Application Report Small Size Custom BSL Based on ROM BSL for MSP430[™] FRAM MCUs

TEXAS INSTRUMENTS

Gary Gao

MSP430 Applications

ABSTRACT

This application report describes a small size custom BSL based on the ROM BSL for MSP430[™] FRAM devices. The boot code size of the custom BSL is less than 50 bytes. The custom BSL can download firmware or update firmware in the target device by UART or I²C (based on the ROM BSL). A GUI for a PC host and example code for the MCU are provided. This custom BSL is easily ported to other devices and supports a debug mode to help with debugging application code.

The software package with examples and the GUI can be downloaded from https://www.ti.com/lit/zip/slaa968.

Table of Contents

1 Introduction	2
1.1 Implementation	2
1.2 Boot Project Introduction	3
1.3 Application Project Introduction	3
1.4 MCU Host Code Introduction	4
2 Example Project Introduction	6
2.1 MCU as Host Example	6
2.2 PC as Host Example	7
3 GUI Introduction	8
3.1 Download Firmware by UART BSL With PC	8
3.2 Generate Linker File and Header File for Boot and Application Project	8
3.3 Convert TXT Image to Header File for Host Project.	9
4 Porting Example Code to Other MCUs	10
5 References	10

Trademarks

MSP430[™] is a trademark of Texas Instruments. All trademarks are the property of their respective owners.



1 Introduction

All of the MSP430 FRAM devices have a ROM-based BSL (for more information, see the MSP430[™] FRAM Devices Bootloader (BSL) User's Guide). Most application use-cases update the firmware using UART or I²C. However, the ROM BSL needs two pins, RST and TEST, for hardware invoke.

The ROM BSL also supports software invoking but it has these risks:

- If an unexpected power off happens during the firmware updating, the device cannot enter BSL mode at the next power up if the device BSL does not support blank memory detection, because the software invoke code has been erased.
- The BSL password is changed by each new firmware upload, so it is easy to send a wrong password, which leads to a mass erase. If the device BSL does not support blank memory detection and the mass erase with wrong password is enabled, the device cannot enter BSL mode at the next power cycle or BOR.

To avoid using the \overline{RST} and TEST pins for hardware invoke and the risks mentioned above for software invoke, this application report provides a solution to solve those problems.



Figure 1-1. Block Diagram of the Update System

For more custom functions such as using a SPI interface or secure features, refer to to MSP430FRBoot – Main Memory Bootloader and Over-the-Air Updates for MSP430 FRAM or Crypto-Bootloader - Secure In-Field Firmware Updates for Ultra-Low Power MCUs.

1.1 Implementation

The custom BSL described in this application report avoids the limitation of the ROM BSL software through the following features:

1. Separate the memory to application area and boot code area. The boot code can always exist to make sure the device can go into BSL if needed.

Memory Range	Function		
Other memory	Application code		
0xFF40 to 0xFF80	Boot code		
0xFF80 to 0xFFFE	JTAG and BSL signature and interrupt vector		
0xFFFF to 0xFFFE	Boot code start address		

Table 1-1. Memory Assignment

The linker file can handle this work and the GUI support to generate the linker files for different devices for CCS or IAR IDEs.



2. Use RAM-based interrupt vector instead of FRAM-based interrupt vector. This makes sure that the BSL password is fixed and is not changed when updating the firmware. The user can define different BSL password by modifying the firmware of boot code before downloading it into the device.

For FRAM devices, support remapping interrupt vectors into the end area of the RAM by setting the SYSRIVECT bit in the SYSCTL register.

3. Define a variable flag _jump_key in FRAM memory at 0xFF3E to guide device go into BSL or application. This flag can just be changed at the start of the application code or when need jump from application to BSL. So when power off or other failure during the firmware updating, the device can go into the BSL or application correctly by check the flag _jump_key at next power cycle or BOR.

1.2 Boot Project Introduction

The boot project include two files: main_boot.c and a linker file generated by the GUI (see Section 3.2 for details).The boot project checks the flag _jump_key to decide go into application code or ROM BSL. That is why this boot code is so small, at less than 50 bytes.



Figure 1-2. Flow Diagram of Boot Code

The BSL password is defined by the boot code image and the value is located at address 0xFFE0 to 0xFFFF (32 bytes). The value in 0xFFFE and 0xFFFF is the boot code start address that is created by the compiler (cannot be changed), but other values can be changed (by default, all are 0xFF). The user can define the password in the array named bsl_password in the main_boot.c file.

1.3 Application Project Introduction

The application project includes three files: main_app.c, bsl_app.h, and a linker file. bsl_app.h and the linker file are generated by the GUI (see Section 3.2). The application project must include functions for the device in the main_app.c file that work with the boot code and host. The blue blocks in Figure 1-3 and Figure 1-4 must be included in the application code.









Figure 1-4. Flow Diagram of Application UART or I²C ISR

The application project supports debug mode for debuging the user tasks. In debug mode, the application project can download into the target device directly by JTAG and can run the code in debug mode. Define "DEBUG" at the start of main_app.c to keep the application project in debug mode. But when you want to download the firmware by PC GUI or the host MCU by this BSL solution, comment out the define "DEBUG" to exit debug mode.

1.4 MCU Host Code Introduction

This code is used for an MCU or other processor as a host to control the process of firmware update. It includes the target device firmware in application_image.h file that is converted from .txt image file by the GUI (see Section 3.3 for details). It also includes the BSL password in the main.c file in the BSL_PW_RESET array. This password is the value in the boot code image at addresses 0xFFE0 to 0xFFFF (32 bytes). Figure 1-5 shows the process of the host project.





Figure 1-5. Flow Diagram of the Host Project



2 Example Project Introduction

This application includes examples for MSP430FR2311 and MSP430FR5969 in CCS and IAR versions, using the MSP430FR5969 as the host MCU. The hardware used in this application is MSP-EXP430FR2311 and MSP-EXP430FR5969.



Figure 2-1. MSP-EXP430FR2311 and MSP-EXP430FR5969

Table 2-1. eUSCI Peripheral Connections

CI	Host device		Target device		
	Signal	MSP430FR5969	Signal	MSP430FR5969(1)	MSP430FR2311
	RXD	P2.6/UCA1RXD	TXD	P2.0/UCA0TXD	P1.7/UCA0TXD
UARI	TXD	P2.5/UCA0TXD	RXD	P2.1/UCA0RXD	P1.6/UCA0RXD
120	SCL	P1.7/UCB0SCL	SCL	P1.7/UCB0SCL	P1.3/UCB0SCL
120	SDA	P1.6/UCB0SDA	SDA	P1.6/UCB0SDA	P1.2/UCB0SDA

2.1 MCU as Host Example

The following process runs the code when using an MCU as the host. This exmaple uses the CCS project for the MSP430FR2311 UART interface:

- 1. Connect the hardware signals as shown in Table 2-1.
- 2. Import the projects, including the target and host projects, into CCS (see Figure 2-2).

) Select search-directory:	C:\Myworks\Image_download\BSL\MSP430_Host_UAR	T\Exi	Browse
) Select archive file:			Browse
iscovered projects:			
🗌 💼 FR2311_App_I2C	[Target_MSP430FR2311/FR2311_App_I2C]	^	Select All
The set of the se	t [Target_MSP430FR2311/FR2311_App_uart]		Deselect All
FR2311_BOOT [1a FR5696 Host RS]	IPGet_IVISP430FR2311/FR2311_B00tj IPC_IHost_MSP430FR5969/FR5696_Host_RSI_I2C1	L.	Deserver rain
	LUART [Host MSP430FR5969/FR5696 Host BSL UART]		Refresh
🗌 👕 FR5969_App [Tai	rget_MSP430FR5969/FR5969_App_UART]		
🗌 👕 FR5969_Boot [Ta	rget_MSP430FR5969/FR5969_Boot]		
T T FR59691 App 120	[Target MSP430FR5969/FR59691 App 12C]	~	
Automatically import ref	erenced projects found in same search-directory		
Copy projects into works	pace		

Figure 2-2. Import Target and Host Projects in CCS



- 3. Select and build the FR2311_Boot project to define the password by modifying the array named bsl_password in the main_boot.c file and generating the FR2311_Boot.txt image file.
- 4. Download the FR2311_Boot.txt image into the MSP430FR2311 MCU through JTAG using the eZ-FET.
- 5. Build the FR2311_App_uart project to generate the FR2311_App_uart.txt firmware (make sure comments define as "DEBUG").
- 6. Convert the FR2311_App_uart.txt firmware to a header file by the GUI (see Section 3.3)
- 7. Copy the contents of convert_output.h into the host project file application_image.h.
- 8. Modify the BSL password array BSL_PW_RESET based on the value at addresses 0xFFE0 to 0xFFFF in the boot code image of the host project.
- 9. Build the host project and download it to the MSP-EXP430FR5969.

10. Push button S1 on the MSP-EXP430FR5969 and LED1 blinks on the MSP-EXP430FR2311.

2.2 PC as Host Example

This is based on the backchannel UART feature of MSP-FET or eZ-FET. The GUI uses the backchannel UART to download the image to the target device.

- 1. Connect the target device UART pins (see Table 2-1) with the MSP-FET or eZ-FET backchannel UART pins.
- 2. Import both target projects to CCS (see Figure 2-3).

🎁 Import CCS Projects	_		×
Import CCS Projects Import existing CCS Project	ts or example CCS Projects.	P	
 Select search-directory: Select archive file: 	C:\Myworks\Image_download\BSL\MSP430_Host_UART\Ex;	Browse	
Discovered projects:			
FR2311_App_I2C FR2311_App_uar FR2311_Boot [Ta FR5696_Host_BS] FR5696_Host_BS] FR5969_App [Ta FR5969_Boot [Ta FR5969_Boot [Ta FR5969_Boot [Ta FR5969_Boot [Ta Copy projects into works Open <u>Resource Explorer</u> to b	[Target_MSP430FR2311/FR2311_App_I2C] t [Target_MSP430FR2311/FR2311_App_uart] urget_MSP430FR2311/FR2311_Boot] L_I2C [Host_MSP430FR5969/FR5696_Host_BSL_I2C] L_UART [Host_MSP430FR5969/FR5696_Host_BSL_UART] urget_MSP430FR5969/FR5969_App_UART] urget_MSP430FR5969/FR5969_Boot] C [Target_MSP430FR5969/FR5969]_App_I2C] erenced projects found in same search-directory pace prowse a wide selection of example projects	Select / Deselect Refres	All h
?	Finish	Cance	1

Figure 2-3. Import Target Projects in CCS

- 3. Select and build the FR2311_Boot project to generate the FR2311_Boot.txt image file.
- 4. Modify the value at addresses 0xFFE0 to 0xFFFD (do not change the value in 0xFFFE to 0xFFFF, which is the start address of the boot code) in FR2311_Boot.txt image if you want to define a different BSL password. This example keeps the default value of 0xFF.
- 5. Download the FR2311_Boot.txt image into MSP430FR2311 through JTAG using the eZ-FET.
- 6. Build the FR2311_App_uart project to generate the FR2311_App_uart.txt firmware. Make sure comments define as "DEBUG".
- 7. Use the GUI to download the FR2311_App_uart.txt firmware (see Section 3.1 for details)
- 8. The LED1 should blink on the MSP-EXP430FR2311.



3 GUI Introduction

The included GUI support these functions:

- 1. Download the firmware by UART BSL
- 2. Generate the linker file and header file for the boot and application projects
- 3. Convert TXT image to header file for host project

3.1 Download Firmware by UART BSL With PC

Before downloading the firmware with the GUI, two files should be prepared: the application firmware and the BSL password. The application firmware is generated by the application project, and the BSL password is based on the value at addresses 0xFFE0 to 0xFFFF in the boot code image. Figure 3-1 shows the format of the password file. An example password file is provided in the installation directory under \MSP430FR_Bootloader_1_00 \GUI_V1\Firmware.

Figure 3-1. BSL Password File Example

Then follow the steps in Figure 3-2 to download the firmware.

Bootloader Host -	
MoreOption	
1 → Application firmware file: C:/Myworks/Image_download/BSL/MSP430_Host_UART/ Choose.tx 2 → Password file: C:/Myworks/Image_download/BSL/MSP430_Host_UART/ Choose.tx 3 → Download	file file
Send package successfully! Operation success! Send the firmware Send package successfully! Operation success! Send package successfully! Operation success! Send package successfully! Operation success! Send package successfully! Operation success! Send firmware successfully! Set the PC to application start address	
Clear	
🔱 Texas Instruments	

Figure 3-2. Steps to Download Firmware With GUI

3.2 Generate Linker File and Header File for Boot and Application Project

The GUI can generate linker files for CCS and IAR, and there will be a header file bsl_app.h to use in the application project.



Bootloader Host	-		×
MoreOption			
Create Linker Files plication firmware file:	Choose	.txt file	
TXT_to_H			
Password file:	Choose	.txt file	
Download			
This GUI is developed with Python version: 3.8.2		^	
Ø Generate linker files	- 0	×]
2 →Input device name: MSP430FR2355 (Example: MSP4 3 → Choose a ouput folder: C:/Myworks/Image_download/BSL/MSP430_Host_UART/ Scan	430FR2355)		
4 → Generate			
Choose a output folder:C:/Myworks/Image_download/BSL/MSP430_Hos Pycharm_work/backup/GUI_V1/Output/FR2355 You have choose device:msp430fr2355 Generate linker files for CCS and IAR Generate bsl_app.h header file for application project	t_UART/	^	

Figure 3-3. Steps to Generate Linker Files and Header File

3.3 Convert TXT Image to Header File for Host Project

When using an MCU or processor as a host, it is necessary to convert the firmware to a header file for them.



Figure 3-4. Steps to Convert TXT File to Header File



4 Porting Example Code to Other MCUs

This application report includes example codes for the MSP430FR2311 and MSP430FR5969 MCUs. To port the example code to other devices:

- 1. Select correct example as template. When porting to MSP430FR2xx or MSP430FR4xx devices, start with the MSP430FR2311 code as a template. When porting to MSP430FR5xx or MSP430FR6xx devices, use the MSP430FR5969 code as a template.
- 2. Create two new projects (one is for the boot project and one is for the application project) with the target device and copy the .c and .h files from the template into the new projects.
- 3. Using the GUI generate linker files and header file for the boot and application project. Add the files into the new projects.
- 4. Modify main_app.c in the application project based on your device. Add any necessary functions and keep the sections that are described in Section 1.3 (for MSP430FR5xx or MSP430FR6xx devices, we recommend to enable the MPU to protect the memory).
- 5. Follow the steps in Section 2 to run the code in hardware.

5 References

- 1. MSP430[™] FRAM Devices Bootloader (BSL) User's Guide
- 2. MSP430FR4xx and MSP430FR2xx Family User's Guide
- 3. MSP430FR58xx, MSP430FR59xx, and MSP430FR6xx Family User's Guide

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to TI's Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265 Copyright © 2022, Texas Instruments Incorporated