Application Report
SLAA793–October 2017

Troubleshooting Guidelines for MSP Devices

Dietmar Walther *Connected MCU Quality*

ABSTRACT

The complexity of applications is increasing, and the integration of functions on a single chip is also increasing. Therefore it is natural that dedicated functions sometimes do not work as a developer expects. In such cases, often a malfunction of the silicon is claimed with a very high-level description. In most cases it is revealed that the silicon itself did not necessarily cause the problem, but the problem was instead application related. Due to complexity, it is sometimes hard to determine the real cause of the malfunction. However there are multiple ways to identify a malfunction, saving time and effort for both the end customer and the vendor. This document provides some principle ways to determine the root cause of a certain number of high-level unintended behaviors. This document help to develop the troubleshooting abilities of a developer by better understanding the relation between application-related issues and silicon-related specifications and functionalities. The process in this application report describe how silicon features and measurement techniques can be used to better understand and solve these issues.

NOTE: This document does not contain a complete list of possible issues or a complete list of items to troubleshoot those issues. The application and the environment always affect whether or not the proposed actions are applicable.

Contents

Trademarks

Code Composer Studio, MSP432, MSP430, E2E are trademarks of Texas Instruments.
IAR Embedded Workbench is a registered trademark of IAR Systems.
All other trademarks are the property of their respective owners.

1 Introduction

Developing electrical applications using semiconductor parts sometimes leads to behaviors that are not expected. These behaviors can quickly lead to extended debug effort under pressure, because they often affect goals for release to market. In first instance the "biggest" component of an application is considered to be the root cause for the malfunction. In most cases, an MCU is realizing this function. In some cases, this triggers a communication from the end user the the semiconductor vendor, reporting a high-level application-dependent issue. The following are examples of issue descriptions that are not always helpful to resolve the situation.

- Application stopped working or application is dead
- No communication
- Measurement does not work
- Draw too high current
- Part is stuck
- LED does not blink

The ultimate goal for both the end user and the vendor should be resolving the malfunction to continue development or production from customer’s point of view and to provide best-in-class support from the vendor's point of view. To achieve these goals, a certain amount of application debug effort is required to isolate the source of the malfunction to certain modules or dedicated functional groups of the microcontroller. This document provides some basic troubleshooting guidelines that can be applied to perform an efficient analysis that gains important initial data and accelerates the whole issue analysis process. Because a large number of issues are only reproducible on a dedicated application board or under dedicated application environmental conditions or use case scenarios, the troubleshooting knowledge is important for both parties working on a solution.

2 General Items to Check

Independent of the actions that apply to specific issues or modes of failure, the following items must be checked to ensure that the basics are verified:

- Verify that supply capacitors are connected according specification and placed close to the device supply pins on the application PCB.
- Replace the suspect TI part with a known-good TI part to determine if the issue is PCB related or device related (for more details, see the *A-B-A Swap Method* in Guidelines for Returns).
- Use different PCs or debug probes to ensure the issue is not related to environment.
- Use code examples provided by TI to cross check general functionality of the a dedicated module.
- Check if the used clocks are set up correctly by routing the clocks to the corresponding GPIO pins and checking them with an oscilloscope.
- Disable the watchdog to prevent unexpected resets due to software related issues.
- Is the behavior dependent on environmental conditions like supply voltage or temperature change?
- Perform a memory check to ensure that the correct firmware is programmed and memory content is as expected.

3 Covered Aspects

Based on long-term data analysis, the most common scenarios have been identified, and methodologies for analysis are provided for the following:

- High current consumption
- JTAG communication problems
- Serial interface communication problems
- Reset related problems
- Hang-up related problems
- Memory related problems

This is not a comprehensive list of all possible issues, but it represents the most commonly reported issues.

The important point of troubleshooting is to go from a high-level application-related issue to a module-related issue in an efficient and effective way. A certain level of application knowledge with respect to hardware and software is required to understand how components interact. The following examples show what can be done on the first level of troubleshooting:

Application does not work

- Is supply available?
- Is the device programmed with the right code, was a read out performed, and content compared to a reference?
- Are there any software-related loops that might prevent the application from start-up?
  - Does the device wait on a trigger (for example, GPIO, temperature, or input voltage)?
  - Do all the clocks run as expected (external clocks and internal clocks)?
  - Are all internal supplies (Vcore) at the right level?
  - Are frequency and power-supply specifications considered?

No communication

- Are the pins properly connected (host or slave connected)?
- Is it possible to probe the signals of the interface with a scope?
- Is it checked if the clock sources are operating properly (for example, external crystals)?
- Is the location of data corruption from the receiver and transmitter (use of CRC)?
- Is the communication module set up correctly (correct GPIO, correct clock settings, and correct protocol)?
- Are the pins connected correctly between receiver and transmitter (contacts, cable length, swapped TX and RX lines, and level shifters, if used)?

Measurement does not work

- Are the inputs signals checked to be as expected (use a scope to probe inputs)?
- Is the measurement MCU in the right mode to perform the measurement (are the modules activated and configured properly)?
- Are the references (clock and voltage) correct?

Draws too high current

- Are all unused ports initialized and properly terminated?
- Is the debug probe still connected during current measurement?
- Does the current disappear after a reset or power cycle?
- Was the device or application current consumption measured?

Reset related problems

- Can the reset event be linked to any external event (for example, supply dip or overvoltage)?
- Does a specific part of the software trigger a reset (can it be traced back to a dedicated part of software; for example, a specific interrupt or a specific loop)?

Hang-up related problems (part is stuck)

- Does the device execute in a code specific endless loop? Can this be checked using "attach on running target" or at port toggles to these loops?

  More details how to use "attach to running target" in CCS and IAR can be found in following documents:

  – Advanced Debugging Using the Enhanced Emulation Module (EEM) With Code Composer Studio™ IDE
  – Advanced Debugging Using the Enhanced Emulation Module (EEM) With IAR Embedded Workbench® IDE
- Is the CPU clock available? Can it be routed to an external pin?
- What is the current consumption? Could it be that the device is in a low-power mode or in a latch-up condition (high current in the range of several to tens of milliamps)?
- Is the device supposed to be waking up from sleep mode or progressing on an interrupt? Is that interrupt enabled and the interrupt condition actually occurring?

Memory related problems

- Which memory locations changed? How did the data change (expected value compared to what is observed)?
- Which data have changed (program code or data values) and which logical transition can be observed?
- Are there any functions inside the application code that might modify memory unintentionally?
- For FRAM devices, it is important to determine if the Memory Protection Unit (MPU) is enabled and configured properly.

3.1 High Current Consumption

A scenario customers often have to deal with on a low-power application is high current consumption. Even if the final issue might be silicon-related and complicated, the following simple issues can each cause high consumption on their own.

1. Check if all unused GPIOs are configured and not floating (configure to output low or high, or use internal pullup or pulldown resistors if switched to input). Improperly terminated GPIOs can cause unreproducible high-current scenarios that take a long time to find, because the symptoms can change with device and environmental conditions like temperature or humidity (see the section *Connection of Unused Pins* in each family user's guide or technical reference manual).
2. Check if input signals are causing crosscurrents on the Schmitt-trigger circuits. The level of any GPIO input must be in the range of DVSS or DVCC ±300 mV as described in the data sheet. If a GPIO is used for an analog function (for example, ADC input), this does not apply.
3. To check if the chip draws high current due to damage, program a standard LPM3 or LPM4 example from ti.com or the CCS Resource Explorer (also considering the application-specific GPIO settings) to see if it draws high current.

The preceding points help to determine if the device executes unexpected code or uses an unexpected CPU frequency during current measurements. This can be found by using the "attach on running target" function of the IDE, available in CCS or IAR. Alternatively, simple port toggles on critical software points (for example, before entering LPM) can help to find out what is wrong.

3.2 JTAG Communication Issues

In this category, end users often report that a device cannot be accessed or programmed using different kind of tools. In most cases, this high-level description is not specific enough and, therefore, TI strongly recommends checking the following items:

1. Is the JTAG security mechanism enabled?
   a. On F1xx, F2xx, F3xx, and F4xx devices, the JTAG fuse can be blown. Measuring the fuse sense current according the data sheet (see Figure 1) gives a good indication if the security mechanism was enabled.

JTAG fuse check mode

MSP430 devices that have the fuse on the TDI/TCLK terminal have a fuse check mode that tests the continuity of the fuse the first time the JTAG port is accessed after a power-on reset (POR). When activated, a fuse check current, I_{TF}, of 1 mA at 3 V, 2.5 mA at 5 V can flow from the TDI/TCLK pin to ground if the fuse is not burned. Care must be taken to avoid accidentally activating the fuse check mode and increasing overall system power consumption.

Activation of the fuse check mode occurs with the first negative edge on the TMS pin after power up or if the TMS is being held low during power up. The second positive edge on the TMS pin deactivates the fuse check mode. After deactivation, the fuse check mode remains inactive until another POR occurs. After each POR the fuse check mode has the potential to be activated.

The fuse check current will only flow when the fuse check mode is active and the TMS pin is in a low state (see Figure 16). Therefore, the additional current flow can be prevented by holding the TMS pin high (default condition).

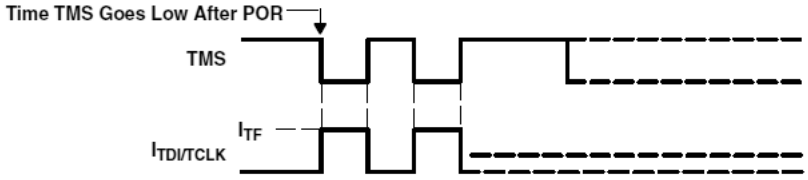

Figure 16. Fuse Check Mode Current: MSP430F13x, MSP430F14x(1)

Figure 1. JTAG Fuse Check Mode of F149

   b. On F5xx and F6xx devices, the JTAG lock is controlled by a signature in the information memory at address 0x17FC to 0x17FF. Check if the application overwrites the bootloader (BSL) memory or if the JTAG lock signature is corrupted. Alternatively the BSL interface can be used to unlock the JTAG signature. The best way is to use the simple BSL Scripter, which comes with ready-to-use examples.
   c. On FRxx devices, the JTAG lock signature is located before the interrupt vectors. Check if stack overflows or user code might have unintentionally written to these locations. Details can be found in the family user's guide in the *JTAG and SBW Lock Mechanism Using the Electronic Fuse* section.
   d. On MSP432™ MCUs, a boot override sequence triggers the JTAG/SWD lock mechanism. Details can be found in the *Boot Overrides* section of the technical reference manual.

In all cases, TI strongly recommends checking MSP430™ Programming With the JTAG Interface for device-specific lock mechanism details.

Check the device pins for shorts to DVCC and DVSS to ensure no physical damage has appeared on the JTAG interface pins.

Having information about the JTAG security fuse and physical conditions of the interface pins helps to further troubleshoot access problems. Also see the MSP430™ Hardware Tools User's Guide for details on JTAG connections.

In addition to the device-related items, also check for any contact issues or wiring problems between the target application and the programmer adapter. TI strongly recommends recording the used JTAG signals in 4-wire or SBW with a digital oscilloscope and comparing good samples to the suspect samples.

2. Which software and hardware tools were used?
   a. It is essential to understand which tools were used. The GUI version and the MSP debug stack version are essential.
   b. The interface used for device programming (4-wire JTAG, SBW, or BSL) is also important. TI recommends trying a different interface to determine if one can be isolated as the source of the issue.
   c. Make sure that the connection between the tool and the device is EXACTLY as recommended (including correct connections and correct capacitance).
2. Tracing the communication with a digital oscilloscope and comparing communication instances that function correctly with ones that do not helps to determine if contact issues are causing the problems or if noise is disturbing the JTAG communication.
3. Has any code been loaded in the part before? Could there be bad code causing the part to reset or preventing JTAG access (try erasing the device using the BSL)?

3.3 Serial Interface Communication Issues

Serial communication is part of a large number of applications, and in such cases it can vary from UART, SPI, I^2C, software-UART, or self-made bit banging interfaces. There are some essentials to verify to determine why a serial communication does not work as expected.

One of the main sources is that the clock source is not correct or disturbed. Especially when the low-frequency oscillator (LFO) is used as clock source. For details, see MSP430™ 32-kHz Crystal Oscillators. This could vary from glitches or drops on the clock up to complete miss of the clock. Checking the clock behavior can provide a good understanding why communication might fail.

In addition to the clock, it is essential to measure the communication signal with an appropriate digital oscilloscope, so that you can attempt to trigger the failing state and compare it with a known-good reference state. Signal disturbances and phase shifts can be detected easily. In addition, the communication can be checked according the protocol in use, and anomalies can be found during inspection. Using a logic analyzer instead of scope might not be sufficient, because glitches and drops might not be seen.

3.4 Start-up Related Cases

There could be several reasons why a device might not start, but there are simple troubleshooting methodologies to quickly find out what might happen.

The more simple devices like the MSP430F1xx to MSP430F4xx devices require different troubleshooting from the more advanced devices like MSP430F5xx or MSP430FRxx, which have multiple power domains.

For the MSP430F1xx to MSP430F4xx devices, the power-up or start-up sequence is quite simple. Power is applied; some voltage supervisors or brownout circuits check the voltage and release the device at a safe voltage. Therefore it is always important to verify that the first instruction is executed. This can be checked by adding a port toggle at the beginning of the code. For C-code projects, TI recommends adding this toggle in the C-start-up code to ensure that the start-up issue is not occurring during low-level initialization. This process is described in the MSP430 Optimizing C/C++ Compiler User's Guide in the chapter *System Initialization*. Also check if the RESET vector stored at address 0xFFFE is set to the expected value (entry point to the program). If this initial port toggle cannot be reached, the problem is likely to be hardware related. Check the supply for high current consumption and check if the supply voltage timing complies with the data sheet specification. It is important to consider the supply ramp specification in the BOR section of the data sheet. Figure 2 shows that the functionality of the whole BOR circuitry is only ensured when DVCC ramps faster than 3 V/s.

The frequency vs supply voltage specification must also be considered, especially on parts without SVS. A common issue is that users increase the CPU frequency during supply ramp without considering that the DVCC has not reached the appropriate level. This leads to a frequency vs supply voltage violation, which can cause unpredictable device behavior due to overclocking.

POR/Brownout Reset (BOR)(1)(2)

over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER		TEST CONDITIONS	V_{CC}	MIN	TYP	MAX	UNIT
$V_{CC(start)}$	See Figure 9	$dV_{CC}/dt \le 3$ V/s			0.7 × $V_{(B\_IT-)}$		V
$V_{(B\_IT-)}$	See Figure 9 through Figure 11	$dV_{CC}/dt \le 3$ V/s				1.71	V
$V_{hys(B\_IT-)}$	See Figure 9	$dV_{CC}/dt \le 3$ V/s		70	130	210	mV
$t_{d(BOR)}$	See Figure 9					2000	µs
$t_{(reset)}$	Pulse length needed at $\overline{RST}$/NMI pin to accepted reset internally		2.2 V/3 V	2			µs

(1) The current consumption of the brownout module is already included in the I_{CC} current consumption data. The voltage level $V_{(B\_IT-)} + V_{hys(B\_IT-)}$ is ≤ 1.8 V.
(2) During power up, the CPU begins code execution following a period of $t_{d(BOR)}$ after $V_{CC} = V_{(B\_IT-)} + V_{hys(B\_IT-)}$. The default DCO settings must not be changed until $V_{CC} \ge V_{CC(min)}$, where $V_{CC(min)}$ is the minimum supply voltage for the desired operating frequency.

Figure 2. POR/BOR Reset Specification of F2132

Such simple debug efforts provide many details for the vendor to judge if the specification was violated.

On the more complex devices like F5xx and F6xx with different power domains, you must observe not only the supply ramp but also the behavior of the internal LDOs. This behavior provides information about what might go wrong internally. Again, it must be emphasized that the correct frequency must be selected for the LDO voltage and external DVCC. Figure 3 shows an example of the F6638 specification for frequency vs supply voltage.

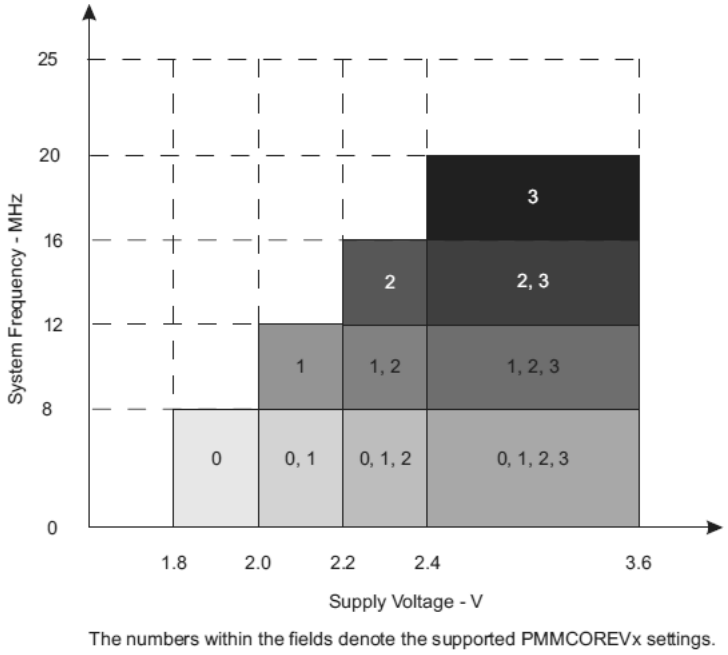

Figure 5-1. Frequency vs Supply Voltage

Figure 3. Frequency vs Supply Voltage Specification of F6638

For example, it is essential to know how the Vcore pin behaves during start-up if the customer increases the Vcore level. Is the Vcore increased step by step during an appropriate DVCC level as specified in the data sheet? Are the delays of the PMM module considered during Vcore increase? In general, TI recommends using the MSP Driver Library functions, which manage all the details on the software level.

In both cases, it is essential to ensure the data sheet specifications for the maximum operating conditions are met (see Figure 4). A scenario quite often seen is that the device is powered up on DVCC after another voltage was applied to a GPIO pin. This can lead to supply through the ESD rails resulting in unpredictable device start-up. More details on this are available in ESD Diode Current Specification.

5.1 Absolute Maximum Ratings(1)

over operating free-air temperature range (unless otherwise noted)

	MIN	MAX	UNIT
Voltage applied at DVCC and AVCC pins to V_{SS}	–0.3	4.1	V
Voltage difference between DVCC and AVCC pins(2)		±0.3	V
Voltage applied to any pin (3)	–0.3	V_{CC} + 0.3 V (4.1 Max)	V
Diode current at any device pin		±2	mA
Storage temperature, T_{stg}(4)	–40	125	°C

(1) Stresses beyond those listed under *Absolute Maximum Ratings* may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under *Recommended Operating Conditions* is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.
(2) Voltage differences between DVCC and AVCC exceeding the specified limits may cause malfunction of the device including erroneous writes to RAM and FRAM.
(3) All voltages referenced to V_{SS}.
(4) Higher temperature may be applied during board soldering according to the current JEDEC J-STD-020 specification with peak reflow temperatures not higher than classified on the device label on the shipping boxes or reels.

Figure 4. Absolute Maximum Ratings of FR6989

3.5 Code Execution or Hang-up Problems

For hang-up related items there are some key items which should be checked to get a better picture what is going wrong.

1. Clocks

   Observe if the system clock (MCLK) or other required clock for modules like the Timer or ADC is present to ensure logic can work. The simplest way is to switch the clocks to the corresponding GPIO pins in their alternate function and measure them with a digital oscilloscope.

2. Supply voltages

   Check if the supply voltages in the 3-V domain or internally regulated voltages (for example, Vcore) are present and do not show unexpected drops or spikes that violate the frequency vs supply voltage specification. The best way to do this is to probe these signals using a digital oscilloscope with a sufficient bandwidth and memory depth.

3. Software

   Are there any software loops inside application code where the CPU is polling for certain flags or waiting for input signals?

   a. The OFIFG flag is favorite candidate because the device will stay in this loop if, for example, the external crystal is damaged or not connected.

   b. Sometimes software hangs in an ADC loop waiting for a certain threshold value that never can be reached, because an external sensor is not working as expected.

4. Current consumption

   The current consumption also gives insight what the device is doing. If the current is very low and comparable with typical low-power mode currents, the device may be staying in sleep mode and not waking up.

   a. In this case, it could be that expected interrupts are not fired due to clock or input related malfunction.

   If the current consumption is unexpectedly high, physical damage may be preventing the device from operating as expected or may be causing a latch-up scenario. In such a case, a power cycle can help to differentiated between permanent damage (high current will stay) or latch-up. For TI recommendations, see MSP430™ System-Level ESD Considerations.

3.6 Memory-Related Problems

For memory-related problems, it is essential to differentiate by memory technology. Three main types of memories are used in MSP families:

1. RAM
2. Flash
3. FRAM

RAM is a volatile memory that loses its information during a power loss. RAM is mainly used for data storage or energy-saving code execution. Typical scenarios where RAM might be corrupted include stack overflow, access violations to data due to incorrect indexing of pointers between functions, and power losses. Short drops or spikes in the supply can lead to temporary frequency vs supply voltage violations and cause code execution problems. Most of these issue modes are introduced by incorrectly written software.

Flash is a nonvolatile memory that does not lose information during power cycling. However, it needs more energy to store the logical information when compared to the other memory technologies in MSP devices. For more details when debugging flash-related problems, see Debugging Flash Issues on the MSP430™ Family of Microcontrollers.

With respect to flash memory, it is always important to determine what changed in the memory. Was calibration memory or application data changed, or was program code changed? If the change was in a part of the memory where the application resides, the probability is that something went wrong during erase or programming. The following key items should be checked in the application during unexpected flash modification:

1. Is supply voltage within specification? Are there spikes or drops on the supply during flash erase or programing?
2. Are the frequency specifications met for flash program during erase or programing?
3. Are there any unexpected resets during flash erase or programing?

For further troubleshooting, the logical information change of the affected flash cells should also be checked. If only changes from a logical 0 to a logical 1 are seen, an erase cycle may have been applied. If there are always changes from logical 1 to 0, something may ben gone wrong during programming. These simple checks might help to identify the software functions related to the behavior.

FRAM is volatile memory that combines the advantages from RAM and flash. It does not require high energy to store data and has very high endurance reliability. For more details, visit MSP430FRxx FRAM Microcontrollers. Typical misuse of FRAM includes unintentional writes caused from stack violations or incorrectly handled function pointers. These unintentional writes can directly change the program code if it is not protected using the Memory Protection Unit. For additional information, see MSP430 FRAM Technology – How To and Best Practices. FRAM is different from flash memory, which requires a dedicated unlock-and-write sequence to store information. Another important factor to consider for FRAM is the frequency specification in the data sheet.

Many of the analytical aspects for memory-related issues are the same for all memory technologies:

1. Which memory locations change? The best practice is to compare memory after failure to a reference memory dump to get all failing locations.
2. What was the logical change for the identified locations (0 to 1 or 1 to 0)?
3. Was supply voltage and frequency within specification? This can be checked by observing supply voltage, regulated voltages, and frequency with a digital oscilloscope.
4. Is the behavior temperature depended?
5. Is the behavior reproducible under certain conditions?
6. What the unexpected change in the identified locations (data or program code)?

4 Conclusion

Even with the strict quality system at Texas Instruments that is in place to prevent malfunctions at the customer site, it is possible that silicon-related issues appear. Therefore, it is essential to get as much as information as possible from the developer or manufacturer about the experienced issue. An efficient and effective electrical issue analysis can be done, and the situation can be resolved as efficiently as possible, only if a good level of collaboration between the end user and the vendor is achieved. However, this document is also in place to give developers some guidance to debug and troubleshoot MCU-related malfunctions and learn how to avoid introducing issues by operating the semiconductor components out of specification.

TI strongly recommends that all users read the official documents (particularly the errata sheet, the data sheet, and the device family user's guide or technical reference manual) to ensure that the conditions specified for device operation are met.

All users can also leverage the engineering community on theTI E2E™ Community to search for similar application-related behaviors experienced by other users and more quickly solve new issues. This forum is actively supported by experienced TI employees in addition to allowing interaction between users.

Table 1 lists other documents that describe certain modules or functions in more detail.

Table 1. Module-Specific Documents

Module or Function	Available Document
BSL Scripter	Bootloader (BSL) Scripter
eUSCI and USCI	Solutions to Common eUSCI and USCI Serial Communication Issues on MSP430™ MCUs
Compiler	MSP430™ Optimizing C/C++ Compiler – User's Guide
Debug	MSP Debugger's Guide
Driver Lib	MSP Driver Library
ESD	MSP430™ System-Level ESD Considerations
ESD	ESD Diode Current Specification
Flash	Debugging Flash Issues on the MSP430™ Family of Microcontrollers
FRAM	MSP430™ FRAM Technology – How To and Best Practices
JTAG	MSP430™ Programming with the JTAG interface
JTAG and tools	MSP430™ Hardware Tools User's Guide
Low-frequency oscillator	MSP430™ 32-kHz Crystal Oscillators

5 References

1. MSP430F13x, MSP430F14x, MSP430F14x1 Mixed-Signal Microcontrollers
2. MSP430F21x2 Mixed-Signal Microcontrollers
3. MSP430F663x Mixed-Signal Microcontrollers
4. MSP430FR698x(1), MSP430FR598x(1) Mixed-Signal Microcontrollers

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