VORAGO VA108x0 Bootloader application note

Jan 2, 2019 Version 1.0

VA10800/VA10820

Abstract

Many applications can benefit from in-system reprogramming without a JTAG probe. This Application note details the implementation of a UART bootloader for the Cortex®-M0 based VA108x0 series of MCUs.

Table of Contents

1 Overview of Bootloader functionality

This document details a bootloader program that will allow the SPI NV memory connected to the VA108x0 ROM_SPI to be reprogrammed. In brief, the 128 Kbyte of program memory space will be divided into two images. The bootloader image will reside in the lower 8 Kbytes of memory (0x0000 – 0x1FFF). When the MCU is reset, the bootloader will begin execution and interrogate the UART interface for incoming messages requesting a reprogramming operation. If a request is not recognized in a short period of time (5 seconds for PC version & 100 mS for MCU to MCU version), code execution will jump to the startup routine in application space (0x2000 – 0x1:FFFF).

Each image (bootloader and user code) will be created separately using a unique linker file and each image will have a separate interrupt vector table (IVT). Since the Cortex-M0 does not support vector redirection and the VA108x0 has RAM based code space, bootloader

software will move the IVT from image 2 to the native Cortex-M0 vector location. This will avoid any interrupt latency that conventional Cortex-M0 bootloaders may incur.

Two port pins with UART Rx and Tx functionality will be routed through an FTDI RS-232 cable to the USB port on a PC. The PC will enumerate the cable and either ExtraPuTTY or Tera Term can be used to communicate with the MCU and upload a new code image via the XMODEM-CRC protocol (128 byte packet size + CRC checking).

Port pins to be used:

UARTA – TX: PA31
UARTA – RX: PA30

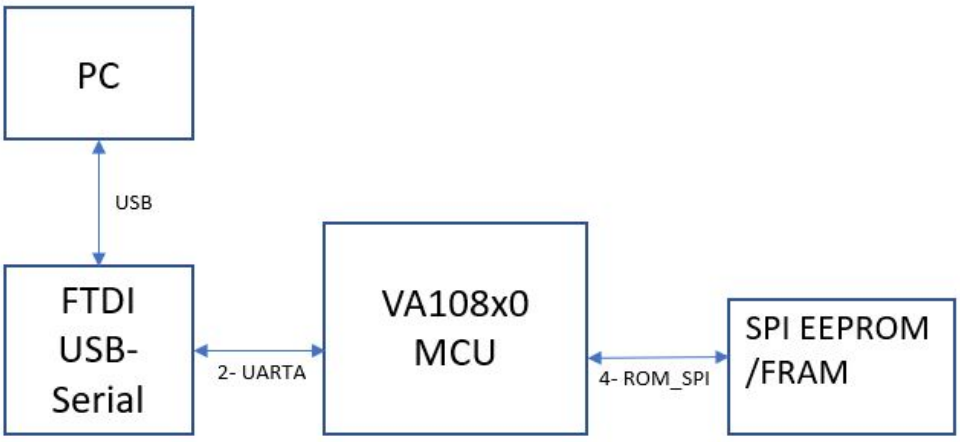


Figure 1- PC to MCU interconnect diagram

1.1 Key Features of the VA108x0 Bootloader

a. Small code size - < 8 Kbytes
b. UART interface for easy connection to a PC or other MCU
c. CRC checking on file transfer and packet re-transmit on NACK for robustness
d. Safe reprogramming operation. Board will never be in jeopardy of becoming non-functional if a reset or power loss occurs during the programming operation.
e. Open source
f. No security implementation

2 VA108x0 Code SRAM

The bootloader code will reside in the lower memory space (0x0000 – 0x1FFF). User code will reside in upper memory space (0x2000 – 0x1:FFFF) as shown in the below figure:

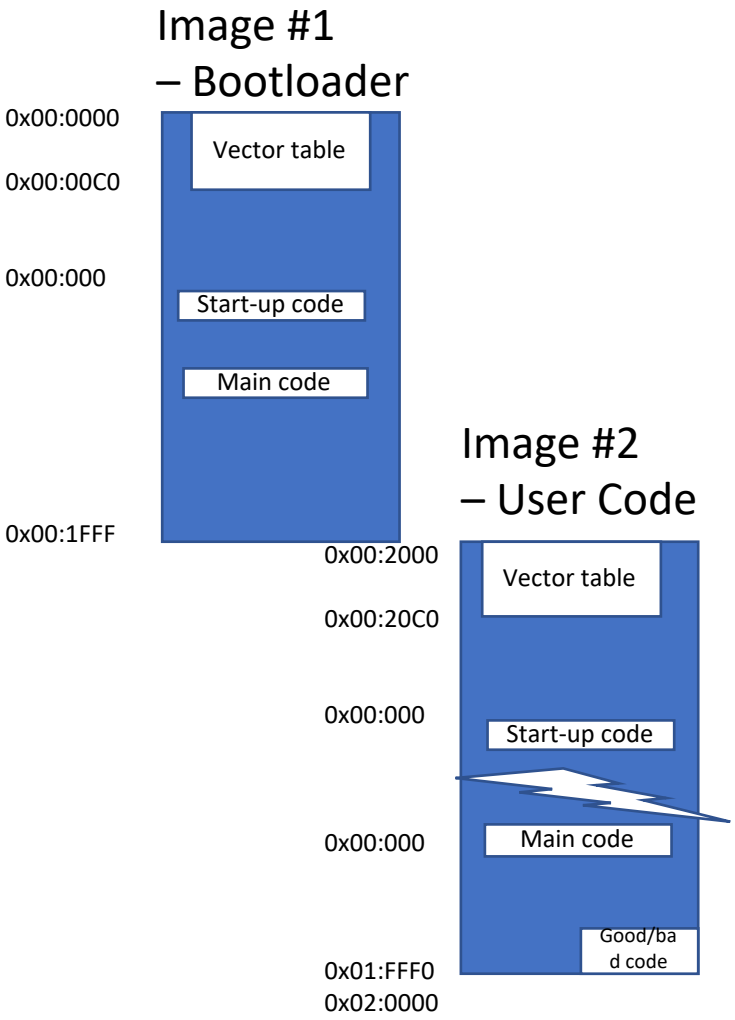


Figure 2- Physical memory locations

When the bootloader runs (after a reset requesting a re-load of instruction code from the boot NVM), if a reprogramming request is not recognized the bootloader will then attempt to run the user code. First, it checks the user application reset vector at 0x2004. The reset vector must point to within user code space for the application image to be considered valid. If this check passes, the entire application vector table (192 bytes) is copied from 0x2000 to the

native Cortex-M0 location 0x0000. Then, bit 2 of the RST_CNTL_ROM register is cleared, disabling a re-load of code RAM upon a SYSRESETREQ software reset (a re-load would change the copied IVT back to the bootloader IVT). Finally, a SYSRESETREQ is initiated and the VA108x0 soft resets, jumping to the user application since the user vector table is now in the native Cortex-M0 location. A subsequent software reset will skip the bootloader and jump straight to the user code. If re-entering the bootloader is desired, set bit 2 of the RST_CNTL_ROM register then initiate the SYSRESETREQ. Other reset sources will run the bootloader first by default, unless that reset source's bit is cleared in RST_CNTL_ROM.

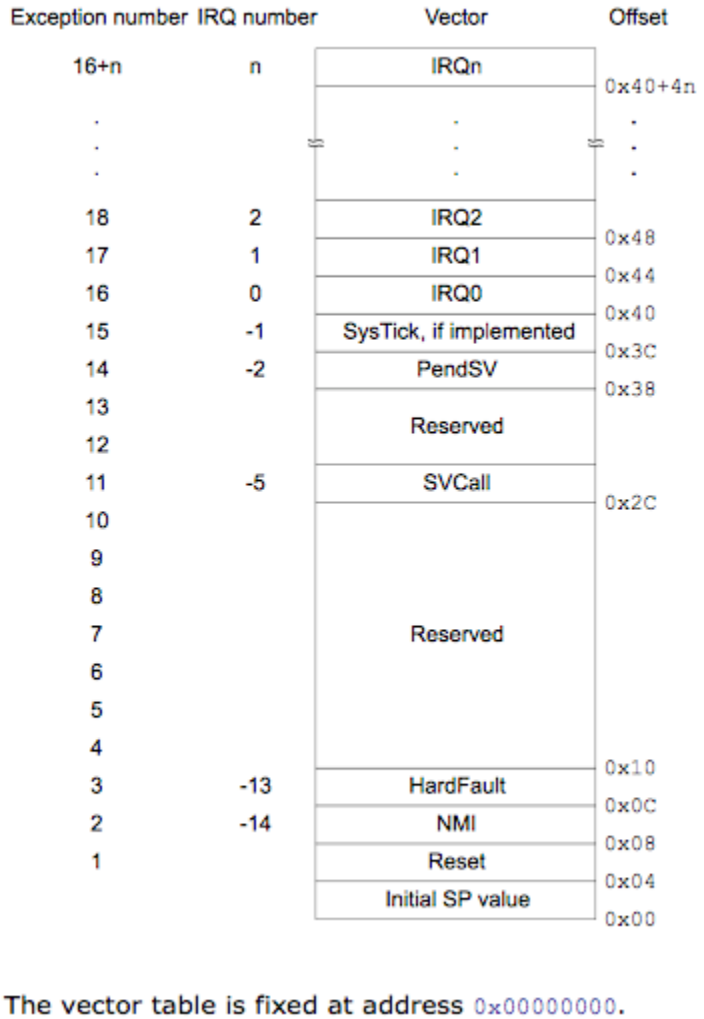


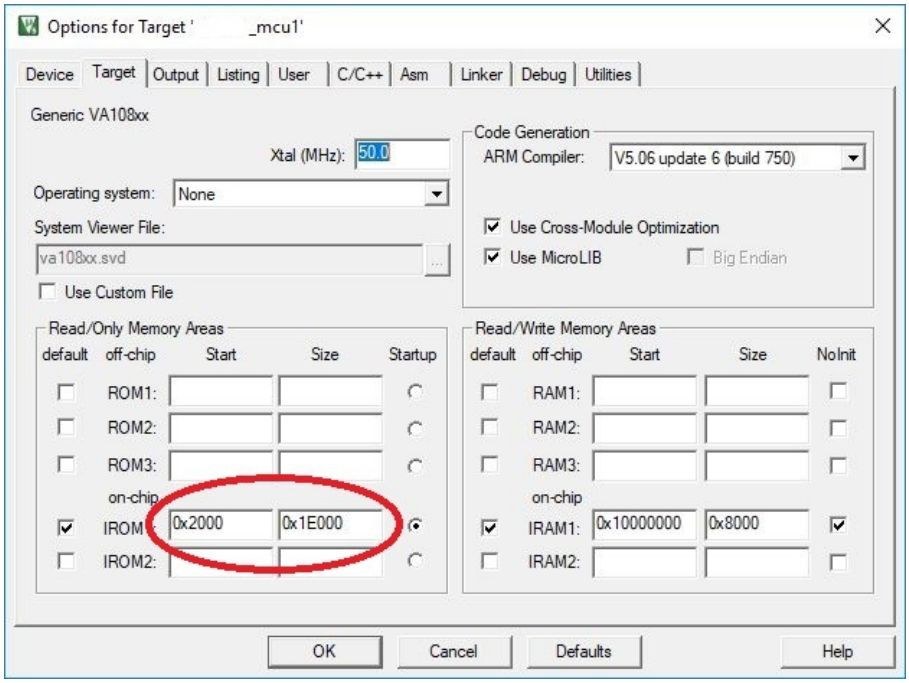
Figure 2 - Cortex-M0 Vector assignment table

2.1 Linker requirements

When compiling the bootloader (using IAR™ or Keil®), set the target code space to be from 0x0000 to 0x1FFF.

When compiling the user application, set the target code space to start at 0x2000 and end at 0x1:FFFF. Set the interrupt vector table location to 0x2000 (necessary in IAR only).

2.1.1 Keil µVision® IDE



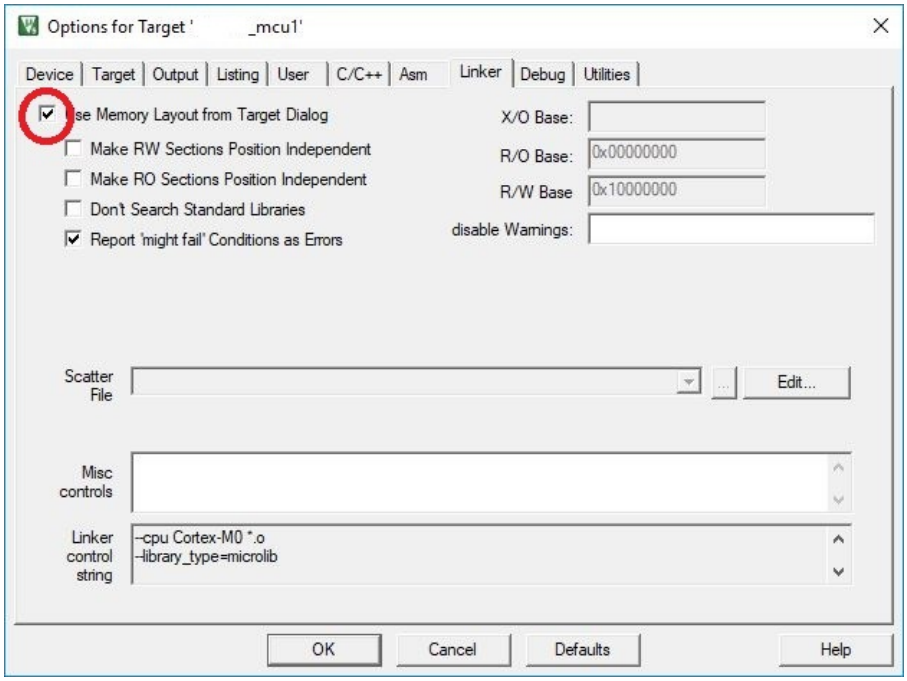
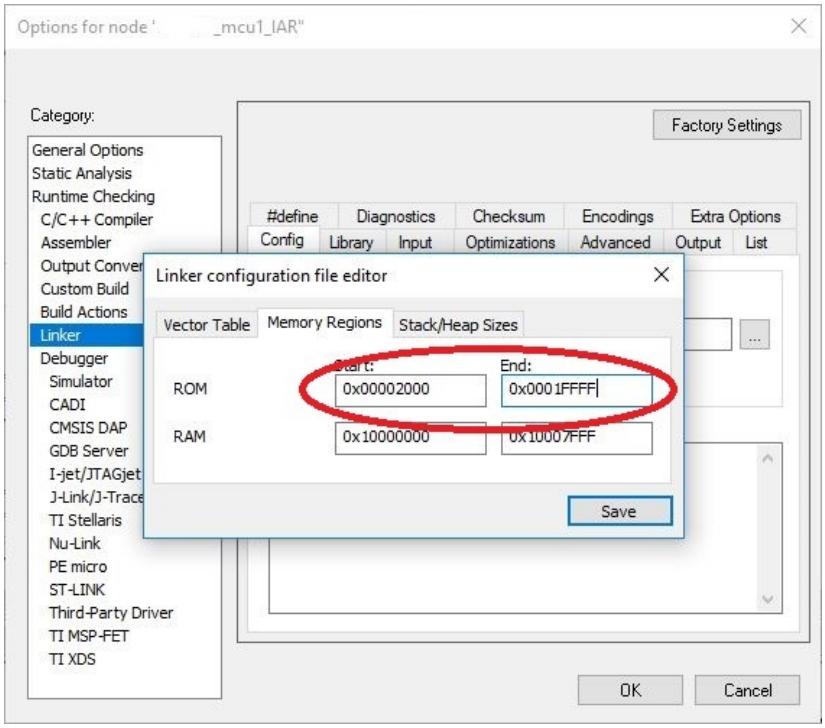


Figure 4- Keil linker settings for compiling user application

2.1.2 IAR Embedded Workbench® IDE



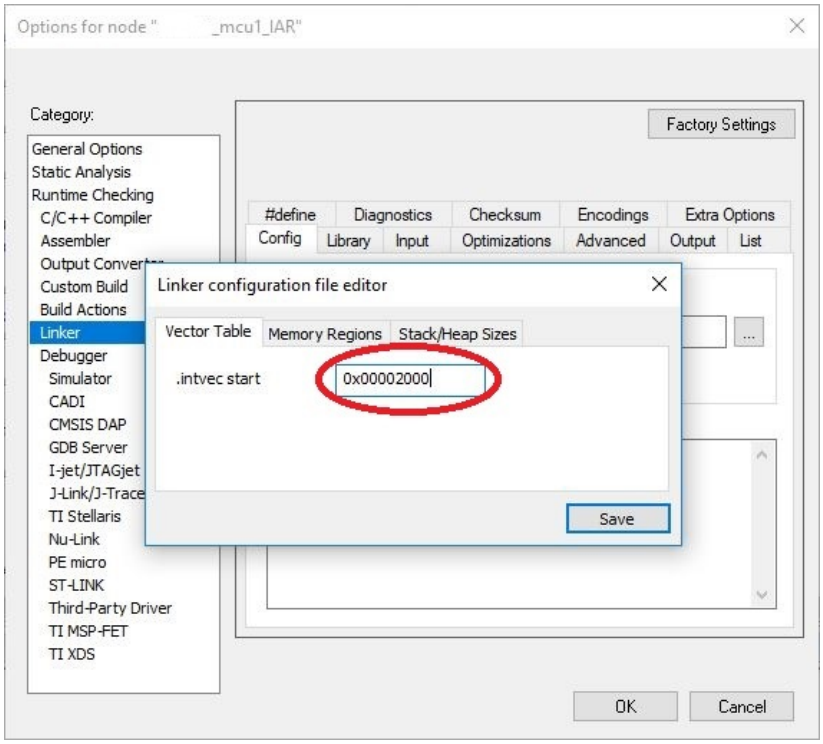


Figure 5- IAR linker settings for compiling user application

2.2 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).

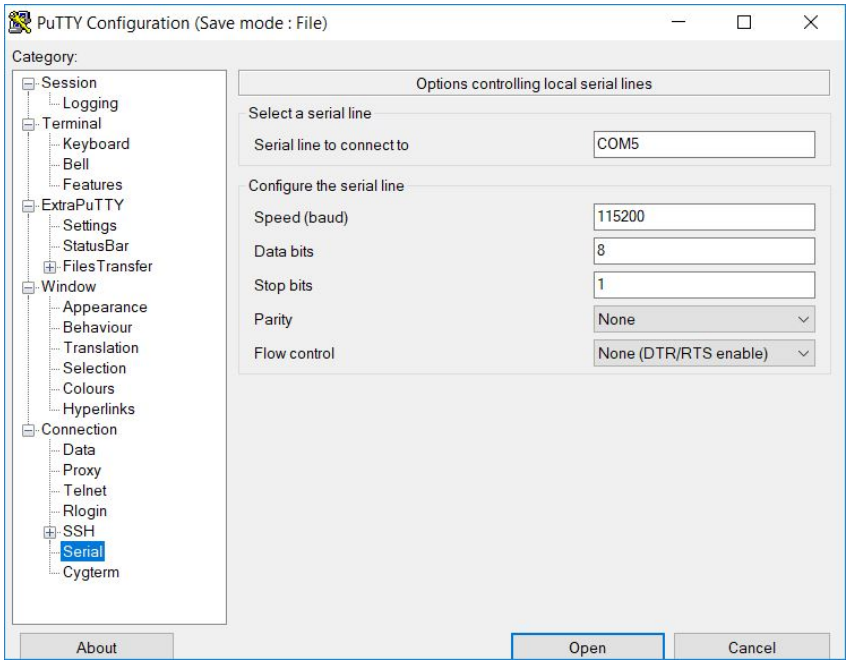


Figure 6- PuTTY configuration for connecting to the VA108x0 Bootloader

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

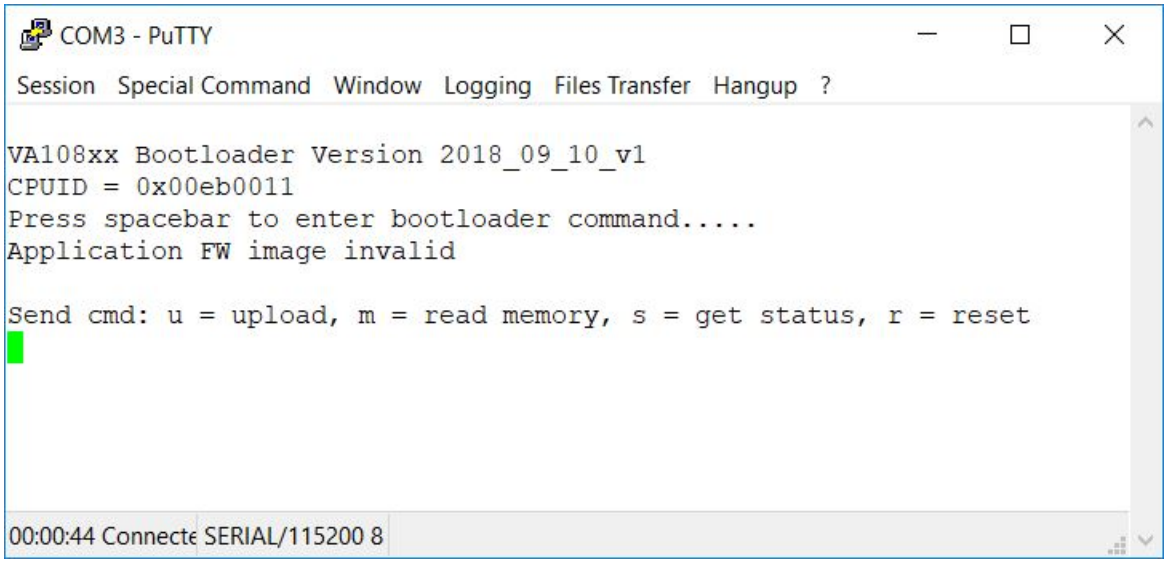


Figure 7- VA108x0 Bootloader startup message – application code area empty/invalid

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:

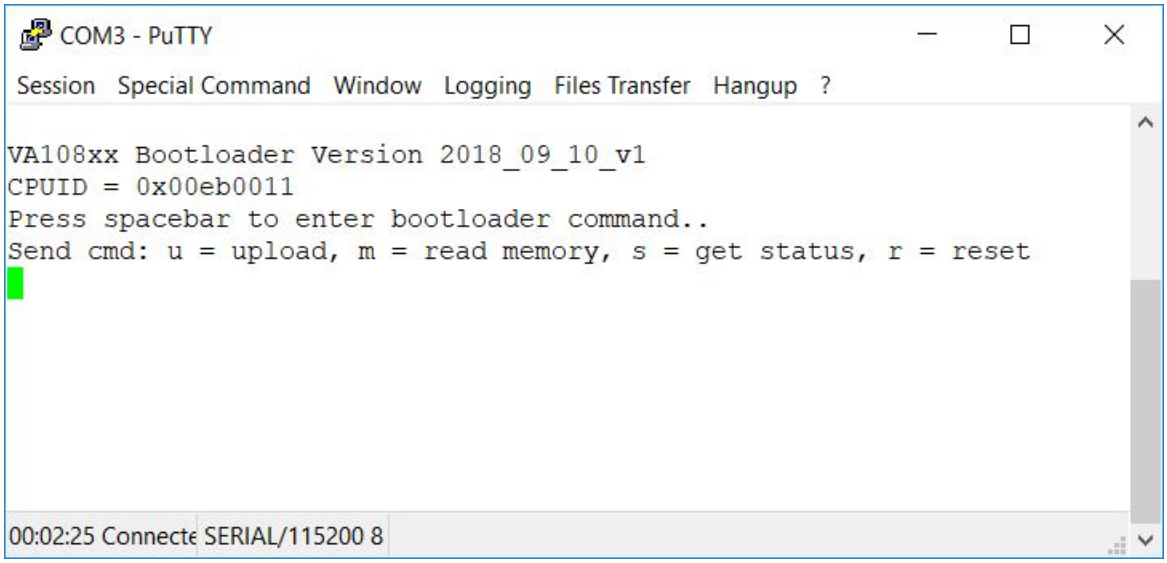


Figure 8- User pressed spacebar upon bootloader startup within 5 seconds

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.

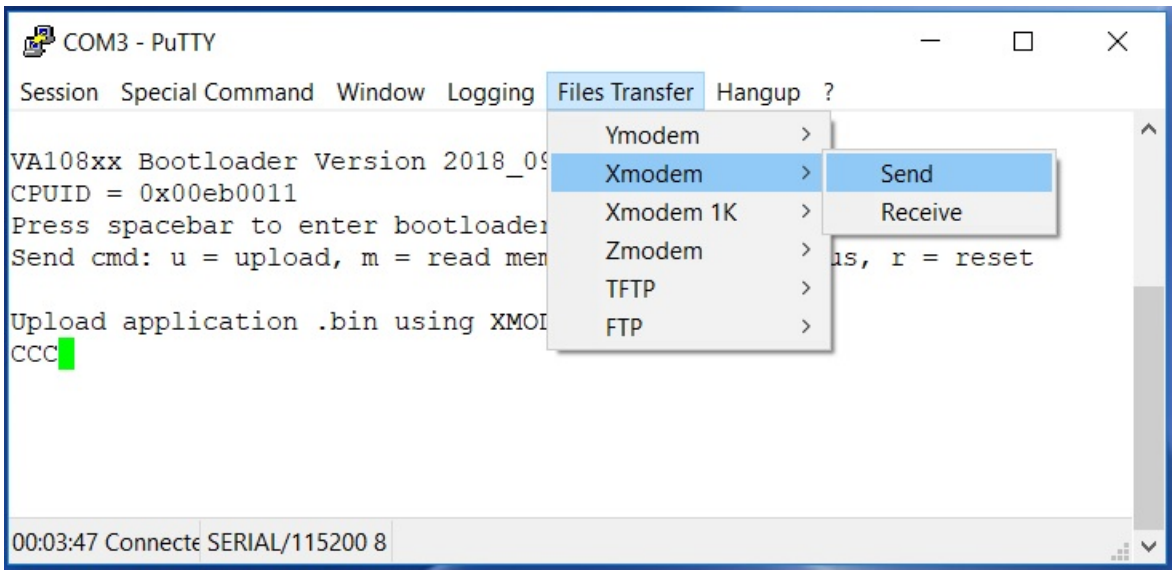


Figure 9- Application binary file upload using Xmodem in ExtraPutty

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):

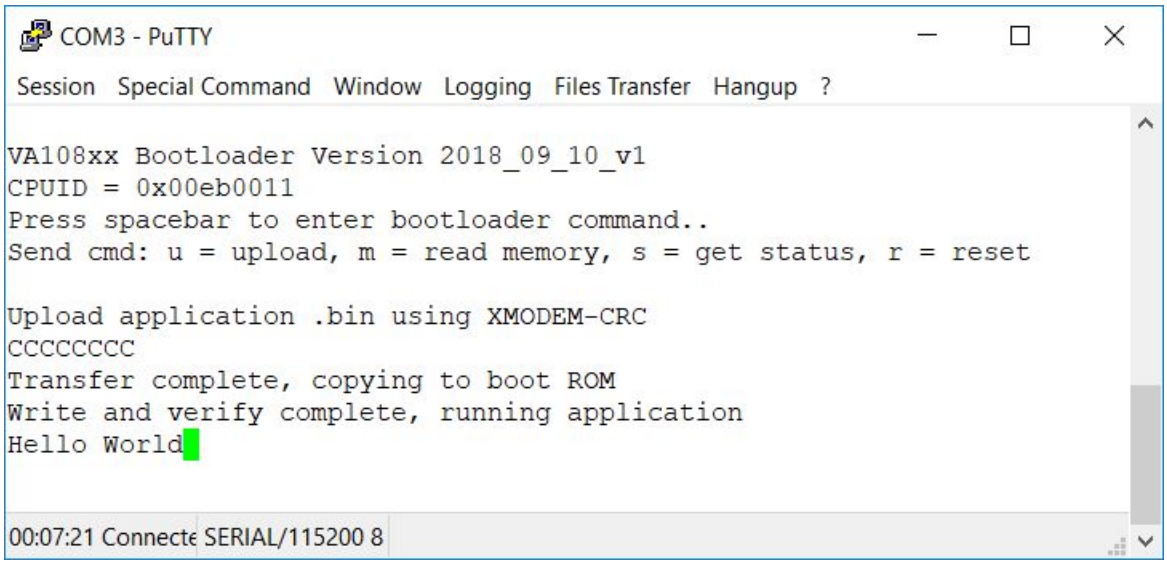


Figure 10- A successful upload, followed by running of user application "Hello World"

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.

2.2.1 File transfer with Tera Term

Uploading a new image using Tera Term is very similar to the process with ExtraPutty. Connect to the desired COM port at 115200 baud rate and press the spacebar to enter the bootloader. Press 'u' to upload a new image. Transfer the desired .bin file using XMODEM.

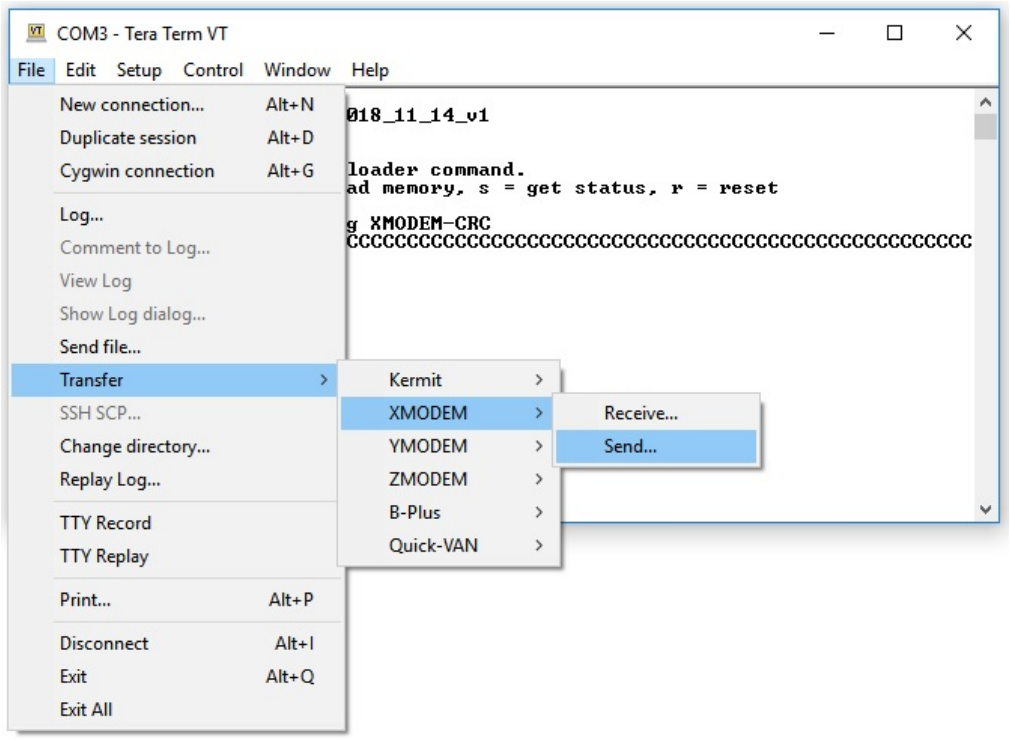


Figure 11- Application binary file upload using Xmodem in Tera Term

3 Programming SPI NVM for the VA108x0

The firmware update process happens in multiple steps to ensure that an incomplete or corrupted code image is not flagged as valid by the bootloader, and that any disruption during the upload process is recoverable.

During the XMODEM file transfer, the user image is reassembled in code RAM space inside the MCU. A CRC on each packet sent (part of the XMODEM protocol) ensures the correct data is transferred. Packets that fail the CRC check will NACK and be resent until successful or a timeout error occurs. At this point, nothing has been written to NVM, so if a power glitch or reset or other disruption occurs, the NVM is not corrupted.

After successful transfer, the complete user code image is copied from code RAM to the SPI NVM. First, the value of the user code reset vector is saved off in a temporary variable for later use. The user reset vector at 0x2004 in code RAM is then modified to a 'garbage' value, typically 0xFFFF:FFFF that will be flagged invalid by the bootloader (pointing outside of user code space 0x2000-0x1:FFFF). This is so that if the write process to the NVM is disrupted for some reason, the user reset vector will be invalid and the 'bad' or incomplete image will never attempt to run. The user code image with 'garbage' reset vector is then written to the SPI NVM (EEPROM or FRAM), starting at 0x2000. Once this is complete, the SPI NVM is read back and verified against the user code stored in RAM. If they match, indicating a successful write, then the final step is to replace the 'garbage' reset vector at 0x2004 in NVM with the correct value saved off earlier, marking the image as valid and bootable. The final reset vector write is also verified. After a successful firmware write, the user image in code RAM is executed (the 'garbage' reset vector in RAM 0x2004 replaced with the correct value, the IVT copied from 0x2000 to 0x0000, bit 2 in the RST_CNTL_ROM register cleared, and a software reset initiated).

3.1 M95 EEPROM on REB1 board

The default configuration of the VA108x0 Bootloader is with the *REB1_M95M01* symbol defined in bootloader.h, indicating use of the flash programming algorithm for the M95 SPI EEPROM.

```
/* REB1_M95M01 */
/* When defined, use programming algorithm for SPI EEPROM, specifically M95M01
   128kB EEPROM used on the REB1 evaluation board. Undef for use of F-RAM
   instead of SPI EEPROM */
#define REB1_M95M01
```

Figure 12- bootloader.h configuration set up for use with M95M01 EEPROM

This symbol being defined invokes the use of the *CopyApplicationImageToEEPROM()* function call inside of the bootloader state machine, when in the *ST_COPY_ROM* state. The M95M01 write/read algorithms are in m95m01.c.

3.2 Cypress FRAM

To use the bootloader with the Cypress FRAM, comment out or otherwise undefine the symbol *REB1_M95M01*. This invokes the use of the *CopyApplicationImageToFRAM()* function call

inside of the bootloader state machine, when in the *ST_COPY_ROM* state. The Cypress FRAM write/read algorithms are in cyp15b102.c.

3.3 Adapting code for other memories

The bootloader can be adapted for use with other memory types. Using other memories involves creating a custom implementation of the *CopyApplicationImageToXXXXX()* function in bootloader.c, calling the write/read algorithms contained in a *devicePartNumber*.c source file. Then a symbol representing that memory type can be defined to determine at compile time which *CopyApplicationImageTo_____()* function to call when the bootloader is in the *ST_COPY_ROM* state.

4 Downloading an Example VA108x0 project

The software project (bootloader and example user application) will be in AN1216_Bootloader.zip on the VORAGO website. The bootloader and example application will include both Keil and IAR project files.

5 Initial programming of the Bootloader

The VA108x0 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.

5.1 Keil µVision IDE

Open the Keil project file for the bootloader, *reb1_va108xx_bootloader.uvprojx*. Under Project->Options for Target 'reb1_va108xx_bootloader', make sure the IROM1 start address is 0x0 and size is set to 0x2000 in the Target section. Make sure 'Use Memory Layout from Target Dialog' in the Linker tab is selected. Build the project. In the Flash->Configure Flash Tools dialog, select 'Use Target Driver for Flash Programming', click 'Settings', and select the 'VA108XX_M95M01_128KB' Programming Algorithm for use with the M95M01 SPI EEPROM, or 'VA108_FM25V20A_FRAM_128KB' for use with FRAM. Click OK and close the flash settings dialog. Connect the target board. Click Flash->Download or press F8 to program the bootloader to the target device.

5.2 IAR Embedded Workbench IDE

Open the IAR project file for the bootloader, *reb1_va108xx_bootloader.eww*. Under Project->Options, choose the category 'Linker' and select the tab 'Config'. Make sure the 'Override Default' checkbox for linker configuration file is checked, and that the file path points to *reb1_va108xx_bootloader.icf* in the project folder. Click 'Edit' to view the linker configuration file settings. Make sure the Vector Table .intvec start address is 0x0, and under 'Memory Regions' the start address is 0x0 and the end address is 0x1FFF. Click Save. Choose the category 'Debugger'. In the 'Setup' tab, under 'Driver', select J-link/J-Trace (for the REB1 on-board or Segger debug adapter). In the 'Download' tab, select 'Use flash loader(s)', select 'Override default .board file', and select 'flash_loader_m95m01.board' for the REB1 SPI EEPROM, or 'flash_loader_fm25v20a.board' for FRAM. Click OK to close the options dialog. Build the project. Connect the target board. Under Project->Download, click 'Download active application' to program the bootloader to the target device.

6 Adapting bootloader for other serial interfaces

This bootloader was designed for use with RS232/RS485 based communications, but it can be adapted for use with other serial interfaces with minimal changes to the core state machine. Mechanisms that enter that state machine and advance through states would require changes and removal of text prints, but the overall functionality and flow through the bootloader would remain the same. For robustness it is recommended to implement some sort of CRC check on the incoming image (as was done by using the RS232/485 XMODEM-CRC protocol) to maintain data integrity.

7 Conclusions

This application note has introduced the VORAGO Technologies VA108x0 Bootloader and examined its usage, mechanism of operation, and installation.

8 Common questions and issues

Q: How do I change the startup wait time?
A: Change the value of #define BOOTLOADER_MSEC_TIMEOUT in bootloader.h

Q: The 'm' (read memory) command doesn't seem to do anything?
A: The 'm' (read memory) functionality is not currently implemented. This functionality will be introduced in a future update.

9 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:

Oct 9, 2018 – Initial template created (V0.1)
November 12, 2018 – Section 1 completed (V0.2)
November 14, 2018 – Section 2 and 4-8 completed (V0.3)
November 16, 2018 – Section 3 (now section 4) completed (V0.4)
January 2, 2019 – Release version 1.0