AN200

USB BOOTLOADER WITH SHARED USBXPRESS® LIBRARY

Relevant Devices

This application note applies to the following devices:
C8051F320, C8051F321, C8051F326, C8051F327, C8051F340, C8051F341, C8051F342, C8051F343, C8051F344, C8051F345, C8051F346, C8051F347

1. Introduction

A bootloader (or code loader) enables convenient updating of device firmware in the field.

The term, “bootloader”, usually refers to a piece of firmware that resides in program code space on the device and possesses the ability to receive new application firmware through a generic communication channel, eliminating the need for a dedicated programming interface with associated programming circuitry. In other words, a bootloader enables "in-application reprogrammability" via a communication channel. This application note describes the design of a USB bootloader built using the USBXpress Application Programming Interface (API), as well as the modifications necessary to make an existing USBXpress application bootloader-aware. The USBXpress API is described in detail in “AN169: USBXpress Programmer’s Guide”.

2. USBXpress Bootloader Overview

A generic bootloader should be able to perform the following functions:

- Send and receive data via a communication channel
- Erase and update application firmware
- Determine the integrity of an application firmware image

In addition to the requirements of a generic bootloader, the following additional requirements are imposed on a USB bootloader that uses the USBXpress firmware library:

- The USBXpress library functions should be shared with the application firmware.
- The USBXpress API interrupt needs to be redirected depending on current operating mode (Application vs. Bootload mode).

Sharing the USBXpress library saves about 3.1 kB of code space that would otherwise have to be duplicated and would significantly reduce the space available for the application project. Moreover, the precompiled library directly claims the USB0 hardware interrupt, preventing the addition of another USBXpress library without modifications to the USBXpress source code itself.

Note: The USBXpress firmware libraries are provided as precompiled library files compatible with the Keil toolchain. As a result, the bootloader firmware is also compatible only with the Keil toolchain.

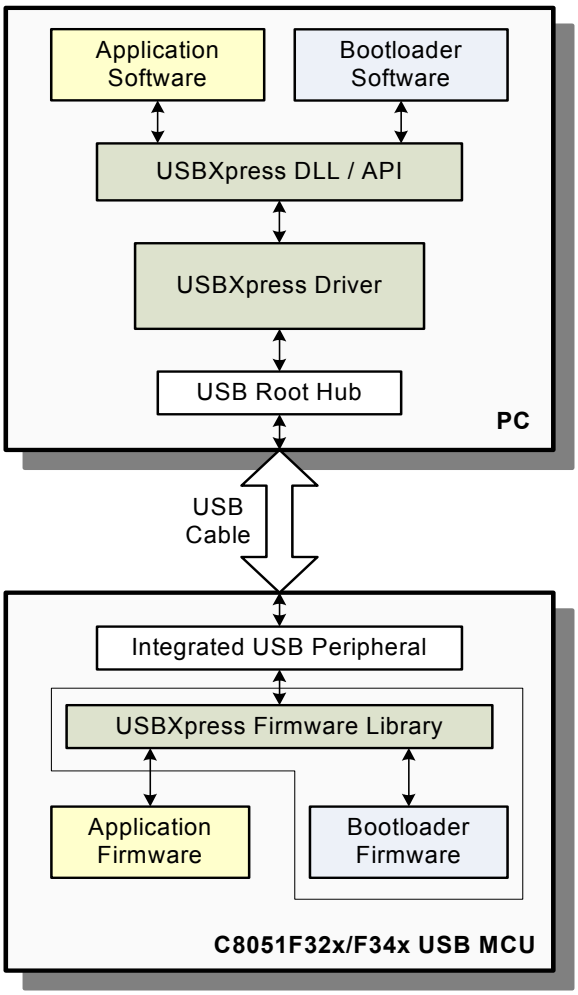


Figure 1. USBXpress Bootloader Overview

3. Bootloader Design

The USBXpress bootloader consists of the bootloader firmware that runs on the MCU and the bootloader software that runs on the PC.

3.1. Bootloader Firmware Memory Usage

The bootloader firmware uses a total of 5 kB of Flash memory. This includes the shared USBXpress library and the shared USB descriptors. The bootloader firmware occupies two sections of code memory:

1. Space from code address 0x0000 through 0x1FFF (4.5 kB)
2. The last page of usable flash memory that includes the lock byte (0.5 kB)

The reason the bootloader has to occupy the area starting from 0x0000 is because a device reset should always execute the bootloader first, which will then decide how to proceed. Instead of placing the entire 5 kB of the bootloader contiguously, 512 bytes of the bootloader firmware are placed on the last page (the page with the lock byte) because that page is not erasable by firmware and thus cannot be used for the application firmware. Figure 2 shows the bootloader firmware code memory map.

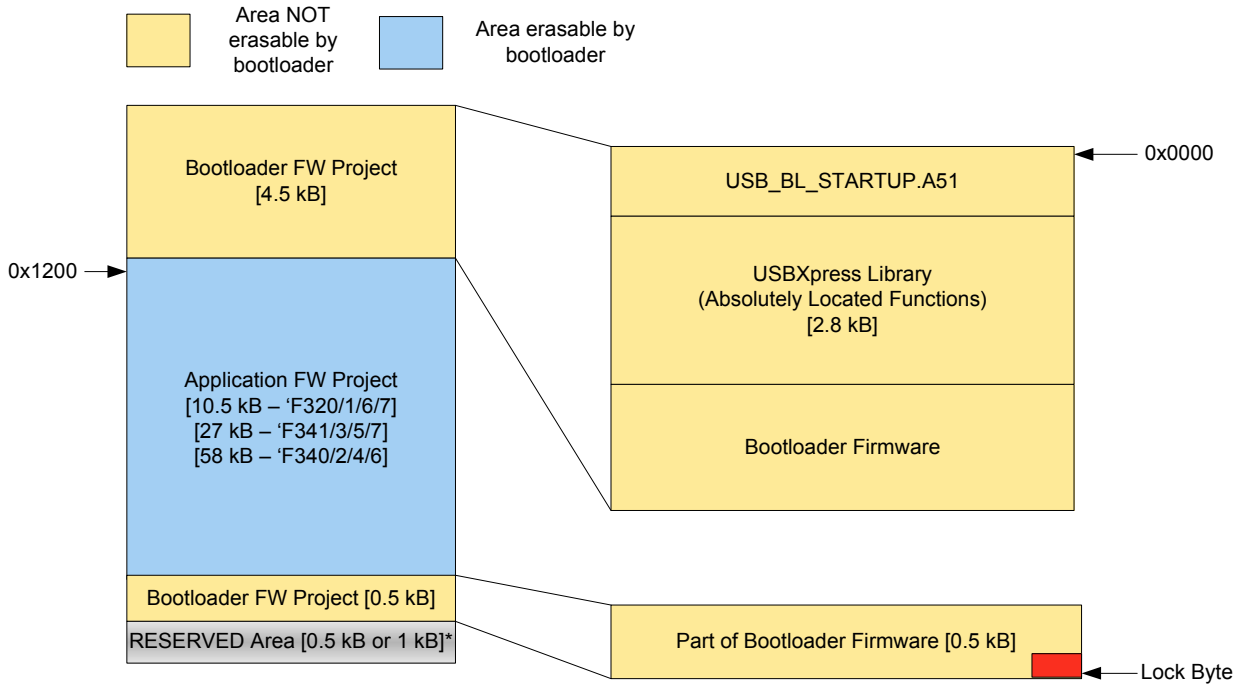


Figure 2. Bootloader Firmware Code Memory Map

Because the bootloader firmware and application firmware are active at mutually-exclusive times, virtually the whole RAM area is usable by each project when it is active. The only area in RAM (in "data" space) used by the bootloader firmware while the application firmware is active is a single-bit variable (DEVICE_MODE) located in the bit-addressable area of the data memory at address 0x2F.7 (bit address 0x7F). In addition, the USBXpress library included within the bootloader firmware reserves a range of addresses in the "xdata" space (on-chip XRAM), which is common to both the projects. More information about the XRAM space used by the USBXpress library is available in Table 1 of “AN169: USBXpress Programmer's Guide".

In addition to sharing the USBXpress firmware library, the bootloader firmware also hosts a set of USB descriptors (USB_BL_USB_Descriptor.c) that can be used by the application firmware. More details about this feature can be found in "4.1.2. USB Descriptors and Bootload Request" on page 10.

3.2. Bootloader Firmware Features

The bootloader firmware includes the following features that are required for any USB bootloader:

- Send and receive data via the USB interface (using the USBXpress API)
- Erase and update application firmware

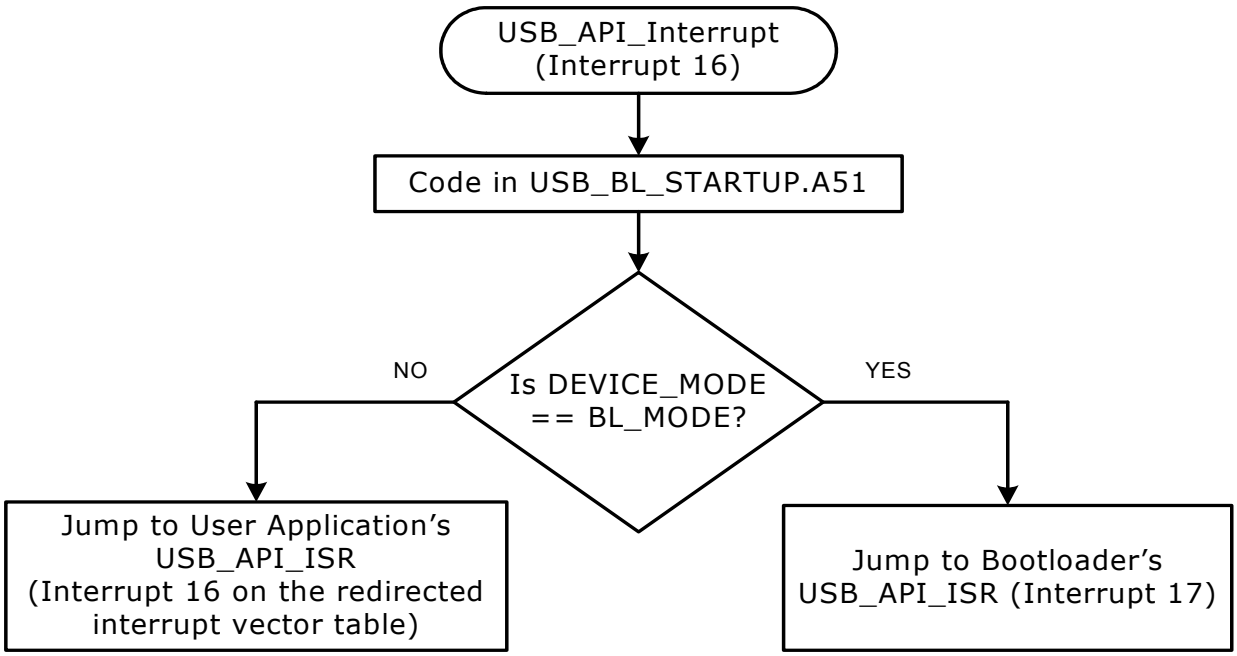
The bootloader firmware also includes advanced features that allow the bootloader to tolerate unexpected events, such as power failures or a cable disconnects during the firmware update process. These fault-tolerance enhancements are described in the following sections.

3.2.1. Self-Update Prevention

The bootloader firmware contains code that includes boundary checks to prevent erasure of the bootloader firmware itself. The SetPage, ErasePage, and WritePage commands work only for pages that are within the application code space. This boundary check ensures that the bootloader is never erased in the case of an accidental command from the PC. Consequently, the USBXpress firmware library cannot be updated because the library is part of the bootloader.

3.2.2. Interrupt Redirection

The hardware interrupt vectors from page 0 are redirected by the bootloader into the application space. All interrupts, except Reset and USB0, are redirected. The Reset interrupt causes the MCU to enter the bootloader and perform the signature check before jumping to the reset vector of the application firmware. The USB0 interrupt is claimed by the USBXpress firmware library included with the bootloader firmware. The USB API interrupt (the pseudo interrupt generated by the USBXpress API) is redirected to either the USB API ISR in the bootloader firmware or the USB API ISR in the application firmware depending on the value of the state variable, DEVICE_MODE. Figure 3 shows the interrupt redirection control flow. Because all hardware interrupts are redirected to the application space using the static code on page 0 (which is never erased), errors during an application firmware download, which could render the device non-bootloadable, are prevented.



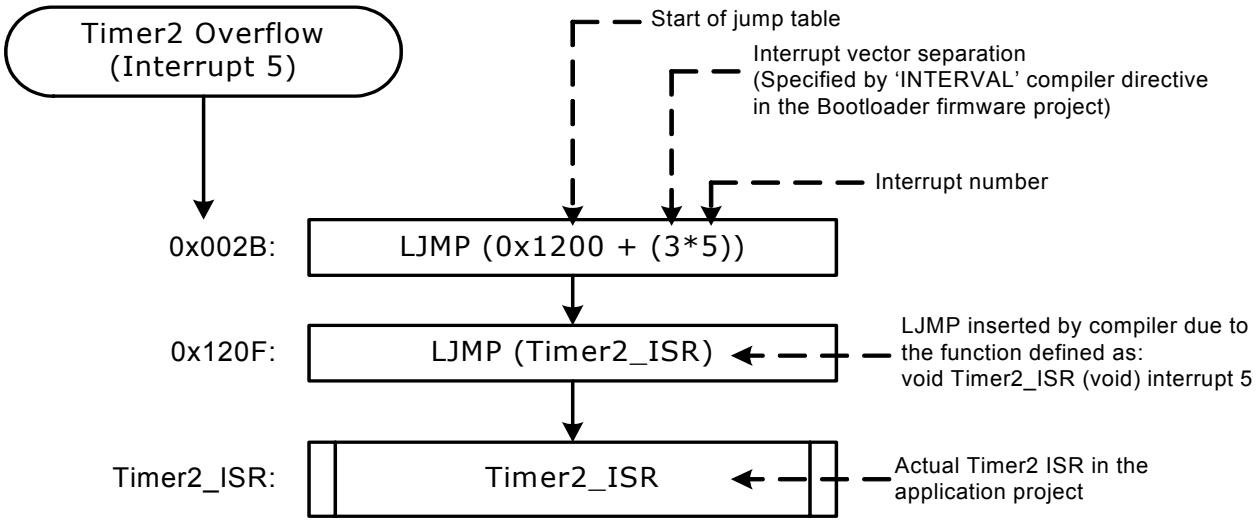


Figure 3. Interrupt Redirection

3.2.3. Post-Reset Signature Check

One of the most important fault-tolerant features included in this bootloader is the post-reset signature check. On a device reset, the bootloader code that is first executed checks whether a specific 16-bit signature value exists in the last two bytes of the application code space. If this check passes, the application is started by the bootloader. If it fails, the bootloader waits for a new application firmware image. To ensure that the device is never rendered non-bootloadable, the application firmware update process is designed such that the page containing the signature is the first to be erased and the signature value is the last data to be written.

3.2.4. Non-Resident Flash Keys

One of the causes of Flash memory corruption is random code execution, which can happen due to an out-of-spec power ramp-up, among other factors. To reduce the possibility of Flash corruption due to random code execution, the Flash unlock key codes are not stored in the non-volatile code memory within the bootloader firmware. In the absense of these key codes, the MCU will not allow any Flash page to be erased or changed, and any attempt to do so will cause a Flash Error Reset.

The Flash keys are sent to the bootloader firmware to be held in RAM at the beginning of the firmware update process and are zeroed out at the end of the bootloading process.

3.2.5. Page-by-Page CRC Check

To ensure that the image sent is written to Flash memory without any errors, a page-by-page CRC check is performed during the firmware update process. The 16-bit CCITT CRC polynomial is implemented on the bootloader software and the firmware. If the 16-bit page CRC calculated on the software image does not match the corresponding CRC reported by the device, the firmware update process is halted, and the user is notified.

3.2.6. Fail-Safe Firmware Update Option

Once a valid application firmware image is present (indicated by a valid signature), the bootloader launches the application firmware on device reset. In this case, the normal procedure to reenter bootload mode is to make the application firmware call the bootloader. Because the application firmware is an unknown, it may become stuck in a state where it is not able to call the bootloader, which could render the device non-bootloadable. To prevent this situation, a fail-safe option is included in the bootloader firmware that checks a port pin state on device reset. If this port pin(P3.0) is pulled low at that time, it will not proceed with the signature check and will stay in bootload mode instead. This option is enabled by default and can be modified to use a different pin or disabled as desired by the user. To disable this feature, remove the macro definition, BOOTLOADER_PIN_OVERRIDE, from the file, USB_BL_Main.h.

3.3. Bootloader Firmware Commands

The commands supported by the bootloader firmware are listed in Table 1.

Table 1. Bootloader Firmware Commands

Command	Description
Get Device Info	Retrieves key device information: Device Code, Firmware Revision, Signature Bytes.
Set Page	Sets the “current” application firmware page to be operated upon by future commands.
Erase Page	Erases the currently-selected page.
Write Page	Writes to the currently-selected page and returns the page CRC.
CRC on Page	Computes and returns the CRC of the currently-selected page.
Software Reset	Causes a software reset of the device.
Set Flash Keys	Sets the Flash Write/Erase Key Codes to be used by Erase Page, Write Page, and Write Signature commands.
Write Signature	Writes the signature bytes to the last two bytes of the application firmware.

3.4. Modifying Bootloader Firmware

The bootloader firmware is built with a specific version of the USBXpress library using the Keil toolchain. The pre-built OMF and hex files of the bootloader firmware for each MCU device family are provided in the AN200SW.zip package. There is usually no need to modify the bootloader firmware in any way, but, if additional features need to be added or existing features need to be removed, it can be done by modifying the bootloader firmware source code supplied in the package. Once the desired modifications are made to the source code, the project can be rebuilt using the included batch files.

The USBXpress API functions and the USB descriptors are shared between the bootloader firmware and the application firmware. Due to this sharing, these are absolutely located at fixed locations in code space using the linker command line options. The linker commands to locate these functions are generated using a spreadsheet that is also included in the package. If the locations of any of the USBXpress functions or descriptors need to be changed, the linker commands need to be recalculated using the spreadsheet, and the resulting command should be used in the linker command input text files (*_LINK.TXT).

3.5. Bootloader Software Features

The bootloader software is a GUI built using Visual C++ 6.0. It lists all available USBXpress devices that are in bootload mode and allows the user to select the target device and the hex file to be downloaded. Once a device and hex file are chosen, it displays the key information retrieved from them, such as device type and firmware revision. It downloads the hex file to the device and verifies the CRC page-by-page. The following sections describe some of the significant features of this software.

3.5.1. Device Mode Detection

It is essential to determine whether a device is in bootload mode or application mode before the PC application starts communicating with it. To make this determination without having to open the handle of each available USBXpress device and sending commands, a simple encoding scheme has been implemented in the "Serial String" of the device's USB descriptors. In this scheme, the device enumerates with different serial numbers depending on whether it is in bootload mode or in application mode. In bootload mode, it uses the same serial string as the application mode, but appends one special predetermined character at the end. The default value for this character is the tilde (~) and is defined in USB_BL_Interface.h. This header file is shared between the bootloader firmware and the bootloader software. The bootloader software is able to use this and list only the devices that are currently in bootload mode.

3.5.2. Intelligent Hex File Processing

To be compatible with this bootloader system, the application firmware image needs to contain key information in the last few bytes of the firmware image. The bootloader software is able to process a hex file that contains a firmware image and extract this key information.

- The MCU device family for which the hex file is meant ('F320/1, 'F326/7, 'F34x).
- The firmware revision specified by the hex file.

3.5.3. Version Compatibility Check

As soon as the bootloader software is connected to a bootloadable device, it issues the Get_Device_Info command and retrieves key details of the device. It then matches this information with the information retrieved from the hex file to determine if the hex file is compatible with the device. The device and hex file are matched on two parameters:

- MCU device family
- Firmware revision

If the revision of the firmware in the hex file is smaller than the one in the device, a warning is displayed, and user confirmation is sought before a firmware "downgrade" operation proceeds.

3.5.4. Surprise Removal Handling

Software applications communicating with a removable USB device need to have code to deal with an unexpected removal of the device. Lack of this code would lead to crashes and unexplained behavior. Unexpected or surprise removal could occur due to a device power loss, cable disconnect, or some other event. The bootloader software is able to detect a surprise removal, abort any ongoing operation, and close the device handle. The Silicon Labs MCU Knowledgebase contains more details on surprise removal handling, and can be accessed from the following URL: http://www.silabs.com/MCU.

3.6. Modifying Bootloader Software

The bootloader software GUI is provided as a pre-built executable. There is usually no need to modify this software, but, if additional features need to be added or existing features need to be removed, it can be done by modifying the bootloader software source code supplied in the package. The source code can be rebuilt using Microsoft Visual C++ 6.0.

3.7. Firmware Update Steps

The firmware update process can be broken down into a series of steps. Following these steps in the specified order minimizes the chance of rendering the device non-bootloadable, even if unexpected events, such as a power failure, occur during the download. Perform the following steps with the bootloader software:

1. Open a handle to the target device, and issue a Get Device Info command.
2. Process the hex file selected by the user, and compare the Device Code and Firmware Version from the file with the data retrieved using the Get Device Info command. If any compatibility issues are detected, notify the user.
3. After confirming that the user wants to proceed with the update, begin the update process by following Steps 4 through 8.
4. Set the flash write/erase key codes using the Set Flash Keys command.
5. Erase the last application page using the Erase Page command. This erases the existing application firmware's signature and must be done before the bootloader firmware will allow any other page to be erased.
6. For every specified page in the hex file beginning with the first application page, perform a page erase, page write, and page CRC check. Verify that the page CRC returned by the device matches the page CRC calculated on the hex file image. If there is a mismatch, notify the user, and abort the download process.
7. After all the application pages have been written to device code memory and verified, use the Write Signature command to write the signature bytes to the end of the application firmware.
8. Reset the device using the Software Reset command, and close the device handle. Upon device reset, the bootloader firmware launches the updated application firmware after verifying the presence of the signature bytes.

4. Making a USBXpress Application Bootloader-Aware

4.1. Application Firmware Modifications

Application firmware designed to work with the USBXpress API as described in “AN169: USBXpress Programmer’s Guide” requires some modifications before it can work with the bootloader firmware. The following steps list the necessary changes to an existing USBXpress application. The same steps can also be adopted when designing a new USBXpress application. Figure 4 shows the application firmware code memory map, which can be used as a quick reference on the space available, and bytes reserved for special use.

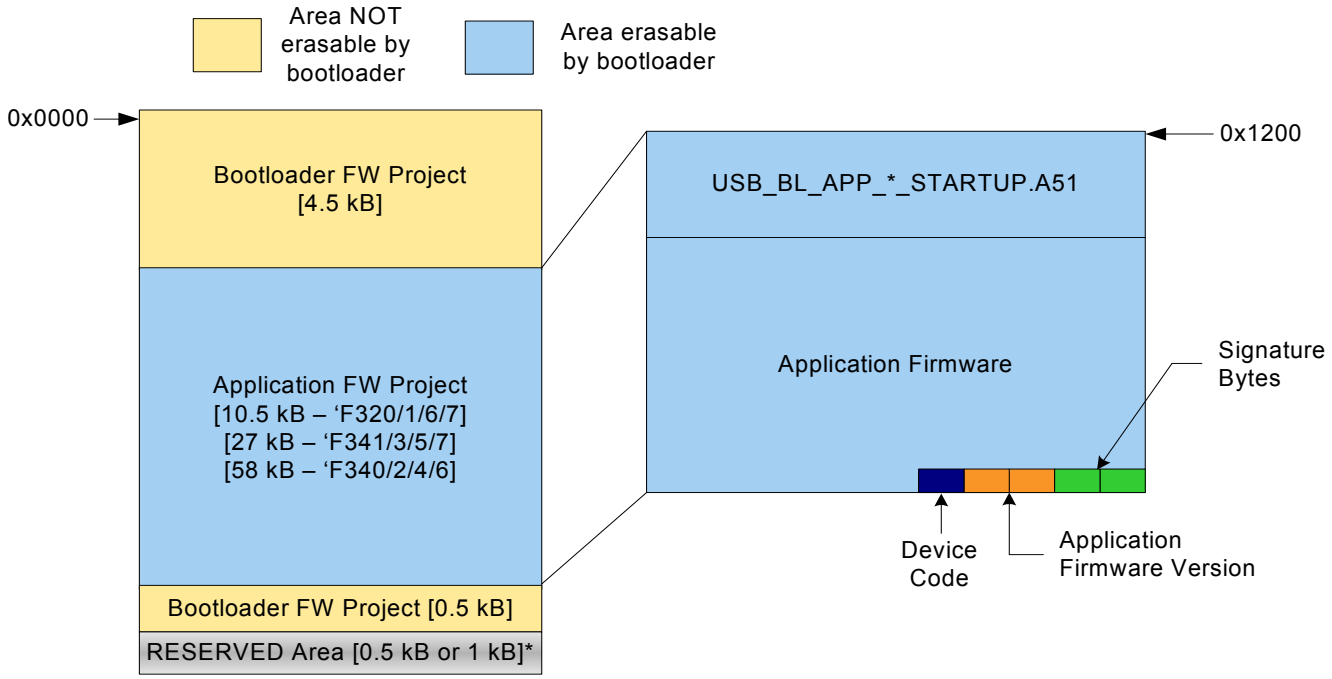


Figure 4. Application Firmware Code Memory Map

4.1.1. Startup Code

Ready-to-use startup code for the different MCU families is provided in the bootloader source code package, AN200SW.zip. The MCU device families and the corresponding startup file names are listed in Table 2.

Table 2. Startup File Names

MCU Part Numbers	Startup File Name
C8051F320/1	USB_BL_APP_F320_1_16k_STARTUP.A51
C8051F326/7	USB_BL_APP_F326_7_16k_STARTUP.A51
C8051F340/2/4/6 (64 k Flash)	USB_BL_APP_F34x_64k_STARTUP.A51
C8051F341/3/5/7 (32 k Flash)	USB_BL_APP_F34x_32k_STARTUP.A51

The startup files are essentially the standard Keil startup files with the following additions/modifications:

- Declares the locations of the shared USB Descriptors
- Declares the locations of the shared USBXpress library functions
- Reserves the XRAM space used by the USBXpress library
- Reserves the bit variable, DEVICE_MODE, used by the bootloader firmware to redirect the USB API ISR.
- Defines the device code and firmware version values.
- Relocates the reset vector for the application firmware from 0x0000 to 0x1200.

The startup file suitable for the MCU being used should be added to the application firmware project.

4.1.2. USB Descriptors and Bootload Request

The shared USB descriptors should be declared as externals before they can be used. These declarations are made in the header file, USB_BL_APP_Shared.h, which is included in the AN220SW.zip package. The header file should be included in any application firmware source file that uses the USB descriptor variables or makes a call to the BOOTLOAD_REQ() function.

Because the USB descriptors are now shared and located within the bootloader firmware area, any previously-existing USB descriptor definitions should be removed from the application firmware. The USB descriptor variable names should be replaced with the external variable names from the above header file wherever they are used in code. Typically, these variables would only be used in the call to the USBXpress USB_Init function.

4.1.3. Command Line Options

The following commands should be added to the Keil C51 compiler command line:

INTVECTOR(0x1200) INTERVAL(3)

These commands inform the Keil compiler that the reset vector begins at code address 0x1200, and the interrupt vectors are spaced three bytes apart.

A linker command that directs the Keil BL51 linker to keep the code within an address range should be added. This address range is specific to the MCU being used and is listed in Table 3.

Table 3. Linker Command Line Options

MCU Part Numbers	Linker Command Line
C8051F320/1	CODE(0x1200-0x3BFD)
C8051F326/7	CODE(0x1200-0x3BFD)
C8051F340/2/4/6 (64k Flash)	CODE(0x1200-0xF9FD)
C8051F341/3/5/7 (32k Flash)	CODE(0x1200-0x7DFD)

4.1.4. USBXpress Firmware Library

Because it is now shared, the USBXpress firmware library should be removed from the application project and stored within the bootloader firmware area.

4.1.5. Space Constraints

The bootloader firmware and the USBXpress library occupy some code space and use some areas of RAM. Table 4 lists the application firmware code and XRAM limits for the various MCU device families. The "data" RAM space can be fully used by the application firmware except for one bit in the bit-addressable area. The last bit in this area at byte address 0x2F.7 (bit address 0x7F) is reserved for the DEVICE_MODE variable.

Table 4. Application Firmware Allowed Address Ranges

MCU Part Numbers	Code Address Range	XRAM Address Range
C8051F320/1	0x1200-0x3BFD	0x0000-0x053F
C8051F326/7	0x1200-0x3BFD	0x0000-0x038B
C8051F340/2/4/6 (64k Flash)	0x1200-0xF9FD	0x0000-0x053F
C8051F341/3/5/7 (32k Flash)	0x1200-0x7DFD	0x0000-0x053F

4.1.6. Adding a Call to the Bootloader

Code allowing one or more conditions to trigger a call to the bootloader should be added to the application firmware. The trigger condition can be anything suitable for the application, such as a button press detected via a GPIO pin. It is recommended that a software trigger condition that calls the bootloader firmware when a specific command is received from the PC application software be included. This allows the user to update the device firmware by only interacting with the application software, without the need for any direct interaction with the hardware via buttons or other means. The device can be put in bootload mode by the application firmware when it makes a call to the “BOOTLOAD_REQ ();” function. This function is declared in the header file, USB_BL_APP_Shared.h. Note that, independent of the methods discussed in this section, the bootloader can also be invoked using the fail-safe option discussed in "3.2.6. Fail-Safe Firmware Update Option" on page 6, if it is enabled.

4.2. Interrupt Latencies

Page 0 of the code space is part of the bootloader and is not erasable. The hardware interrupt vectors are all present on page 0. To allow the application firmware to use and modify interrupt vectors, all the interrupts are redirected from the page 0 hardware interrupt vector locations to the first application page. The USB0 hardware interrupt is not redirected. See "3.2.2. Interrupt Redirection" on page 4 for more details.

Due to this redirection, additional interrupt latencies will be observed in application firmware's ISRs. For all the redirected interrupts (except the Reset interrupt), an additional LJMP instruction is executed. The reset interrupt triggers a signature check by the bootloader before it jumps to the reset vector of the application firmware.

4.3. Non-Volatile Storage Considerations

It is possible to use the on-chip flash memory as non-volatile storage for data, such as calibration constants. When this is done by the application firmware, care should be taken to only use the space within the application firmware area for storage. The limits are listed in Table 4.

4.4. Locking Flash Memory

The code memory can be protected from access via the debug interface (C2) as described in the "Flash Security" chapter in the MCU datasheet. If this is desired, it is recommended that the entire code space be locked rather than locking only some sections of code. Locking only a section of code space can cause problems with code execution. For example, if the pages occupied by the bootloader firmware are locked, leaving the application firmware space unlocked, code in the application firmware that wants to read the shared USB descriptors will cause a Flash Error Reset because code from unlocked space is trying to read code from locked space.

4.5. Application Software Modifications

The only necessary modification to the application software is to add a way to read the device serial number and use that to recognize whether the device is in bootload mode or application mode. See "3.5.1. Device Mode Detection" on page 7 for details on device mode. The application software needs to look at available USBXpress devices and only attempt to open a handle when the serial string does not end with a “~” character. The suffix character for the serial string used by the bootloader firmware to indicate bootload mode can be modified in USB_BL_Interface.h. If a device is found in bootload mode, the application software can automatically launch the bootloader software or prompt the user to do so.

5. . Test Panel Example Application

The AN200SW.zip package includes a modified version of the Testpanel USBXpress example application that demonstrates the use of the bootloader. The following components are included:

- Testpanel PC application software
- Testpanel device-specific firmware examples

For each device, there are two versions of the Testpanel firmware example. The second one (Testpanel2) differs from the first one (Testpanel1) by sending an additional byte of data to the PC application. This difference serves to demonstrate the change in the application firmware after a bootload operation. The steps to be followed for a demonstration of the bootloader with the Testpanel example can be found in the file "Demo.txt" included in AN200SW.zip.

NOTES:

CONTACT INFORMATION

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
Tel: 1+(512) 416-8500
Fax: 1+(512) 416-9669
Toll Free: 1+(877) 444-3032

Email: MCUinfo@silabs.com
Internet: www.silabs.com