Application Report
SLAA721A – October 2016 – Revised December 2016

MSP430FRBoot – Main Memory Bootloader and Over-the-Air Updates for MSP430™ FRAM Large Memory Model Devices

Ryan Brown and Katie Pier
MSP430 Apps

ABSTRACT

This application report is an extension to *MSPBoot – Main Memory Bootloader for MSP430 Microcontrollers* and describes the implementation of a main-memory resident bootloader for MSP430™ FRAM microcontrollers using either universal asynchronous receiver/transmitter (UART) communication or a serial peripheral interface (SPI) bus and CC110x RF transceivers to accomplish over-the-air downloads (OAD). While still being highly flexible and modular, this bootloader maintains a small footprint, making it a very cost-effective solution, and supports the large memory model (devices with a memory footprint greater than 16 KB).

A software package with examples and source code for both master and slave devices is available from http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSP430FRBoot/latest/index_FDS.html. Section 5 provides step-by-step procedures that explain how to run the examples.

This bootloader is not to be confused with the MSP430 Bootloader (BSL), which resides in protected memory (ROM) in MSP430 FRAM microcontrollers. For more information on the BSL, see the *MSP430FR57xx, MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Bootloader (BSL) User's Guide*.

Contents

List of Figures

List of Tables

1 Introduction

Expanding on the original theory behind *MSPBoot – Main Memory Bootloader for MSP430™ Microcontrollers*, many FRAM applications require a solution that allows for easy field upgrades. MSP430FRBoot has been designed to accomplish this task with any custom communication peripheral and entry sequence as defined by the user. Two different examples have been included to further demonstrate these capabilities. One example uses the UART protocol to create a simple two-wire communication link between devices, while the other example incorporates SPI buses and two CC110x devices to accomplish wireless over-the-air downloads. Above all, these solutions can maintain high performance, high integration, and ultra-low power in a cost-effective design.

MSP430 FRAM devices are equipped with the very useful UART Bootloader (BSL) which allows for a simple way to do field upgrades. MSP430 FRAM devices have a ROM-resident BSL that supports UART but cannot be modified to support I^{2}C or other interfaces (the exception to this is FRxxxx1 devices, which implement an I^{2}C BSL solution instead). Furthermore, the BSL cannot include custom entry sequences that might be required for application. For more information on the BSL, see the *MSP430FR57xx, MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Bootloader (BSL) User's Guide*.

Given these limitations, it becomes necessary to create a bootloader that resides in main memory and still allows for an easy implementation of the application. This application report describes the implementation of the MSP430FRBoot bootloader with the following characteristics:

- Small footprint (less than 4KB in size required)
- 20-bit incorporation for large memory models
- Supports the eUSCI peripherals offered on FRAM devices
- UART communication offers the most simple wired interface using a small memory space.
- SPI bus offers over-the-air downloads (using the CC110x) at a slightly larger footprint.
- Different options that allow for customizable levels of robustness
- Optional dual image support in case of communication interruption
- Allows for use of all interrupts in application
- Application can reuse the low-level drivers from the bootloader or implement its own drivers.
- Configurable entry sequence
- Optional validation of application using CRC-CCITT
- Source code available, allowing for additional customizations

Source code for the bootloader with different sample configurations, application examples, and host examples are included to allow for easy testing, customization, and implementation. Knowledge of UART and SPI specifications as well as sub-1 GHz RF communication protocol is assumed.

1.1 Glossary

BOR	Brownout reset
BSL	MSP430 Bootloader
CI	MSPBoot Communication Interface
CRC	Cyclic Redundancy Check
eUSCI	Enhanced Universal Serial Communication Interface
MCU	Microcontroller
MI	MSPBoot Memory Interface
MSPBoot	The bootloader described by *MSPBoot – Main Memory Bootloader for MSP430™ Microcontrollers*
MSP430FRBoot	The bootloader described by this application report
OSI	Open Systems Interconnection
OAD	Over-the-Air Download
SPI	Serial Peripheral Interface
ROM	Read-Only Memory
UART	Universal Asynchronous Receiver/Transmitter

1.2 Conventions

This document contains some UART transfer examples that use the following form:

☐	Host→Target	St: Start
▨	Target→Host	Sp: Stop

SPI transfer examples use the following form:

☐	Master→Slave	X: Don't care
▨	Slave→Master	

2 Implementation

A modular approach is used to allow for an easy migration between MSP430 devices and allow for customization of each layer. Figure 1 shows the software layers.

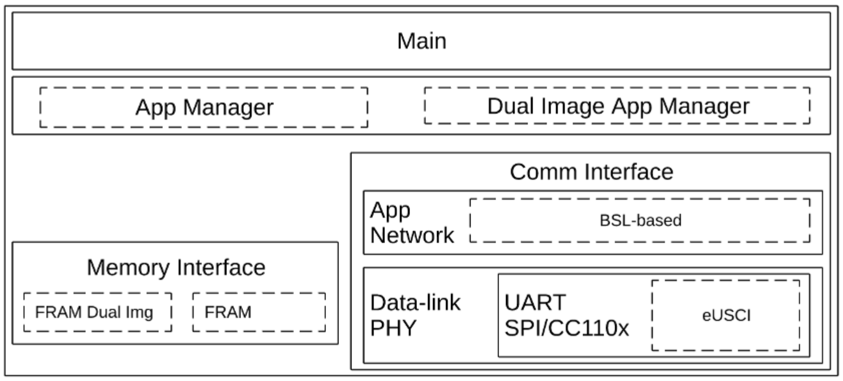


Figure 1. MSPBoot Software Architecture

Each module is described in more detail in the following sections.

2.1 Main

The main routine has the following purpose:

- Initialize basic functionality of the MSP430 MCU
- Initialize the other MSP430FRBoot layers
- Implement the main loop which polls the communication interface and processes commands

Figure 2 shows the state diagram of the main routine.

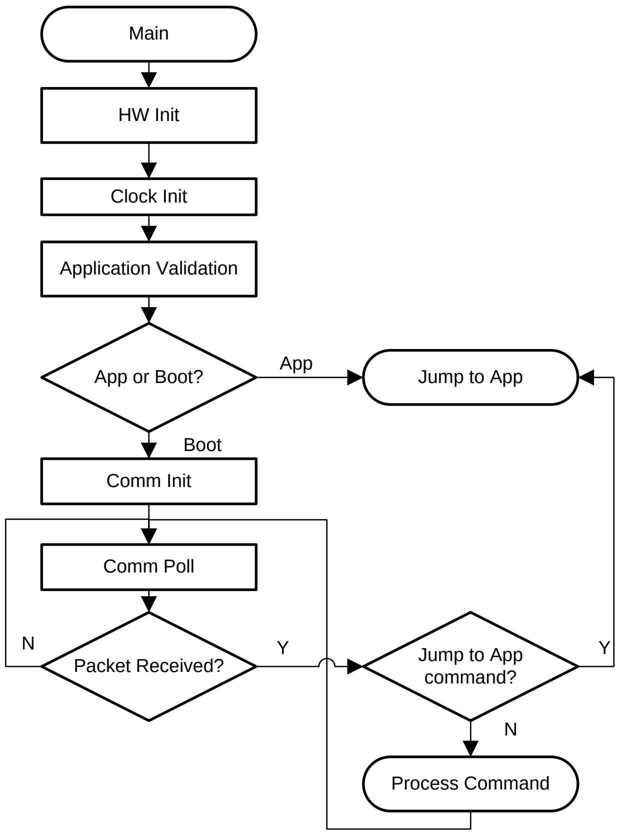


Figure 2. Flow Diagram of Main

2.2 Application Manager

The main functions of the Application Manager are:

- Detecting when the device should be in bootloader mode versus application mode
- Validating the application
- Redirecting interrupt vectors
- Jumping from bootloader to application
- Recovering a valid image when in dual-image mode

2.2.1 Bootloader and Application Detection

The Application Manager detects if the bootloader or the application should be executed by applying the following rules:

- Application is executed if
  - The application is valid (see Section 2.2.1.2)
    AND
  - The bootloader is not forced by an external event or by application (see Section 2.2.1.1)
- Bootloader is executed if
  - It is forced by an external event or by the application
    OR
  - The application is invalid

Figure 3 shows the implementation of this decision process.

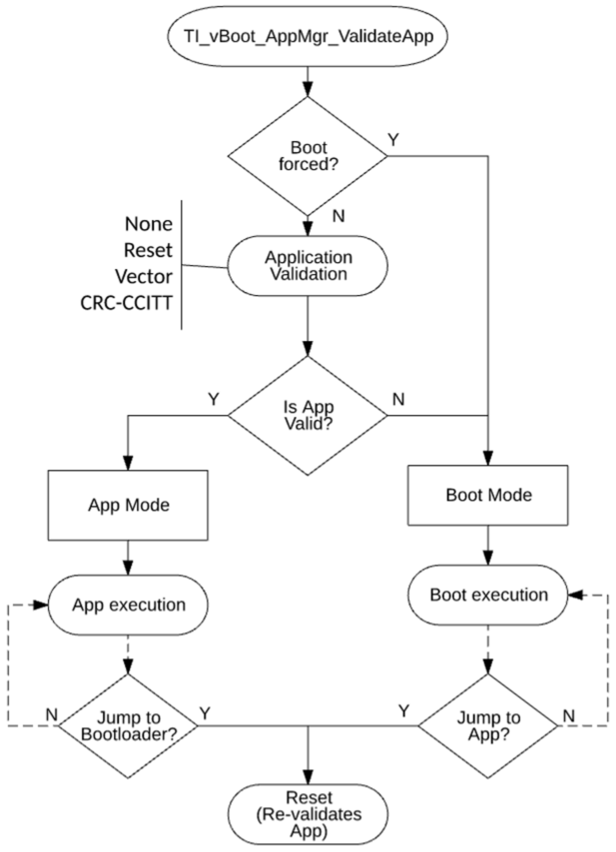


Figure 3. Application Validation by App Manager

2.2.1.1 *Forcing Bootloader Mode*

Even with a valid application, the bootloader mode can be forced by the following:

- Option1: An external event such as the state of a GPIO after reset.

  By default, the software checks if the following GPIOs are low after reset to force bootloader mode:
  - P1.1 in MSP430FR5969 (S2 button on MSP-EXP430FR5969).
  - P1.2 in MSP430FR6989 (S2 button on MSP-EXP430FR6989).
  - P5.5 in MSP430FR5994 (S2 button on MSP-EXP430FR5994).

  This event can be modified as needed in TI_MSPBoot_AppMgr_BootisForced().
- Option2: An application calls for the execution of bootloader mode.

  The variables StatCtrl and PassWd are reserved and shared between application and bootloader. To force bootloader mode, the application sets these variables to:

  ```
  PassWd = 0xC0DE
  StatCtrl.BIT0 = 1
  ```

2.2.1.2 *Application Validation*

The application validation mechanism allows the bootloader to validate the application before executing it. Three methods are implemented to allow for different levels of code footprint and security:

- None: The application is not validated and is assumed to be always valid. An external event can be used to force Boot mode. This method is not recommended.
- Reset vector: If the reset vector is different from 0xFFFF (erased state), the application is assumed to be valid and is executed.
- CRC_CCITT: A CRC-CCITT is calculated for the whole application image and compared to an expected value. The BSL-based protocol (see Section 2.4.2.1) uses CRC-CCITT, so this validation method is recommended when using this protocol.

The validation methods can prevent executing corrupted applications but they do not ensure the integrity and functionality of the application, which is the user's responsibility. If the application does not have the intended functionality, the MSP430 can still be recovered using a hardware entry sequence.

2.2.1.3 *Jumping to Application*

MSPBoot forces a reset when the Communication Protocol detects that the download is complete and the device should jump to the application.

FRAM devices use a software BOR to force reset, which provides an efficient method to restore the MSP430 MCU to a default state. Declaration HW_RESET_BOR is enabled by default.

2.2.2 Memory Assignment

MSPBoot cannot erase or reprogram the bootloader area. This limitation provides a more secure implementation, because the bootloader is always accessible, and the MSP430 MCU can be recovered by forcing bootloader mode.

The reset vector is an integral part of the bootloader, because it forces the MSP430 MCU to always jump to the bootloader entry sequence and, thus, should not be erased. Because the reset vector resides in the top of 16-bit FRAM space (0xFFFE), the bootloader code is placed in the contiguous locations (see Figure 4).

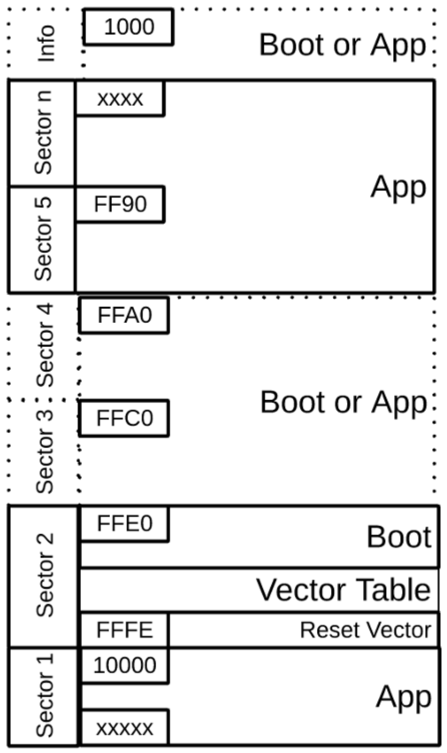


Figure 4. Memory Assignment

The interrupt vector table is also in the protected boot area. Because the value of the interrupt tables is expected to change based on the application, this means that special considerations must be followed to allow for application interrupts. Additional 20-bit space is available for the application (0x10000 and above).

2.2.3 Interrupt Vectors in FRAM Devices

FRAM does not have the limitation of a minimum erase size, so all interrupts can be reprogrammed in FRAM devices without risking erasure of the reset vector. By default, MSP430FRBoot enables protection of the bootloader area using MPU, but this feature is disabled while reprogramming interrupt vectors.

Some MSP430 MCUs support redirecting vectors to RAM in hardware (SYSRIVECT), which could be a good alternative especially for devices with sufficient RAM. This also allows for full protection of the bootloader using the MPU module.

2.2.4 Dual Image Support

The Application Manager can also support dual image mode. In this mode, a valid application is always expected to reside in main memory even if an image download is interrupted or if a newly downloaded image is corrupted. This mode is critical for OAD where communication can be interrupted unexpectedly.

In dual image mode, the main memory is divided, creating a Download area and Application area as explained in Section 2.3.1. The application validation process in this mode (see Figure 5) is different from the usual procedure.

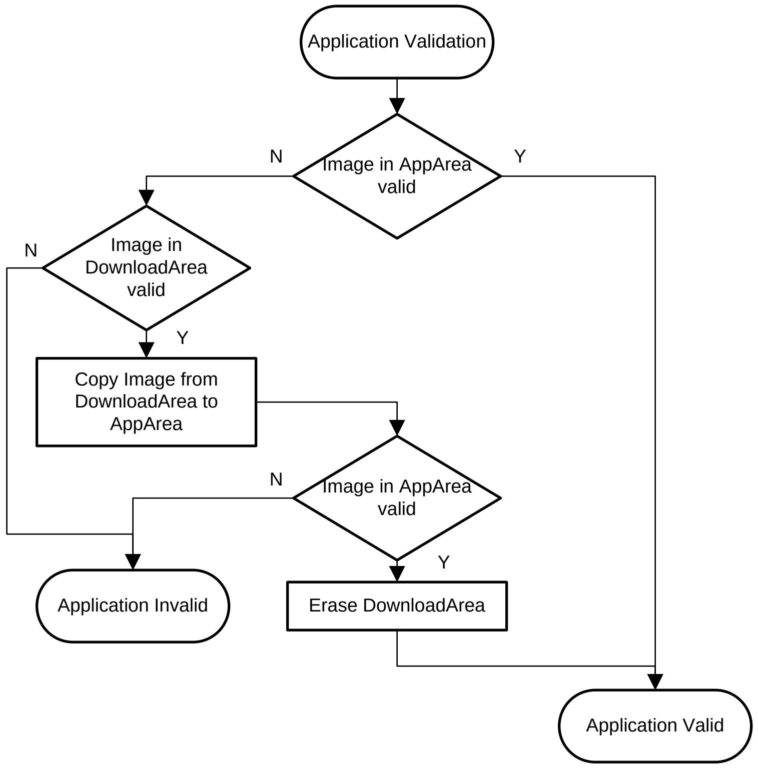


Figure 5. Dual Image Application Validation

2.2.4.1 *Jump to Application in Dual Image Mode*

When an application download process is completed, MSPBoot performs the following steps before jumping to the new application:

1. Validate the new image in the Download area.
   (a) If invalid, exit. A reset forces the bootloader again and executes the application only if the original image is valid.
   (b) Continue otherwise.
2. Replace Application area with Download area.
3. Validate image in Application area.
   (a) If valid, erase Download area. A reset will execute the application, because the image in the Application area is valid.
   (b) Exit otherwise. This is an unexpected state, but a reset will validate both images again.

2.3 Memory Interface (MI)

To protect the bootloader area, the MSP430 MCU is logically partitioned in two sections:

- Application area: Writable section with user application and redirected vector table
- Bootloader area: Nonwritable section with bootloader and vector table

The size of each section is defined in the project linker file. Examples showing different memory sizes are available in the example projects for the Code Composer Studio™ IDE (CCS).

The memory interface provides an API that is used to program and erase the application memory area and protect the bootloader area. This memory protection is implemented as follows for FRAM devices:

- FRAM does not require erasing, but the application memory is written with 0xFF when an erase is performed to calculate a valid CRC.
- The address being erased or programmed is validated to avoid accidental corruption of the bootloader area.
- The MPU protects the bootloader area. The user can modify the MPU settings according to the application, but TI recommends always protecting the bootloader area.
- MPU protection is disabled only when updating interrupt vectors as discussed in Section 2.2.3.

NOTE: MSPBoot does not allow write or erase access to the bootloader area when executing updates, but it cannot protect against accidental erase when executing an application. The bootloader area is hardware-protected using the MPU.

2.3.1 Dual Image Support

When Dual Image support is enabled, the Memory Interface module partitions the MSP430 application area in two subsections, resulting in the following logical memory map:

- Nonboot area:
  - Download area: Section used as temporary buffer to store a new application image. Physical addresses in this area are inaccessible to the host, but this area is written when the host attempts to download to logical addresses in the application area.
  - Application area: Section used to execute the current application image. Logical addresses in this area are available to the host, but the host cannot write to the physical addresses. The bootloader updates this area when a new image in Download area is validated. This procedure is explained in Section 2.2.4.
- Boot Area: Read-only section with bootloader and vector table.

The size of each sector is defined in the project linker file. Examples showing different memory sizes are available in the example projects for CCS.

2.4 Communication Interface (CI)

The purpose of the CI is to:

- Receive data from and send data to a host
- Implement a communication protocol
- Parse the data, validate a packet, and execute the appropriate command
- Based on the output of the function, generate a response

Following the Open Systems Interconnection (OSI) model, the CI is divided into two modules:

- Physical-DataLink (PHY-DL)
- Network-Application (NWK-APP)

2.4.1 Physical-DataLink (PHY-DL)

The PHY-DL layer provides a hardware abstraction layer (HAL) to simplify the migration process to a different MSP430 derivative or peripheral. The PHY-DL layer provides a stable channel for sending data to and receiving raw data from the host. The current bootloader was implemented using UART or SPI and it supports the eUSCI, but other options could be included if desired. The PHY-DL layer is initialized by providing a pointer to a structure with the callback functions in Table 1.

Table 1. PHY-DL Callback Structure

t_CI_Callback	Structure type definition
.RxCallback	Called when a new byte is received
.TxCallback	Called when a byte needs to be transmitted
.ErrorCallback[1]	Called when an error is detected in PHY-DL (for example, a time-out)

[1] Callback is optional. The protocol or CI may not require a callback.

A higher level layer (NWK-APP) uses the callback functions to implement the communication protocol. Depending on the protocol, some callbacks are not required and can be disabled in the PHY-DL layer to reduce the footprint. NWK-APP layer is described in Section 2.4.2.

2.4.1.1 UART

The UART interface is implemented using 8-N-1 format (8 data bits, no parity bit, and 1 stop bit) (see Figure 6).

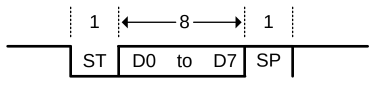


Figure 6. UART 8-N-1 Format

The default baud rate is defined as CONFIG_CI_PHYDL_UART_BAUDRATE = 57600.

2.4.1.2 SPI

The SPI interface, used for CC110x communication, is implemented using the following configuration (also see Figure 7):

- 8-bit data
- MSB first
- Clock polarity = 0 (inactive state is low)
- Clock phase = 1 (data captured on first clock edge, changed on following edge)
- 3-pin configuration with STE implemented using GPIO

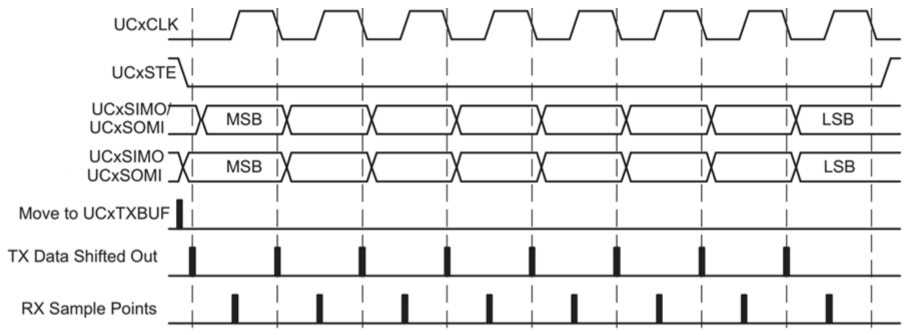


Figure 7. SPI Format

2.4.1.3 CC110x

The CC110x devices send data using a packet structure shown in Table 2.

Table 2. CC110x Data Packet Structure

Header	Length[1]	Command	Address[2]	Data[2]	Checksum
0x80	N	1 byte	3 bytes	N-6 bytes	2 bytes

[1] The maximum packet length is 24 bytes; therefore, the most data bytes (N) allowed per packet is 16.
[2] If the length is equal to one (command only), these sections are not included in the packet.

The configuration for the CC110x in MSP430FRBoot is as follows:

- 250-kbps data speeds
- Carrier frequency of 902750 Hz

The data speed can be set to either 1.2 or 38.4 kbps through the variable sent by the radio_init function inside of TI_MSPBoot_CI_PHYDL_CC1101.c. Radio frequency can be altered in TI_MSPBoot_Config.h. Changes must be made to both the target and host firmware projects. See the *CC1101 Low-Power Sub-1 GHz RF Transceiver* data sheet for more information regarding common CC110x command and other communication details.

This packet structure is identical to the BSL-based protocol, therefore it can be directly transferred from the PHY-DL to the NWK-APP layer with the expected formatting. This is also referred to in Section 2.4.2.1.2.

2.4.1.4 Comm Sharing

The user application can use the communication interface as desired (UART, GPIO, or other purpose), because the resources are released when the microcontroller jumps to the application. Optionally, the CI PHY-DL can be shared with the application, allowing it to use the same communication interface and reducing the application footprint. When this feature is enabled, the bootloader shares the function pointers from Table 3.

Table 3. Boot2App_Vector_Table Definition

Boot2App_Vector_Table	Table with addresses of shared CI PHY-DL functions
TI_MSPBoot_CI_PHYDL_Init	Function used to initialize PHY-DL passing a pointer to an application t_CI_Callback
TI_MSPBoot_CI_PHYDL_Poll	Function checks all relevant flags and calls corresponding callbacks when required.
TI_MSPBoot_CI_PHYDL_TxByte	Function used to write the TX Buffer

The application must declare its own callbacks, which are passed during initialization of CI PHY-DL and called when the corresponding event is detected. The PHY-DL layer is designed with low footprint being a top priority. The application can always implement its own drivers if the PHY-DL implementation is inadequate. Application examples showing how to share CI PHY-DL are included in the software package.

2.4.2 NWK-APP

The CI Network-Application layer implements the communication protocol, interpreting the raw data from PHY-DL, and validates such data before executing the appropriate commands. For means of simplicity, MSP430FRBoot only uses the BSL-based protocol.

2.4.2.1 BSL-Based Protocol

The MSP430 BSL is the standard bootloader included in MSP430 MCUs. The BSL is described in detail in *MSP430 Programming With the Bootloader (BSL)*.

The BSL-based protocol implemented in MSP430FRBoot maintains robustness but does not implement all the commands and exactly the same format as the BSL protocol to reduce its footprint. The protocol is packet-based and has the format shown in Table 4.

Table 4. BSL-Based Protocol Command Format

Header	Length	Payload	Checksum[L]	Checksum[H]
0x80	1 to PAYLOAD_MAX_SIZE[1]	1 to PAYLOAD_MAX_SIZE bytes	1 byte	1 byte

[1] PAYLOAD_MAX_SIZE is set to 19 by default (1 CMD + 3 Addr + 16 Data)

Header: Fixed to 0x80.

Length: 1 byte with the length of the payload. Valid values are 1 to PAYLOAD_MAX_SIZE.

Payload: 1 to PAYLOAD_MAX_SIZE bytes containing the command, address, and data (optional depending on the command type).

Checksum: 16-bit CRC-CCITT of the payload.

The commands in Table 5 are implemented as a payload.

Table 5. BSL-Based Protocol Commands

Command	CMD	Byte$_1$	Byte$_2$	Byte$_3$	Byte$_4$	…	Byte$_{length-1}$
ERASE_SEGMENT	0x12	ADDR[L]	ADDR[M]	ADDR[H]	X	X	X
ERASE_APP	0x15	X	X	X	X	X	X
RX_DATA_BLOCK	0x10	ADDR[L]	ADDR[M]	ADDR[H]	DATA0	X	DATAn
TX_VERSION	0x19	X	X	X	X	X	X
JUMP2APP	0x1C	X	X	X	X	X	X

ERASE_SEGMENT: Erases the memory segment (512B in FRAM) addressed by ADDR.

ERASE_APP: Erases the application area.

RX_DATA_BLOCK: Programs n bytes of data starting at address ADDR.

TX_VERSION: Requests the MSPBoot version from the target.

JUMP2APP: Instructs the target to jump to the application image (after validation).

Response from the target are always a single byte (Table 6 lists the valid values).

Table 6. BSL-Based Protocol Slave Response

Response	Value	Description
OK	0x00	Previous command executed correctly
HEADER_ERROR	0x51	Frame had incorrect header
CHECKSUM_ERROR	0x52	Frame checksum incorrect
PACKETZERO_ERROR	0x53	Length of packet = 0
PACKETSIZE_ERROR	0x54	Length of packet > MAX_LEN
UNKNOWN_ERROR	0x55	Error in protocol
INVALID_PARAMS	0xC5	Parameters received for command are incorrect
INCORRECT_COMMAND	0x56	Received command is not valid
MSPBOOT_VERSION	0 to 0xFF	Sent as response for TX_VERSION command (default is 0xA0)

2.4.2.1.1 Security

The contents of each packet are validated with a 16-bit CRC that provides additional robustness to the bootloader. The host can check the result of each command and retry if the previous command was unsuccessful.

The ERASE_SEGMENT and RX_DATA_BLOCK commands can erase and write any area within the 16-bit memory map, thus potentially corrupting the bootloader. To avoid this possibility, TI recommends including the CONFIG_MI_MEMORY_RANGE_CHECK MI definition to validate the address before a program or erase operation. The application area can be corrupted if the process is interrupted, so TI recommends using one of the application validation methods described in Section 2.2.1.2 or use the dual-image approach.

2.4.2.1.2 BSL-Based Protocol Using CC110x

The CC110x implementation of this protocol follows the same guidelines used for UART, but it includes slight changes since the information is received in complete packets instead of bytes. Because the incoming CC110x packet outlined in Table 2 is the same as the expected BSL-based protocol in Table 4, data from the PHY-DL layer can be directly transferred to the NWK-APP without the need for conversion.

2.4.2.1.3 Examples Using UART or CC110x

The following considerations apply when using UART with BSL-based protocol:

- Address is not required, because communication is expected to be point-to-point.
- All bytes in UART are in 8-N-1 format as described in Section 2.4.1.1.
- The target responds with the command result when ready and not when requested by the host.
- The host should wait for the response from the target after sending a command, preferably with a time-out.
- Different commands have different processing times.
- Example: Host erases the microcontroller application area.

0x80	0x01	0x15	0x64	0xA3
Header	Length	ERASE_APP	Checksum_L	Checksum_H

The target device processes the command and responds with the result when ready.

0x00
OK

The same considerations apply when using a CC110x with BSL-based protocol. The exception to this is Section 2.4.2.1.2 where it is stated that all bytes received in packets from the CC110x are in the same format expected for BSL-based protocol; therefore, they can be directly transferred from the PHY-DL to the NWK-APP. Although processing times are the same, over-the-air communication can be expected to be slightly slower than UART, and host wait times should be lengthened to compensate.

3 Customization of MSP430FRBoot

MSPBoot was designed with low cost and low footprint being top priorities; however, some applications require or can benefit from having a higher level of security and robustness. Based on the application requirements, different levels of customizations have been added to the MSP430FRBoot code and they can be adjusted to particular needs. These options are selected either by adding the appropriate files or by enabling or disabling preprocessor definitions. Table 7 list the options that can be configured in TI_MSPBoot_Config.h.

Table 7. Optional Configurations

Value	Description	Change in Code Size
`NDEBUG`		
Defined	ASSERT_H functions are ignored. Watchdog is enabled.	–
Undefined	Used during debugging. ASSERT_H functions are checked. Watchdog is disabled.	Adds approximately 20 bytes
`CONFIG_MI_MEMORY_RANGE_CHECK`		
Defined	The address being erased or programmed is validated to be within the Application area.	Adds approximately 40 bytes
Undefined	Address being erased or programmed is not validated. Host must send correct address.	–
`CONFIG_APPMGR_APP_VALIDATE`		
1	Application is validated by checking its reset vector.	Adds approximately 10 bytes
2	Application is validated by checking its CRC_CCITT.	Adds approximately 94 bytes
`CONFIG_CI_PHYDL_COMM_SHARED`		
Defined	Communication Interface PHY-DL layer is shared with application.	Adds approximately 4 bytes
Undefined	CI PHY-DL is not shared with application.	–
`CONFIG_CI_PHYDL_TIMEOUT`		
Defined	Detect time-out in CI PHY-DL.	Adds approximately 48 to 62 bytes
Undefined	CI PHY-DL does not detect time-out.	–
`CONFIG_CI_PHYDL_ERROR_CALLBACK`		
Defined	A callback function is called when a time-out error is detected.	Adds approximately 16 to 20 bytes
Undefined	A callback function is not called when a time-out is detected.	–

Other customizations are selected by adding and using the appropriate files in the project. Table 8 lists the files that are interchangeable in the project.

Table 8. Customization Files

File	Comments
CI PHY-DL	
TI_MSPBoot_CI_PHYDL_USCI_UART.c	Use eUSCI as UART
TI_MSPBoot_CI_PHYDL_CC1101.c	Use CC110x
MI	
TI_MSPBoot_MI_FRAM.c	API used to program application FRAM
TI_MSPBoot_MI_FRAMDualImg.c	API implementing dual image in FRAM
App Manager	
TI_MSPBoot_AppMgr.c	Standard App Manager
TI_MSPBoot_AppMgrDualImg.c	App Manager that supports dual image

3.1 *Predefined Customizations*

The software package includes projects for Code Composer Studio IDE that support three devices (MSP430FR5969, MSP430FR6989, MSP430FR5994) with two communication interfaces (UART or SPI with CC110x) and two predefined configurations (single image, dual image) per device. In the provided CCS examples, devices and communication interfaces are separated by project selection, and the predefined configurations can be chosen under Project → Build Configurations → Set Active.

4 Building MSPBoot

This section provides a step-by-step guide that explains how to build the bootloader and demo applications for a target device. Section 5 explains how to build and use the host applications to run a demo.

4.1 *LaunchPad™ Development Kit Hardware*

This software package includes examples for the MSP430FR5969, MSP430FR6989, and MSP430FR5994 on their LaunchPad™ development kits (MSP-EXP430FR5969, MSP-EXP430FR6989, and MSP-EXP430FR5994, respectively) (see Figure 8).

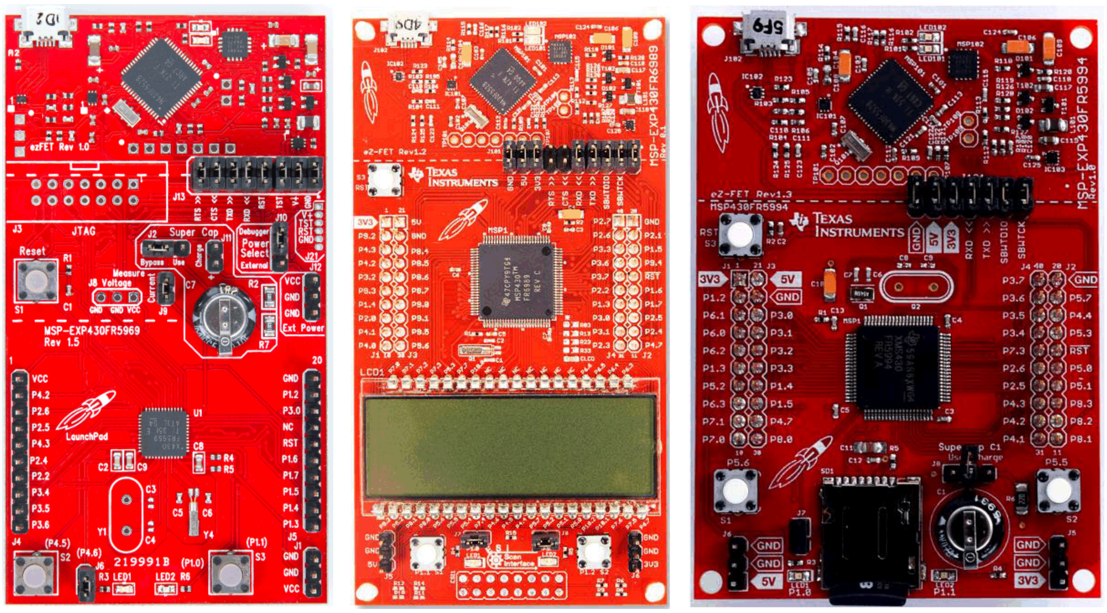


Figure 8. MSP-EXP430FR5969, MSP-EXP430FR6989, and MSP-EXP430FR5994

The bootloader and demo applications use the same LED (LED1 and LED2) and push button (S1 and S2) notations across all variants of the LaunchPad development kits. The pin assignments that correspond to these I/O peripherals are different for each board derivative. For ease of use, the examples have been designed so that the host and target LaunchPad development kits should be the same derivative, although this can be modified for different configurations if desired.

4.2 CC110x Hardware

Two hardware options are available for using CC110x communication with the MSP430FRBoot examples. The first is a combination of the CC1101EMK868-915 and BOOST-CCEMADAPTER, but the simplest solution is with a 430BOOST-CC110L. Figure 9 shows both options.

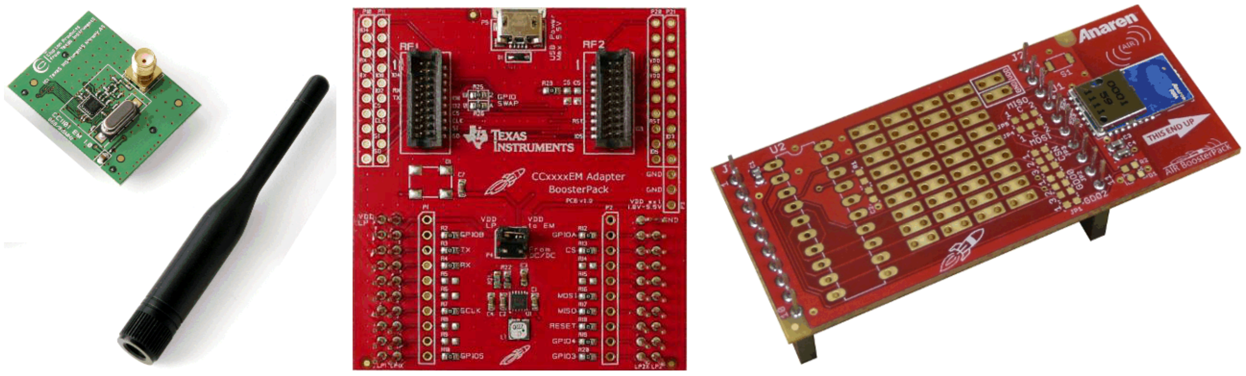


Figure 9. CC1101EMK868-915, BOOST-CCEMADAPTER, and 430BOOST-CC110L

Two units of either option are required, one for the host device and the other for the target. Both solutions are compatible across all LaunchPad development kits and are directly connected such that no other hardware is required to run the provided examples. More information about the ecosystem for the LaunchPad development kits and BoosterPack plug-in modules can be found on the TI LaunchPad tools page.

4.3 Software

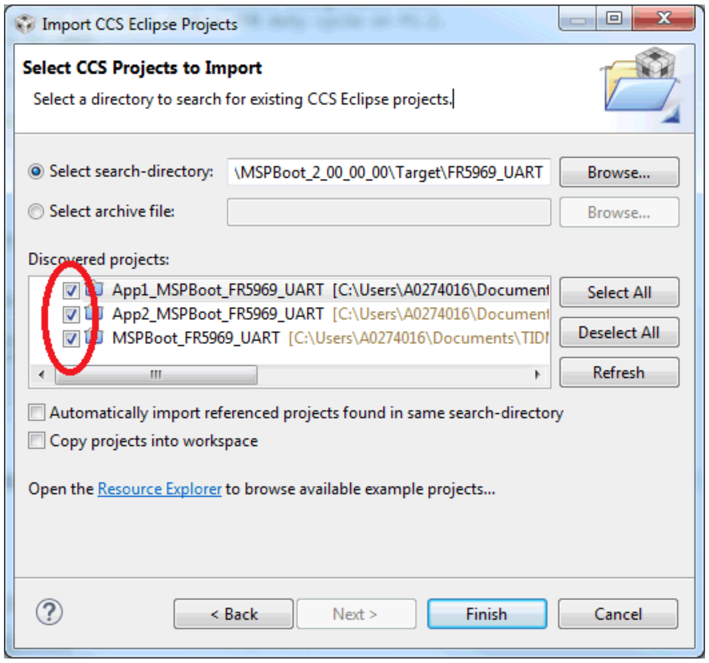
The software package includes the following folders:

- **Target**: Target bootloader and demo applications.
  – **FR5969_UART**, **FR5969_CC1101**, **FR5994_UART**, **FR5994_CC1101**, **FR6989_UART**, **FR6989_CC1101**: Projects that support the appropriate FRAM derivative with the communication specified.
    - **CCS**: CCS project files.
      - **MSPBoot**: CCS project files for the bootloader.
        - Config: CCS Linker files for the bootloader.
      - **App1_MSPBoot**: CCS project files for Application Example 1.
        - Config: CCS Linker files for App1.
      - **App2_MSPBoot**: CCS project files for Application Example 2.
        - Config: CCS Linker files for App2.
    - **Src**: Source code.
      - **MSPBoot**: Source code for the bootloader.
        - **AppMgr**: App Manager source code files.
        - **Comm**: CI source code files.
        - **MI**: MI source code files.
      - **App1**: Source code for Application Example 1.
      - **App2**: Source code for Application Example 2.

- **Host**: Host demo application.
  - MSP-EXP430FR5969, MSP-EXP430FR5994, MSP-EXP430FR6989: Host project that supports the corresponding LaunchPad development kits (see Section 4.3.2).
    - CCS: CCS project files.
    - Src: Source code.
      - TargetApps: Converted target application examples.
- **430txt_converter**: Scripts and applications used to convert CCS output files to host TargetApps. See Section 4.3.2 for details.

4.3.1 Building the Target Software

1. Select a target processor: MSP430FR5969, MSP430FR5994, or MSP430FR6989.
2. Open CCS and select or create a workspace.
3. Import the MSPBoot CCS projects into the workspace. The projects are located in MSPBoot\Target\<target>\CCS\


Import CCS Eclipse Projects

Select CCS Projects to Import

Select a directory to search for existing CCS Eclipse projects.

(● Select search-directory: \MSPBoot_2_00_00_00\Target\FR5969_UART) Select archive file:

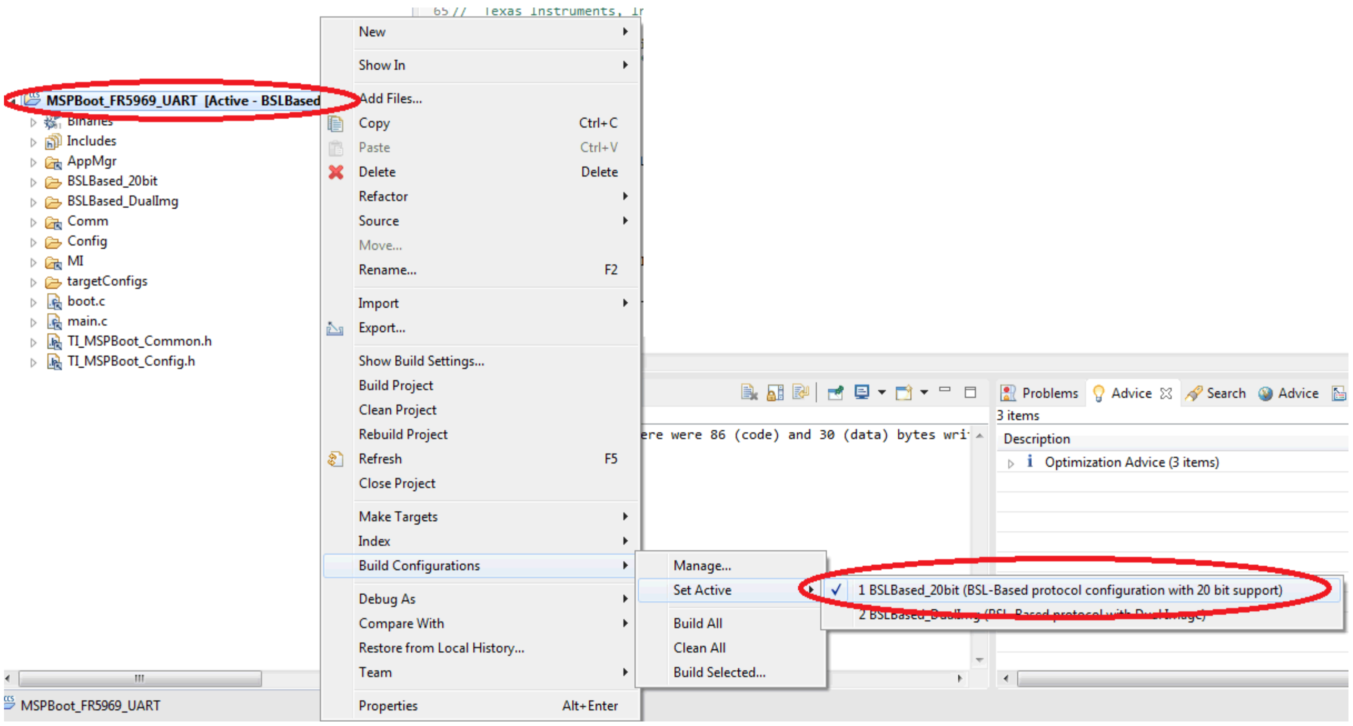
Discovered projects:
- [✓] App1_MSPBoot_FR5969_UART [C:\Users\A0274016\Document
- [✓] App2_MSPBoot_FR5969_UART [C:\Users\A0274016\Document
- [✓] MSPBoot_FR5969_UART [C:\Users\A0274016\Documents\TIDI

[ ] Automatically import referenced projects found in same search-directory
[ ] Copy projects into workspace

Open the Resource Explorer to browse available example projects...

Figure 10. Import MSPBoot CCS Projects

4. Build the bootloader.
   (a) Select the MSPBoot project.
   (b) Select the proper target configuration based on Section 3.1.



ere were 86 (code) and 30 (data) bytes wri

Figure 11. Select Target Configuration

(c) Build and Download . Only the target LaunchPad development kit should be connected to the PC.

5. Build both applications.

(a) Select the App1_MSPBoot project and select the same configuration as the bootloader:

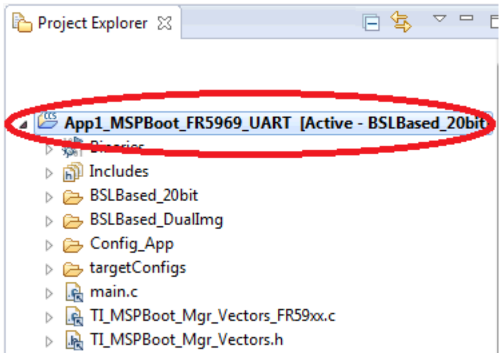


Figure 12. Select App1_MSPBoot Project

(b) Click the Build project. The output is generated after this step, but the output will be converted and downloaded through the Host processor. Section 4.3.2 explains how to convert the image, and Section 5 explains how to download it using a host demo.

(c) Repeat Step 5 for App2_MSPBoot.

4.3.2 Convert Application Output Images

The CCS and IAR projects generate outputs in MSP430 .txt format in the MSPBoot\Target\<target>\CCS\<App>\<Configuration>\ directories, where,

<target> = FR5969_UART, FR5969_CC1101, FR5994_UART, FR5994_CC1101, FR6989_UART, or FR6989_CC1101

<App> = App1_MSPBoot or App2_MSPBoot

<Configuration> = BSLBased_20bit or BSLBased_DualImg

This .txt file does not include CRC, and it needs to be converted to a format usable by the Host project. To make this easier, the software package includes 430txt2C, a Perl script used to convert an MSP430 .txt file to a C array:

Location: MSPBoot\430txt_converter\430txt2C.pl

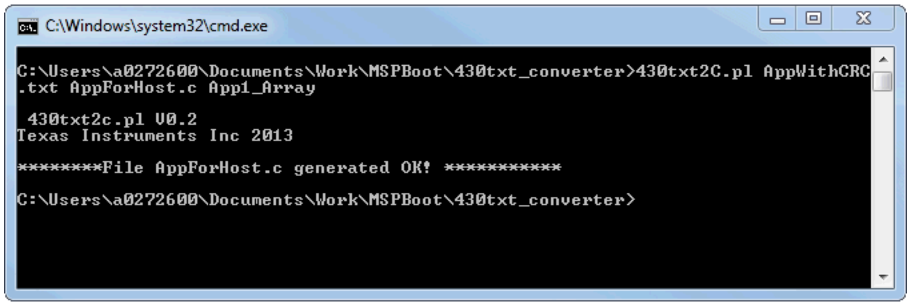
Syntax: 430txt2c.pl src dest struct

`src` = Source file in .txt format

`dest` = Destination file in .c format

`struct` = Name of array in C file

Figure 13 shows an example of running 430txt2C.



```
C:\Users\a0272600\Documents\Work\MSPBoot\430txt_converter>430txt2C.pl AppWithCRC
.txt AppForHost.c App1_Array

 430txt2c.pl V0.2
Texas Instruments Inc 2013

*********File AppForHost.c generated OK! ************

C:\Users\a0272600\Documents\Work\MSPBoot\430txt_converter>
```

Figure 13. 430txt2C Example

NOTE: A Perl interpreter is required to run this script. Visit http://www.perl.org/ to download an interpreter if needed.

The software package includes several Windows .bat files in the MSPBoot\430txt_converter folder. The purpose of these files (in addition to serving as examples) is to automate the process of calculating the CRC for applications, converting to .C, and copying to host projects.

4.3.3 Generating Linker Files

Linker files for CCS and IAR are included for all target configurations, and they can be used as a starting point for other devices or custom projects. MSP430FRBoot includes a linker generator script that can also help with this process.

MSPFRBootLinkerGen: Generates application and bootloader linker files for CCS (also see Figure 14).

Location:	MSPFRBoot\linkerGen\MSPFRBootLinkerGen.pl
Syntax:	`MSPFRBootLinkerGen.pl [-help] -file <filename> -lnk_file <lnk_msp430.cmd> -boot_size <size> [-dual_image] [-shared_vectors] <number>`

Where (all numbers are hex values),

`-file <filename>` = Specifies output file prefixes. The following files are generated:

./output/<filename>/_Boot.cmd (linker file for Bootloader in CCS).

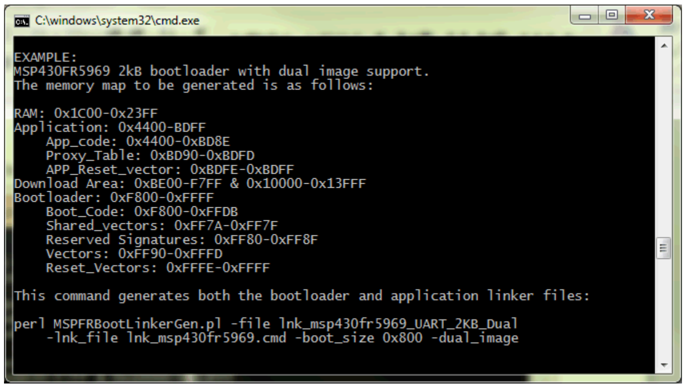
./output/<filename>/_App.cmd (linker file for Application in CCS).

`-lnk_file <lnk_msp430.cmd>` = Specifies the default linker file for the device being used.

`-boot_size <size>` = Specifies size of bootloader area. Only increments of 0x400 are allowed because of the granularity of the MPU (Memory Protection Unit).

`-dual_image` = Optional parameter. If defined, linker files for dual-image support are generated, with separate download and application areas.

`-shared_vectors <number>` = Optional parameter. Specifies the number of shared vectors (in hex). If not specified, default value is 3 vectors.



```
EXAMPLE:
MSP430FR5969 2kB bootloader with dual image support.
The memory map to be generated is as follows:

RAM: 0x1C00-0x23FF
Application: 0x4400-BDFF
    App_code: 0x4400-0xBD8E
    Proxy_Table: 0xBD90-0xBDFD
    APP_Reset_vector: 0xBDFE-0xBDFF
Download Area: 0xBE00-F7FF & 0x10000-0x13FFF
Bootloader: 0xF800-0xFFFF
    Boot_Code: 0xF800-0xFFDB
    Shared_vectors: 0xFF7A-0xFF7F
    Reserved Signatures: 0xFF80-0xFF8F
    Vectors: 0xFF90-0xFFFD
    Reset_Vectors: 0xFFFE-0xFFFF

This command generates both the bootloader and application linker files:

perl MSPFRBootLinkerGen.pl -file lnk_msp430fr5969_UART_2KB_Dual
    -lnk_file lnk_msp430fr5969.cmd -boot_size 0x800 -dual_image
```

Figure 14. Example Command

NOTE: A Perl interpreter is required to run this script. Visit http://www.perl.org/ to download an interpreter if needed.

This script uses templates that are located in the same folder. These templates can be modified as needed but they are required to run the script.

The software package includes a Windows .bat file that is used to generate the linker files for all predefined projects and targets configurations. This file can also be used as a base for any further customizations.

5 Demo Using FRAM LaunchPad Development Kit as Host

This software package includes projects and source code for a host device running on MSP-EXP430FR5969, MSP-EXP430FR5994, and MSP-EXP430FR6989 LaunchPad development kits. Each supports its own MSPBoot protocol with all target derivatives. As an example, the MSP-EXP430FR5969 supports MSP430FR5969 MSPBoot targets for either UART or CC110x communication for both single-image or dual-image modes, and so forth. This can be tailored such that any LaunchPad development kit host can be used to program any target MSP derivative with the proper MSP430FRBoot firmware.

5.1 Hardware

From the options described earlier, this demo uses FRAM LaunchPad development kits to connect to a target of the same derivative. For UART communication, the eUSCI TXD and RXD lines need to be connected in addition to ground. Make sure that the host TXD line is connected to the target RXD line, and the host RXD line is connected to the target TXD line.

No wiring is required between the target and host devices when using CC110x communication. Simply connect the CC1101EMK868-915 with BOOST-CCEMADAPTER or the 430BOOST-CC110L to the corresponding LaunchPad development kit or device pins to complete the setup. Make sure that the BoosterPack plug-in modules are correctly oriented on the LaunchPad development kit boards.

Table 9 lists the specific eUSCI peripheral used for each MSP device and communication type.

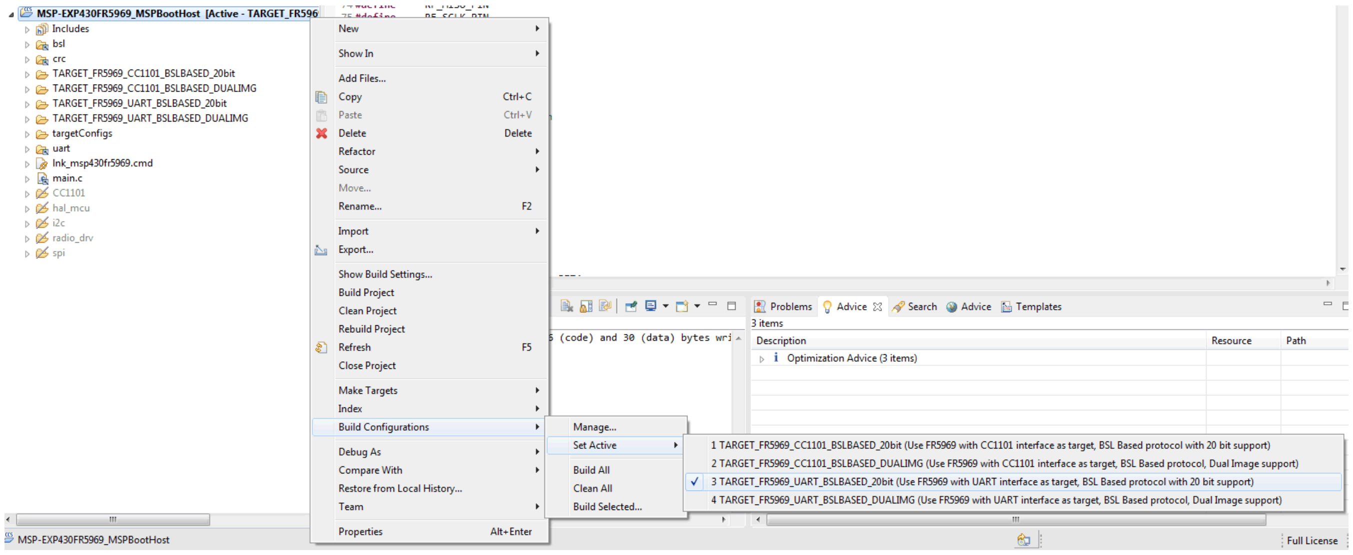
Table 9. eUSCI Peripheral Connections

CI	Pin	MSP Derivative		
		MSP430FR5969	**MSP430FR5994**	**MSP430FR6989**
UART	RXD	P2.6/UCA1RXD	P6.1/UCA3RXD	P4.3/UCA0RXD
	TXD	P2.5/UCA1TXD	P6.0/UCA3TXD	P4.2/UCA0TXD
	GND			
SPI (CC110x)	MISO	P1.7/UCB0SOMI	P5.1/UCB1SOMI	P1.7/UCB0SOMI
	MOSI	P1.6/UCB0SIMO	P5.0/UCB1SIMO	P1.6/UCB0SIMO
	CLK	P2.2/UCB0CLK	P5.2/UCB1CLK	P1.4/UCB0CLK
	SS	P3.0	P4.4	P1.5

5.2 Building the Host Project

The host project can be built following the next steps:

1. Import the project to CCS. The project files are located in MSPBoot\Host\ <host>\CCS, where <host> is the variant of the LaunchPad development kit (MSP-EXP430FR5969, MSP-EXP430FR5994, or MSP-EXP430FR6989).
2. Select the target derivative. This can be selected using the different target configurations in CCS (see Figure 15).



1 TARGET_FR5969_CC1101_BSLBASED_20bit (Use FR5969 with CC1101 interface as target, BSL Based protocol with 20 bit support)
2 TARGET_FR5969_CC1101_BSLBASED_DUALIMG (Use FR5969 with CC1101 interface as target, BSL Based protocol, Dual Image support)
✓ 3 TARGET_FR5969_UART_BSLBASED_20bit (Use FR5969 with UART interface as target, BSL Based protocol with 20 bit support)
4 TARGET_FR5969_UART_BSLBASED_DUALIMG (Use FR5969 with UART interface as target, BSL Based protocol, Dual Image support)

Figure 15. Target Selection for Host Project in CCS

3. Build and download the application. Only the host LaunchPad development kit should be connected to the PC at this time to avoid FET target confusion in CCS.

This project uses Application images located in the following folder:

MSPBoot\Host\<host>\Src\TargetApps

Prebuilt images are included, but target Applications can be replaced or updated by following the procedure described in Section 4.3.1 and Section 4.3.2.

5.3 Running the Demo

The host LaunchPad development kit project sends two different images to the target device, using a push button for user interaction. USB connection to a computer is not required on either LaunchPad development kit to run the demo; however, each kit should be powered either by a USB connection through the eZ-FET or with a steady 3.3-V external power supply to the V_{CC} and GND pins (ensure that the eZ-FET is disconnected in this instance). Because both the host and target LaunchPad development kits are of the same derivative, it might be helpful to label each board accordingly to avoid confusion. The demo is run using these steps, regardless of communication type or image mode used:

1. Build and download MSPBoot as described in Section 4.3.1, and build App1 and App2.
2. Convert App1 and App2 according to Section 4.3.2.

NOTE: Batch file PrepareCCSOutput_[FR derivative].bat shows how to convert to C and copy the output files. In this host implementation, the MSP430 MCU holds the target image without CRC, so it calculates the CRC value assuming that unimplemented locations are 0xFF.

3. Build and download the host application as described in Section 5.2.
4. Connect the boards according to the desired communication type (UART as described in Section 5.1 or one of the CC110x solutions described in Section 4.2).
5. Reset and execute code in both devices.
6. To enter the target bootloader mode (indicated by both LED1 and LED2 remaining on):
   (a) If the target does not have a valid application (default), the target stays in bootloader mode.
   (b) Bootloader mode can be forced in hardware by pressing and holding the S2 button on the target device while pressing and releasing the reset button.
   (c) If running an application:
      (i) APP1 jumps to bootloader mode when the S2 button is pressed on the target device.
      (ii) APP2 jumps to bootloader mode when it receives the Force Boot command (supported only if

CI PHY-DL is shared).

7. Press the S1 button on the host board. The host device performs the following sequence of commands:
   (a) Toggles LED1 twice.
   (b) Sends “Force Boot” command (0xAA).
      (i) If the target device is already in bootloader mode, it discards the packet, because the CRC is incorrect.
      (ii) If the target is running APP2, the target device enters bootloader mode.
   (c) Requests the bootloader version (sends the TX_VERSION command).
      (i) If the target response is 0xA0 (expected from BSL protocol), the host continues.
      (ii) If the target response is any other value, the host aborts transaction.
   (d) Erases the target application area (sends the ERASE_APP command).
   (e) Sends APP1 (uses the RX_DATA_BLOCK commands).
   (f) Programs CRC of APP1 (uses the RX_DATA_BLOCK command).
   (g) Forces the target application to run (sends the JUMP2APP command).
   (h) Toggles LED1 twice to indicate successful transfer, and keeps LED1 on to show that the host is now ready to send APP2.
8. Target starts running APP1 upon completion of transfer.
   (a) The target device blinks LED1.
   (b) LED1 blinks at a periodic interval using the timer.
   (c) Press the S2 button on the target board to enter bootloader mode.
9. With the target in bootloader mode, press S2 button on the host board to send APP2. When finished and done toggling, LED1 of the host board stays off to indicate that APP1 is now ready to be sent.
10. Target starts running APP2 upon completion of transfer.
   (a) The target device blinks LED2.
   (b) Press the S2 button on the target board to toggle LED2.
   (c) Because the CI is initialized, the host can send a Force Boot command to force bootloader mode in the target device at the start of a new transfer sequence.
11. Press the S1 button on the host to start a new sequence sending APP1 again.

Dual-image mode contains a brief pause from the host after the transfer is complete while it validates the download area, transfers the memory into the application space, and erases the download area after the application area is validated by a CRC-CCITT check.

6 References

1. *MSPBoot – Main Memory Bootloader for MSP430 Microcontrollers*
2. *MSP430FR57xx, MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Bootloader (BSL)*
3. *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide*
4. *CC1101 Low-Power Sub-1 GHz RF Transceiver*

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from October 13, 2016 to December 20, 2016	Page
• Added new Byte$_2$ column and renumbered following bytes in Table 5, *BSL-Based Protocol Commands*	12

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		