

MSP430™ Spy-Bi-Wire With SimpleLink™ MCUs

Nima Eskandari

MSP430 Applications

ABSTRACT

The embedded memory of MSP430™ microcontrollers (MCUs) can be programmed using the on-chip JTAG interface. The MSP430 MCUs support 2-wire JTAG interface. The 2-wire JTAG interface is referred to as Spy-Bi-Wire (SBW). Using Spy-Bi-Wire a host can access the programmable memory (flash memory), the data memory (RAM), and in FRAM devices, the nonvolatile FRAM memory. The host MCU 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 SimpleLink™ MSP432P401R, CC3220, and CC2640R2F devices as the hosts for the SBW communication with the target MSP430 MCU. Both flash-based and FRAM-based MSP430 MCUs are used in this report to showcase the differences between the two device families, related to SBW communication. Software examples are provided for each of the SimpleLink devices mentioned. The software examples also make use of the SimpleLink Software Development Kit (SDK), making it easy to port to other SimpleLink devices.

The source code and other files described in this application report can be downloaded from <http://www.ti.com/lit/zip/slaa754>. The example source code demonstrates how a SimpleLink MCU can access the memory of a target MSP430 MCU (flash or FRAM based) through the SBW communication.

Contents

1	Introduction	2
1.1	Supplementary Online Information	2
2	Software Example	3
2.1	Software Example File Descriptions.....	3
3	Spy-Bi-Wire (SBW) Connections	5
3.1	Spy-Bi-Wire Connections for the Target MSP430 MCU	5
3.2	Spy-Bi-Wire (SBW) Connections for the Host SimpleLink Device	7
4	How to Use the Software Examples	10
4.1	Import the Projects to CCS	10
4.2	Run the Software Examples	14
5	Create Custom MSP430 Firmware Image	22
5.1	MSP430 Flash Firmware Image.....	22
5.2	MSP430 FRAM Firmware Image	23
5.3	Generating Custom Firmware Image Header Files.....	24
6	Error Messages.....	28

List of Figures

1	MSP430G2553 SBW Pins.....	6
2	MSP430FR2311 SBW Pins	6
3	MSP432P401R LaunchPad Development Kit SBW Host Pins	8
4	CC3220SF LaunchPad Development Kit SBW Host Pins	8
5	CC2640R2 LaunchPad Development Kit SBW Host Pins	9
6	Import Software Example Projects to CCS	10
7	Import All Projects to CCS	11
8	Building Software Example Projects.....	12

9	TI RTOS Build Project Dependency	13
10	MSP432P401R LaunchPad Development Kit Connected to MSP430G2553	14
11	MSP432P401R LaunchPad Development Kit SBW Host Connections	14
12	MSP430G2553 LaunchPad Development Kit SBW Connections	15
13	Application UART COM Port Number in Windows Device Manager	15
14	PuTTY Serial Communication Settings	16
15	MSP432Host_SBW_MSP430 Project Debug Mode	17
16	MSP432Host_SBW_MSP430 Project Serial Output (DEBUG is 1)	17
17	MSP432Host_SBW_MSP430 Project Serial Output (DEBUG is 0)	18
18	MSP430G2553 Programmed Successfully	19
19	MSP432P401R LaunchPad Development Kit Connected to MSP430FR2311	20
20	MSP430FR2311 LaunchPad Development Kit SBW Connections	20
21	MSP432Host_SBW_MSP430FR Serial Output (DEBUG is 1)	21
22	MSP430FR2311 Programmed Successfully	21
23	Enabling MSP430 Hex Utility	25
24	MSP430 TI-TXT Hex Format	25
25	MSP430FR2311 TI-TXT Hex Format Firmware Image	26
26	FRAM TI-TXT Hex to Firmware Image Converter	27
27	Firmware Image Python Script Console Output	27
28	SBW Error	28

List of Tables

1	Top-Level Folder Description	3
2	CC2640R2 Host Example Project Content	4
3	CC3220 Host Example Project Content	4
4	MSP432 Host Example Project Content	4
5	SBW Folder Content	5

Trademarks

MSP430, SimpleLink, E2E, LaunchPad, Code Composer Studio are trademarks of Texas Instruments. All other trademarks are the property of their respective owners.

1 Introduction

Spy-Bi-Wire (SBW) provides a method for the MSP430 MCUs to be programmed. For the MSP430 MCU to be programmed through SBW, commands must be sent to the dedicated pins of the target MSP430 MCU, using the SBW protocol.

To enable the SBW connection, an entry sequence must be applied to the dedicated SBW pins. After the SBW entry sequence is applied, commands can be sent to the target MSP430 MCU. The SBW 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 MSP432P401R, CC3220, and CC2640 as the SBW host to the target MSP430 MCU. The SBW protocol in these examples is implemented on top of the SimpleLink SDK, which makes it extremely easy to port to other SimpleLink devices.

1.1 Supplementary Online Information

For more information, visit [Replicator for MSP430 MCU](#). This page contains links to additional SBW and JTAG user's guides and source code for using an MSP430F5437 to program other MSP430 MCUs.

For more information on JTAG and SBW communication, see [MSP430™ Programming With the JTAG Interface](#), which describes how the JTAG and SBW interface can be used to program MSP430 MCUs. It also describes the commands used by SBW protocol.

Additional support is provided by the [TI E2E™ Community](#).

2 Software Example

Example software is available for SimpleLink devices to act as the SBW host to target MSP430 MCUs.

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

- [SimpleLink MSP432P401R LaunchPad Development Kit](#)
- [SimpleLink Wi-Fi® CC3220SF Wireless Microcontroller LaunchPad Development Kit](#)
- [SimpleLink Bluetooth® low energy CC2640R2F wireless MCU LaunchPad development kit](#)

2.1 Software Example File Descriptions

The example software can be downloaded from <http://www.ti.com/lit/zip/slaa754>.

Table 1 describes the content of the top-level folder, and the following tables describe the content of the software examples.

Table 1. Top-Level Folder Description

Folder Name	Description
CC2640R2Host_SBW_MSP430	Example SimpleLink SDK project for CC2640R2. CC2640R2 acts as a SBW host for MSP430 flash-based devices. This example must be used with the CC2640R2F wireless MCU LaunchPad development kit. Note: This is a Code Composer Studio™ IDE project and must be imported into the IDE. This project depends on tirtos_builds_CC2640R2_LAUNCHXL_release_ccs, which also must be imported to the IDE.
CC2640R2Host_SBW_MSP430FR	Example SimpleLink SDK project for CC2640R2. CC2640R2 acts as a SBW host for MSP430 FRAM-based devices. This example must be used with the CC2640R2F wireless MCU LaunchPad development kit. Note: This is a Code Composer Studio IDE project and must be imported into the IDE. This project depends on tirtos_builds_CC2640R2_LAUNCHXL_release_ccs, which also must be imported to the IDE.
CC3220Host_SBW_MSP430	Example SimpleLink SDK project for CC3220. CC3220 acts as a SBW host for MSP430 flash-based devices. This example must be used with the CC3220 wireless MCU LaunchPad development kit. Note: This is a Code Composer Studio IDE project and must be imported into the IDE. This project depends on tirtos_builds_CC3220SF_LAUNCHXL_release_ccs, which also must be imported to the IDE.
CC3220Host_SBW_MSP430FR	Example SimpleLink SDK project for CC3220. CC3220 acts as a SBW host for MSP430 FRAM based devices. This example must be used with the CC3220 wireless MCU LaunchPad development kit. Note: This is a Code Composer Studio IDE project and must be imported into the IDE. This project depends on tirtos_builds_CC3220SF_LAUNCHXL_release_ccs, which also must be imported to the IDE.
MSP432Host_SBW_MSP430	Example SimpleLink SDK project for MSP432P401R. MSP432P401R acts as a SBW host for MSP430 flash-based devices. This example must be used with the MSP432P401R LaunchPad development kit. Note: This is a Code Composer Studio project and must be imported into the IDE. This project depends on tirtos_builds_MSP_EXP432P401R_release_ccs, which also must be imported to the IDE.
MSP432Host_SBW_MSP430FR	Example SimpleLink SDK project for MSP432P401R. MSP432P401R acts as a SBW host for MSP430 FRAM based devices. This example must be used with the MSP432P401R LaunchPad development kit. Note: This is a Code Composer Studio IDE project and must be imported into the IDE. This project depends on tirtos_builds_MSP_EXP432P401R_release_ccs, which also must be imported to the IDE.
tirtos_builds_CC2640R2_LAUNCHXL_release_ccs	TI RTOS project to be used with CC2640R2 example projects. This project is referenced by all CC2640R2 projects.

Table 1. Top-Level Folder Description (continued)

Folder Name	Description
tirtos_builds_CC3220SF_LAUNCHXL_release_ccs	TI RTOS project to be used with CC3220 example projects. This project is referenced by all CC3220 projects.
tirtos_builds_MSP_EXP432P401R_release_ccs	TI RTOS project to be used with MSP432 example projects. This project is referenced by all MSP432 projects.
Python_Scripts/FLASH_TI_txt_hex_to_image.py and Python_Scripts/FRAM_TI_txt_hex_to_image.py	Converts TI HEX 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_image.py

The following tables describe the contents of each project.

Table 2. CC2640R2 Host Example Project Content

Name	Description
SBW (Directory)	Contains the SBW implementation, example firmware image and peripheral drivers.
Board.h	Contains board specific macros.
main_tirtos.c	Entry point for TI RTOS.
CC2640R2_LAUNCHXL.h	CC2640R2_LAUNCHXL board-specific APIs.
CC2640R2_LAUNCHXL.c	CC2640R2 specific for TI RTOS. Responsible for setting up the board specific items for the CC2640R2_LAUNCHXL board.
cc2640r2host_sbwnbsp_msp430.c or cc2640r2host_sbwnbsp_msp430fr.c	Driver Initialization. Starts the SBW programming with the example firmware image.

Table 3. CC3220 Host Example Project Content

Name	Description
SBW (Directory)	Contains the SBW implementation, example firmware image and peripheral drivers.
Board.h	Contains board specific macros.
main_tirtos.c	Entry point for TI RTOS.
CC3220SF_LAUNCHXL.h	CC3220SF_LAUNCHXL board specific APIs.
CC3220SF_LAUNCHXL.c	CC3220 specific for TI RTOS. Responsible for setting up the board specific items for the CC3220SF_LAUNCHXL board.
cc3220host_sbwnbsp_msp430.c or cc3220host_sbwnbsp_msp430fr.c	Driver Initialization. Starts the SBW programming with the example firmware image.

Table 4. MSP432 Host Example Project Content

Name	Description
SBW (Directory)	Contains the SBW implementation, example firmware image and peripheral drivers.
Board.h	Contains board specific macros.
main_tirtos.c	Entry point for TI RTOS.
MSP_EXP432P401R.h	MSP_EXP432P401R board specific APIs.
MSP_EXP432P401R.c	MSP432 specific for TI RTOS. Responsible for setting up the board specific items for the MSP_EXP432P401R board.
msp432host_sbwnbsp_msp430.c or msp432host_sbwnbsp_msp430fr.c	Driver Initialization. Starts the SBW programming with the example firmware image.

The SBW folder in each project contains the drivers for the peripherals used by the SimpleLink host. This folder also contains the SBW commands and an example firmware image for the target MSP430 MCU.

Table 5 describes the content of the SBW folder.

Table 5. SBW Folder Content

Name	Description
image/msp430_image.h or image/msp430fr_image.h	Contains the example firmware image for flash or FRAM target MSP430 MCU. The msp430_image.h contains the blinky example for MSP430G2553 device, while the msp430fr_image.h contains the blinky example for MSP430FR2311 device.
SBW430.c or SBW430FR.c	Implementation of the SBW commands. This file contains the commands used to program an MSP430 MCU through Spy-Bi-Wire.
SBW430.h or SBW430FR.h	Contains the function declarations for the SBW commands.
sbw_main.c	Programs the target device with the specified example firmware image. It resets the device and allows it to run after reset.
sbw_main.h	Contains the function declarations for the sbw_main.c
config.h	Contains configurable options such as number of time to retry when a time-out occurs 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 SBW entry sequence pins and send SBW commands.
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 data through UART protocol. (This file is not used in the example project)
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

3 Spy-Bi-Wire (SBW) Connections

The SimpleLink host and target MSP430 MCU must be connected to through the SBW pins and share the GND signal. The SBW pins include the two dedicated pins used for 2-wire JTAG interface.

3.1 Spy-Bi-Wire Connections for the Target MSP430 MCU

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

The SBW entry sequence must be applied to the TST and RST pins on the target MSP430 MCU to invoke the SBW communication. For more information on the SBW entry sequence, see [MSP430™ Programming With the JTAG Interface](#).

Figure 1 and Figure 2 show the dedicated pins for SBW on the MSP430G2553 and the MSP430FR2311, respectively.

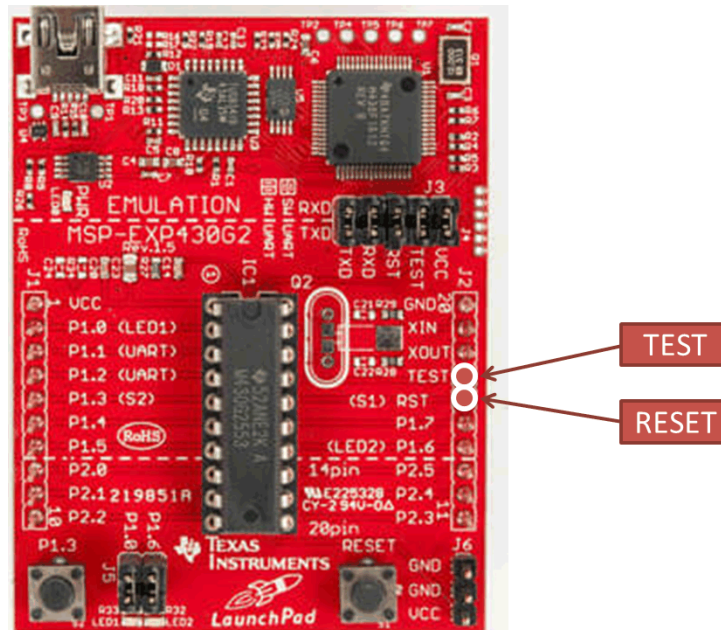


Figure 1. MSP430G2553 SBW Pins

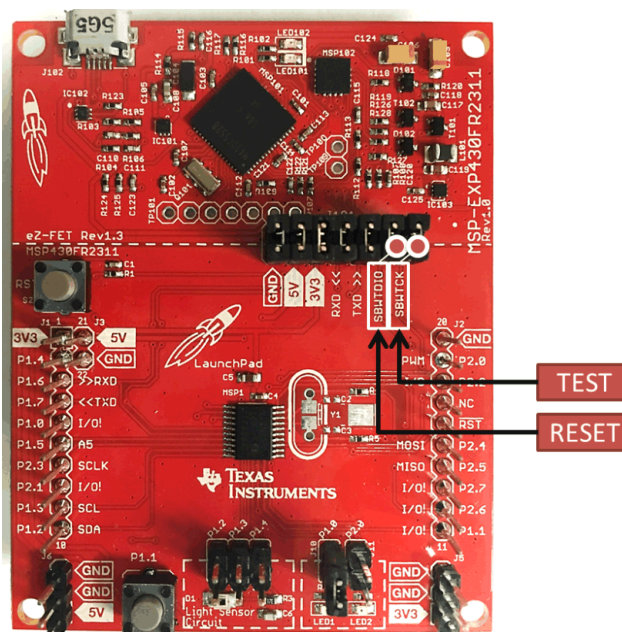


Figure 2. MSP430FR2311 SBW Pins

These pins are connected to the GPIO of the SimpleLink host. After the SBW entry sequence is completed, the host can use these same pins to send and receive SBW commands.

The pins for SBW interface 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)

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 *Spy-Bi-Wire (SBW) Connections for the Host SimpleLink Device*

The host device uses two GPIO pins to communicate through the SBW interface to the target MSP430 MCU. The host also uses one UART module. The UART module communicates the status of code execution to the PC through the backchannel serial port. These pins are defined in the board-specific file of every project (MSP_EXP432P401R.c, CC3220SF_LAUNCHXL.c, and CC2640R2_LAUNCHXL.c).

The software example provided for the supported LaunchPad development kits uses the following pins on each board:

MSP432P401R LaunchPad development kit:

- Target Reset Pin: P6.0
- Target Test Pin: P6.1
- Backchannel UART to PC:
 - TX: P1.3
 - RX: P1.2

CC3220SF wireless microcontroller LaunchPad development kit:

- Target Reset Pin: PIN 03 (GPIO 12)
- Target Test Pin: PIN 04 (GPIO 13)
- Backchannel UART to PC:
 - TX: PIN 57
 - RX: PIN 55

CC2640R2 wireless microcontroller LaunchPad development kit:

- Target Reset Pin: DIO23
- Target Test Pin: DIO25
- Backchannel UART to PC:
 - TX: DIO03
 - RX: DIO02

[Figure 3](#), [Figure 4](#), and [Figure 5](#) show each LaunchPad development kit and the pins used for the software example.

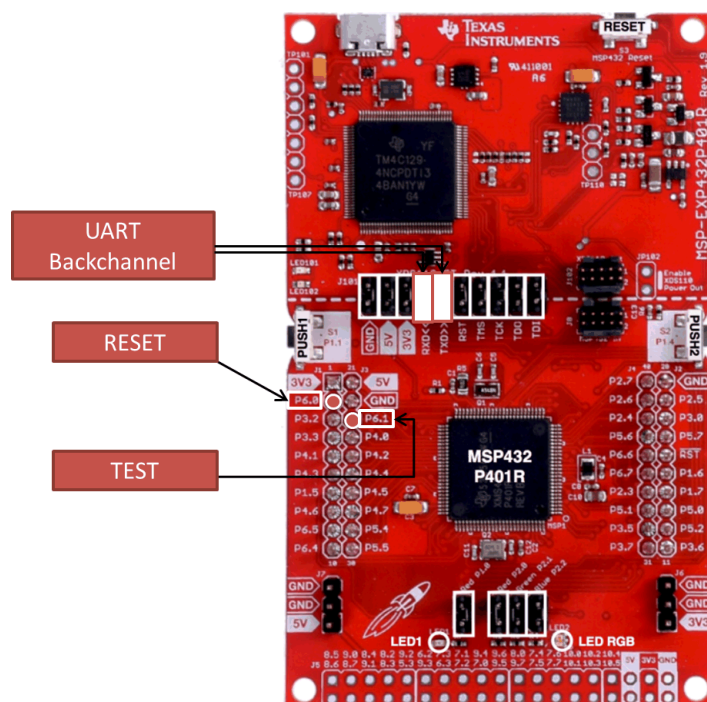


Figure 3. MSP432P401R LaunchPad Development Kit SBW Host Pins

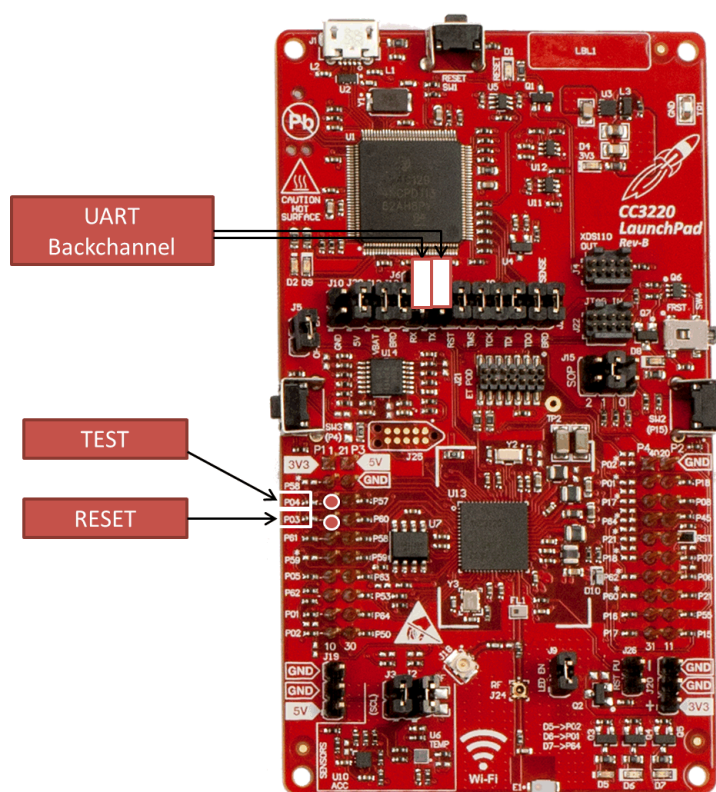


Figure 4. CC3220SF LaunchPad Development Kit SBW Host Pins

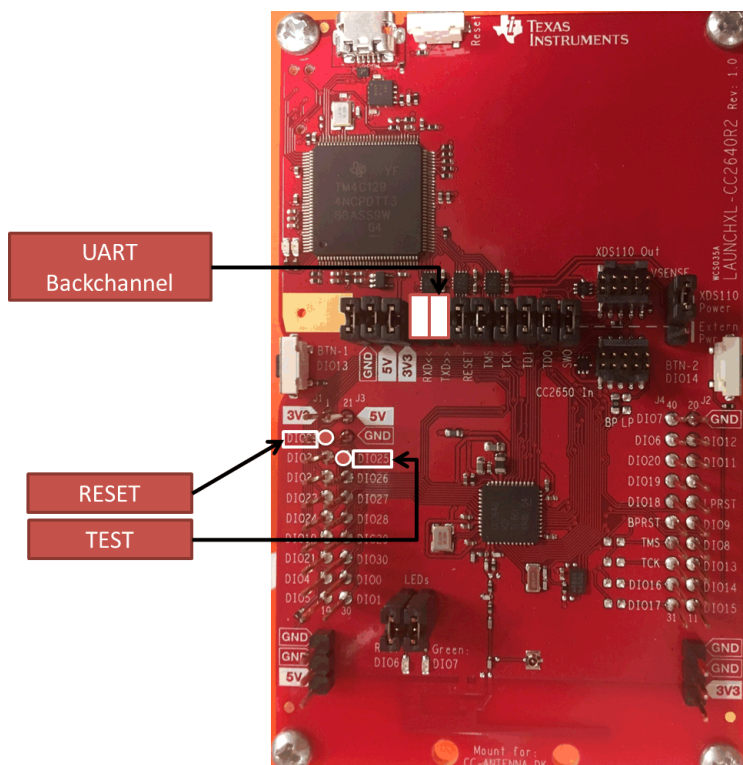


Figure 5. CC2640R2 LaunchPad Development Kit SBW Host Pins

The SimpleLink Host connects to the MSP430 MCU through the Reset (SBW data input/output pin) and Test (SBW clock pin) pins. The SimpleLink host and MSP430 MCU must share the ground signal. Also, the MSP430 MCU must be powered up. Finally, through the backchannel UART, a PC can be used to view the status of the MSP430 firmware update.

4 How to Use the Software Examples

The software examples in this application report used the following SimpleLink SDKs:

- SimpleLink CC32xx SDK v:1.40.00.03
- SimpleLink MSP432 SDK v:1.40.01.00
- SimpleLink CC2640R2 SDK v:1.35.00.33

Code Composer Studio (CCS) IDE Version 7.1.0.00016 was used to compile and debug the projects.

Download and extract the zip file containing the software examples.

4.1 Import the Projects to CCS

To import the projects, click Project > Import CCS Projects.

Select Browse and navigate to the extracted file from the software example zip file.

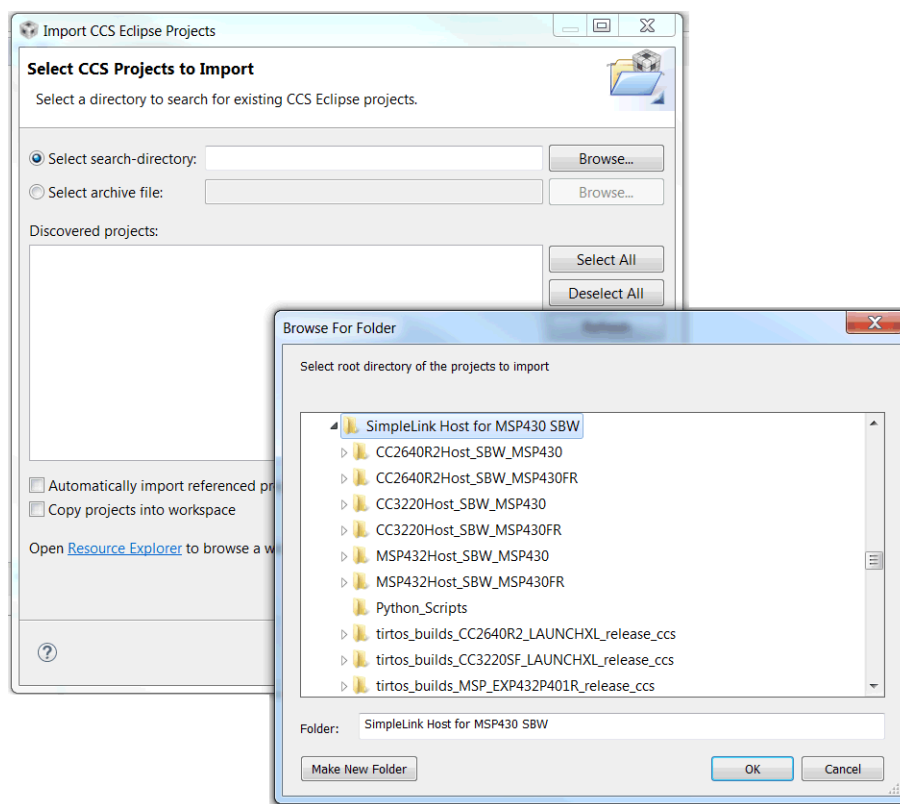


Figure 6. Import Software Example Projects to CCS

CCS automatically finds all of the projects inside the folder. Select all projects including the TI RTOS build projects and import them to your workspace.

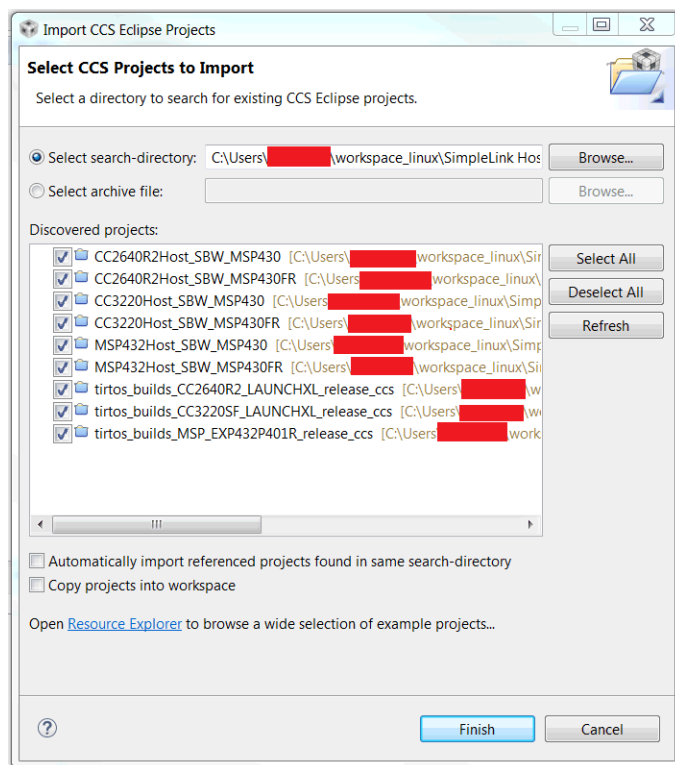


Figure 7. Import All Projects to CCS

After importing, the projects must be built.

Select all of the imported projects, except for the TI RTOS build projects. Right click and press Rebuild Project.

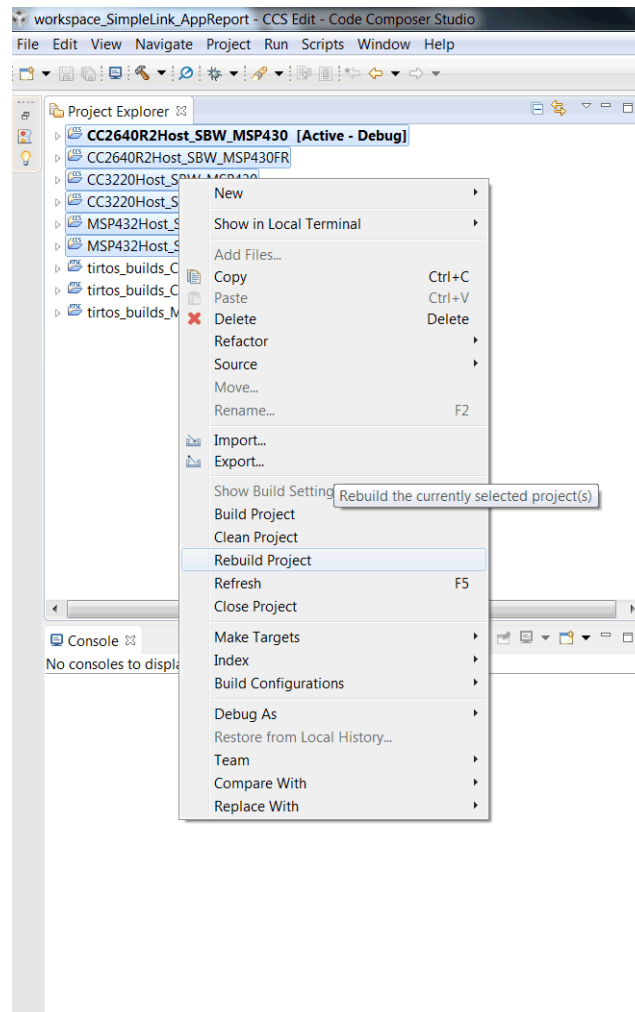


Figure 8. Building Software Example Projects

The `tirtos_builds_CC2640R2_LAUNCHXL_release_ccs`, `tirtos_builds_CC3220SF_LAUNCHXL_release_ccs`, and `tirtos_builds_MSP_EXP432P401R_release_ccs` will be rebuilt automatically. These projects are referenced by the other projects with the same platform.

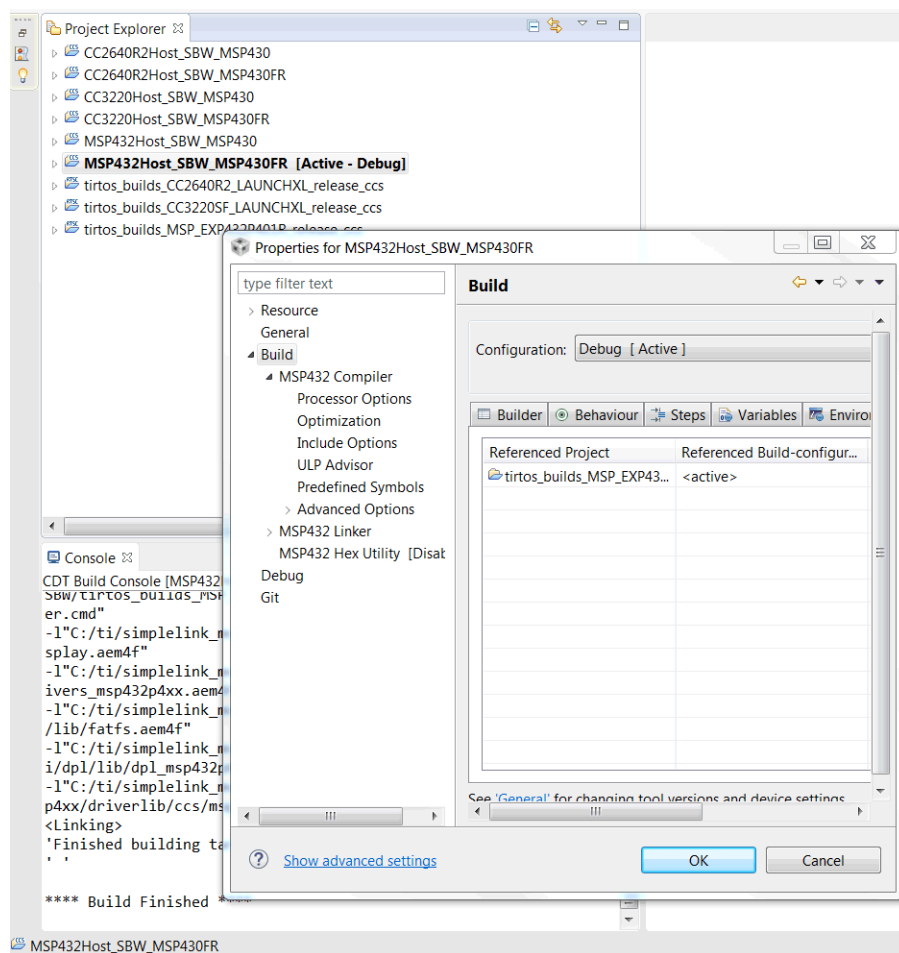


Figure 9. TI RTOS Build Project Dependency

4.2 Run the Software Examples

Connect the SimpleLink host to the MSP430 target. For the first part of this demo, the MSP432P401R LaunchPad and MSP430G2553 are used. As for the second part, MSP430FR2311 is used as the target.

4.2.1 MSP430G2553 SBW Target Example

Connect the two devices as shown in the following figures.

The pin-to-pin connection is also available in the README file.

Figure 10, Figure 11, and Figure 12 show the connections for the MSP432P401R and MSP430G2553 LaunchPad development kits.

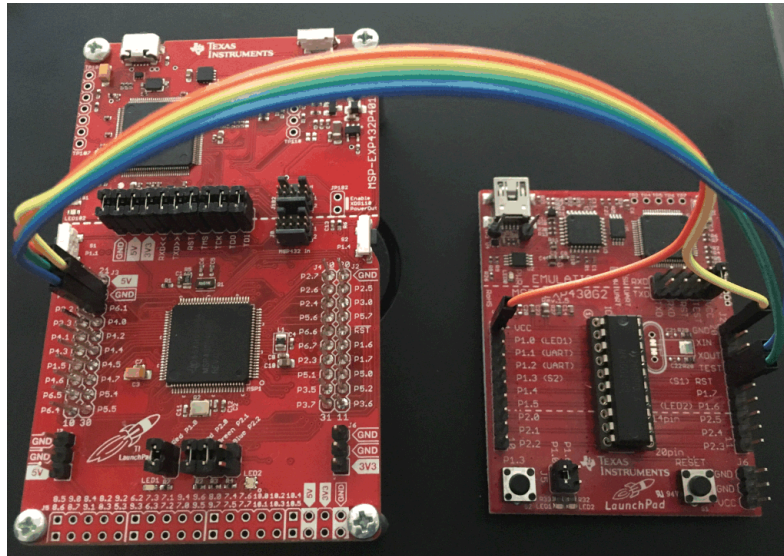


Figure 10. MSP432P401R LaunchPad Development Kit Connected to MSP430G2553

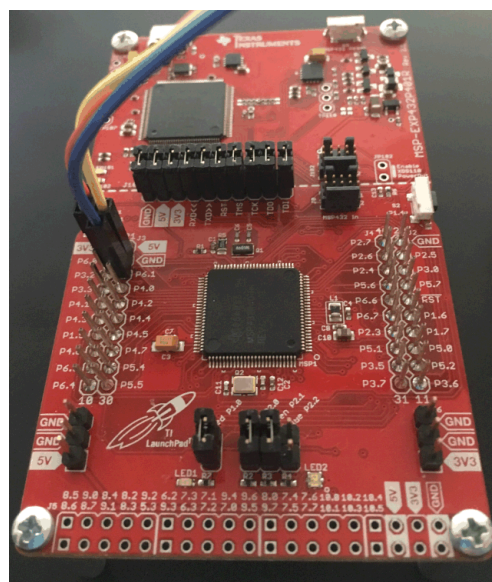


Figure 11. MSP432P401R LaunchPad Development Kit SBW Host Connections

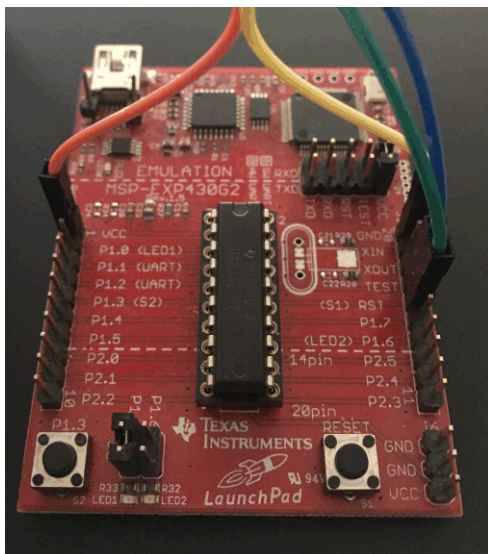


Figure 12. MSP430G2553 LaunchPad Development Kit SBW Connections

In this setup, the MSP430G2553 is powered from the MSP432P401R LaunchPad development kit. The MSP432P401R LaunchPad development kit is powered through the USB connection to PC.

After connecting the MSP432 LaunchPad development kit to the PC, the COM port number for serial communication to the board can be found in Windows Device Manager.

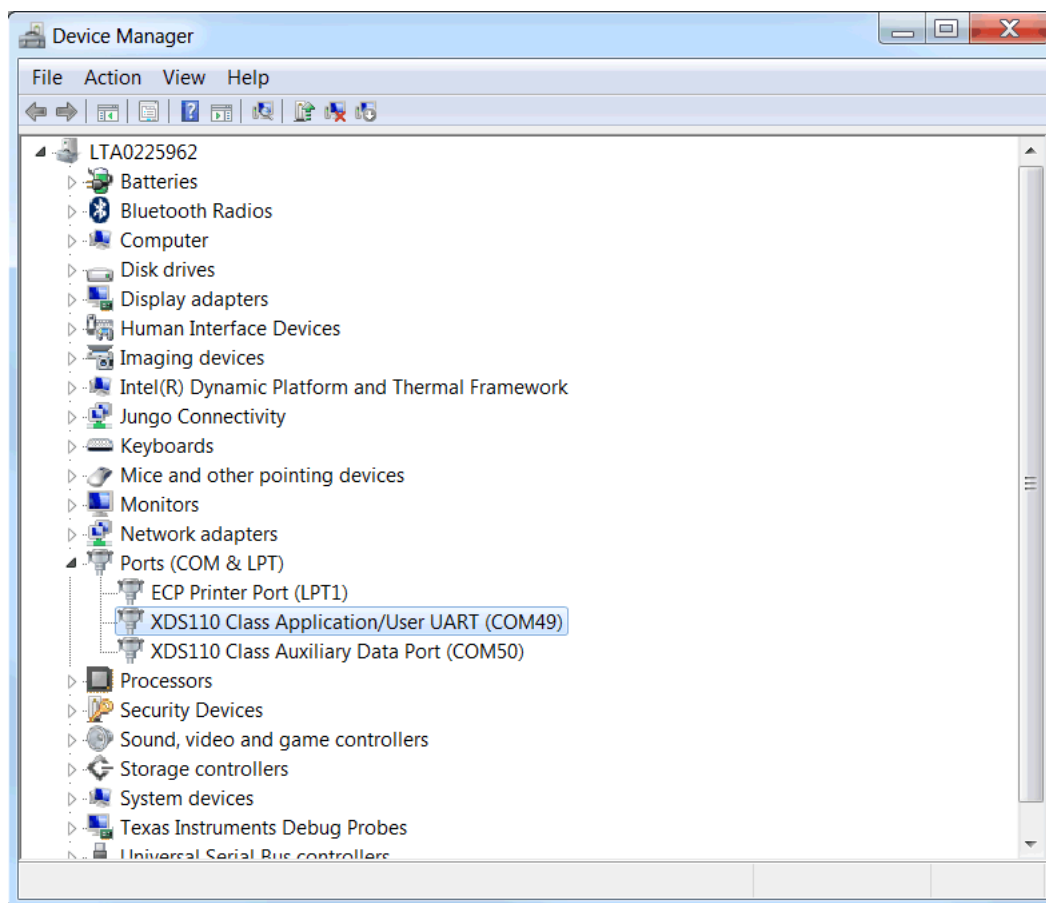


Figure 13. Application UART COM Port Number in Windows Device Manager

A serial communication terminal program such as PuTTY can be used to view the execution of the example software.

Using PuTTY, open a serial communication with the following specification to the COM port found in the Windows Device Manager (see [Figure 14](#)).

- Baud rate: 9600
- Data bits: 8
- Stop bits: 1
- Parity: None

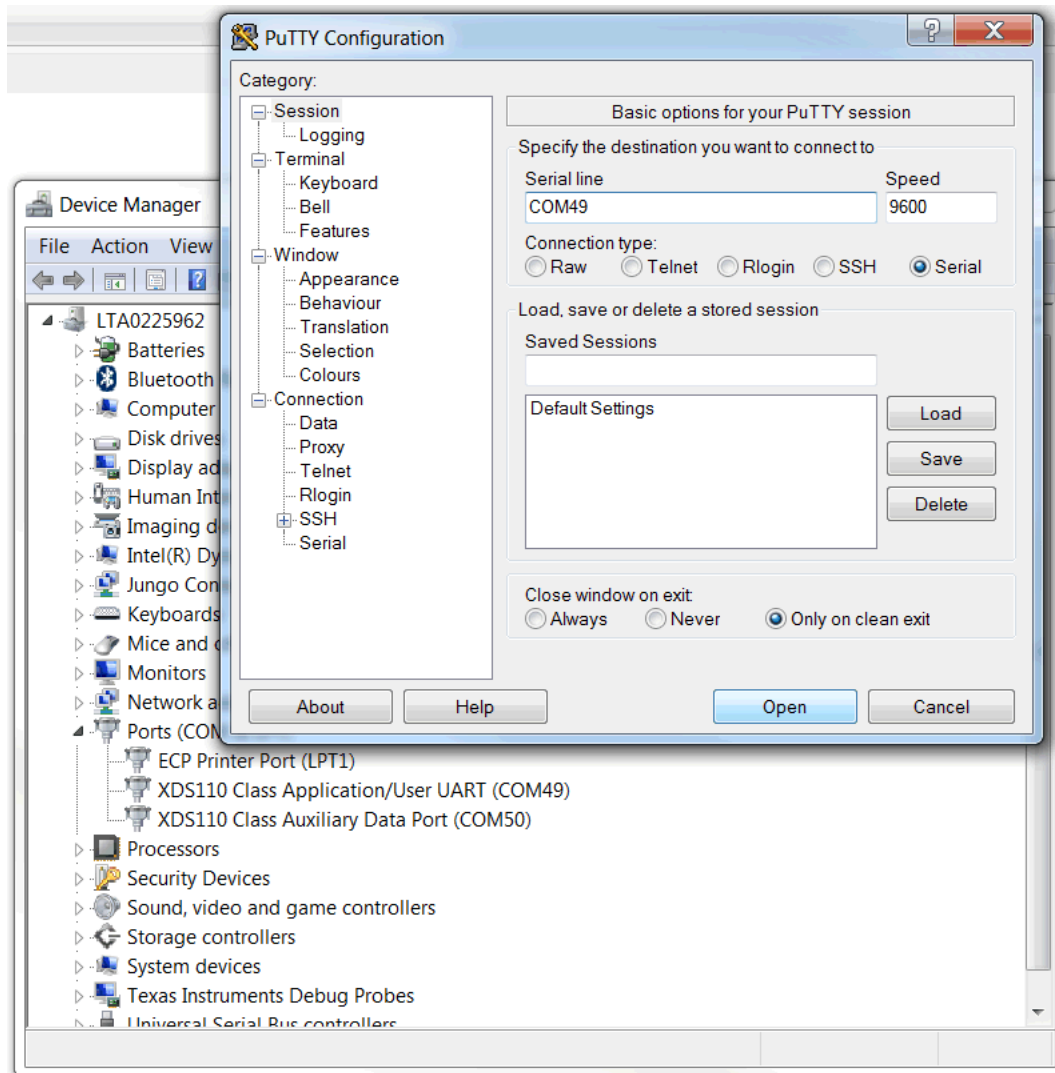


Figure 14. PuTTY Serial Communication Settings

Next, debug the project MSP432Host_SBW_MSP430 by clicking Run > Debug.

After the binary firmware image is uploaded to MSP432P401R, run the example by clicking the green play button.

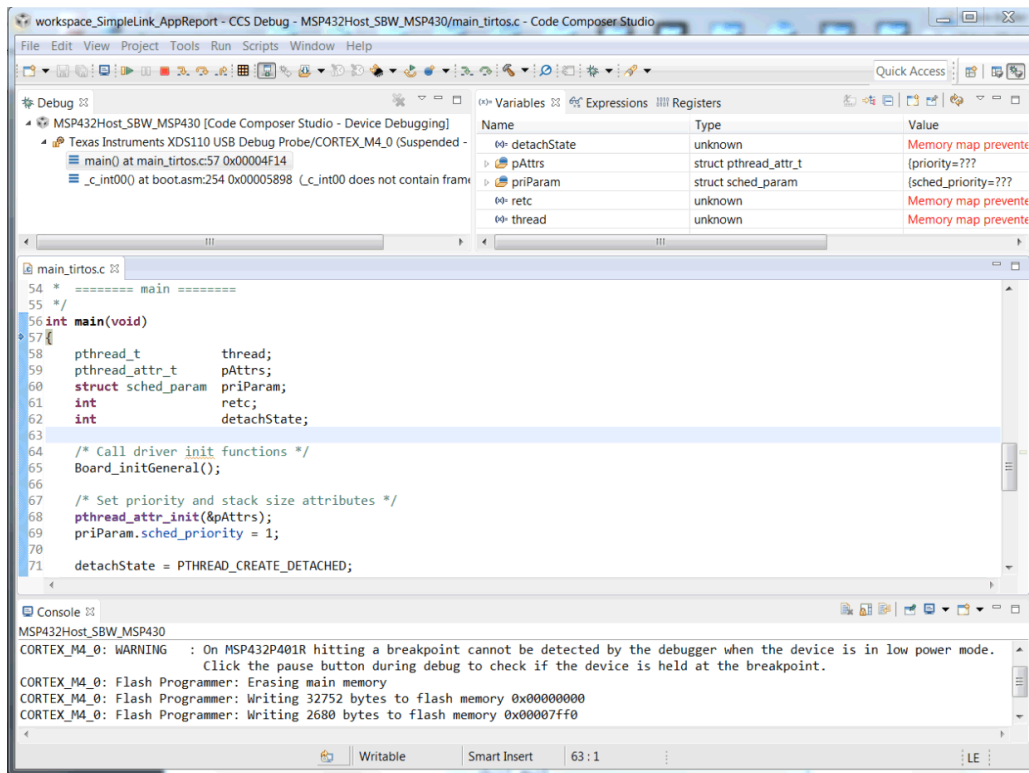


Figure 15. MSP432Host_SBW_MSP430 Project Debug Mode

The next step is to view the serial output on PuTTY (see [Figure 16](#)).

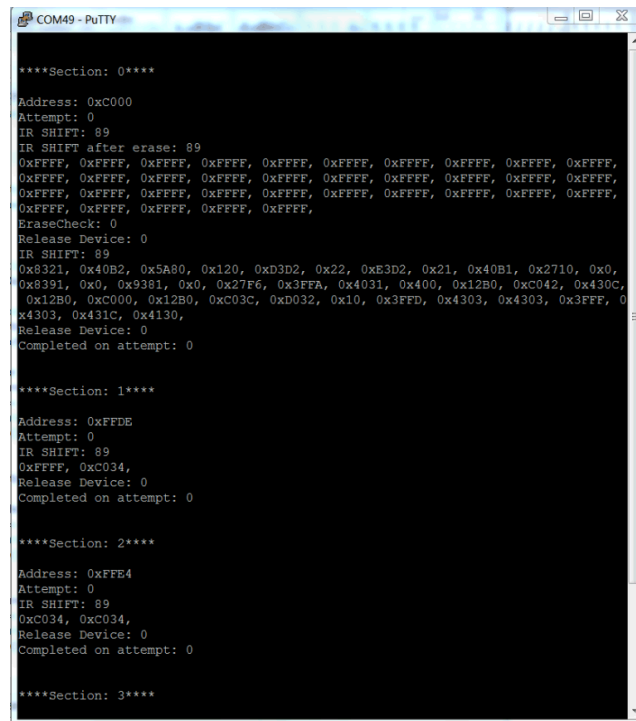
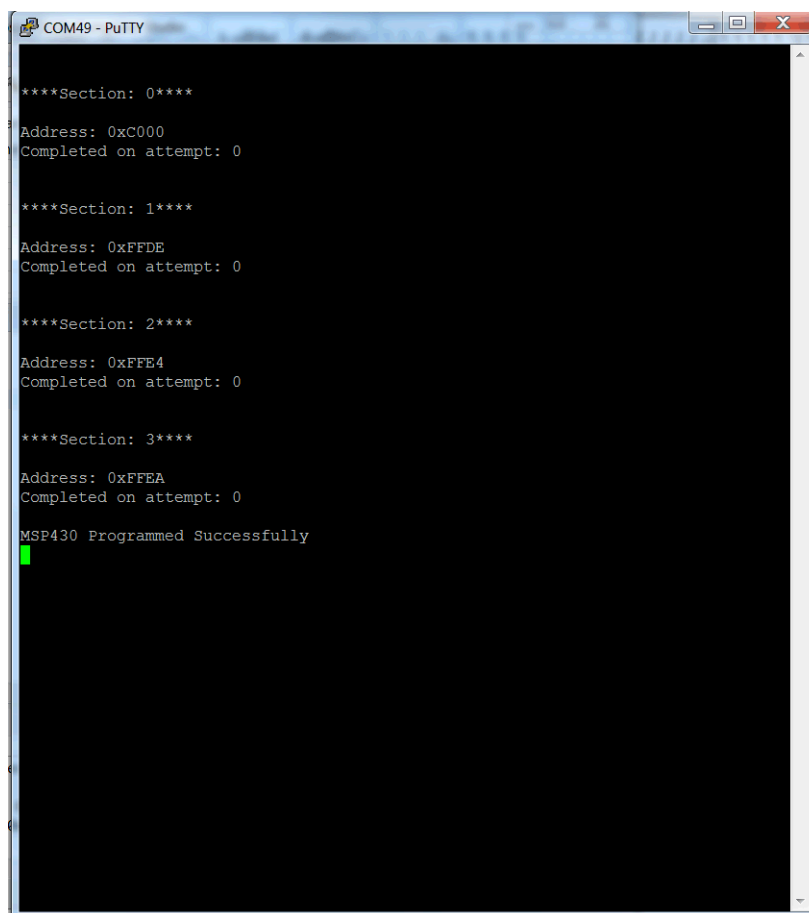


Figure 16. MSP432Host_SBW_MSP430 Project Serial Output (DEBUG is 1)

If the `DEBUG` macro is defined as 1 in the `config.h` file, the serial output looks similar to Figure 16. In this case, the execution of the code can be viewed in detail. If the `DEBUG` macro is defined as 0, the serial output looks similar to Figure 17.



```

COM49 - PuTTY

****Section: 0****
Address: 0xC000
Completed on attempt: 0

****Section: 1****
Address: 0xFFDE
Completed on attempt: 0

****Section: 2****
Address: 0xFFE4
Completed on attempt: 0

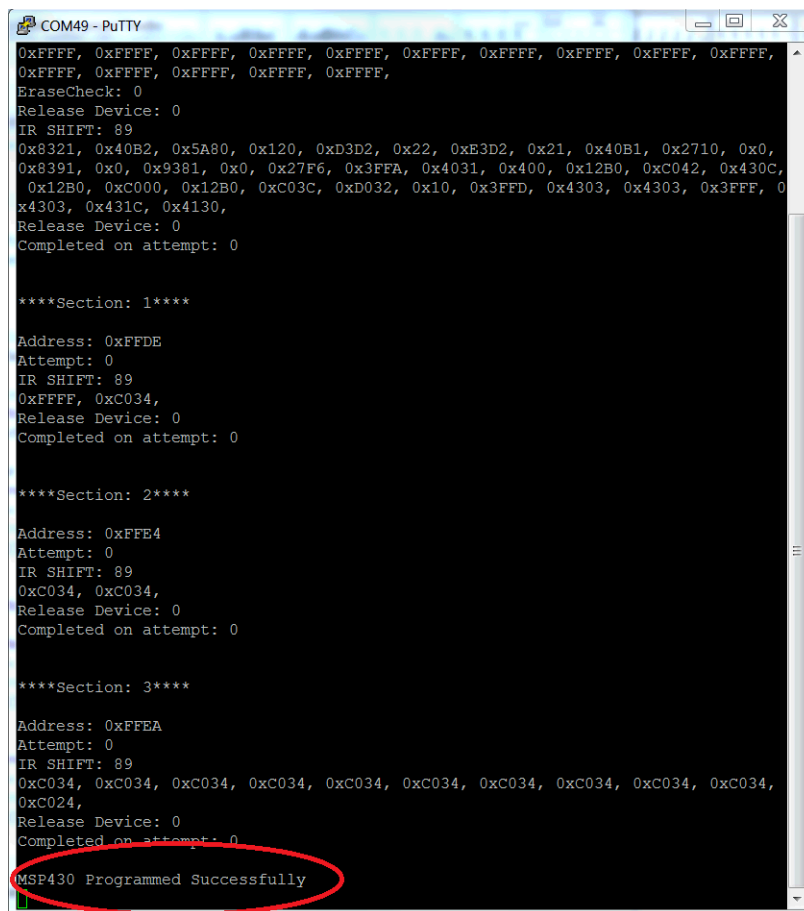
****Section: 3****
Address: 0xFFEA
Completed on attempt: 0
MSP430 Programmed Successfully

```

Figure 17. MSP432Host_SBW_MSP430 Project Serial Output (DEBUG is 0)

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

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



```

COM49 - PuTTY
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
EraseCheck: 0
Release Device: 0
IR SHIFT: 89
0x8321, 0x40B2, 0x5A80, 0x120, 0xD3D2, 0x22, 0xEBD2, 0x21, 0x40B1, 0x2710, 0x0,
0x8391, 0x0, 0x9381, 0x0, 0x27F6, 0x3FFA, 0x4031, 0x400, 0x12B0, 0xC042, 0x430C,
0x12B0, 0xC000, 0x12B0, 0xC03C, 0xD032, 0x10, 0x3FFD, 0x4303, 0x4303, 0x3FFF, 0
x4303, 0x431C, 0x4130,
Release Device: 0
Completed on attempt: 0

****Section: 1****
Address: 0xFFDE
Attempt: 0
IR SHIFT: 89
0xFFFF, 0xC034,
Release Device: 0
Completed on attempt: 0

****Section: 2****
Address: 0xFFE4
Attempt: 0
IR SHIFT: 89
0xC034, 0xC034,
Release Device: 0
Completed on attempt: 0

****Section: 3****
Address: 0xFFEA
Attempt: 0
IR SHIFT: 89
0xC034, 0xC034, 0xC034, 0xC034, 0xC034, 0xC034, 0xC034, 0xC034, 0xC034,
0xC024,
Release Device: 0
Completed on attempt: 0
MSP430 Programmed Successfully

```

Figure 18. MSP430G2553 Programmed Successfully

The same steps must be followed to program any flash-based MSP430 MCU using a CC2640R2 or CC3220 device.

4.2.2 MSP430FR2311 SBW Target Example

To program an MSP430FR2311 using the MSP432P401R through SBW, the same steps as in [Section 4.2.1](#) must be followed.

[Figure 19](#) shows the connections between the MSP430FR2311 and the MSP432P401R

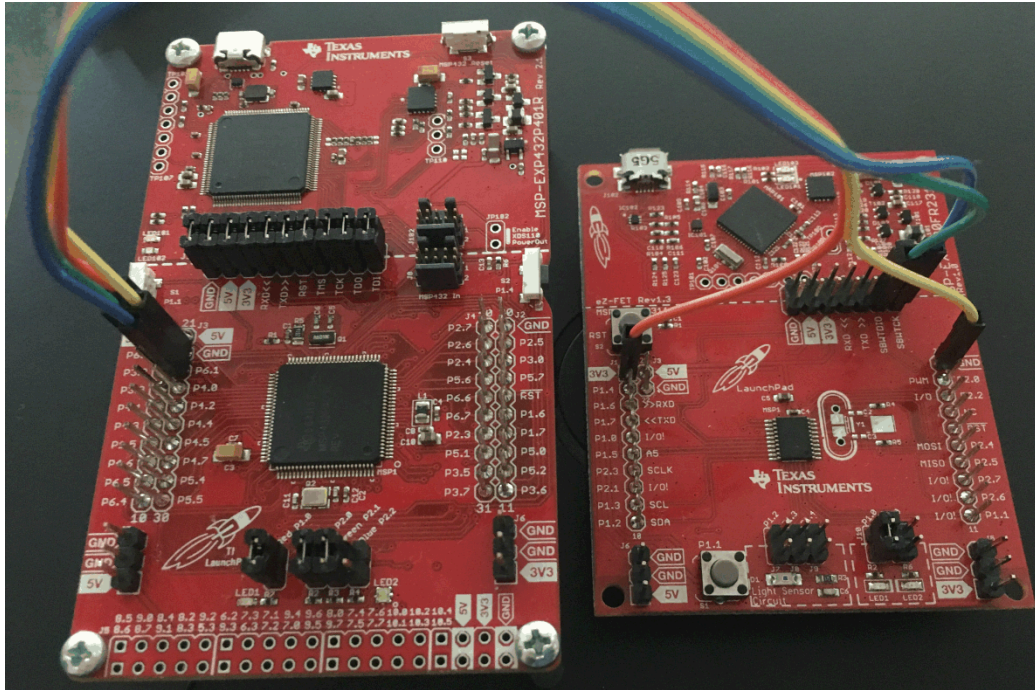


Figure 19. MSP432P401R LaunchPad Development Kit Connected to MSP430FR2311

The MSP432P401R connections are the same as in [Section 4.2.1](#).

[Figure 20](#) shows the MSP430FR2311 connections.

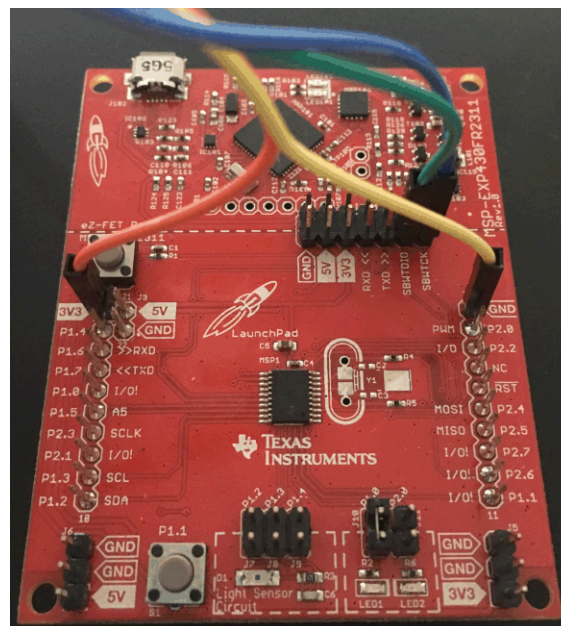
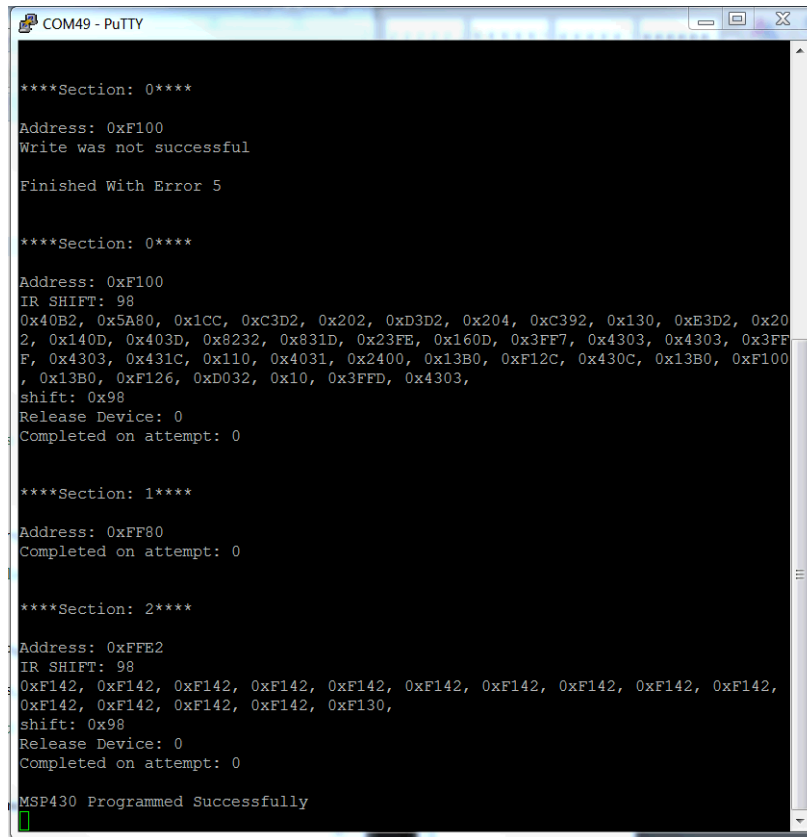


Figure 20. MSP430FR2311 LaunchPad Development Kit SBW Connections

After the connections are made, the example project MSP432Host_SBW_MSP430FR can be debugged and the result can be viewed using PuTTY (see [Figure 21](#)).



```

COM49 - PuTTY

****Section: 0****
Address: 0xF100
Write was not successful
Finished With Error 5

****Section: 0****
Address: 0xF100
IR SHIFT: 98
0x40B2, 0x5A80, 0x1CC, 0xC3D2, 0x202, 0xD3D2, 0x204, 0xC392, 0x130, 0xE3D2, 0x20
2, 0x140D, 0x403D, 0x8232, 0x831D, 0x23FE, 0x160D, 0x3FF7, 0x4303, 0x4303, 0x3FF
F, 0x4303, 0x431C, 0x110, 0x4031, 0x2400, 0x13B0, 0xF12C, 0x430C, 0x13B0, 0xF100
, 0x13B0, 0xF126, 0xD032, 0x10, 0x3FFD, 0x4303,
shift: 0x98
Release Device: 0
Completed on attempt: 0

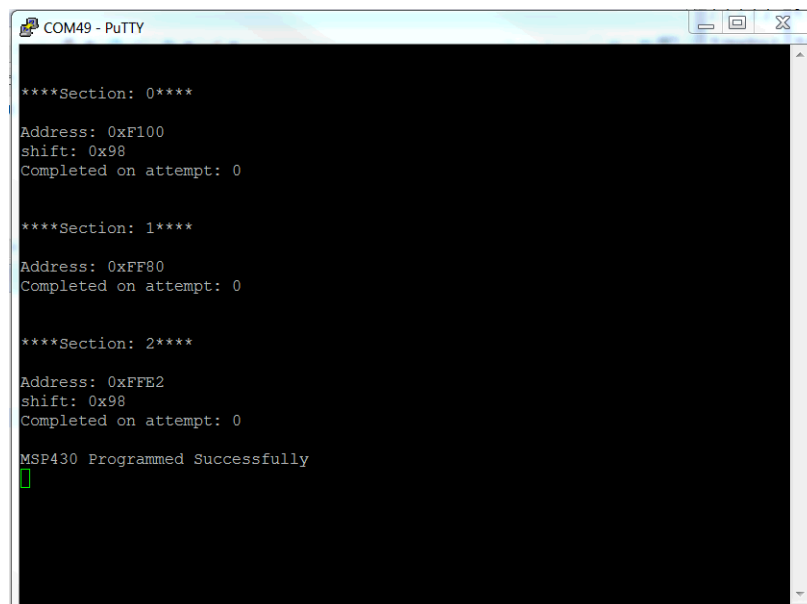
****Section: 1****
Address: 0xFF80
Completed on attempt: 0

****Section: 2****
Address: 0xFFE2
IR SHIFT: 98
0xF142, 0xF142, 0xF142, 0xF142, 0xF142, 0xF142, 0xF142, 0xF142, 0xF142,
0xF142, 0xF142, 0xF142, 0xF142, 0xF130,
shift: 0x98
Release Device: 0
Completed on attempt: 0
MSP430 Programmed Successfully

```

Figure 21. MSP432Host_SBW_MSP430FR Serial Output (DEBUG is 1)

After running the example, the following result is seen in the serial terminal (see [Figure 22](#)).



```

COM49 - PuTTY

****Section: 0****
Address: 0xF100
shift: 0x98
Completed on attempt: 0

****Section: 1****
Address: 0xFF80
Completed on attempt: 0

****Section: 2****
Address: 0xFFE2
shift: 0x98
Completed on attempt: 0
MSP430 Programmed Successfully

```

Figure 22. MSP430FR2311 Programmed Successfully

The same steps must be followed to program any FRAM-based MSP430 MCU using a CC2640R2 or CC3220 device.

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.

To create a new firmware image for any MSP430 MCU from an existing code composer studio 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 MCU 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 as 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
 */

const uint16_t eprom[] =
{
//0xc000
0x8321, 0x40B2, 0x5A80, 0x0120, 0xD3D2, 0x0022, 0xE3D2, 0x0021,
0x40B1, 0xC350, 0x0000, 0x8391, 0x0000, 0x9381, 0x0000, 0x27F6,
0x3FFA, 0x4031, 0x0400, 0x12B0, 0xC042, 0x430C, 0x12B0, 0xC000,
0x12B0, 0xC03C, 0xD032, 0x0010, 0x3FFD, 0x4303, 0x4303, 0x3FFF,
0x4303, 0x431C, 0x4130,
//0xffde
0xFFFF, 0xC034,
//0xffe4
0xC034, 0xC034,
//0xffea
0xC034, 0xC034, 0xC034, 0xC034, 0xC034, 0xC034, 0xC034, 0xC034,
0xC034, 0xC034, 0xC022,

};

const uint32_t eprom_address[] =
{
0xc000, 0xffde, 0xffe4, 0xffea,
};

```

```
const uint32_t eprom_length_of_sections[] =
{
    35, 2, 2, 11,
};

const uint32_t eprom_sections      = 4;

const uint32_t eprom_termination = 0x00000000; /*Check device data sheet*/
const uint32_t eprom_start       = 0x00000000; /*Check device data sheet*/
const uint32_t eprom_finish      = 0x00000000; /*Check device data sheet*/
const uint32_t eprom_length      = 0x00000000; /*Check device data sheet*/
```

The first variable is flash. The flash_sections variable, hold 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 4 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:

- For address 0xC000
The first 35 16-bit words (70 bytes) of data in flash must be written to the MSP430 MCU, starting at address 0xC000.
- For address 0xFFDE
Starting from the 36st element in the flash variable, 2 16-bit words (4 bytes) must be written to the MSP430 MCU, starting at address 0xFFDE.
- For address 0xFFE4
Starting at the 38th element in the flash variable, 2 16-bit words (4 bytes) must be written to the MSP430 MCU, starting at address 0xFFE4.
- For address 0xFFEA
Starting at the 40th element in the flash variable, 11 16-bit words (22 bytes) must be written to the MSP430 MCU, 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 same 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 as 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
 */

uint16_t fram[] =
{
    //0xf100
    0x40B2, 0x5A80, 0x01CC, 0xC3D2, 0x0202, 0xD3D2, 0x0204, 0xC392,
    0x0130, 0xE3D2, 0x0202, 0x140D, 0x403D, 0x8232, 0x831D, 0x23FE,
```

```

0x160D, 0x3FF7, 0x4303, 0x4303, 0x3FFF, 0x4303, 0x431C, 0x0110,
0x4031, 0x2400, 0x13B0, 0xF12C, 0x430C, 0x13B0, 0xF100, 0x13B0,
0xF126, 0xD032, 0x0010, 0x3FFD, 0x4303,
//0xff80
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
//0xffe2
0xF142, 0xF142, 0xF142, 0xF142, 0xF142, 0xF142, 0xF142, 0xF142,
0xF142, 0xF142, 0xF142, 0xF142, 0xF142, 0xF142, 0xF130,

};

const uint32_t fram_address[] =
{
0xf100, 0xff80, 0xffe2,
};

const uint32_t fram_length_of_sections[] =
{
37, 4, 15,
};

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*/
const uint32_t fram_finish      = 0x00000000; /*Check device data sheet*/
const uint32_t fram_length      = 0x00000000; /*Check device data sheet*/

```

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

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.

[Figure 23](#), [Figure 24](#), and [Figure 25](#) show how to generate a TI-TXT hex file in CCS.

1. Right click the project and click the properties option.
2. Check the *Enable MSP430 Hex Utility* checkbox.

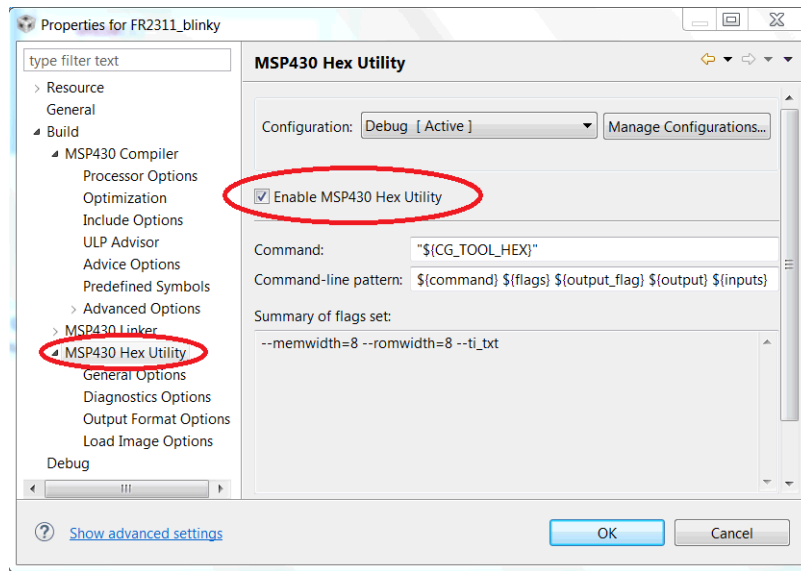


Figure 23. Enabling MSP430 Hex Utility

3. In Output Format Options, select TI-TXT hex format.

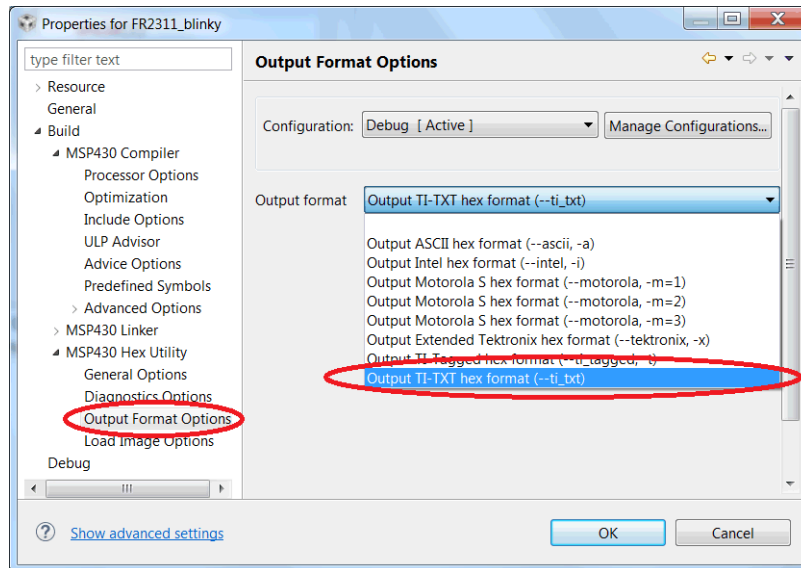


Figure 24. MSP430 TI-TXT Hex Format

4. After the project is built, the .txt containing the full firmware image is created. [Figure 25](#) shows this file.

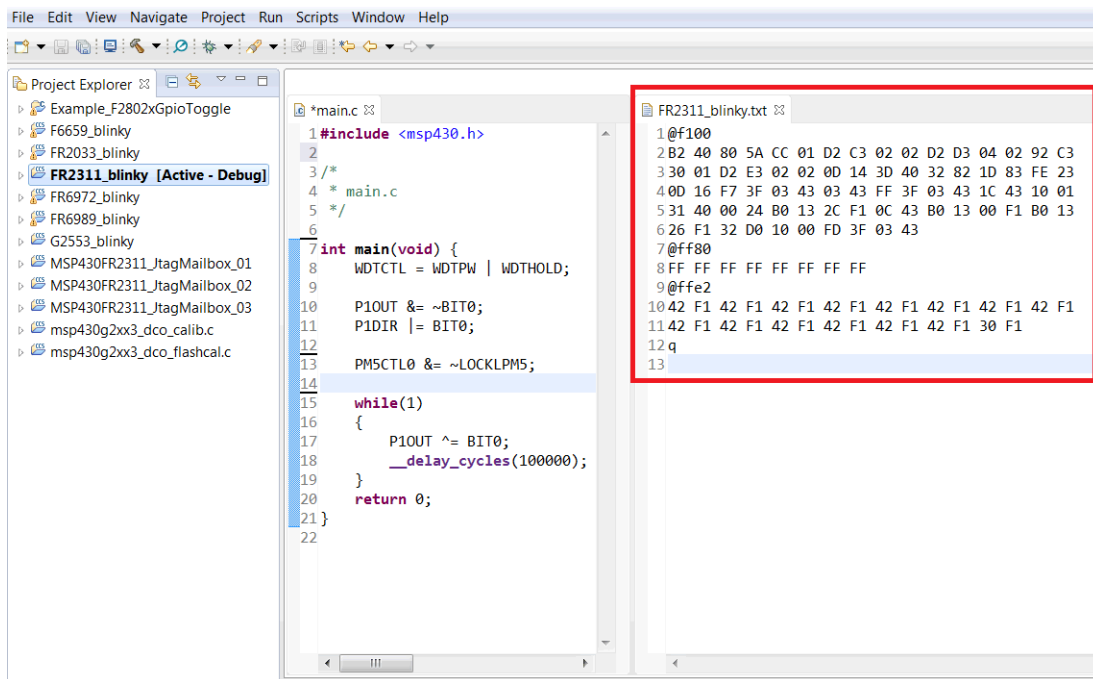


Figure 25. MSP430FR2311 TI-TXT Hex Format Firmware Image

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.

```

1  """
2  Created By: Nima Eskandari
3  Description: Creates a .h file from TI TXT HEX file.
4  """
5
6  # TODO: Change th sourcePath variable to the path of the TI HEX TXT file.
7
8  sourcePath = r"C:\Users\████████\workspace_blinky\FR231i_blinky\Debug\FR231i_blinky.txt"
9  imagePath = "msp430fr_image.h"
10
11 class MemorySection:
12
13     def __init__(self, content):
14         self.Content = content
15         self.ContentSplitBySpace = []
16         content_split_by_line = self.Content.split('\n')
17         for line in content_split_by_line:
18             line_split_by_space = line.split(' ')
19             for i in range(0, len(line_split_by_space)):
20                 line_split_by_space[i] = line_split_by_space[i].replace(' ', '')
21                 if line_split_by_space[i] != '':
22                     self.ContentSplitBySpace.append(line_split_by_space[i])
23
24         self.StartAddress = self.get_start_address()
25         self.SizeInBytes = self.get_number_of_bytes()
26         self.SizeInWords = int(self.SizeInBytes / 2)
27         self.ContentInBytes = self.get_content_in_bytes()
28         self.ContentInWords = self.get_content_in_words()
29         self.HexPaddedContentInWords = self.get_hex_padded_content_in_words()
30
31     def print_section(self):
32         print("Start Address: " + self.StartAddress)
33         print("Size in words: " + str(self.SizeInWords))
34         print("Content in words: ")
35         print(self.HexPaddedContentInWords)
36
37     def get_start_address(self):
38         return "0x" + self.ContentSplitBySpace[0]

```

Figure 26. FRAM TI-TXT Hex to Firmware Image Converter

The variable `sourcePath` must be modified 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\████████\AppData\Local\Programs\Python\Python35\python.exe
C:/Users/████████/PycharmProjects/Test/Replicator/FRAM_TI_txt_hex_to_image.py
Start Address: 0xf100
Size in words: 37
Content in words:
['0x40B2', '0x5A80', '0x01CC', '0xC3D2', '0x0202', '0xD3D2', '0x0204',
'0xC392', '0x0130', '0xE3D2', '0x0202', '0x140D', '0x403D', '0x8232',
'0x831D', '0x23FE', '0x160D', '0x3FF7', '0x4303', '0x4303', '0x3FFF',
'0x4303', '0x431C', '0x0110', '0x4031', '0x2400', '0x13B0', '0xF12C',
'0x430C', '0x13B0', '0xF100', '0x13B0', '0xF126', '0xD032', '0x0010',
'0x3FFD', '0x4303']
Start Address: 0xff80
Size in words: 4
Content in words:
['0xFFFF', '0xFFFF', '0xFFFF', '0xFFFF']
Start Address: 0xffe2
Size in words: 15
Content in words:
['0xF142', '0xF142', '0xF142', '0xF142', '0xF142', '0xF142',
'0xF142', '0xF142', '0xF142', '0xF142', '0xF142', '0xF142', '0xF142',
'0xF142', '0xF130']

Process finished with exit code 0

```

Figure 27. Firmware Image Python Script Console Output

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

6 Error Messages

If the target to host connection setup is incorrect or the JTAG fuse is blown, the error in [Figure 28](#) can be seen in the serial terminal.

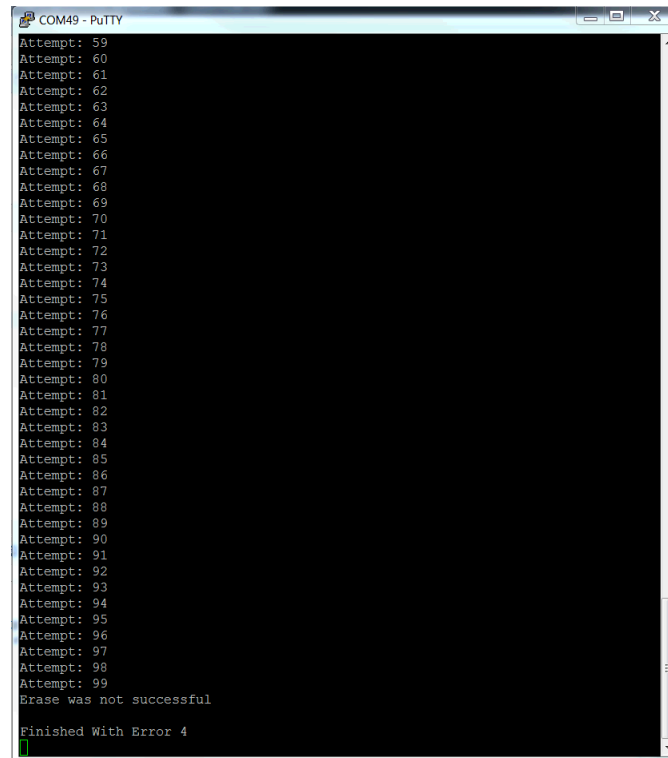


Figure 28. SBW Error

To debug this issue further, change the `DEBUG` macro in `config.h` to 1, which prints more information into the serial terminal. If that does not give any hints as to what the source of the failure is, debug and single step through the `Program_MSPFR()` or `Program_MSP()` in `SBW430FR.c` or `SBW430.c`. Also double check the connections between the host and the target. Finally, cycle power to the target MSP430 MCU and run the host again.

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 that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such 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 non-compliance 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 (<http://www.ti.com/sc/docs/stdterms.htm>), [evaluation modules](#), and [samples](http://www.ti.com/sc/docs/sampterm.htm) (<http://www.ti.com/sc/docs/sampterm.htm>).

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