Application Report
SLAA729A–January 2017–Revised March 2017

Debugging Flash Issues on the MSP430™ Family of Microcontrollers

Caleb Overbay
Uwe Haensel

MSP430 Applications
MSP430 Quality

ABSTRACT

Flash memory is widely used in the MSP430™ family of microcontrollers (MCUs) for nonvolatile data storage. While specific device user's guides and data sheets explain how to properly program and utilize flash, common issues still arise when working with this type of memory. In many circumstances these issues are caused by improper handling of flash according to data sheet specifications. However, more advanced debug techniques are sometimes required to find the root cause. This application report provides an explanation of common flash issues, what causes them, and best practices for resolution.

Contents

List of Figures

List of Tables

Trademarks

1 Introduction

Flash memory is a form of nonvolatile electronic storage used in MSP430 MCUs. The most common use cases are factory or production programming and field updates or data logging. These scenarios are also where many common issues arise. While they may seem difficult to solve, this application report guides you through identifying and solving the most common flash issues. Finally, if answers cannot be found in this document, a form is located in Section 6 that can be used to aid in submitting a request for assistance from a TI representative.

Use Table 1 as a quick guide to what may be causing a flash issue you may be experiencing. For more detailed information, see the respective section for each issue within this document.

Table 1. Common Flash Issues and Solutions

Issue	Solutions
Supply voltage interruption or loss	Check for good connection. Ensure supply is regulated properly. Reduce EMI and ESD sources.
Violation of system frequency	Check safe operating area for voltage and frequency.
Violation of supply voltage needed for flash programming	Check the voltage changed during programming error (VPE) bit. A change may indicate a possible programing issue. Ensure the supply voltage is stable during programming and meets data sheet specifications.
Incorrect or no clock provided to flash timing generator	Provide an active clock to the flash controller. Properly divide source clock to meet frequency requirements in the data sheet.
Violation of cumulative programming time	Keep track of the cumulative programming time for a row. Refresh the row by erasing and writing back data.
Unexpected erases	Be mindful of flash segment boundaries. Save data to RAM before erasing a segment. Follow segment boundaries when allocating memory.
Logical coding errors	Check flash addresses are within range before operating on them.
Programming tool not recognizing device	Update tool to latest firmware. Check hardware connections.

2 Flash Basics

There are several basic characteristics of flash memory that need to be understood when developing an application utilizing it. These include flash erase and writes, memory segmentation, the meaning of unexpected 0s or 1s, and the endurance of flash. Having a basic knowledge of these topics helps immensely when debugging any flash issue observed in your application.

2.1 Erasing Flash

The act of erasing a flash bit always changes its state to a 1. On MSP430 MCUs, the smallest amount of flash memory that can be erased is called a *segment*. Even if the user wants to erase only a single bit, all other information in that segment is also erased. This proposes a unique challenge when trying to maintain portions of a segment during an erase. Any additional information that needs to remain unaltered must first be saved to RAM or another flash segment and then written back to the original location after the erase has been performed. More information on segmentation and segment sizes can be found in the device user's guide in a section titled *Flash Memory Segmentation*.

Conversely, TI's FRAM-based MSP430 MCUs have no need for segmentation. The FRAM memory can be treated as if it were RAM, while still maintaining the desired property of being nonvolatile. More information on the benefits of TI's FRAM portfolio can be found in *FRAM FAQs*.

2.2 Writing Flash

The act of writing to flash is destructive, meaning that a bit can only be transitioned from a 1 to a 0 during the write process. Therefore, if the state of a bit needs to be changed from 0 to 1, an erase must first take place on the *entire segment* the bit is located within. Otherwise, there is a possibility of an overwrite occurring. Section 2.1 describes the process necessary for maintaining data in a segment during an erase. Also, writes, unlike erases, can be performed in single bit, byte, or word sizes on an MSP430 MCU.

2.3 Unexpected 0s or 1s

Often, the *signature* of an issue can help guide the debug process. When 0s are consistently observed where 1s are expected and vice versa, it can give a clue as to what caused the unexpected state. One of the most effective techniques is to read out the affected flash memory and compare it with what was expected in that location.

If 0s are consistently observed where 1s are expected, there is a possibility of data being overwritten. 'Writing' to flash means that bits are being transitioned from a 1 to a 0 state. If multiple writes or overwrites occur at the same location, eventually the majority of those bits will be in a 0 state. Section 2.2 describes the process of writing to flash and how to avoid overwriting data.

If 1s are consistently observed where 0s are expected, there is a possibility that an unexpected erase occurred. In flash memory, the state of a bit can become 1 only as a result of an erase. Also, the smallest amount of flash that can be erased is a segment. This could signal that an erase occurred without taking into account the entire segment would also be erased. See Section 2.1 for a more detailed description of flash erases and segmentation.

2.4 Flash Endurance

Flash endurance is a measure of the number of erase and write cycles that a flash array can achieve while retaining data integrity. Endurance on MSP430 MCUs is specified to be a minimum of 10000 read/write cycles, but they typically fulfill 100000 read/write cycles. *MSP430 Flash Memory Characteristics* provides a detailed description of flash endurance and the factors that affect this specification.

NOTE: If your application is at risk of pushing the limits of flash endurance, consider writing a flash wear leveling algorithm or switching to one of TI's FRAM-based MSP430 microcontrollers.

2.5 Flash Data Retention

Data retention is an important aspect of flash memory and can be a concern when an application undergoes extreme temperatures. *Understanding MSP430 Flash Data Retention* provides a detailed discussion of factors that influence this parameter, the various figures of merit to interpret flash data retention, along with tips to help prevent failures on MSP430 MCUs.

3 Supply Voltage Related Flash Issues

Insufficient supply voltage is the root cause of many flash programming issues. Monitoring V_{CC} while trying to duplicate the issue determines if data sheet specifications are being met. Violations of these specifications can cause erroneous code execution and incorrect device operation. Always see the device-specific data sheet to find the appropriate voltage levels required for programming flash and operating at the desired system frequency. Also, be cautious of ESD or EMI causing V_{CC} interruptions.

3.1 Supply Voltage Interruption

If voltage glitches occur during flash programming, incorrect cell currents can cause unexpected results. Some MSP430 MCU device variants feature a VPE bit in the flash memory control register that signals when DV_{CC} changes significantly during programming. The changing of this bit during programming can possibly indicate an invalid result and can be used to debug supply voltage related flash problems. See the device-specific data sheet and user's guide to determine if this functionality is available on your device.

To determine if a significant change to supply voltage occurred during a programming, the value of the VPE bit must be observed to change from the start of programming flash to the end of the programming sequence. For example, if VPE = 1 when the programming sequence starts, and VPE = 0 when the programming sequence terminates, then a significant change of voltage during programming occurred. The same is true if the VPE bit flips from a 0 value to a 1 value during programming. Additional checks, such as a CRC, must be done on the recently programmed memory to determine if the programmed values were corrupted.

A multitude of sources can cause supply voltage interruption or loss, including:

- Battery interruption or losing charge
- Bouncing or bad contacts
- Socket wear out
- Bent leads
- Contamination
- ESD or EMI
- High current consumption elsewhere in system

Information regarding the required supply voltage levels during flash programming can be found in the device-specific data sheet. See Section 3.3 for a detailed description of supply voltage requirements during flash programming.

NOTE: When experiencing V_{CC} issues, always be sure to switch off the supply voltage before removing an IC from its socket.

3.2 System Frequency Dependency on Supply Voltage

The maximum allowable system frequency has a strong dependency on the voltage supplied to the MSP430 MCU. Violating the supply voltage needed to operate at a given frequency can cause erroneous code execution and adversely affect any flash programming that may occur in the system. A commonly overlooked scenario is violating the maximum system frequency during device power up. Always ensure your application is meeting the required supply voltage level before operating at a desired frequency.

This dependency is represented differently based on the MSP430 variant being used. However, the specification is always located in the ‘Recommend Operating Conditions' section of a device data sheet. For example, Figure 1 shows how this dependency is represented in *MSP430G2x53, MSP430G2x13 Mixed-Signal Microcontrollers*.

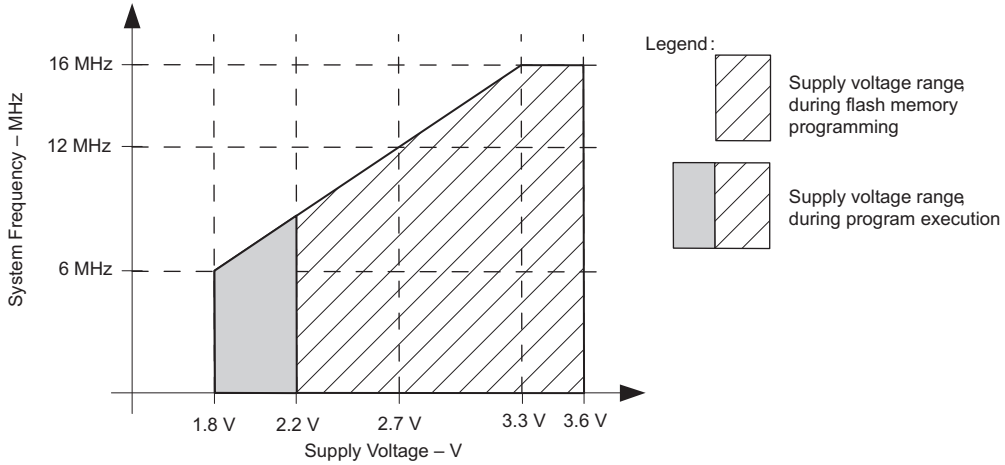
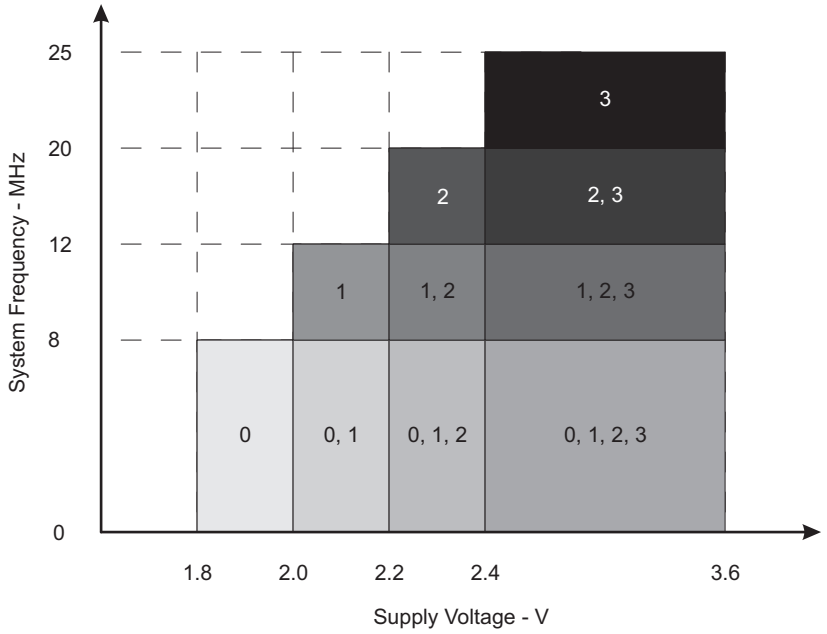


Figure 1. MSP430G2x53, MSP430G2x13 Safe Operating Area

As can be seen in Figure 1, the system frequency is represented on the y-axis, and the supply voltage is represented on the x-axis. The minimum supply voltage to properly operate this microcontroller is 1.8 V, but it can only safely run up to 6 MHz at this voltage. It can also be seen that the minimum supply voltage required to program flash is 2.2 V. Finally, as the supply voltage increases, so does the maximum allowable system frequency.

Figure 2 shows another example of how the dependency is represented in *MSP430F552x, MSP430F551x Mixed-Signal Microcontrollers*. This depiction is considerably different than the one shown in Figure 1 but shows similar information.



NOTE: The numbers within the fields denote the supported PMMCOREVx settings.

Figure 2. MSP430F552x, MSP430F551x Safe Operating Area

Figure 2 shows the core voltage divided into levels 0, 1, 2, and 3. The diagram also shows which core level the MCU can operate at given the desired system frequency and supply voltage. Again, the important thing to note here is that as the system frequency increases, the required supply voltage increases as well. Always be sure to check your device-specific data sheet for a similar diagram when deciding a system frequency and supply voltage.

Finally, be mindful of operating conditions at device start-up. In many applications, V_{CC} can take a relatively long time to reach the required voltage for safe operation at the desired system frequency. This could be due to several factors including slow voltage ramp up or ripple on the line. Trying to operate at an elevated system frequency during start-up could cause erroneous code execution and adversely affect your system. Many MSP430 MCUs provide a Supply Voltage Supervisor (SVS) that can be used to hold the device in reset until DV_{CC} is at an appropriate level for the application to function properly. This is a module that is device-specific and must be initialized appropriately by the user for their specific application. See your device-specific data sheet and user guide to discover if this feature is available.

3.3 Supply Voltage Required for Flash Programming

There is a minimum supply voltage that must be met to properly program flash. Typically this information is located in the 'Specifications' section of a device-specific data sheet. In that section, $DV_{CC(PGM,ERASE)}$ specifies the minimum and maximum voltage levels required to program flash. Table 2 is an example of how this is represented in a given MCU data sheet.

Table 2. Example of MSP430F5529 $DV_{CC(PGM,ERASE)}$

PARAMETER		MIN	TYP	MAX	UNIT
$DV_{CC(PGM,ERASE)}$	Program and erase supply voltage	1.8		3.6	V

In some applications, such as data logging, there is a possibility of code that modifies flash throughout device execution. Operating conditions at device start-up are often overlooked during development. If the microcontroller attempts to program flash without the supply voltage meeting requirements specified in the data sheet, the data written may be invalid. This could be caused by slow ramp up or interruption of the supply voltage. As discussed in Section 3.2, many MSP430 MCUs provide a Supply Voltage Supervisor (SVS) that can be used to hold the device in reset until DV_{CC} is at an appropriate level for the application to function properly. See your device-specific data sheet and user's guide to discover if this feature is available.

4 Proper Flash Setup and Utilization

Improperly setting up and utilizing flash memory will cause unexpected behavior in your system. Several factors must be considered including configuring the flash controller clock, cumulative programming time, how to allocate and partition flash, and proper in-app modification of flash. Additionally, some MSP430 MCUs enable the ability to verify flash integrity by using marginal read mode. Fully understanding these elements and how they influence flash operation helps not only when debugging but also when developing a flash-based application.

4.1 Flash Timing Generator Source Clock

In specific MSP430 device variants including the MSP430F1xx, MSP430F2xx, and MSP430F4xx MCUs, the flash timing generator requires user specification of a clock source. The timing generator can be sourced from the ACLK, SMCLK, or MCLK but special care must be used to ensure this source is divided down to an appropriate frequency, f_{FTG}. Figure 3 shows a block diagram depicting how the source clock can be selected for the flash timing generator.

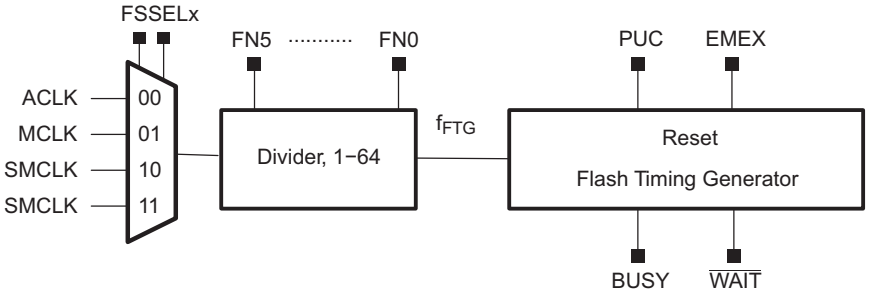


Figure 3. Example of MSP430x2xx Flash Timing Generator

The source clock is selected by setting the FSSELx bits and the divider is selected by setting the FNx bits. Consult the device-specific data sheet and user guide for more information on properly setting up the flash timing generator and the required range of f_{FTG}.

On other MSP430 device variants including the MSP430F5xx and MSP430F6xx MCUs, the timing generator does not require a source clock be provided. This alleviates the need to ensure f_{FTG} is in the proper frequency range and makes programming flash simpler.

4.2 Cumulative Programming Time

To program a single flash bit, byte, or word, a high voltage must be applied to a complete 64-byte row of cells. This high voltage generates a stress on the entire row regardless of the size of flash being programmed and must be time limited to avoid damage. Erasing a row resets the cumulative programming time to zero, and programming can start again. Section 3.3 of *MSP430 Flash Memory Characteristics* provides an in-depth description of this issue and how to avoid violating the cumulative programming time.

4.3 Flash Memory Allocation and Partitioning

On MSP430 MCUs, the default linker file partitions flash into main and information memory sections. There can be multiple information memory sections, and there is no operational difference between main and information memory. However, by default, no code is placed in an information memory section, making them ideal for small amounts of user-defined data.

Flash memory can also be partitioned into user-defined sections to keep code and data separate. This can be accomplished by modifying the default linker command file generated when a new project is created in Code Composer Studio™ IDE or IAR Embedded Workbench® IDE. See Section 4.3.1 and Section 4.3.2 for a brief description of how to modify the linker file to include a new flash section and place a variable in that newly defined section.

When allocating or partitioning flash, always keep in mind segment boundaries. It is best practice to align any new section of flash with segment boundaries to avoid unexpectedly erasing data outside of the section when attempting to program it. See Section 2.1 for more information on flash segments and their importance.

4.3.1 Modifying Code Composer Studio™ IDE Linker Command File

1. First you need to modify the linker command file for your specific MSP430 device variant. This example uses the MSP430G2553. It is good practice to make a copy of the default linker command file, lnk_msp430g2553.cmd, and rename it, lnk_msp430g2553_modified.cmd, so you can make modifications while still maintaining the original.
   (a) Set aside enough room in your memory for the data you would like to store. It is best practice to align your partition with segment boundaries so that erases do not affect data outside the partition. First find "FLASH" in the memory map at the top of the linker file and comment it out. Then, make a new area in flash, renaming it "MY_FLASH", for example:

```
//FLASH: origin = 0xC000, length = 0x3FDE
MY_FLASH: origin = 0xC000, length = 0x0200 // New area, aligned with segment boundary
FLASH: origin = 0xC200, length = 0x3DDE    // Rest of flash, modified start and length
```

   (b) Specify what is going to be allocated for that area in memory. Scroll down to the "SECTIONS" area of the linker file. Above the .infoA and other allocations, add the following (name the section whatever you prefer):

```
.MY_MEM :{} > MY_FLASH // Place section MY_MEM in area MY_FLASH
```

2. In your main application code .c file where global variables are declared at the top of code, use a pragma to place the variable in your newly created flash section:

```
#pragma DATA_SECTION(data, ".MY_MEM")
unsigned int data;
```

3. Finally, ensure that when you build your project, Code Composer Studio IDE is using the modified linker file. This can be checked by following Project → Properties → General. The linker command file listed should be your modified one. If it is not, change it to use your updated one.

4.3.2 Modifying IAR Embedded Workbench® IDE Linker Command File

1. First you need to modify the linker command file for your specific MSP430 device variant. This example uses the MSP430G2553. It is good practice to make a copy of the default linker command file, lnk_msp430g2553.xcl, and rename it, lnk_msp430g2553_modified.xcl, so you can make modifications while still maintaining the original.

   (a) All you need to do is add a line under placement directives to specify where in flash you want to put your data:

```
// ---------------------------------
// Placement directives
//

-Z(DATA)MY_MEM=C000-C200  // Area in flash for data, aligned with segment boundaries
```

2. In your main application code .c file where global variables are declared at the top of code, use a pragma to place the variable in your newly created flash section:

```
#pragma segment="MY_MEM"
#pragma location="MY_MEM"
__no_init unsigned int data;
```

3. Finally, ensure that when you build your project, IAR is using the modified linker file. This can be checked by following Project → Options → Linker. Under Linker Configuration File, check the "override default" box and then browse to your modified file.

4.4 *In-App Modification of Flash (Data Logging)*

Data logging is present in a wide variety of applications. In this scenario flash memory is modified by the application itself and can present a number of challenges. Below is a list of the most common issues that arise when developing code that modifies flash and how to solve them:

- System frequency vs supply voltage specification violated
  - Stay mindful of device conditions during power up and power down.
  - See Section 3.2 and Section 3.3 for further information.
- Flash controller clock is not properly configured
  - Applies to specific MSP430 device variants including the MSP430F1xx, MSP430F2xx, and MSP430F4xx MCUs.
  - See Section 4.3 for further information.
- Logical coding errors on flash addresses
  - Ensure flash addresses are within range.
  - Be mindful of segment boundaries and sizes.
- Stack overflow
  - Causes erroneous code execution
  - Commonly see random bytes being overwritten
  - Possibility of jumping to a flash routine in application code
  - Alleviate by reducing RAM usage
    - Reduce global variables and arrays.

Finally, some operations must be executed from either RAM or flash and some can operate from both. For these operations, see the device user's guide for a detailed description of how to operate from RAM and sample code to get started.

4.5 Flash Integrity Verification Using Marginal Read Mode

Marginal read mode is a device-specific functionality that can be used as an early indication of weak memory cells. Check your MCU data sheet to see if this feature is available. When using marginal read mode, marginally programmed flash memory bits can be detected. This could be caused by improper f_{FTG} settings or a violation of the minimum supply voltage. For more information on proper f_{FTG} settings, see Section 4.2, and for more information on the minimum supply voltage, see Section 3.

A common method of using marginal read mode to verify flash integrity is to periodically perform a checksum calculation over a section of flash, then repeat the same calculation with marginal read mode enabled. If these two checksums do not match, it could indicate an insufficiently programmed flash memory location. It is possible to refresh the affected flash memory location by disabling marginal read mode, copying to RAM, erasing the flash segment, then writing back from RAM.

5 Programming Devices With Application Code

Some users experience flash issues when programming application code onto an MSP430 MCU. This can occur during development or during production. This section discusses issues that may occur during production programming, how to use the JTAG locking feature, and where to look for help when having trouble with a TI-provided debugging tool.

5.1 Production Programming

Even at production phase, there are common issues that arise when programming flash, the most common being insufficient supply voltage during programming. For more information regarding supply voltage and flash programming, see Section 3.

If the issue observed is not related to supply voltage, the focus then shifts to the interface used to program the device. This includes JTAG, BSL, or a custom solution that may have been created. When debugging JTAG and BSL issues, see *MSP430 Programming With the JTAG Interface* and *MSP430 Programming With the Bootloader (BSL)*, respectively, and ensure your application is meeting all specifications discussed in these documents.

5.2 JTAG Lock

MSP430 MCUs offer a JTAG locking feature to help protect against unwanted memory access. This feature is available on all MSP430 devices but the locking mechanism is implemented differently depending on the device variant. Some devices have a physical fuse that, once blown, permanently locks the JTAG. This fuse can be blown by applying a high voltage to the TEST pin of the device rendering the interface unusable. The MSP430F1xx, MSP430F2xx, and MSP430F4xx MCUs support this permanent locking feature.

Others devices contain an eFuse (or "soft"-fuse) that can be locked using a software write to specific memory signature areas and can be reversible though bootloader (BSL) access to clear the signatures. Additionally, the MSP430FRxx JTAG interface has the option of being locked with a JTAG password that will grant users access if they provide the correct password through the JTAG tool chain. Conversely, the MSP430F5xx and MSP430F6xx devices can only be locked without this JTAG password option. Section 1.4 of *MSP430 Programming With the JTAG Interface* explains the process of locking the JTAG both permanently and temporarily. Also see chapter 2 of MSP Code Protection Features for a description of the different JTAG locking features across families.

Occasionally there is a need to reprogram flash when the fuse has already been blown. Unfortunately in this scenario, the JTAG cannot be used to access the microcontroller. However, if the bootloader (BSL) is available on the device, it can be used for reprogramming. For more information on how this is done, see *MSP430 Programming With the Bootloader (BSL)*. Additionally, corrupt flash can sometimes make the JTAG appear to be locked due to unpredictable code execution preventing JTAG access. When this happens, attempt to erase the part through BSL to gain access again.

5.3 Programming Tools

A variety of tools are available for programming an MSP430 MCU and TI strongly encourages using TI recommended tools to ensure compatibility. When experiencing an issue with these devices, ensure you have the latest tool and the latest firmware revision for that tool. Often, bug fixes are released and updates can alleviate some of the issues observed. Also the MSP Debuggers User's Guide can be helpful when learning how to use TI MSP debug tools.

The firmware on a TI provided programming tool reads the MSP430 device ID and revision to know which device it is trying to program. Sometimes, newer MCUs and revisions are not recognized due to older firmware on the tools. As described previously, updating to the latest firmware revision should alleviate this issue. To do this, first you need to check that the latest MSP debug stack is installed. In Code Composer Studio IDE, select Help → About Code Composer Studio → Installation Details and then expand "TI MSP430 tool-chain." Select "MSP Debug Probe Drivers" and in the text box below it, scroll to "Components:" as seen in Figure 4.

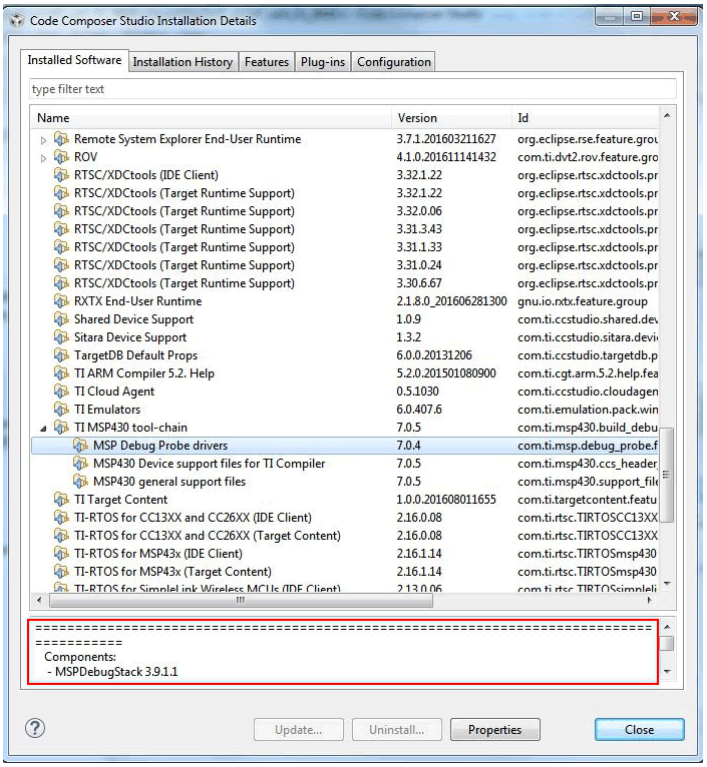


Figure 4. Example of Checking MSP Debug Stack Version Installed

Ensure that the MSP Debug Stack version you see here is the same as the latest version on the MSPDS tool folder. If it does not match this version, update to the latest in Code Composer Studio IDE by selecting Help → Check for Updates then following the prompts. When the latest version has been installed and you attempt to use the programming tool, Code Composer Studio IDE prompts you to update the firmware and program it with the latest available in the debug stack.

Also, be sure to use the verification function when programming with a TI provided tool. By using this feature, it can be verified that flash programming is working correctly or the appropriate password is being provided to the BSL/JTAG. This can be enabled in Code Composer Studio IDE by following Project → Properties → Debug and selecting "Full verification" under Verification Options. Note that full verification should be set by default in a new Code Composer Studio IDE project.

Finally, if you are using a custom programming tool, see Chapter 2 of *MSP430 Programming With the JTAG Interface*. This chapter provides a description of how to develop an example MSP430 flash programmer.

5.4 References

1. FRAM FAQs
2. MSP430G2x53, MSP430G2x13 Mixed-Signal Microcontrollers
3. MSP430F552x, MSP430F551x Mixed-Signal Microcontrollers
4. MSP430 Programming With the JTAG Interface
5. MSP430 Programming With the Bootloader (BSL)
6. Understanding MSP430 Flash Data Retention
7. MSP Debuggers User's Guide
8. MSP Code Protection Features

6 TI Flash Questionnaire Form

If after following the guidance of previous sections flash issues are still observed, please provide the answers to the following questions to your local TI sales representative or on the MSP forum of the TI E2E™ Community.

Condition at Flash Programming Execution

V_{CC} Measured	
Frequency	
Reset pin low or high	

Programming Tools Used

Tool:	
Software revision	
Hardware revision	

Latest version can be found at www.ti.com

Programming Method

JTAG	
BSL	
Application code routines	
Fuse blown after programming intended?	

Programming Flow (How is the Issue Detected?)

Please outline the sequence.	
Verify OK?	
Multiple programming attempts done? Result?	

Other

How many units are experiencing the issue?	
If field failure, how long was the device in the field?	
Were the failing field units reprogrammed? Result?	
Are 0s being seen where 1s are expected or are 1s being seen where 0s are expected?	
Is it always the same address that fails?	
Can a code readout or comparison to original be provided?	
Any changes to the programming flow, customer code done?	
Was there a different programming tool or alternate programming method used to confirm?	

Revision History

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

Changes from January 12, 2017 to March 16, 2017 **Page**

- Changed the solution description for the "Violation of supply voltage needed for flash programming" issue in Table 1, *Common Flash Issues and Solutions* 2
- Updated the first paragraph and added the second paragraph in Section 3.1, *Supply Voltage Interruption* 3

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/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265