

# MSP430<sup>™</sup> Bootloader With Sitara<sup>™</sup> Embedded Linux Host

Nima Eskandari

MSP430 Applications

### ABSTRACT

The MSP430<sup>™</sup> bootloader (BSL) allows a host processor to communicate with embedded memory in the MSP430 microcontroller (MCU). The host can access the programmable memory (flash memory), the data memory (RAM), and in FRAM devices, the nonvolatile FRAM memory. The host processor can access the memory of the target MSP430 MCU during the prototyping phase, final production, and in service (field software updates).

This application report uses the BeagleBone Black, featuring the Sitara<sup>™</sup> AM3358BZCZ100 ARM® Cortex®-A8 32-Bit RISC processor, as the host for the BSL communication with the target MSP430 MCUs. Both flash-based and FRAM-based MSP430 MCUs are used in this document to showcase the differences between the BSLs. A software example is provided for the BeagleBone Black board, to showcase communication with the embedded memory of MSP430 MCUs. The software examples run on top of the Linux<sup>®</sup> operating system for Sitara processors.

The source code and other files described in this application report can be downloaded from http://www.ti.com/lit/zip/slaa760. The example source code demonstrates how a Sitara processor with a Linux operating system can access the memory of a target MSP430 MCU (flash or FRAM based) through UART BSL.

	Contents	
1	Introduction	2
	1.1 Supplementary Online Information	2
2	Software Example	3
	2.1 Software Example File Descriptions	
3	Bootloader (BSL) Connections	
	3.1 Bootloader Connections for the Target MSP430 MCU	5
	3.2 Bootloader Connections for the Linux Host Device (BeagleBone Black)	8
4	How to Use the Software Examples	9
	4.1 Transmit the Example Projects to BeagleBone Black (PSCP)	9
	4.2 Run the Software Examples	11
5	Create Custom MSP430 Firmware Image	16
	5.1 MSP430 Flash Firmware Image	16
	5.2 MSP430 FRAM Firmware Image	17
	5.3 Generating Custom Firmware Image Header Files	19
6	UART BSL Command Line Utility For MSP430 FRAM Devices	22
7	Error Messages	23
	List of Figures	
1	MSP430G2553 BSL Entry Sequence Pins	5
2	MSP430FR2311 BSL Entry Sequence Pins	6
3	MSP430G2553 UART BSL Pins	7
4	MSP430FR2311 UART BSL Pins	7

# Contonto

5

6 7

1

Transmit UART\_BSL\_MSP430FR to BeagleBone Black ...... 10



#### Introduction

### www.ti.com

8	Transmit UART_BSL_MSP430 to BeagleBone Black	10
9	Building Software Example Projects	11
10	BeagleBone Black Connected to MSP430G2553	11
11	MSP430G2553 LaunchPad Development Kit UART BSL Connections	12
12	UART_BSL_MSP430 Project Console Output	12
13	MSP430G2553 Programmed Successfully	13
14	BeagleBone Black Connected to MSP430FR2311	14
15	MSP430FR2311 LaunchPad Development Kit UART BSL Connections	14
16	UART_BSL_MSP430FR Console Output	15
17	MSP430FR2311 Console Output (DEBUG is 1)	15
18	Enabling MSP430 Hex Utility	
19	MSP430 TI-TXT Hex Format	19
20	MSP430FR2311 TI-TXT Hex Format Firmware Image	
21	FRAM TI-TXT Hex to Firmware Image Converter	21
22	Firmware Image Python Script Console Output	21
23	UART_BSL_MSP430FR_Command_Line_Utility Console Output	22
24	Error Messages With DEBUG Set to 0	23
25	Error Messages With DEBUG Set to 1	24

### Trademarks

MSP430, Sitara, E2E, Code Composer Studio are trademarks of Texas Instruments. ARM, Cortex are registered trademarks of ARM Ltd. Linux is a registered trademark of Linus Torvalds. All other trademarks are the property of their respective owners.

### 1 Introduction

The bootloader provides a method for the MSP430 MCUs to be programmed. For the MSP430 MCU to be programmed through BSL, commands must be sent using the UART protocol to the MSP430 MCU.

To invoke the bootloader, a BSL entry sequence must be applied to dedicated pins. After the BSL entry sequence is applied, UART commands can be sent to the target MSP430 MCU. The BSL UART commands are different depending on whether the MSP430 MCU is flash or FRAM based.

Software examples are provided for both flash and FRAM-based MSP430 MCUs. These software examples use the BeagleBone Black, featuring a Sitara AM3358 processor, as the UART BSL host to the target MSP430 MCU. The UART BSL protocol in these examples is implemented on top of the Linux operating system, which makes it extremely easy to port to other Linux-based devices.

# 1.1 Supplementary Online Information

For more information and tool, visit Bootloader (BSL) for MSP low-power microcontrollers. This page contains links to additional BSL user's guides, source code, firmware images, and the BSL scripter with documentation and code examples.

For FRAM-based MSP430 MCUs:

MSP430FR5xx and MSP430FR6xx Bootloader (BSL) User's Guide

MSP430FR4xx and MSP430FR2xx Bootloader (BSL) User's Guide

For flash-based MSP430 MCUs:

MSP430<sup>™</sup> Flash Device Bootloader (BSL) User's Guide

Additional support is provided by the TI E2E<sup>™</sup> Community.

MSP430<sup>™</sup> Bootloader With Sitara<sup>™</sup> Embedded Linux Host



### 2 Software Example

Example software is available for the BeagleBone Black to act as the UART BSL host to target MSP430 MCUs. This code has also been tested on the BeagleBone (original).

Software Example

The following development kits were used to develop the software examples:

- BeagleBone Black
- BeagleBone (Original)

The example code is developed and tested on top of the Linux operating system.

The BeagleBone image used in the software example can be downloaded from Bone Debian 8.6.

To get the BeagleBone Black (or original) ready for the software example provided in this report, follow the steps provided in BeagleBoard.com Start Your Beagle (Getting Started Guide).

### 2.1 Software Example File Descriptions

The example software can be downloaded from http://www.ti.com/lit/zip/slaa760.

Table 1 describes the content of the top-level folder.

Folder Name	Description
UART_BSL_MSP430	Example source code for BeagleBone Black (or original) running the Linux operating system. BeagleBone Black acts as a UART BSL host for MSP430 flash based devices. This example was developed and tested with BeagleBone Black (and original) running Debian 8.6.
UART_BSL_MSP430FR	Example source code for BeagleBone Black (or original) running the Linux operating system. BeagleBone Black acts as a UART BSL host for MSP430 FRAM based devices. This example was developed and tested with BeagleBone Black (and original) running Debian 8.6.
UART_BSL_MSP430FR_Command_Line_Utility	Example source code for BeagleBone Black (or original) running the Linux operating system. BeagleBone Black acts as a UART BSL host for MSP430 FRAM based devices. This project creates a command line utility that takes in the GPIO pins used for UART BSL entry sequence, the UART module number for UART BSL commands, and the TI-TXT hex firmware image file (for the target MSP430 MCU). Using the given parameters, this utility updates the firmware on the target MSP430 MCU. This example was developed and tested with BeagleBone Black (and original) running Debian 8.6.
Python_Scripts/flash_TI_txt_hex_to_byte_image.py and Python_Scripts/FRAM_TI_txt_hex_to_byte_image.py	Converts a hex TI-TXT file to the header file format used by the host. These files are python scripts and must be run using python. Example: python flash_TI_txt_hex_to_byte_image.py

### **Table 1. Top-Level Folder Description**

Each project contains the drivers for the peripherals used by the Linux host. Each project folder also contains the BSL UART commands and an example firmware image for the target MSP430 MCU.

Table 2 describes the content of each project.

Name	Description
image/msp430_image.h or image/msp430fr_image.h	Contains the example firmware image for flash or FRAM target MSP430. The msp430_image.h contains the blinky example for MSP430G2553 device, while the msp430fr_image.h contains the blinky example for MSP430FR2311 device.
bsl.c	Implementation of the UART BSL commands. This file contains the commands used to program an MSP430 MCU through UART BSL. This file generates the BSL Command packages to send. It also receives and unpacks the BSL Responses from the target. For the UART_BSL_MSP430 project, the BSL commands defined in this folder are the flash-based BSL commands. For the UART_BSL_MSP430FR project, the BSL commands are the FRAM-based BSL commands.
bsl.h	Contains the function declarations for the BSL commands.
main.c	Initializes the hosts peripherals and programs the target device with the specified example firmware image.
config.h	Contains configurable options such as UART read timeout and whether or not to output software progress to the serial terminal.
debug.c	Initializes the backchannel UART for communication with PC (serial terminal programs such as PuTTY)
debug.h	Contains the function declarations for the debug.c
gpio_if.c	Device specific interface file to access the GPIOs. The GPIOs are used to control the dedicated BSL entry sequence pins.
gpio_if.h	Contains the function declarations for the gpio_if.c
uart_if.c	Interfaces with the UART module on the device to send and receive BSL commands.
uart_if.h	Contains the function declarations for the uart_if.c
utils.c	Interfaces with the timer modules on the device to generate specified delays. Also contain other utilities for debugging.
utils.h	Contains the function declarations for the utils.c

Other than the UART\_BSL\_MSP430 and UART\_BSL\_MSP430FR projects, there is also a command line utility for FRAM based MSP430 MCUs available in the folder.

The UART\_BSL\_MSP430FR\_Command\_Line\_Utility project is based on the UART\_BSL\_MSP430FR project. The UART\_BSL\_MSP430FR project programs the target MSP430 FRAM based device with the example firmware image, using GPIO 50, GPIO 51 and UART4. Meanwhile, the UART\_BSL\_MSP430FR\_Command\_Line\_Utility takes in two GPIO numbers (for TEST and RESET pins), the UART module number (for example 4) and the path to a TI-TXT hex format firmware image file. After parsing the firmware image file, the utility programss the target MSP430 MCU with the specified firmware image, using the specified pins.



# **3 Bootloader (BSL) Connections**

The host Sitara processor and target MSP430 MCU must be connected to through the BSL pins and share the GND signal. The BSL pins include the dedicated BSL entry pins and the UART BSL pins.

# 3.1 Bootloader Connections for the Target MSP430 MCU

To access the MSP430 memory through the BSL, an entry sequence must be applied to the dedicated pins. In this application report, the MSP430G2553 and MSP430FR2311 are used as the target for BSL communication.

The BSL entry sequence must be applied to the TST and RST pins on the target MSP430 MCU, to invoke the bootloader. For more information on the BSL entry sequence see the appropriate bootloader user's guide listed in Section 1.1.

Figure 1 and Figure 2 show the dedicated pins for BSL entry sequence for the MSP430G2553 and MSP430FR2311.

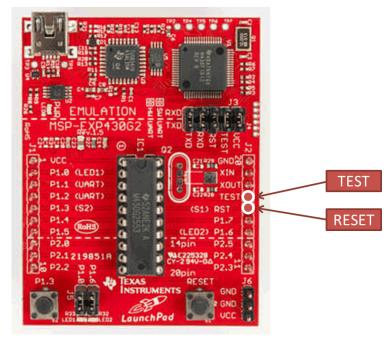


Figure 1. MSP430G2553 BSL Entry Sequence Pins

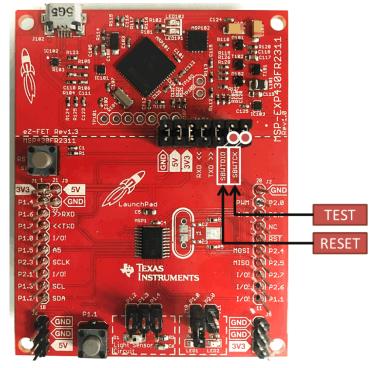


Figure 2. MSP430FR2311 BSL Entry Sequence Pins

These pins are connected to the GPIO of the host. After the BSL entry sequence is completed, the host can use the dedicated UART BSL pins to send and receive BSL commands.

The pins for BSL entry sequence are Reset and Test pins. Some LaunchPad development kits specify these two pins as SBWTDIO and SBWTCK.

- SBWTDIO: Spy-By-Wire Data Input/Output (RESET PIN)
- SBWTCK: Spy-By-Wire Clock (TEST PIN)

Figure 3 and Figure 4 show the dedicated UART pins for the MSP430G2553 and MSP430FR2311. To find the UART BSL pins for any MSP430 MCU, see the device-specific data sheet. The UART BSL pins are specific to each device and package. For example, the UART BSL pins for the MSP430G2553 are P1.1 and P1.5, while the UART BSL pins for the MSP430FR2311 are P1.6 and P1.7.



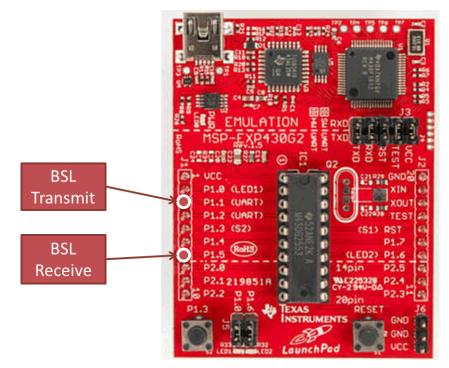


Figure 3. MSP430G2553 UART BSL Pins

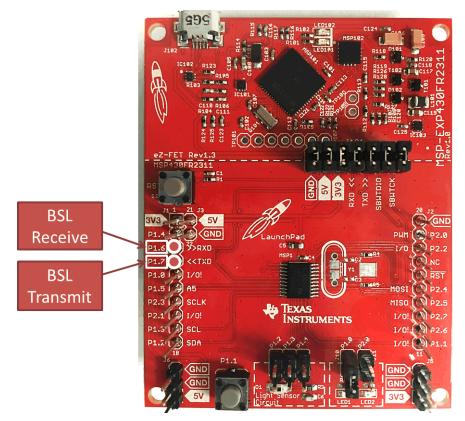


Figure 4. MSP430FR2311 UART BSL Pins



### Bootloader (BSL) Connections

www.ti.com

The final connections needed on the target MSP430 MCU are the power connections. All power pins must be connected to the required voltages. In this case, both MSP430FR2311 and MSP430G2553 are connected to a 3.3-V supply through the VCC and GND pins.

# 3.2 Bootloader Connections for the Linux Host Device (BeagleBone Black)

The host device uses two GPIO pins to execute the BSL entry sequence. The host also uses a UART module. The UART module communicates to the BSL target to transmit commands and receive the responses.

The software example provided for the BeagleBone Black uses the following pins:

- Target Reset Pin: GPIO 50 (P9.14)
- Target Test Pin: GPIO 51 (P9.16)
- BSL Communication UART:
  - TX: UART4\_TXD (P9.11)
  - RX: UART4\_RXD (P9.13)

The female header on the left side of the board is named P9, and the female header on the right side of the board is named P8. Figure 5 shows the BeagleBone Black and the pins used for the software example.

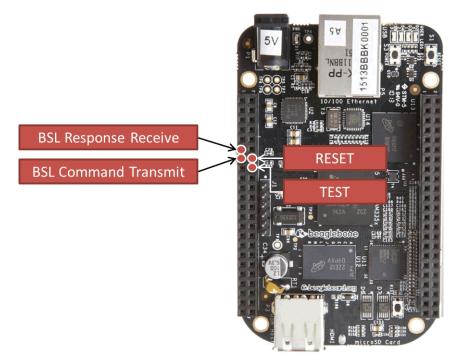


Figure 5. BeagleBone Black UART BSL Host Pins

For a description of all pins on the BeagleBone Black, see http://beagleboard.org/support/bone101.

The Linux host connects to the MSP430 MCU through the Reset (BSL entry pin), Test (BSL entry pin), BSL UART TX pin, and BSL UART RX pin. The Linux host and MSP430 MCU must share the ground signal. Also, the MSP430 MCU must be powered up. Finally, a PC can be used to view the status of the MSP430 firmware update through SSH.



# 4 How to Use the Software Examples

The software examples in this application report use the BeagleBone Black board with Debian 8.6 operating system.

The software example must be compiled using the GCC available on the BeagleBone Black.

Download and extract the zip file containing the software examples. The files available after extraction must be sent to the BeagleBone Black. After downloading PuTTY, PSCP can be used to transmit the files from a Windows PC to the BeagleBone Black.

# 4.1 Transmit the Example Projects to BeagleBone Black (PSCP)

To transmit the projects from a Windows PC to the BeagleBone Back, download PuTTY and use PSCP.

The IP Address of the BeagleBone is required for PSCP to transmit the files. Navigate to the folder where PuTTY programs are located. Execute the following commands to transmit the projects to the BeagleBone Black.

- pscp -r "C:\<PATH TO THE EXAMPLE PROJECTS FOLDER>\UART\_BSL\_MSP430FR\_Command\_Line\_Utility" root@BEAGLEBONE\_IP\_ADDRESS:<PATH OF THE DESTINATION FOLDER> Example (see Figure 6): pscp -r "C:\Users\UART\_BSL\_MSP430FR\_Command\_Line\_Utility" root@128.247.87.83:/home/root/final
- pscp -r "C:\<PATH TO THE EXAMPLE PROJECTS FOLDER>\UART\_BSL\_MSP430FR " root@BEAGLEBONE\_IP\_ADDRESS:<PATH OF THE DESTINATION FOLDER> Example (see Figure 7):

pscp -r "C:\Users\UART\_BSL\_MSP430FR" root@128.247.87.83:/home/root/final

 pscp -r "C:\<PATH TO THE EXAMPLE PROJECTS FOLDER>\UART\_BSL\_MSP430" root@BEAGLEBONE\_IP\_ADDRESS:<PATH OF THE DESTINATION FOLDER> Example (see Figure 8):

pscp -r "C:\Users\UART\_BSL\_MSP430" root@128.247.87.83:/home/root/final

Command Prompt		
C:\Users\a0225962\Document	s\putty>pscp -r "C:\Users\	workspace_linux\UAR
T BSL MSP430FR Command Lin	e_Utility" root@128.247.87	.83:/home/root/final
COMMIT_EDITMSG		00:00:00   100%
config		00:00:00   100%
description		00:00:00   100%
HEAD		00:00:00   100%
applypatch-msg.sample		00:00:00   100%
commit-msg.sample		00:00:00   100%
post-update.sample		00:00:00   100%
pre-applypatch.sample		00:00:00   100% 00:00:00   100%
pre-commit.sample pre-push.sample		00:00:00   100% 00:00:00   100%
pre-rebase.sample		00:00:00   100%
prepare-commit-msq.sample		00:00:00   100%
update.sample		00:00:00   100%
index		00:00:00   100%
exclude		00:00:00   100%
HEAD		00:00:00 100%
master	0 kB 0.4 kB/s ETA:	00:00:00   100%
master		00:00:00   100%
1c563a32253d0d3b0f695a89d	0 kB   0.5 kB/s   ETA:	00:00:00   100%

Figure 6. Transmit UART\_BSL\_MSP430FR\_Command\_Line\_Utility to BeagleBone Black



Command Prompt						
C:\Users\a0225962\Documents T_BSL_MSP430FR" root@128.24	5\putty: 17.87.83	⊳pscp -r "C:\l B:/home/root/f	Jsers` Final	10W/	*kspace_1	inux\UAR
COMMIT_EDITMSG	0 kB	0.0 kB/s	ETA: 0		100%	
config	0 kB	0.3 kB/s			100%	
description	0 kB	0.1 kB/s			100%	
HEAD	0 kB	0.0 kB/s			100%	
applypatch-msg.sample commit-msg.sample	0 kB   0 kB	0.5 kB/s 0.9 kB/s			100% 100%	
post-update.sample	0 kB	0.9  kB/s 0.2 kB/s			100%	
pre-applypatch.sample	0  kB	0.2  kB/s			100%	
pre-commit.sample	1 kB	1.6 kB/s			100%	
pre-push.sample	1 kB	1.3 kB/s			100%	
pre-rebase.sample	4 kB	4.8 kB/s	ETA: 0	0:00:00 j	100%	
prepare-commit-msg.sample	1 kB	1.2 kB/s			100%	
update.sample	3 kB	3.5 kB/s			100%	
index	1 kB	1.1 kB/s			100%	
exclude	0 kB	0.2 kB/s			100%	
HEAD	1 kB	1.3 kB/s			100%	
master	1 kB	1.3 kB/s			100%	
master 1c563a32253d0d3b0f695a89d	1 kB   0 kB	1.2 kB/s 0.5 kB/s			100% 100%	<b>—</b>
	UKB	0.J KB/S	ETA: U	0.00.00	100%	

Figure 7. Transmit UART\_BSL\_MSP430FR to BeagleBone Black

Command Prompt				
				· · · · · · · · · · · · · · · · · · ·
C:\Users\a0225962\Document	s\putty>p	oscp -r "C:\Us	ers' (we	orkspace_linux\UAR
T_BSL_MSP430" root@128.247				
COMMIT_EDITMSG	0 kB		ETA: 00:00:00	100%
config	0 kB		ETA: 00:00:00	100%
description	0 kB		ETA: 00:00:00	100%
HEAD	0 kB		ETA: 00:00:00	100%
applypatch-msg.sample	0 kB		ETA: 00:00:00	100%
commit-msg.sample	0 kB		ETA: 00:00:00	100%
post-update.sample	0 kB	0.2 kB/s	ETA: 00:00:00	100%
pre-applypatch.sample	0 kB	0.4 kB/s	ETA: 00:00:00	100%
pre-commit.sample	1 kB	1.6 kB/s	ETA: 00:00:00	100%
pre-push.sample	1 kB	1.3 kB/s	ETA: 00:00:00	100%
pre-rebase.sample	4 kB	4.8 kB/s	ETA: 00:00:00	100%
prepare-commit-msg.sample	1 kB	1.2 kB/s	ETA: 00:00:00	100%
update.sample	3 kB	3.5 kB/s	ETA: 00:00:00	100%
index	1 kB	1.1 kB/s	ETA: 00:00:00	100%
exclude	i0 kB i	0.2 kB/s	ETA: 00:00:00	100%
HEAD	1 kB		ETA: 00:00:00	100%
master	1 kB		ETA: 00:00:00	100%
master	1 kB i		ETA: 00:00:00	100%
de2d7735942bf92637c987959	0 kB		ETA: 00:00:00	100% -

Figure 8. Transmit UART\_BSL\_MSP430 to BeagleBone Black

If the PSCP command did not work the first time and resulted in errors, try the PSCP command again. After transmitting, the projects must be built.

Execute the following commands in the corresponding project folder to build each project.

```
    UART_BSL_MSP430
```

gcc -I ./ main.c uart\_if.c pinmux.c gpio\_if.c utils.c bsl.c -o msp430\_bsl

• UART\_BSL\_MSP430FR

gcc -I ./ main.c uart\_if.c pinmux.c gpio\_if.c utils.c bsl.c -o msp430fr\_bsl

• UART\_BSL\_MSP430FR\_Command\_Line\_Utility

gcc -I ./ main.c uart\_if.c pinmux.c gpio\_if.c utils.c bsl.c -o command\_line\_bsl



root@beagl	.ebone:/# cd /home	/root/final		
	.ebone:/// ed /nome/root/			
			SL MSP430FR Command	Line Utility
	ebone:/home/root/			
			MSP430# gcc -I ./ ma	in.c uart if.c pir
	if.c utils.c bsl			
coot@beagl	_ ebone:/home/root/	final/UART BSL 1	MSP430# ls	
EADME .md	config.h image	e pinmux.c	uart if.h	
sl.c	gpio_if.c main.	c pinmux.h	utils.c	
sl.h	gpio_if.h msp43	0_bsl uart_if.	c utils.h	
coot@beagl	.ebone:/home/root/	final/UART_BSL_1	MSP430# cd	
	ebone:/home/root/			
			MSP430FR# gcc -I ./	main.c uart_if.c p
	io_if.c utils.c k			
	ebone:/home/root/			
	config.h image			
osl.c				
sl.h	SF			
	ebone:/home/root/			
			3SL_MSP430FR_Command MSP430FR Command Lin	
			ls.c bsl.c -o command	
			MSP430FR Command Lin	
			nmux.c uart if.h	e_ocriticy# is
	config.h			
	gpio if.c			

Figure 9. Building Software Example Projects

The content of each project is shown after the software examples have been built. Each project now has an executable file. msp430\_bsl, msp430fr\_bsl, and command\_line\_bsl are the three executable files.

### 4.2 Run the Software Examples

Connect the BeagleBone Linux host to the target MSP430 MCU. For the first part of this demo, the BeagleBone Black and MSP430G2553 are used. As for the second part, MSP430FR2311 is used as the target.

### 4.2.1 MSP430G2553 UART BSL Target Example

Connect the two devices. The pin to pin connection is also available in the README file.

Figure 10 and Figure 11 show the connections for the BeagleBone Black and MSP430G2553 LaunchPad development kit.

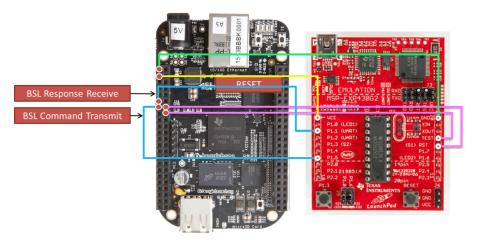
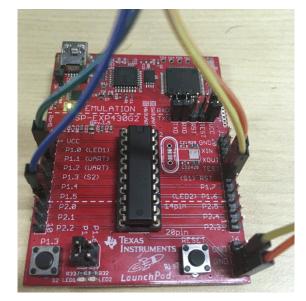


Figure 10. BeagleBone Black Connected to MSP430G2553





# Figure 11. MSP430G2553 LaunchPad Development Kit UART BSL Connections

Run the UART\_BSL\_MSP430 example by executing ./msp430\_bsl. Figure 12 shows the output.

Putty 128.247.87.83 - Putty	
*****New Attempt****	<b>^</b>
Write Password was successful	
Device ID: 0x2553	
****Section: 0****	
Address: 0xC000 Erase successful	
Erase Successiui	
****Section: 1****	
Address: 0xFFDE	
Erase successful	
****Section: 2****	
Address: 0xFFE4	
Erase successful	
****Section: 3****	
Address: 0xFFEA Erase successful	
****Section: 0****	
Address: 0xC000	
Erase check successful Write Large Data To Memory successful	
Write Large Data to Memory Successful	
****Section: 1****	
Address: 0xFFDE	
Erase check successful	
Write Large Data To Memory successful	
****Section: 2****	
Address: 0xFFE4	-

Figure 12. UART\_BSL\_MSP430 Project Console Output



The target MSP430G2553 is programmed with the example firmware image. Figure 13 shows the result of successful programming.

🛃 128.247.87.83 - PUTTY	X
	-
Address: 0xFFDE	
Erase successful	
****Section: 2****	
Address: 0xFFE4	
Erase successful	
****Section: 3****	
Address: 0xFFEA	
Erase successful	
****Section: 0****	
Address: 0xC000	
Erase check successful	
Write Large Data To Memory successful	
****Section: 1****	
Address: 0xFFDE Erase check successful	
Write Large Data To Memory successful	
Wite harge back to Memory Successful	
****Section: 2****	
Address: 0xFFE4	
Erase check successful Write Large Data To Memory successful	
Wille harge baca to Memory Successivi	
****Section: 3****	
Address: 0xFFEA	
Erase check successful Write Large Data To Memory successful	
Mille Large Data to Memory Succession	
MSP430 programmed successfully	
Device is reset	=
root@beaglebone:/home/root/final/UART_BSL_MSP430#	-

Figure 13. MSP430G2553 Programmed Successfully

The MSP430G2533 device ID is 0x2553. All MSP430 MCUs have a unique device ID. For example, the Device ID for MSP430FR2311 is 0xFF80, which is shown in Section 4.2.2.

Finally, the MSP430 MCU is programmed and the message, "MSP430 Programmed Successfully" is shown.

The example program that is downloaded to the target MSP430 MCU toggles P1.0, which can be seen by the toggling of the LED on the MSP430G2553 LaunchPad development kit.

Follow these same steps to program any flash-based MSP430 MCU using the BeagleBone Black.



# 4.2.2 MSP430FR2311 UART BSL Target Example

To program an MSP430FR2311 using the BeagleBone Black through UART BSL, follow the steps in Section 4.2.1.

Figure 14 shows the connections between the MSP430FR2311 and the BeagleBone Black.

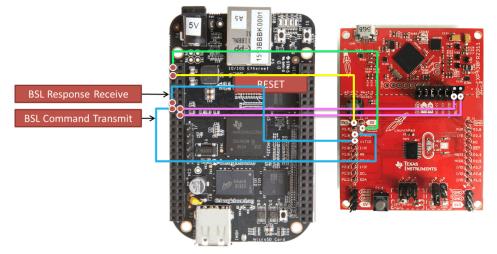


Figure 14. BeagleBone Black Connected to MSP430FR2311

Figure 15 shows the MSP430FR2311 connections.



Figure 15. MSP430FR2311 LaunchPad Development Kit UART BSL Connections

Run the UART\_BSL\_MSP430 example by executing ./msp430fr\_bsl. Figure 16 shows the console output.



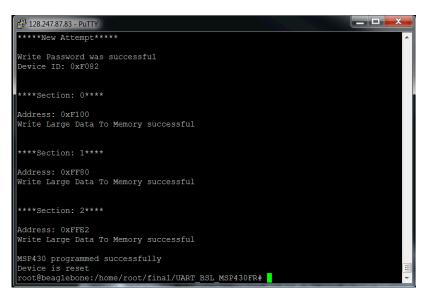


Figure 16. UART\_BSL\_MSP430FR Console Output

To view the execution of the example software with more details, change the DEBUG macro in config.h from 0 to 1. Figure 17 shows an example console output.



Figure 17. MSP430FR2311 Console Output (DEBUG is 1)



### Create Custom MSP430 Firmware Image

Follow the same steps to program any FRAM-based MSP430 MCU using BeagleBone Black.

The default firmware image in all MSP430 flash-based examples is the MSP430G2553 blinky firmware image. This program toggles P1.0 on and off.

The default firmware image in all MSP430 FRAM-based examples is the MSP430FR2311 blinky firmware image. This program toggles P1.0 on and off.

For information on creating a new firmware image for any MSP430 MCU from an existing Code Composer Studio<sup>™</sup> IDE (CCS) project, see Section 5.

# 5 Create Custom MSP430 Firmware Image

The software example provides a default example firmware image which is inside the msp430\_image.h or msp430fr\_image.h header file (based on whether the target MSP430 is flash or FRAM based).

The following sections describe the format of the firmware image header file.

### 5.1 MSP430 Flash Firmware Image

The default example firmware image for the flash based MSP430 MCUs is the blinky example for MSP430G2553.

```
/*
* This file was automatically generated.
* The memory sections should be quickly double checked.
* Some sections such ad Start, Finish, Termination and Length must be
* modified based on device datasheet. (These values aren't used by the
* default program.
* Created by: Nima Eskandari
* /
uint8_t flash[] =
{
//0xc000
0x21, 0x83, 0xB2, 0x40, 0x80, 0x5A, 0x20, 0x01,
0xD2, 0xD3, 0x22, 0x00, 0xD2, 0xE3, 0x21, 0x00,
0xB1, 0x40, 0x50, 0xC3, 0x00, 0x00, 0x91, 0x83,
0x00, 0x00, 0x81, 0x93, 0x00, 0x00, 0xF6, 0x27,
0xFA, 0x3F, 0x31, 0x40, 0x00, 0x04, 0xB0, 0x12,
0x42, 0xC0, 0x0C, 0x43, 0xB0, 0x12, 0x00, 0xC0,
0xB0, 0x12, 0x3C, 0xC0, 0x32, 0xD0, 0x10, 0x00,
0xFD, 0x3F, 0x03, 0x43, 0x03, 0x43, 0xFF, 0x3F,
0x03, 0x43, 0x1C, 0x43, 0x30, 0x41,
//0xffde
0xFF, 0xFF, 0x34, 0xC0,
//0xffe4
0x34, 0xC0, 0x34, 0xC0,
//Oxffea
0x34, 0xC0, 0x34, 0xC0, 0x34, 0xC0, 0x34, 0xC0,
0x34, 0xC0, 0x34, 0xC0, 0x34, 0xC0, 0x34, 0xC0,
0x34, 0xC0, 0x34, 0xC0, 0x22, 0xC0,
};
const uint32_t flash_address[] =
0xc000, 0xffde, 0xffe4, 0xffea,
};
```



```
const uint32_t flash_length_of_sections[] =
{
70, 4, 4, 22,
};
const uint32_t flash_sections = 4;
const uint32_t flash_termination = 0x00000000; /*Check device data sheet*/
const uint32_t flash_start = 0x0000000; /*Check device data sheet*/
const uint32_t flash_finish = 0x0000000; /*Check device data sheet*/
const uint32_t flash_length = 0x0000000; /*Check device data sheet*/
```

The first variable is *flash*. The *flash\_sections* variable holds the number of start addresses that is required to be programmed. In this case, the *flash\_sections* variable is set to 4. This is consistent with the size of *flash\_address* array. The *flash\_address* holds four addresses. These addresses are 0xC000, 0xFFDE, 0xFFE4, and 0xFFEA. The *flash\_length\_of\_sections* array specifies the number of bytes in the *flash* variable for each of the addresses specified in *flash\_address* array.

The example default image is interpreted as follows:

• For address 0xC000

The first 70 bytes of data in flash must be written to the MSP430 memory, starting at address 0xC000.

• For address 0xFFDE

Starting from the 71<sup>st</sup> element in the flash variable, 4 bytes must be written to the MSP430 memory, starting at address 0xFFDE.

For address 0xFFE4

Starting at the 75<sup>th</sup> element in the flash variable, 4 bytes must be written to the MSP430 memory, starting at address 0xFFE4.

For address 0xFFEA

Starting at the 79<sup>th</sup> element in the flash variable, 22 bytes must be written to the MSP430 memory, starting at address 0xFFEA.

# 5.2 MSP430 FRAM Firmware Image

The default example firmware image for the FRAM based MSP430 MCUs is the blinky example for MSP430FR2311. The firmware image file is formatted the exact same way as the flash based MSP430 MCUs. The only difference is that variables are all renamed from flash to fram.

```
/*
* This file was automatically generated.
* The memory sections should be quickly double checked.
* Some sections such ad Start, Finish, Termination and Length must be
* modified based on device datasheet. (These values aren't used by the
* default program.
* Created by: Nima Eskandari
*/
uint8_t fram[] =
{
//0xf100
0xB2, 0x40, 0x80, 0x5A, 0xCC, 0x01, 0xD2, 0xC3,
0x02, 0x02, 0xD2, 0xD3, 0x04, 0x02, 0x92, 0xC3,
0x30, 0x01, 0xD2, 0xE3, 0x02, 0x02, 0x1E, 0x14,
0x3D, 0x40, 0x3C, 0x82, 0x3E, 0x40, 0x0E, 0x00,
0x1D, 0x83, 0x0E, 0x73, 0xFD, 0x23, 0x0D, 0x93,
```



Create Custom MSP430 Firmware Image

```
0xFB, 0x23, 0x1D, 0x16, 0xF2, 0x3F, 0x03, 0x43,
0x03, 0x43, 0xFF, 0x3F, 0x03, 0x43, 0x1C, 0x43,
0x10, 0x01, 0x31, 0x40, 0x00, 0x24, 0xB0, 0x13,
0x36, 0xF1, 0x0C, 0x43, 0xB0, 0x13, 0x00, 0xF1,
0xB0, 0x13, 0x30, 0xF1, 0x32, 0xD0, 0x10, 0x00,
0xFD, 0x3F, 0x03, 0x43,
//0xff80
OxFF, OxFF, OxFF, OxFF, OxFF, OxFF, OxFF, OxFF,
//0xffe2
0x4C, 0xF1, 0x4C, 0xF1, 0x4C, 0xF1, 0x4C, 0xF1,
0x4C, 0xF1, 0x4C, 0xF1, 0x4C, 0xF1, 0x4C, 0xF1,
0x4C, 0xF1, 0x4C, 0xF1, 0x4C, 0xF1, 0x4C, 0xF1,
0x4C, 0xF1, 0x4C, 0xF1, 0x3A, 0xF1,
};
const uint32_t fram_address[] =
0xf100, 0xff80, 0xffe2,
};
const uint32_t fram_length_of_sections[] =
{
84, 8, 30,
};
const uint32_t fram_sections
                               = 3;
const uint32_t fram_termination = 0x00000000; /*Check device data sheet*/
const uint32_t fram_start = 0x00000000; /*Check device data sheet*/
                           = 0x00000000; /*Check device data sheet*/
const uint32_t fram_finish
                               = 0x0000000; /*Check device data sheet*/
const uint32_t fram_length
```

To generate a custom firmware image header file, follow the instructions in Section 5.3.

INSTRUMENTS

ÈXAS

www.ti.com

# 5.3 Generating Custom Firmware Image Header Files

To generate firmware image header files, the firmware must be compiled and the output must be in TI-TXT hex format.

The following steps describe how to generate a TI-TXT hex file in CCS.

- 1. Right click on the project and click on the properties option.
- 2. Check the Enable MSP430 Hex Utility checkbox (see Figure 18).

Very Properties for FR2311_blinky	MSP430 Hex Utility		•
<ul> <li>&gt; Resource</li> <li>General</li> <li> <ul> <li>Build</li> <li>MSP430 Compiler</li> <li>Processor Options</li> </ul> </li> </ul>	Configuration: Debug	[Active ]   Manage Configurations	•
Optimization Include Options ULP Advisor Advice Options Predefined Symbols > Advanced Options	Enable MSP430 Hex U Command: Command-line pattern: Summary of flags set:	"\$(CG_TOOL_HEX)" \${command} \${flags} \${output_flag} \${output} \${inputs}	H
<ul> <li>MSP430 Lieker</li> <li>MSP430 Hex Utility General Options</li> <li>Diagnostics Options</li> <li>Output Format Options</li> <li>Load Image Options</li> </ul>	memwidth=8romw	idth=8ti_txt	
Debug		-	Ŧ
Show advanced settings		OK Cancel	

Figure 18. Enabling MSP430 Hex Utility

3. In Output Format Options, select TI-TXT hex format (see Figure 19).

type filter text	Output Format Options 🗢 👻 🖘			
<ul> <li>&gt; Resource General</li> <li>&gt; Build</li> <li>&gt; MSP430 Compiler Processor Options</li> </ul>	Configuration:	Debug [Active]		
Optimization Include Options ULP Advisor Advice Options Predefined Symbols Advanced Options MSP430 Linker MSP430 Hex Utility General Options Diagnostics Options Output Format Options Load Image Options Debug	Output format	Output TI-TXT hex format (ti_txt)  Output ASCII hex format (ascii, -a) Output Intel hex format (intel, -i) Output Motorola S hex format (motorola, -m=1) Output Motorola S hex format (motorola, -m=2) Output Motorola S hex format (motorola, -m=3) Output Extended Tektronix hex format (tektronix, -x) Output TI-TSgeed hex format (ti_tagged, S) Output TI-TXT hex format (ti_txt)		
Show advanced settings		OK Cancel		

# Figure 19. MSP430 TI-TXT Hex Format

4. After the project is built, the .txt containing the full firmware image is created. Figure 20 shows this file.

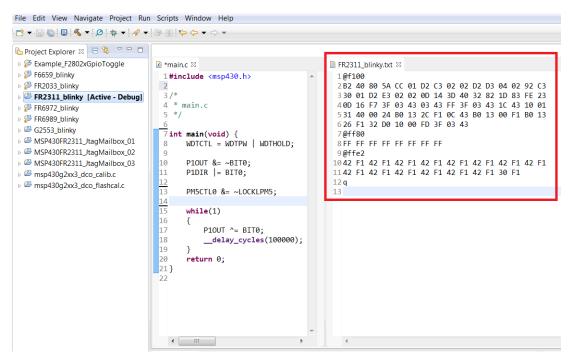


Figure 20. MSP430FR2311 TI-TXT Hex Format Firmware Image

The generated TI-TXT file can be converted to a firmware image header file.

Python scripts are provided to convert the TI-TXT file to a firmware image header file. The scripts are in the Python\_Scripts folder of the example software zip file.

There are two Python scripts. The only difference between the two scripts is whether the created header file is for an FRAM-based device or a flash-based device.

Texas

*TRUMENTS* 

D		
📑 👘 FRA		iex_to_byte_image.py × 🛛 🖡 FLASH_TI_txt_hex_to_byte_image.py ×
1		
2		ated By: <u>Nima Eskandari</u>
З		pription: Creates a .h file from TI TXT HEX file.
4		
5		
6		DDO: Change th sourcePath variable to the path of the TI HEX TXT file.
7		
8		ccePath = r"C:\Users\Note: CePath = r"C:\Users\\text{Note: CePath = r"C:\users\_\text{Note: CePath = r"C:\users\_\text{No
9	imag	gePath = "msp430fr_image.h"
10		
11		MemorySection:
12		
13		<pre>definit(self, content):</pre>
14		self.Content = content
15		<pre>self.ContentSplitBySpace = []</pre>
16		<pre>content_split_by_line = self.Content.split('\n')</pre>
17		for line in content_split_by_line:
18		<pre>line_split_by_space = line.split(' ')</pre>
19		<pre>for i in range(0, len(line_split_by_space)):</pre>
20		<pre>line_split_by_space[i] = line_split_by_space[i].replace(' ', '')</pre>
21		<pre>if line_split_by_space[i] != '':</pre>
22		<pre>self.ContentSplitBySpace.append(line_split_by_space[i])</pre>
23		
24		<pre>self.StartAddress = self.get_start_address()</pre>
25		<pre>self.SizeInBytes = self.get_number_of_bytes()</pre>
26		<pre>self.SizeInWords = int(self.SizeInBytes / 2)</pre>
27		<pre>self.ContentInBytes = self.get_content_in_bytes()</pre>
28		<pre>self.ContentInWords = self.get_content_in_words()</pre>
29		<pre>self.HexPaddedContentInWords = self.get_hex_padded_content_in_words()</pre>
30		<pre>self.HexPaddedContentInBytes = self.get_hex_padded_content_in_bytes()</pre>
31		
32		<pre>def print_section(self):</pre>
33		<pre>print("Start Address: " + self.StartAddress)</pre>
34		<pre>print("Size in words: " + str(self.SizeInWords))</pre>
35		print("Content in Bytes: ")
36		<pre>print(self.HexPaddedContentInBytes)</pre>
37		
38		<pre>def get_start_address(self):</pre>
39	Â	return "0x" + self.ContentSplitBySpace[0]

Figure 21. FRAM TI-TXT Hex to Firmware Image Converter

Modify the variable *sourcePath* to point to the TI-TXT hex file to convert to a firmware image header file. The *imagePath* variable defines the location for the output firmware image header file. Using Python 3, the script can be executed and the firmware image header file is generated.

C:\Users\	.\Programs	\Python\Python35	\python.exe			
C:/Users/						
/FRAM_TI_txt_hex_to_byte	e_image.py					
Start Address: 0xf100						
Size in words: 37						
Content in Bytes:						
['0xB2', '0x40', '0x80',	'0x5A', '0xCC',					
\$'0x02', '0xD2', '0xD3',	'0x04', '0x02',		'0x30', '0x01', 2			
S'0xD2', '0xE3', '0x02',		'0x14', '0x3D',	'0x40', '0x32', 2			
5'0x82', '0x1D', '0x83',	'0xFE', '0x23',	'0x0D', '0x16',	'0xF7', '0x3F', 2			
\$'0x03', '0x43', '0x03',	'0x43', '0xFF',	'0x3F', '0x03',	'0x43', '0x1C', 2			
\$'0x43', '0x10', '0x01',	'0x31', '0x40',	'0x00', '0x24',	'0xB0', '0x13', 2			
S'0x2C', '0xF1', '0x0C',	'0x43', '0xB0',	'0x13', '0x00',	'0xF1', '0xB0', 2			
S'0x13', '0x26', '0xF1',	'0x32', '0xD0',	'0x10', '0x00',	'0xFD', '0x3F', 2			
\$'0x03', '0x43']						
Start Address: 0xff80						
Size in words: 4						
Content in Bytes:						
['0xFF', '0xFF', '0xFF',	'OxFF', 'OxFF',	'OxFF', 'OxFF',	'OxFF']			
Start Address: 0xffe2						
Size in words: 15						
Content in Bytes:						
['0x42', '0xF1', '0x42',	'0xF1', '0x42',	'0xF1', '0x42',	'0xF1', '0x42',			
'0xF1', '0x42', '0xF1',	'0x42', '0xF1',	'0x42', '0xF1',	'0x42', '0xF1',			
'0x42', '0xF1', '0x42',	'0xF1', '0x42',	'0xF1', '0x42',	'0xF1', '0x42',			
'0xF1', '0x30', '0xF1']						
Process finished with exit code 0						

Figure 22. Firmware Image Python Script Console Output



UART BSL Command Line Utility For MSP430 FRAM Devices

www.ti.com

After the conversion is completed, the firmware image header file is generated and can be used as a replacement for the default firmware image of the software example.

# 6 UART BSL Command Line Utility For MSP430 FRAM Devices

The UART\_BSL\_MSP430FR\_Command\_Line\_Utility, is an example command line application which can be used as a general purpose utility for updating the firmware on an MSP430 target using any GPIO pin or UART module available on the BeagleBone Black. To use this command line utility, a firmware image file in TI-TXT hex format is needed. The path to this firmware image file is passed to the utility as the last argument. The pins used as GPIO and UART TX/RX must be configured before the command line utility is used.

There is no need for the Python scripts to convert the TI-TXT hex file to a header file. The TI-TXT hex file can be directly passed to the command line utility, where it is automatically parsed.

Figure 23 shows how the command line utility can be used to update the firmware of the MSP430FR2311 using the FR2311\_blinky.txt file. GPIO 50 is used as the RESET pin, GPIO 51 is used as the TEST pin, and the UART4 module transmits and receives UART BSL commands and responses.

🛃 128.247.87.83 - PuTTY					
bsl.h gpioif.c main.c uart if.c utils.h					
root@beaglebone:/home/root/final/UART_BSL_MSF430FR_Command_Line_Utility# ./comma					
nd_line_bsl 50 51 4 image/FR2311_blinky.txt					
Reset Pin GPIO: 50					
Test Pin GPIO: 51					
UART Module: 4					
Reset Pin set to GPIO: 50					
Test Pin set to GPIO: 51 UART Module set to: 4					
TI TXT Hex: image/FR2311 blinky.txt					
Initializing UART: /dev/tty04					
Waiting for timeout to ensure input buffer is empty!					
Waiting for timeout to ensure input buffer is empty: Starting the BSL commands!					
Starting the BSE commands:					
*****New Attempt****					
Write password failed					
*****New Attempt****					
Write Password was successful					
Device ID: 0xF082					
****Section: 0****					
Address: 0xF100					
Write Large Data To Memory successful					
****Section: 1****					
****Section: 1****					
Address: 0xFF80					
Write Large Data To Memory successful					
White harge back to Memory Successful					
****Section: 2****					
Address: 0xFFE2					
Write Large Data To Memory successful					
MSP430 programmed successfully					
Device is reset					
root@beaglebone:/home/root/final/UART_BSL_MSP430FR_Command_Line_Utility#					

Figure 23. UART\_BSL\_MSP430FR\_Command\_Line\_Utility Console Output

The source code for this example can be used to create a custom command line application to communicate to any MSP430 MCU through UART BSL.



# 7 Error Messages

If the connection between the host and the target are not correct, or the UART BSL state machine of the target MSP430 MCU is out of sync with the host, error messages can be seen in the console output (see Figure 24).

If the macro DEBUG (in config.h) is defined as 0, then fewer error messages are shown. However, if this macro is set to 1, the actual content of the UART BSL commands and responses can be viewed (see Figure 25).

🛃 128.247.87.83 - PuTTY	
<pre>root@beaglebone:/home/root/final/UART_BSL_MSP430# ./msp430_bs1</pre>	^
****New Attempt****	
Write password failed	
****New Attempt****	
Write password failed	
****New Attempt****	
Write password failed	
****New Attempt****	
Write password failed	
*****New Attempt*****	
Write password failed Max retries exceeded	
<pre>max retries exceeded root@beaglebone:/home/root/final/UART_BSL_MSP430# ./msp430_bsl</pre>	Ŧ

Figure 24. Error Messages With DEBUG Set to 0



Figure 25. Error Messages With DEBUG Set to 1

The example issue in Figure 25 is usually caused by the UART BSL state machine being out of sync with the host. This is suggested because the 0x80 byte is received as the third and fourth byte in the packet.

If no UART command and response are sent or received, check the hardware connections and the pin configurations.

### IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your noncompliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products <a href="http://www.ti.com/sc/docs/stdterms.htm">http://www.ti.com/sc/docs/stdterms.htm</a>), evaluation modules, and samples (<a href="http://www.ti.com/sc/docs/stdterms.htm">http://www.ti.com/sc/docs/stdterms.htm</a>), evaluation

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265 Copyright © 2017, Texas Instruments Incorporated