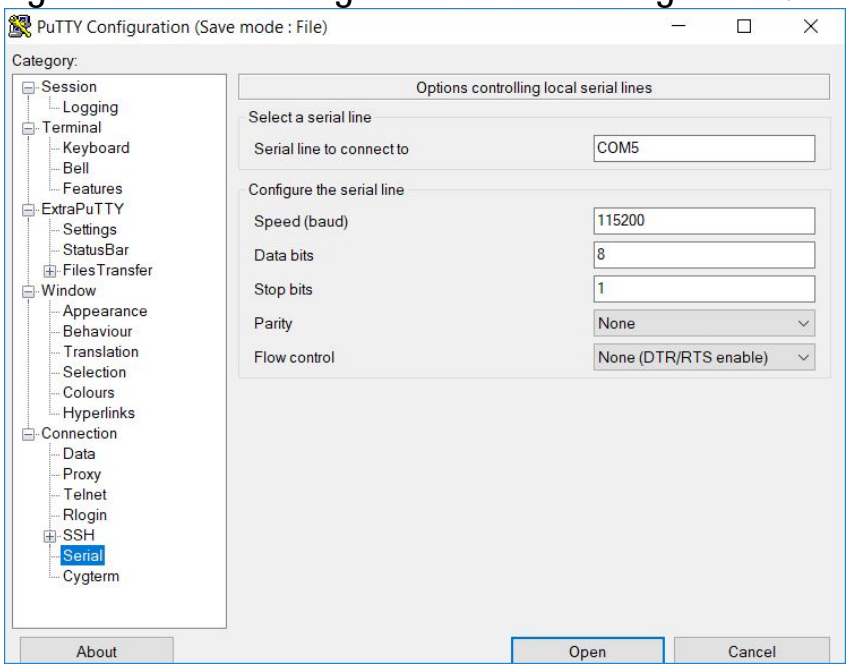
Figure 11 - Target memory setup for Image B



2.4 File transfer with XMODEM and ExtraPuTTY

ExtraPuTTY is a fork of PuTTY terminal software that adds features, notably XMODEM file transfer. It can be found here: http://www.extraputty.com/
Use the Windows Device Manager to find the COM port number of the USB to serial adapter. Point ExtraPuTTY to this COM port number, with 115200 baud rate, 8 data bits, no parity, and one stop bit (8-N-1). Alternate terminal windows like TerraTerm or PuTTY can be used but they do not support the XMODEM download making programming impossible.

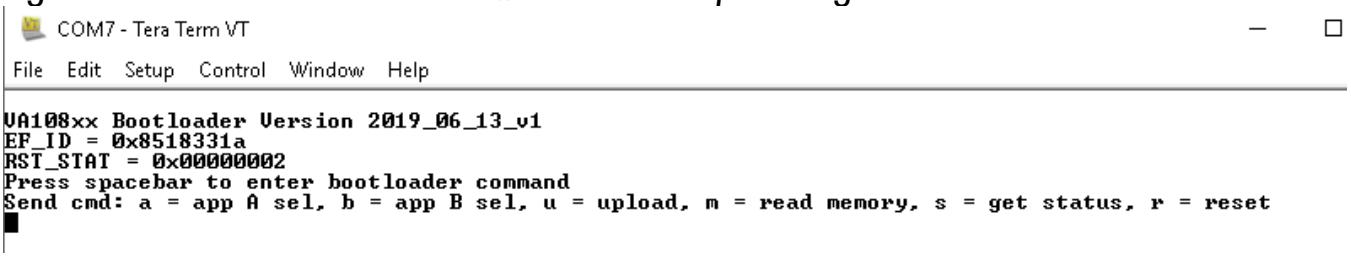
Figure 12 - PuTTY configuration for connecting to the VA108x0 Bootloader



Upon power-up of the connected MCU, the following message will appear on the terminal window if the application code space is empty or corrupted.

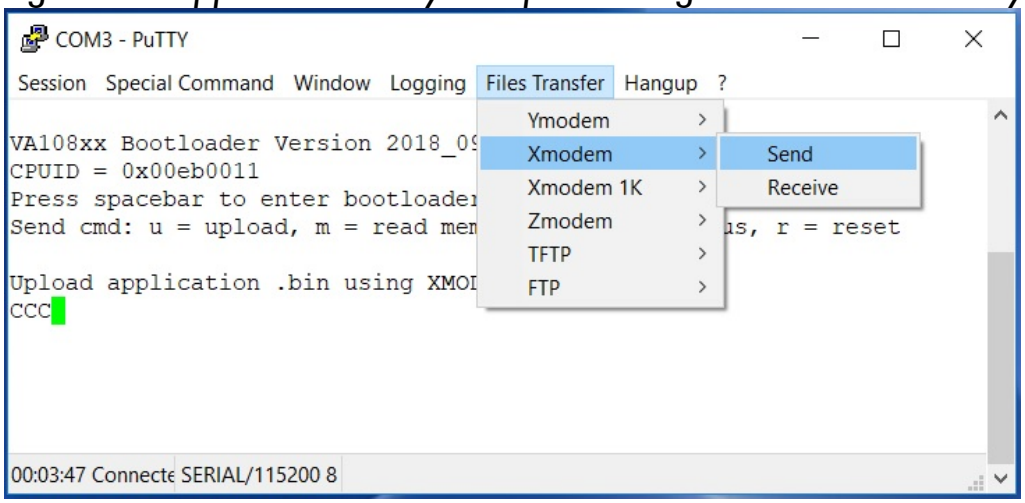
If the application area is not empty (and valid), the user will have 5 seconds to press the spacebar in the terminal window to enter the bootloader instead of running user code. If the spacebar is not pressed within 5 seconds, the application firmware will run. This timeout changes to 100ms when "PC Mode" in the bootloader code is disabled (it is a compile flag in bootloader.h). An example of the text output in PC Mode when user application is valid, but the spacebar has been pressed within the timeout period is shown:

Figure 13 - VA108x0 Bootloader with CRC startup message –



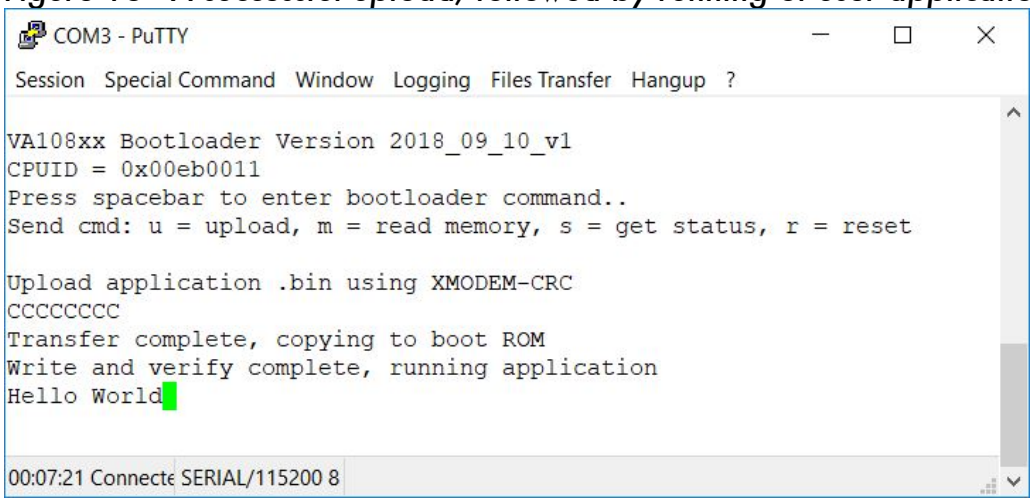
A code image compiled for use with the VA108x0 Bootloader is different from a normal firmware image only in that the code start address and IVT location has been changed from 0x0000 to 0x2000. To upload a .bin file, press the 'u' key in the terminal window. The bootloader will then send a sequence of 'C' characters, a signal to the XMODEM protocol to use the CRC-16 packet check mode (XMODEM-CRC). Then, in PuTTY, under 'Files Transfer', under 'Xmodem', select 'Send'. Choose the desired .bin file.

Figure 14 - Application binary file upload using Xmodem in ExtraPutty



The file will upload, copy over to the SPI boot ROM (EEPROM or FRAM), and then the MCU will attempt to run the uploaded code image. If successful, it will look like the following (Note: The 'Hello World' text is printed from a demo user application, depending on the user code the text shown after 'running application' may look different):

Figure 15 - A successful upload, followed by running of user application "Hello World"



If there is an error, the error will be reported, and the bootloader will reset and give the option to retry the upload. It will not attempt to run an image that fails the write verification or reset vector check.

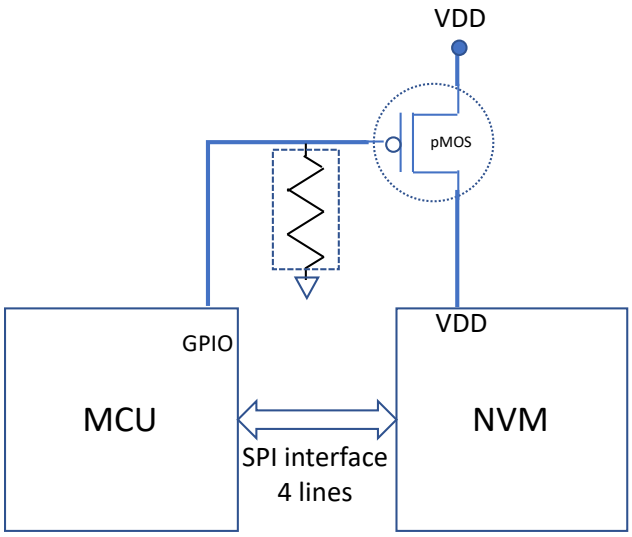
3 Avoiding Single-Event-Functional-Interrupts

If an IC is not being used, placing it in a low power mode is recommended. In many devices, the low power mode disables power to all, but the essential circuits needed for waking up. This keeps those powered down circuits immune to particle strikes causing a lock-up event.

In order to remove functional interruption errors such as when a chip is non-responsive, it will most likely require power to be removed and then reapplied. Figure 16 illustrates one method of removing power to the SPI memory without impacting other circuits on the board using a small p-channel FET with a pull-down resistor on the gate. As the MCU goes through the RESET sequence, the port pins will be tri-stated, and the resistor will turn the FET on and power is applied to the NVM. After the boot sequence, software can turn the FET off, removing power and making it immune to SEFI events.

Note that the I/O lines to the SPI memory should be driven low during the power-off interval otherwise, it is possible that the SPI memory would try power the IC through a logic input pin.

Figure 16 - Power control of SPI NVM with p-channel FET



4 Conclusions

This application note has introduced the Vorago Technologies VA108x0 Bootloader and examined its usage, mechanism of operation, and installation. To avoid SEFI events, hardware considerations with powering down NVM when not being accessed were also provided.

5 Other Resources

VORAGO VA108x0 programmers guide & VORAGO MCU products:
http://www.voragotech.com/VORAGO-products

VORAGO Application notes: http://www.voragotech.com/resources

VORAGO VA108x0 REB1board user guide: Part of Board Support Package (BSP)
http://www.voragotech.com/products/reb1

Revision log:

July 2019 – Initial release