Tiva™ C Series TM4C129x Microcontrollers Silicon Revisions 1 and 2

Silicon Errata

Literature Number: SPMZ850A
October 2013–Revised October 2013

Contents

Tiva™ C Series TM4C129x Microcontrollers Silicon Revisions 1 and 2

1 Introduction

This document describes known exceptions to the functional specifications for all of the Tiva™ C Series TM4C129x microcontrollers. Note that some features are not available on all devices in the series, so not all errata may apply to your device. See your device-specific data sheet for more details.

For details on ARM® Cortex™-M4F CPU advisories, see the *ARM Core Cortex™-M4 (AT520) and Cortex-M4F (AT521) Errata Notice* (literature number: SPMZ637).

2 Device Nomenclature

To designate the stages in the product development cycle, TI assigns prefixes to the part numbers of all microcontroller (MCU) devices. Each Tiva C series family member has one of two prefixes: XM4C or TM4C (for example, **XM4C**129XNCPDTI). These prefixes represent evolutionary stages of product development from engineering prototypes (XM4C) through fully qualified production devices (TM4C).

Device development evolutionary flow:

XM4C — Experimental device that is not necessarily representative of the final device's electrical specifications and may not use production assembly flow.

TM4C — Production version of the silicon die that is fully qualified.

XM4C devices are shipped against the following disclaimer:

"Developmental product is intended for internal evaluation purposes."

TM4C devices have been characterized fully, and the quality and reliability of the device have been demonstrated fully. TI's standard warranty applies.

Predictions show that prototype devices (XM4C) have a greater failure rate than the standard production devices. Texas Instruments recommends that these devices not be used in any production system because their expected end-use failure rate still is undefined. Only qualified production devices are to be used.

3 Device Markings

Figure 1 shows an example of the Tiva™ C Series TM4C129x microcontroller package symbolization.

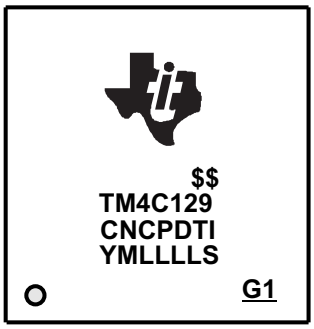

Figure 1. Example of Device Part Markings

This identifying number contains the following information:

- **Lines 1 and 5:** Internal tracking numbers
- **Lines 2 and 3:** Part number

For example, TM4C129C on the second line followed by NCPDTI on the third line indicates orderable part number TM4C129CNCPDTI. Note that the first letter in the part number indicates the product status. A T indicates the part is fully qualified and released to production; an X indicates the part is experimental (pre-production) and requires a waiver. Pre-production parts also include a revision number in the part number, for example, **X**M4C129C followed by NCPDTI**2** indicates revision 2. Production parts do not include a revision number in the part number. The **DID0** register identifies the version of the microcontroller, as shown in Table 1. The MAJOR and MINOR bit fields indicate the die revision number. Combined, the MAJOR and MINOR bit fields indicate the TM4C129x microcontroller silicon revision number.

Table 1. Tiva™ C Series TM4C129x Silicon Revision Codes

MAJOR Bit Field Value	MINOR Bit Field Value	Die Revision	Silicon Revision
0x0	0x0	A0	1
0x0	0x1	A1	2

- **Line 4:** Date code The first two characters on the fourth line indicate the date code, followed by internal tracking numbers. The two-digit date code YM indicates the last digit of the year, then the month. For example, a 34 for the first two digits of the fourth line indicates a date code of April 2013.

4 Advisory to Silicon Revision Correlation

Table 2. Advisory to Silicon Revision Matrix

Advisory Number	Advisory Title	Silicon Revision(s) Affected	
		1	2
ADC			
ADC#09	First two ADC Samples From the Internal Temperature Sensor Must be Ignored	X	X
ADC#13	A Glitch can Occur on pin PE3 When Using any ADC Analog Input Channel to Sample	X	X
ADC#14	The First two ADC Samples may be Incorrect	X	X
ADC#15	ADC Global Synchronization does not function	X	
CRC			
CRC#01	Any Data Read From a Non-CRC Register After Accessing the CRCRSLTPP Register is Incorrect	X	
DMA			
DMA#02	µDMA Data may be Corrupted if Transferred or Received While Entering or Exiting Deep-Sleep Mode	X	X
ELEC			
ELEC#02	V_{BAT} Supply pin may be Damaged if the pin Voltage Ramps Faster Than 0.7 V/µs	X	X
EPI			
EPI#01	Data Reads can be Corrupted when the Code Address Space in the EPI Module is Used	X	X
Ethernet			
ETH#01	The LED Polarity Bit in the Ethernet MAC Clock Configuration Register Does Not Function	X	
GPIO			
GPIO#03	GPIO Pins may Glitch on Power up	X	
General-Purpose Timers			
GPTM#09	General-Purpose Timers do not Synchronize When Configured for RTC Mode	X	X
GPTM#12	The GPTMPP Register Does not Correctly Indicate Alternate Clock Capability	X	
GPTM#13	Reading the GPTMTnV, the GPTMTnR, or the GPTMRTCPD Registers may Return Incorrect Values When Using an Alternate Clock Source	X	X
GPTM#14	General-Purpose Timer ADC and µDMA Triggers may not be Captured in Certain Modes When Using PIOSC	X	
GPTM#15	Counter Does not Immediately Clear to 0 When MATCH is Reached In Edge Count Up Mode	X	X
GPTM#16	Special Configuration is Required when Operating the GPTM in 32-bit Mode with the Alternate Clock Source	X	
GPTM#17	The GPTMSYNC Register Bits Must be Manually Cleared when Using an Alternate Clock Source	X	
Hibernation			
HIB#10	If MEMCLR is set to a Non-Zero Value, a Tamper Event may not Clear all of the Bits in the HIBDATA Register	X	X
HIB#12	Tamper Logging Failure on XOSC Fail Event	X	
HIB#13	Tamper Events may be Missed in log	X	
HIB#15	The VDDFAIL Interrupt bit in the HIBIC Register is not Properly Cleared	X	
HIB#16	Application Code May Miss New Tamper Event During Clear	X	X
HIB#17	$\overline{\text{WAKE}}$ Cannot be Used to Wake From Hibernate Mode	X[(1)]	
HIB#18	Can get two Matches per day in Calendar Mode	X	X
LCD			
LCD#01	LIDD-Mode DMA Transactions in the LCD Controller Cause the Microcontroller to Become Unresponsive	X	X
LCD#02	LCD DMA FIFO Underflow Interrupt Occurs When EPI is Mapped to an External SDRAM With an Address That is not 0x1000.0000	X	X

(1) Applicable to devices with date codes earlier than 0x38 (August 2013).

Table 2. Advisory to Silicon Revision Matrix (continued)

Advisory Number	Advisory Title	Silicon Revision(s) Affected	
		1	2
LCD#03	LCD Module Does not Restart if an Underflow Occurs	X	X
	Memory		
MEM#03	EEPROM Data May be Corrupted if an EEPROM Write or Erase is Interrupted	X	
MEM#09	ROM_SysCtlClockFreqSet() does not Properly Configure MOSC	X	
ONEWIRE			
ONEWIRE#01	A Delay is Needed for the 1-Wire Receive Configuration	X	
PWM			
PWM#01	Under Certain Circumstances, the PWM Load Interrupt is Triggered as Soon as the PWM is Enabled	X	
PWM#02	Setting the PWMSYNC Bits May Not Synchronize the PWM Counters if PWMDIV is Used	X	
PWM#03	The PWM Generators May Not Generate Interrupts or ADC Triggers	X	
QEI			
QEI#01	When Using the Index Pulse to Reset the Counter, a Specific Initial Condition in the QEI Module Causes the Direction for the First Count to be Misread	X	X
SSI			
SSI#03	SSI1 can Only be Used in Legacy Mode	X	X
SSI#04	The First Byte Sent by the QSSI in Master Mode is Incorrect when Using the Alternate Clock	X	
SSI#05	Bus Contention in Bi- and Quad-Mode of SSI	X	X
System Control			
SYSCTL#03	The MOSC Verification Circuit Does not Detect a Loss of Clock After the Clock has been Successfully Operating	X	X
SYSCTL#09	Some Devices may not Start Properly During Power Up if V_{DDC} is Decaying Between 200 and 100 mV When Power is Reapplied	X	
SYSCTL#12	MOSC Does not Power Down in Deep-Sleep when it is not the Deep-Sleep Clock Source	X	
SYSCTL#13	The NMIC register does not indicate NMI sources when read	X	
SYSCTL#14	Power Consumption is Higher When MOSC is Used in Single-Ended Mode	X	
SYSCTL#15	Watchdog Reset Improperly Updates the NMIC Register	X	
UART			
UART#01	When UART SIR Mode is Enabled, µDMA Burst Transfer Does not Occur	X	X
USB			
USB#02	USB Controller Sends EOP at end of Device Remote Wake-Up	X	
USB#03	Any Data Read From a Non-USB Register After Accessing the USBPP, USBPC, or the USBCC Register is Incorrect	X	
Watchdog Timers			
WDT#08	Reading the WDTVALUE Register may Return Incorrect Values When Using Watchdog Timer 1	X	X

5 Known Design Exceptions to Functional Specifications

Table 3. Advisory List

Table 3. Advisory List (continued)

ADC#09 *First two ADC Samples From the Internal Temperature Sensor Must be Ignored*

Revision(s) Affected: 1 and 2.

Description: The analog source resistance (R_S) to the ADC from the internal temperature sensor exceeds the specified amount of 500Ω. This causes a settling time requirement that is longer than the sampling interval to the converter.

Workaround(s): A small delay must be inserted between samples to allow enough time for the voltage to settle. Configure the respective ADC Sample Sequence n Sample and Hold Time (ADCSSTSH) register for a sample and hold width of at least 16 ADC clocks (TSHx = 0x4).

Alternatively, three consecutive samples from the same channel must be taken to accurately sample the internal temperature sensor using the ADC. The first two consecutive samples should be discarded and the third sample can be kept. These consecutive samples cannot be interrupted by sampling another channel.

ADC#13 *A Glitch can Occur on pin PE3 When Using any ADC Analog Input Channel to Sample*

Revision(s) Affected: 1 and 2.

Description A glitch may occur on PE3 when using any ADC analog input channel (AINx) to sample. This glitch can occur when PE3 is configured as a GPIO input or as a GPIO open drain and happens at the end of the ADC conversion. These glitches will not affect analog measurements on PE3 when configured as AIN0 as long as the specified source resistance is met.

Workaround(s) A 1kΩ external pull-up or pull-down on PE3 will help to minimize the magnitude of the glitch to 200 mV or less.

ADC#15 *ADC Global Synchronization Does not Function*

Revision(s) Affected: 1 only.

Description: The SYNCWAIT and GSYNC bits in the ADC Processor Sample Sequence Initiate (ADCPSSI) register are set to allow sample sequencers in ADC0 and ADC1 to be synchronized. These bits do not function.

Workaround(s): None

CRC#01 *Any Data Read From a Non-CRC Register After Accessing the CRCRSLTPP Register is Incorrect*

Revision(s) Affected: 1 only.

Description: A CRC Post Processing Result (CRCRSLTPP) register read followed by a read from any other non-CRC register on the AHB results in incorrect data in the non-CRC register.

Workaround(s): Read any CRC register after reading the CRCRSLTPP register and before reading any other register on the AHB. To determine which modules are on the AHB, refer to Figure 1-1 in the data sheet.

DMA#02 *µDMA Data may be Corrupted if Transferred or Received While Entering or Exiting Deep-Sleep Mode*

Revision(s) Affected: 1 and 2.

Description: Transferred or received data using the µDMA from either the UART or the SSI peripherals may get corrupted when entering Deep-Sleep mode from Run mode or exiting Deep-Sleep mode to Run mode if the Run mode clock configuration is not the same as the Deep-Sleep mode clock configuration.

Workaround(s): Program the Run mode clock configuration to match the Deep-Sleep mode clock configuration right before entering Deep-Sleep mode.

ELEC#02 *V_{BAT} Supply pin may be Damaged if the pin Voltage Ramps Faster Than 0.7 V/µs*

Revision(s) Affected: 1 and 2.

Description The V_{BAT} supply pin may be damaged if the pin voltage ramps faster than 0.7 V/µs. Fast V_{BAT} ramps are a concern when a battery is being connected or the V_{BAT} supply is hard switched.

Workaround(s) An RC circuit as shown should be added to the V_{BAT} pin to prevent the damage. The R_1 and C_1 should be placed close to the microcontroller for best protection. In systems that do not require Hibernate when the VDD supply is off, the V_{BAT} pin should be tied to the VDD supply, which typically ramps at a rate slower than 0.7 V/µs. The R1 and C1 components are not required for a V_{BAT} supply ramp less than 0.7 V/µs.

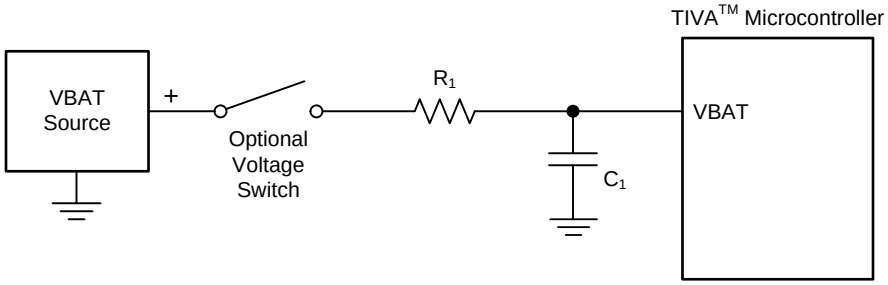

Figure 2. RC Circuit

EPI#01 *Data Reads can be Corrupted when the Code Address Space in the EPI Module is Used*

Revision(s) Affected: 1 and 2.

Description: The external code address space at address 0x1000.0000 is specified for the EPI module using the ECSZ and ECADR fields in the EPI Address Map (EPIADDRMAP). However, data reads can be corrupted when using this address space.

Workaround(s): Code cannot be executed from the 0x1000.0000 address space. The EPI address spaces at 0x6000.0000 and 0x8000.0000 can be used instead.

In addition, when reading from EPI memory mapped to the code address space at 0x1000.0000, replace direct EPI memory reads via pointers with calls to the EPIWorkaroundWordRead(), EPIWorkaroundHWordRead() or EPIWorkaroundByteRead() functions depending on the data size for the read operation. Similarly, when writing to the EPI code address space, replace direct writes with calls to the EPIWorkaroundWordWrite(), EPIWorkaroundHWordWrite() or EPIWorkaroundByteWrite() functions. These APIs are new and can be found in Appendix 2. For Keil, IAR, GCC, and Code Bench, these functions are defined as inline functions in the epi.h file in *C:\ti\TivaWare_C_Series-2.0\driverlib*. For CCS, which doesn't support this structure, these should be added to a new file placed in the *\driverlib* directory called epi_workaround_ccs.s, and this file should be added to the project. Note that the new DriverLib APIs and the CCS file mentioned in Appendix 2 are included in TivaWare release 2.0.1 and later releases.

ETH#01 *The LED Polarity Bit in the Ethernet MAC Clock Configuration Register Does Not Function*

Revision(s) Affected: 1 only.

Description: The LED Polarity bit (POL) in the Ethernet MAC Clock Configuration Register (EMACCC) does not function. The LED outputs coming from the Ethernet PHY are active high and their polarity cannot be configured.

Workaround(s): None.

GPIO#03 *GPIO Pins may Glitch on Power up*

Revision(s) Affected: 1 only.

Description: Some devices may drive the GPIOs to ground during power up for less than 5 µs when V_{DDC} is ~ 400-500 mV.

Workaround(s): None.

GPTM#09 *General-Purpose Timers do not Synchronize When Configured for RTC Mode*

Revision(s) Affected: 1 and 2.

Description: When attempting to synchronize the General-Purpose Timers using the GPTM Synchronize (GPTMSYNC) register, they do not synchronize if any of the timers are configured for RTC mode. This applies even if the timers are using the alternate clock.

Workaround(s): None.

GPTM#12 *The GPTMPP Register Does not Correctly Indicate Alternate Clock Capability*

Revision(s) Affected: 1 only.

Description The ALTCLK bit in the GPTM Peripheral Properties (GPTMPP) register reads as 0x0, the ALTCLK source is available to the timer. When the ALTCLK bit reads as 0x1, the ALTCLK source is not available to the timer. This is opposite from the intended implementation of these bits. The ALTCLK bit should read as 0x0 when the ALTCLK source is not available for use by the timer and the ALTCLK bit should read as 0x1 when the ALTCLK source is available for use by the time.

Workaround(s) None.

GPTM#13 *Reading the GPTMTnV, the GPTMTnR, or the GPTMRTCPD Registers may Return Incorrect Values When Using an Alternate Clock Source*

Revision(s) Affected: 1 and 2.

Description Incorrect values may be read from the GPTM Timer n Value (GPTMTnV), the GPTM Timer n (GPTMTnR), and the GPTM RTC Predivide (GPTMRTCPD) registers when using an alternate clock source.

Workaround(s) None.

GPTM#14 *General-Purpose Timer ADC and µDMA Triggers may not be Captured in Certain Modes When Using PIOSC*

Revision(s) Affected: 1 only.

Description An ADC or µDMA event that is triggered by the general-purpose timer may not be triggered when the timer is configured to use the alternate clock and the system clock frequency is greater than 17 MHz. This affects the following counting modes:

Mode	Direction	Trigger Affected
Edge Count	Down	Capture Match
One-shot	Up	Time Out
One-shot	Down	Time Out

Workaround(s) None.

GPTM#15 *Counter Does not Immediately Reset to 0 When MATCH is Reached In Edge Count Up Mode*

Revision(s) Affected: 1 and 2.

Description When configured for input edge count mode and count up mode, after counting to the match value, the counter uses one additional edge to reset the timer to 0. As a result, after the first match event, all subsequent match events occur after the programmed number of edge events plus one.

Workaround(s) In software, account for one additional edge in the programmed edge count after the first match interrupt is received.

GPTM#16 *Special Configuration is Required when Operating the GPTM in 32-bit Mode with the Alternate Clock Source*

Revision(s) Affected: 1 only.

Description: When the alternate clock source is selected by setting the ALTCLK bit in the GPTM Clock Configuration (GPTMCC) register and the timer is in 32-bit mode, the GPTM Timer B Match (GPTMTBMATCHR) and GPTM Timer B Interval Load (GPTMTBILR) registers are not writable. As a result, special configuration is required when operating the GPTM in 32-bit mode with the alternate clock source.

Workaround(s): Clear the ALTCLK bit before configuring the GPTMTBMATCHR and GPTMTBILR registers and then set the ALTCLK bit prior to enabling the timer.

GPTM#17 *The GPTMSYNC Register Bits Must be Manually Cleared when Using an Alternate Clock Source*

Revision(s) Affected: 1 only.

Description: If an alternate clock source is configured in the Alternate Clock Configuration (ALTCLKCFG) register and enabled using the GPTM Clock Configuration (GPTMCC) register, the bits in the GPTM Synchronize (GPTMSYNC) register are not cleared automatically after being set.

Workaround(s): When using the bits in the GPTMSYNC register, software must clear the bits prior to setting them for a subsequent update.

HIB#10 *If MEMCLR is set to a Non-Zero Value, a Tamper Event may not Clear all of the Bits in the HIBDATA Register*

Revision(s) Affected: 1 and 2.

Description: If the MEMCLR bit field in the HIB Tamper Control (HIBTPCTL) register is set to a non-zero value, the Hibernation Data (HIBDATA) register may not clear the specified bits. The MEMCLR bit field provides the option to clear all, the upper half, lower half, or none of the Hibernate memory on a tamper event.

Workaround(s): After clearing the tamper event by setting the TPCLR bit, the application should clear the data in the Hibernate memory in the HIBDATA register (write either the upper half, the lower half, or all of the bits to all zeros).

HIB#12 *Tamper Logging Failure on XOSC Fail Event*

Revision(s) Affected: 1 only.

Description: When XOSC failure is created in the Tamper module, the XOSC bit does not get set in HIBTPLOG1, HIBTPLOG3, HIBTPLOG5, or HIBTPLOG7. The XOSC fail can be a result of shorting or grounding one of the XOSC pins or by removing the 32.768-kHz oscillator source. The user will know the XOSC fail has occurred by reading the XOSCFAIL bit in the HIBTPSTAT register. The user may not be able to correlate with an RTC time stamp.

Workaround(s): None.

HIB#13 *Tamper Events may be Missed in log*

Revision(s) Affected: 1 only.

Description: The HIB Tamper module log captures up to four events. Events 1, 2, 3 and the last event are captured. However, events 4 through the last event minus one are overwritten. If an event occurs after the 3rd and then de-asserts before the last entry, the event will be lost.

Workaround(s): Systems that have VDD available can be configured to wake up on a tamper event and clear the log.

HIB#15 *The VDDFAIL Interrupt bit in the HIBIC Register is not Properly Cleared*

Revision(s) Affected: 1 only.

Description: The VDDFAIL bit in the Hibernation Interrupt Clear (HIBIC) register should be cleared by writing a 1 to bit 7 of the register, but it is not cleared.

Workaround(s): To clear the VDDFAIL bit, write a 1 to both bit 7 and the reserved bit 1 of the HIBIC register.

HIB#16 *Application Code May Miss New Tamper Event During Clear*

Revision(s) Affected: 1 and 2.

Description: During the clear of a tamper event, a new tamper event could be missed or the tamper log could be corrupted. The clear of a tamper event starts with the Tamper Clear (TPCLR) bit in the HIB Tamper Control register (HIBTPCTL) being written. The write takes 3 rising edges of the 32.768-kHz clock to complete the clear.

Workaround(s): To prevent missing a tamper event during these three Hibernate clock cycles and restoring the tamper log to its reset state, workaround code must be implemented in the NMI handler as shown in Appendix 1. The new DriverLib APIs mentioned in Appendix 1 are included in TivaWare release 2.0.1 and later releases.

HIB#17 *$\overline{\text{WAKE}}$ Cannot be Used to Wake From Hibernate Mode*

Revision(s) Affected: 1 only. Applicable to devices with date codes before 0x38 (August, 2013). See Section 3 on how to read the date code.

Description: The external $\overline{\text{WAKE}}$ pin cannot be used to wake from Hibernate mode. In addition, the WAKENC bit in the Hibernation Peripheral Properties (HIBPP) register that indicates the presence of the $\overline{\text{WAKE}}$ pin is clear.

Workaround(s): Use any of the following methods of waking from Hibernate mode:

- RTC match wake event
- Low battery wake event
- External $\overline{\text{RST}}$
- GPIO K[7:4]
- Tamper TMPR[3:0]

HIB#18 *Can get two Matches per day in Calendar Mode*

Revision(s) Affected: 1 and 2.

Description: When the CAL24 bit in the Hibernation Calendar Control (HIBCALCTL) register is clear, the RTC counts in 12 hour, AM/PM mode. The AM/PM bit in the Hibernation Calendar Match 0 (HIBCALM0) specifies whether the match should occur in the AM or the PM. However, this bit is ignored when determining if a match is occurring. As a result, an RTC match could occur twice in one day.

Workaround(s): Adjust the match time to 24 hour mode before configuring the HIBCALM0 register and set the CAL24 bit. Alternatively, when the match occurs, check the AM/PM bit in the Hibernation Calendar (HICAL0) register to determine if the match is correct.

LCD#01 *LIDD-Mode DMA Transactions in the LCD Controller Cause the Microcontroller to Become Unresponsive*

Revision(s) Affected: 1 and 2.

Description: Whenever a LIDD-mode DMA transaction is requested by setting the DMAEN bit of the LCD LIDD Control (LCDLIDDCTL) register, after setting the LCD DMA frame buffer n base and ceiling addresses in the LCDDMABAFB and LCDDMACAFB registers, respectively, the next attempt to access any LCD controller register causes the microcontroller and the JTAG connection to become unresponsive. A power-on reset recovers the device.

Workaround(s): The following code must be inserted prior to each new LCD DMA transaction. The code resets the LCD DMA engine. Additional code around the LCDClockReset() function call is required when using LIDD mode. This code temporarily configures the CS signal to the display as a GPIO output to ensure that it remains high during the LCD DMA reset operation. Without this GPIO code, a low glitch will likely occur on the CS signal, which may cause the attached display to operate incorrectly.

```
//
// If using LIDD mode, revert the CS pin to GPIO control.
// Configure the CS pin as a GPIO output, drive the pin high, and
// clear the respective alternate function register bit. This is
// done to prevent glitches on CS during the time the LCD
// controller is reset after each transaction. This step is not
// needed for Raster mode.
//
GPIOPinTypeGPIOOutput(GPIO_PORTJ_BASE, 0x40);
GPIOPinWrite(GPIO_PORTJ_BASE, 0x40, 0x40);
HWREG(GPIO_PORTJ_BASE + GPIO_O_AFSEL) &= ~0x40;
//
// Reset the module (but preserve all register values)
//
LCDClockReset(LCD0_BASE, LCD_CLOCK_MAIN);
//
// If using LIDD mode, set the CS pin back to hardware control.
// This step is not needed for Raster mode.
//
HWREG(GPIO_PORTJ_BASE + GPIO_O_AFSEL) |= 0x40;
```

LCD#02 *LCD DMA FIFO Underflow Interrupt Occurs When EPI is Mapped to an External SDRAM With an Address That is not 0x1000.0000*

Revision(s) Affected: 1 and 2.

Description: If the EPI controller is mapped to allow indirect access to SDRAM where the ECADR bit field is 0x0 and the ERADR bit field is not 0x0 in the EPI Address Map (EPIADDRMAP) register, a DMA FIFO underflow interrupt occurs. This is a result of the LCD being a lower priority than the CPU when they both try to access the SDRAM.

Workaround(s): Map the EPI to use the external code area 0x1000.0000 (ECADR = 0x1). This makes the LCD higher priority than the CPU and prevents this FIFO underflow condition.

LCD#03 *LCD Module Does not Restart if an Underflow Occurs*

Revision(s) Affected: 1 and 2.

Description: The LCD module does not restart if a DMA FIFO underflow interrupt occurs. Writes to the UFLOWRST bit in the LCD Control (LCDCTL) register have no effect.

Workaround(s): If a DMA FIFO underflow interrupt occurs (the FIFOU interrupt bit is set), restart the LCD Raster controller by calling LCDRasterEnable().

MEM#03 *EEPROM Data May be Corrupted if an EEPROM Write is Interrupted*

Revision(s) Affected: 1 and 2 - XM4C devices only. This issue will be fixed for TM4C devices.

Description: Corrupted EEPROM data can occur if an EEPROM write is interrupted with any of the following power events:

- Power failure
- External reset ($\overline{\text{RST}}$) (if configured for a simulated POR sequence in the RESBEHAVCTL register (EXTRES = 0x3))
- Brown-out (BOR) event (if configured for a simulated POR sequence in the RESBEHAVCTL register (BOR = 0x3))
- Watchdog reset (if configured for a simulated POR sequence in the RESBEHAVCTL register (WDOGn = 0x3))

The corrupted EEPROM data that can result from this sequence is not limited to the current word being written. If these events do not apply to your system, then normal EEPROM operation is expected. If a failure occurs, there will not be any indication of the failed erase or corrupted data (for example in the PRETRY and the ERETRY bits in the EEPROM Support Control and Status (EESUPP) register.

Workaround(s): Depending on the system, there are a few potential workarounds:

1. Program the EEPROM only when the device is guaranteed to not have power removed and when a brown-out reset and an external reset will not occur. There are no restrictions on EEPROM reads.
2. Use the Flash memory with application software to store data instead of the EEPROM controller.
3. Limit the number of lifetime EEPROM writes to 7 writes per word.
4. Use an external EEPROM.

MEM#09 *ROM_SysCtlClockFreqSet() Does not Properly Configure MOSC*

Revision(s) Affected: 1 only.

Description: The ROM_SysCtlClockFreqSet() function does not properly configure the MOSC.

Workaround(s): Use the TivaWare function of SysCtlClockFreqSet() in Flash memory.

ONEWIRE#01 *A Delay is Needed for the 1-Wire µDMA Receive Configuration*

Revision(s) Affected: 1 only.

Description: A 1-Wire µDMA receive configuration includes a write to the DMAOP field of the ONEWIREDMA register followed by a write to the ONEWIREDATW register with the value 0xFFFF.FFFF to prime the read operations. If a sufficient delay is not inserted between these two instructions, the write to the ONEWIREDATW register is not recognized and the read does not start.

Workaround(s): After writing to the DMAOP field in the ONEWIREDMA register, insert a delay of at least 187.5 ns (3 PIOSC clock cycles) before writing the ONEWIREDATW register.

PWM#01 *Under Certain Circumstances, the PWM Load Interrupt is Triggered as Soon as the PWM is Enabled*

Revision(s) Affected: 1 only.

Description: A spurious PWM interrupt occurs immediately when the PWM is enabled under the following conditions:

- The PWM Load register contains a nonzero value and
- Either of the PWM Compare registers contains a value less than the value in the PWM Load register and
- PWM interrupts are enabled.

Workaround(s): None

PWM#02 *Setting the PWMSYNC Bits May Not Synchronize the PWM Counters if PWMDIV is Used*

Revision(s) Affected: 1 only.

Description: The bits in the PWM Time Base Sync (PWMSYNC) register are used to synchronize the counters in the PWM generators. The PWMDIV field in the PWM Clock Configuration (PWMCC) register is used to specify a fractional version of the system clock to use for the counters. If the PWMSYNC bits are set when the PWMDIV field is configured to anything other than 0x0, the counters may not be synchronized.

Workaround(s): None.

PWM#03 *The PWM Generators May Not Generate Interrupts or ADC Triggers*

Revision(s) Affected: 1 only.

The PWM Generators May Not Generate Interrupts or ADC Triggers

Description: The PWM generators do not reliably generate interrupts or ADC triggers.

Workaround(s): None.

QEI#01 *When Using the Index Pulse to Reset the Counter, a Specific Initial Condition in the QEI Module Causes the Direction for the First Count to be Misread*

Revision(s) Affected: 1 and 2.

Description: When using the index pulse to reset the counter with the following configuration in the QEI Control (QEICTL) register:

- SIGMODE is 0 indicating quadrature mode
- CAPMODE is 1 indicating both PhA and PhB edges are counted

and the following initial conditions:

- Both PhA and PhB are 0
- The next quadrature state is in the counterclockwise direction

the QEI interprets the state change as an update in the clockwise direction, which results in a position mismatch of 2.

Workaround(s): None.

SSI#03 *SSI1 can Only be Used in Legacy Mode*

Revision(s) Affected: 1 (all modules) and 2 (only SSI1).

Description: Bi-, quad-, and advance-modes of operation do not function correctly on the specified SSI module(s). As a result, any affected module can only be used for legacy operation.

Workaround(s): On revision 2 devices, use SSI0, SSI2, or SSI3 for bi-, quad-, and advance-mode operation. Only use SSI1 for legacy-mode operation.

SSI#04 *The First Byte Sent by the SSI in Master Mode is Incorrect when Using the Alternate Clock*

Revision(s) Affected: 1 only.

Description: When the alternate clock source is selected by setting the ALTCLK bit in the SSI Clock Configuration (SSICC) register and the SSI is in master mode, the first byte transmitted is incorrect.

Workaround(s): Use the system clock for the QSSI when in master mode.

SSI#05 *Bus Contention in Bi- and Quad-Mode of SSI*

Revision(s) Affected: 1 and 2.

Description: When the SSI is configured in Bi- or Quad-mode, and a write to external memory is followed by a read from external memory, bus contention can occur on the SSI data pins.

Workaround(s): Perform a dummy read from memory after a write operation and before a valid read operation. For example:

```
SSIConfigSetExpClk(SSI0_BASE,SysCtlClockFreqSet (),
SSI_FRF_MOTO_MODE_0, SSI_MODE_MASTER, 1000000, 8);
SSIAdvModeSet(SSI0_BASE, SSI_ADV_BI_WRITE);
SSIDataGet(SSI0_BASE, &pui32DataRx[ui32Index]); //write
SSIAdvModeSet(SSI0_BASE, SSI_ADV_BI_READ);
SSIDataGetNonBlocking(SSI0_BASE, ui32Dummy); //dummy read
SSIDataGetNonBlocking(SSI0_BASE, &pui32DataRx[0])); //read
```

Note that if the transfer normally requires any dummy operations, the switch from write to read and the dummy operation above should occur before the normal dummy operations.

SYSCTL#03 *The MOSC Verification Circuit Does not Detect a Loss of Clock After the Clock has been Successfully Operating*

Revision(s) Affected: 1 and 2.

Description: If the MOSC clock source has been powered up and operating correctly and is subsequently removed or flatlines, the MOSC verification circuit does not indicate an error condition.

Workaround(s): Use Watchdog module 1, which runs off of PIOSC, to reset the system if the MOSC fails.

SYSCTL#09 *Some Devices may not Start Properly During Power up*

Revision(s) Affected: 1 only.

Description: In very rare cases, the internal LDOs may not start properly during power up. If the LDOs do not start properly, the device may not begin operating, and VDDC may not reach its specified levels.

Workaround(s): Power cycle the device until the device starts up correctly. This issue has not been seen on devices when the VDD rise time from 0 V to 3.0 V is less than 100 us. However, meeting this condition does not guarantee that the issue will not occur.

SYSCTL#12 *MOSC Does not Power Down in Deep-Sleep when it is not the Deep-Sleep Clock Source*

Revision(s) Affected: 1 only.

Description: The main oscillator (MOSC) continues to run in Deep-Sleep mode if it was used as the system clock source for Run mode. The Deep-Sleep clock source programmed in the DSCLKCFG register is still applied in Deep-Sleep, though the current consumption is about 350 µA higher in Deep-Sleep mode than it would be if the MOSC was disabled.

Workaround(s): Turn off the main oscillator before entering Deep-Sleep mode by switching to an alternate clock source and then setting the PWRDN bit in the Main Oscillator Control (MOSCCTL) register.

SYSCTL#13 *The NMIC Register Does not Indicate NMI Sources when Read*

Revision(s) Affected: 1 only.

Description: The NMI Cause Register (NMIC) register should indicate which possible NMI source caused an NMI - MOSCFAIL, Tamper, WDT, BOR or external NMI signal. However, this register does not function correctly.

Workaround(s): Use alternate sources to determine which possible event caused the NMI:

- MOSCFAIL – check the MOFRIS bit in the Raw Interrupt Status (RIS) register
- Tamper – check the STATE field in the HIB Tamper Status (HIBTPSTAT) register
- WDT – check the WDTRIS bit in the Watchdog Raw Interrupt Status (WDTRIS) register
- BOR – check the BORRIS bit in the Raw Interrupt Status (RIS) register
- External signal – configure the GPIO interrupt registers to trigger an interrupt when the NMI pin is asserted and check the GPIO Raw Interrupt Status (GPIORIS) register

SYSCTL#14 *Power Consumption is Higher When MOSC is Used in Single-Ended Mode*

Revision(s) Affected: 1 only.

Description: The MOSC internal oscillator continues to run, even when a single-ended clock source is attached to OSC0. This issue does not affect proper operation but does result in additional power consumption of up to 3.5 mA.

Workaround(s): None

SYSCTL#15 *Watchdog Reset Improperly Updates the NMIC Register*

Revision(s) Affected: 1 only.

Description: When the watchdog expires and causes a reset, the WDT bit in the NMI Cause (NMIC) register is set, and it should not be. Note that the Reset Cause Register (RESC) is appropriately updated to indicate the watchdog reset.

Workaround(s): After a watchdog reset occurs, manually clear the WDT bit in the NMIC register to remove the erroneous NMI indication.

UART#01 *When UART SIR Mode is Enabled, µDMA Burst Transfer Does not Occur*

Revision(s) Affected: 1 and 2.

Description: If the IrDA Serial Infrared (SIR) mode is enabled in the UART peripheral and the µDMA is mapped to either UARTn RX or UARTn TX and is configured to do a burst transfer, the burst data transfer does not occur.

Workaround(s): Clear the $\overline{\text{SET}}$ bit in the DMA Channel Useburst Set (DMAUSEBURSTSET) register to have the µDMA channel mapped to the UART to respond to single or burst requests to ensure that the data transfer occurs.

USB#02 *USB Controller Sends EOP at end of Device Remote Wake-Up*

Revision(s) Affected: 1 only.

Description: When the USB controller is operating as a Device and is suspended by the Host, and the USB controller issues a remote wake-up, an end of packet (EOP) is sent to the Host at the end of the Device's remote wake-up signal. Although this EOP is not expected, issues related to remote wake-up have not been observed. This does not affect USB certification.

Workaround(s): None.

USB#03 *Any Data Read From a Non-USB Register After Accessing the USBPP, USBPC, or the USBCC Register is Incorrect*

Revision(s) Affected: 1 only.

Description: A read from any of the following registers followed by a read from any other non-USB register on the AHB results in incorrect data in the non-USB register:

- USB Peripheral Properties (USBPP)
- USB Peripheral Configuration (USBPC)
- USB Clock Configuration (USBCC)

Workaround(s): Read any USB register after reading any of the registers in the above list and before reading any other non-USB register on the AHB. To determine which modules are on the AHB, refer to Figure 1-1 in the data sheet.

WDT#08 *Reading the WDTVALUE Register may Return Incorrect Values When Using Watchdog Timer 1*

Revision(s) Affected: 1 and 2.

Description Incorrect values may be read from the Watchdog Value (WDTVALUE) register at the Watchdog Timer 1 base address when using Watchdog Timer 1.

Workaround(s) None.

6 Appendix 1

To address the erratum HIB#16, “Application Code May Miss New Tamper Event During Clear,” the HibernateTamperEventsClear() API must be replaced with the following APIs:

```
HibernateTamperEventsClearNoLock();
HibernateTamperUnLock();
HibernateTamperLock();
```

The API definitions are as follows:

```
//*****************************************************************************
//
//! Clears the tamper feature events without Unlock and Lock.
//!
//! This function is used to clear all tamper events without unlock/locking
//! the tamper control registers, so API HibernateTamperUnLock() should be
//! called before this function, and API HibernateTamperLock() should be
//! called after to ensure that tamper control registers are locked.
//!
//! This function doesn't block until the write is complete.
//! Therefore, care must be taken to ensure the next immediate write will
//! occure only after the write complete bit is set.
//!
//! This function is used to implement a software workaround in NMI interrupt
//! handler to fix an issue when a new tamper event could be missed during
//! the clear of current tamper event.
//!
//! \note The hibernate tamper feature is not available on all Tiva
//! devices.  Please consult the data sheet for the Tiva device that you
//! are using to determine if this feature is available.
//!
//! \return None.
//
//*****************************************************************************
void
HibernateTamperEventsClearNoLock(void)
{
    //
    // Wait for write completion.
    //
    _HibernateWriteComplete();

    //
    //
    // Set the tamper event clear bit.
    //
    HWREG(HIB_TPCTL) |= HIB_TPCTL_TPCLR;

}

//*****************************************************************************
//
//! Unlock temper registers.
//!
//! This function is used to unlock the temper control registers.  This
//! function should be only used before calling API
//! HibernateTamperEventsClearNoLock().
//!
//! \note The hibernate tamper feature is not available on all Tiva
//! devices.  Please consult the data sheet for the Tiva device that you
//! are using to determine if this feature is available.
//!
//! \return None.
//
//*****************************************************************************
void
HibernateTamperUnLock(void)
```

```
{
    //
    // Unlock the tamper registers.
    //
    HWREG(HIB_LOCK) = HIB_LOCK_HIBLOCK_KEY;
    _HibernateWriteComplete();
}
//*****************************************************************************
//
//! Lock temper registers.
//!
//! This function is used to lock the temper control registers.  This
//! function should be used after calling API
//! HibernateTamperEventsClearNoLock().
//!
//! \note The hibernate tamper feature is not available on all Tiva
//! devices.  Please consult the data sheet for the Tiva device that you
//! are using to determine if this feature is available.
//!
//! \return None.
//
//*****************************************************************************
void
HibernateTamperLock(void)
{
    //
    // Wait for write completion.
    //
    _HibernateWriteComplete();

    //
    // Lock the tamper registers.
    //
    HWREG(HIB_LOCK) = 0;
    _HibernateWriteComplete();
}
```

The software workaround must be added in the NMI Handler. The code mainly polls the tamper log entries during the tamper clear synchronization.

An example of an NMI handler with this workaround is shown below, with the block code of the workaround highlighted in RED.

```
static uint32_t g_ui32RTCLog[4];
static uint32_t g_ui32EventLog[4];
//*****************************************************************************
//
// Handles an NMI interrupt generated by a Tamper event.
//
//*****************************************************************************
void
NMITamperEventHandler(void)
{
    uint32_t ui32NMIStatus, ui32TamperStatus;
    uint32_t pui32Buf[3];
    uint8_t ui8Idx, ui8StartIdx;
    bool     bDetectedEventsDuringClear;

    //
    // Get the cause of the NMI event.
    //
    ui32NMIStatus = SysCtlNMIStatus();

    //
    // We should have got the cause of the NMI event from the above function.
    // But in Snowflake RA0 the NMIC register is not set correctly when an
    // event occurs.  So as a work around check if the NMI event is caused by a
```

```
// tamper event and append this to the return value from SysCtlNMIStatus().
// This way only this section can be removed once the bug is fixed in next
// silicon rev.
//
ui32TamperStatus = HibernateTamperStatusGet();
if(ui32TamperStatus & (HIBERNATE_TAMPER_STATUS_EVENT |
                       HIBERNATE_TAMPER_STATUS_EXT_OSC_FAILED))
{
    ui32NMIStatus |= SYSCTL_NMI_TAMPER;
}

//
// Check if SysCtlNMIStatus() returned a valid value.
//
if(ui32NMIStatus)
{

    //
    // Check if the NMI Interrupt is due to a Tamper event.
    //
    if(ui32NMIStatus & SYSCTL_NMI_TAMPER)
    {
        //
        // If the previous NMI event has not been processed by main
        // thread, we need to OR the new event along with the old ones.
        //
        if(g_ui32NMIEvent == 0)
        {
            //
            // Reset variables that used for tamper event.
            //
            g_ui32TamperEventFlag = 0;
            g_ui32TamperRTCLog = 0;

            //
            // Clean the log data for debugging purpose.
            //
            memset(g_ui32RTCLog, 0, (sizeof(g_ui32RTCLog))<<2);
            memset(g_ui32EventLog, 0, (sizeof(g_ui32EventLog))<<2);
        }


        //
        // Log the tamper event data before clearing tamper events.
        //
        for(ui8Idx = 0; ui8Idx< 4; ui8Idx++)
        {
            if(HibernateTamperEventsGet(ui8Idx,
                                        &g_ui32RTCLog[ui8Idx],
                                        &g_ui32EventLog[ui8Idx]))
            {
                //
                // Event in this log entry, store it.
                //
                g_ui32TamperEventFlag |= g_ui32EventLog[ui8Idx];
                g_ui32TamperRTCLog = g_ui32RTCLog[ui8Idx];
            }
            else
            {
                //
                // No event in this log entry. Done checking the logs.
                //
                break;
            }
        }
```

```
//
// Process external oscillator failed event.
//
if(ui32TamperStatus & HIBERNATE_TAMPER_STATUS_EXT_OSC_FAILED)
{
    g_ui32TamperXOSCFailEvent++;
    g_ui32TamperEventFlag |= HIBERNATE_TAMPER_EVENT_EXT_OSC;
    g_ui32TamperRTCLog = HWREG(HIB_TPLOG0);
}
//
// The following block of code is to workaround hardware defect
// which results in missing new tamper events during tamper clear
// synchronization.
//
// There is a window after the application code writes the tamper
// clear where a new tamper event can be missed if the application
// requires more than one tamper event pins detection.
//
// The tamper Clear is synchronized to the hibernate 32kHz clock
// domain.  The clear takes 3 rising edges of the 32KHz clock.
// During this window, new tamper events could be missed.
// A software workaround is to poll the tamper log during the
// tamper event clear synchronization.
//

//
// Clear the flag for the case there are events triggered
// during clear execution.
//
bDetectedEventsDuringClear = false;

//
// Unlock the Tamper Control register. This is required before
// calling HibernateTamperEventsClearNoLock().
//
HibernateTamperUnLock();
do
{
    //
    // We will start to poll the log registers at index 1 for
    // any new events.
    //
    ui8StartIdx = 1;
    //
    // Clear the Tamper event.
    // Note this API doesn't wait for synchronization, which
    // allows us to check the tamper log during
    // synchronization.
    //
    HibernateTamperEventsClearNoLock();

    //
    // Check new tamper event during tamper event clear
    // synchronization.
    // This will take about 92us(three clock cycles) at most.
    //
    while(HibernateTamperStatusGet() & HIBERNATE_TAMPER_STATUS_EVENT)
    {

        //
        // Clear execution isn't done yet , poll for new events.
        // If there were any new event, it will be logged in log 1
        // registers and so on.
        //
        for(ui8Idx = ui8StartIdx; ui8Idx< 4; ui8Idx++)
        {
```

```
            if(HibernateTamperEventsGet(ui8Idx,
                                        &g_ui32RTCLog[ui8Idx],
                                        &g_ui32EventLog[ui8Idx]))
            {
                //
                // detected new event, store it.
                //
                g_ui32TamperEventFlag |= g_ui32EventLog[ui8Idx];

                //
                // check for more event.
                //
                continue;
            }
            else
            {
                //
                // no new event in this log, update the log index
                // to be checked next, and break out of loop.
                //
                ui8StartIdx = ui8Idx;
                break;
            }
        }

        //
        // all last three logs have info. Check if all 4 logs
        // have the same info. This is to detect the case that
        // events happen during clear execution.
        //
        if(ui8Idx == 4)
        {
            //
            // If events happens during clear
            // execution, all four log registers will be
            // logged with the same event, to detect this
            // condition, we will compare with all four log data.
            //
            if(HibernateTamperEventsGet(0, &g_ui32RTCLog[0], &g_ui32EventLog[0]))
            {
                if((g_ui32RTCLog[0] == g_ui32RTCLog[1])     &&
                   (g_ui32EventLog[0] == g_ui32EventLog[1]) &&
                  (g_ui32RTCLog[0] == g_ui32RTCLog[2])     &&
                   (g_ui32EventLog[0] == g_ui32EventLog[2]) &&
                   (g_ui32RTCLog[0] == g_ui32RTCLog[3])     &&
                   (g_ui32EventLog[0] == g_ui32EventLog[3]))
                {
                    //
                    // Detected events during clear execution.
                    // Event logging takes priority, the clear
                    // will not be done in this case. We will need
                    // to go back to the beginning of the loop and
                    // clear the events.
                    //
                    if(bDetectedEventsDuringClear)
                    {
                        //
                        // This condition has already detected,
                        // we have cleared the event,
                        // clear the flag.
                        //
                        bDetectedEventsDuringClear = false;
                    }
                    else
                    {
                        // This is the first time it has been
```

```
                                    // detected, set the flag.
                                    //
                                    bDetectedEventsDuringClear = true;
                                }

                                //
                                // Break out of while loop so that we can
                                // clear the events, and start the
                                // workaround all over again.
                                //
                                break;
                            }
                        }
                        else
                        {
                            //
                            // Log 0 didn't detect any events. So this is not
                            // the case of missing events during clear
                            // execution.
                            // Update the log index at which we will poll next.
                            // It should be the last log entry that OR all the
                            // new events.
                            //
                            ui8StartIdx = 3;
                        }
                    }
                }
            }
            while(bDetectedEventsDuringClear);
            //
            // Lock the Tamper Control register.
            //
            HibernateTamperLock();

            //
            // Save the tamper event and RTC log info in the Hibernate Memory
            //
            HibernateDataGet(pui32Buf, 3);
            pui32Buf[1] = g_ui32TamperEventFlag;
            pui32Buf[2] = g_ui32TamperRTCLog;
            HibernateDataSet(pui32Buf, 3);

            //
            // Signal the main loop that an NMI event occurred.
            //
            g_ui32NMIEvent++;
        }

        //
        // Clear NMI events
        //
        SysCtlNMIClear(ui32NMIStatus);
    }

}
```

7 Appendix 2

To address the erratum EPI#01, "Data Reads can be Corrupted when the Code Address Space in the EPI Module is Used", the following code should be added to the epi.h file in the *C:\ti\TivaWare_C_Series-2.0\driverlib*:

```
#ifdef rvmdk
//*****************************************************************************
//
// Keil case.
//
//*****************************************************************************
inline void
EPIWorkaroundWordWrite(uint32_t *pui32Addr, uint32_t ui32Value)
{
  uint32_t ui32Scratch;

    __asm
    {
        //
        // Add a NOP to ensure we don’t have a flash read immediately before
        // the EPI read.
        //
        NOP

        //
        // Perform the write we're actually interested in.
        //
        STR ui32Value, [pui32Addr]

        //
        // Read from SRAM to ensure that we don't have an EPI write followed by
        // a flash read.
        //
        LDR ui32Scratch, [__current_sp()]
    }
}

inline uint32_t
EPIWorkaroundWordRead(uint32_t *pui32Addr)
{
  uint32_t ui32Value, ui32Scratch;

    __asm
    {
        //
        // Add a NOP to ensure we don’t have a flash read immediately before
        // the EPI read.
        //
        NOP

        //
        // Perform the read we're actually interested in.
        //
        LDR ui32Value, [pui32Addr]

        //
        // Read from SRAM to ensure that we don't have an EPI read followed by
        // a flash read.
        //
        LDR ui32Scratch, [__current_sp()]
    }

    return(ui32Value);
}

inline void
```

```
EPIWorkaroundHWordWrite(uint16_t *pui16Addr, uint16_t ui16Value)
{
    uint32_t ui32Scratch;

    __asm
    {
        //
        // Add a NOP to ensure we don’t have a flash read immediately before
        // the EPI read.
        //
        NOP

        //
        // Perform the write we're actually interested in.
        //
        STRH ui16Value, [pui16Addr]

        //
        // Read from SRAM to ensure that we don't have an EPI write followed by
        // a flash read.
        //
        LDR ui32Scratch, [__current_sp()]
    }
}

inline uint16_t
EPIWorkaroundHWordRead(uint16_t *pui16Addr)
{
    uint32_t ui32Scratch;
    uint16_t ui16Value;

    __asm
    {
        //
        // Add a NOP to ensure we don’t have a flash read immediately before
        // the EPI read.
        //
        NOP

        //
        // Perform the read we're actually interested in.
        //
        LDRH ui16Value, [pui16Addr]

        //
        // Read from SRAM to ensure that we don't have an EPI read followed by
        // a flash read.
        //
        LDR ui32Scratch, [__current_sp()]
    }

    return(ui16Value);
}

inline void
EPIWorkaroundByteWrite(uint8_t *pui8Addr, uint8_t ui8Value)
{
  uint32_t ui32Scratch;

    __asm
    {
        //
        // Add a NOP to ensure we don’t have a flash read immediately before
        // the EPI read.
        //
        NOP
```

```
        //
        // Perform the write we're actually interested in.
        //
        STRB ui8Value, [pui8Addr]

        //
        // Read from SRAM to ensure that we don't have an EPI write followed by
        // a flash read.
        //
        LDR ui32Scratch, [__current_sp()]
    }
}

inline uint8_t
EPIWorkaroundByteRead(uint8_t *pui8Addr)
{
    uint32_t ui32Scratch;
    uint8_t ui8Value;

    __asm
    {
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        NOP

        //
        // Perform the read we're actually interested in.
        //
        LDRB ui8Value, [pui8Addr]

        //
        // Read from SRAM to ensure that we don't have an EPI read followed by
        // a flash read.
        //
        LDR ui32Scratch, [__current_sp()]
    }

    return(ui8Value);
}

#else
#ifdef ccs

//*****************************************************************************
//
// Code Composer Studio versions of these functions can be found in separate
// source file epi_workaround_ccs.s.
//
//*****************************************************************************
extern void EPIWorkaroundWordWrite(uint32_t *pui32Addr, uint32_t ui32Value);
extern uint32_t EPIWorkaroundWordRead(uint32_t *pui32Addr);
extern void EPIWorkaroundHWordWrite(uint16_t *pui16Addr, uint16_t ui16Value);
extern uint16_t EPIWorkaroundHWordRead(uint16_t *pui16Addr);
extern void EPIWorkaroundByteWrite(uint8_t *pui8Addr, uint8_t ui8Value);
extern uint8_t EPIWorkaroundByteRead(uint8_t *pui8Addr);

#else
//*****************************************************************************
//
// GCC and IAR case.
//
//*****************************************************************************
inline void
```

```
EPIWorkaroundWordWrite(uint32_t *pui32Addr, uint32_t ui32Value)
{
    volatile register uint32_t ui32Scratch;

    __asm volatile (
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        "    NOP\n"
        "    STR %[value],[%[addr]]\n"
        "    LDR %[scratch],[sp]\n"
         :  [scratch] "=r" (ui32Scratch)
         :  [addr] "r" (pui32Addr), [value] "r" (ui32Value)
     );

    //
    // Keep the compiler from generating a warning.
    //
    ui32Scratch = ui32Scratch;
}

inline uint32_t
EPIWorkaroundWordRead(uint32_t *pui32Addr)
{
    volatile register uint32_t ui32Data, ui32Scratch;

    //
    // ui32Scratch is not used other than to add a padding read following the
    // "real" read.
    //

    __asm volatile(
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        "    NOP\n"
        "    LDR %[ret],[%[addr]]\n"
        "    LDR %[scratch],[sp]\n"
         : [ret] "=r" (ui32Data),
           [scratch] "=r" (ui32Scratch)
         : [addr] "r" (pui32Addr)
    );


    //
    // Keep the compiler from generating a warning.
    //
    ui32Scratch = ui32Scratch;

    return(ui32Data);
}

inline void
EPIWorkaroundHWordWrite(uint16_t *pui16Addr, uint16_t ui16Value)
{
    volatile register uint32_t ui32Scratch;

    __asm volatile (
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        "    NOP\n"
        "    STRH %[value],[%[addr]]\n"
```

```
            "    LDR %[scratch],[sp]\n"
             :  [scratch] "=r" (ui32Scratch)
             :  [addr] "r" (pui16Addr), [value] "r" (ui16Value)
         );


    //
    // Keep the compiler from generating a warning.
    //
    ui32Scratch = ui32Scratch;
}

inline uint16_t
EPIWorkaroundHWordRead(uint16_t *pui16Addr)
{
    register uint16_t ui16Data;
    register uint32_t ui32Scratch;

    //
    // ui32Scratch is not used other than to add a padding read following the
    // "real" read.
    //

    __asm volatile(
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        "    NOP\n"
        "    LDRH %[ret],[%[addr]]\n"
        "    LDR %[scratch],[sp]\n"
         : [ret] "=r" (ui16Data),
           [scratch] "=r" (ui32Scratch)
         : [addr] "r" (pui16Addr)
    );

    //
    // Keep the compiler from generating a warning.
    //
    ui32Scratch = ui32Scratch;

    return(ui16Data);
}

inline void
EPIWorkaroundByteWrite(uint8_t *pui8Addr, uint8_t ui8Value)
{
    volatile register uint32_t ui32Scratch;

    __asm volatile (
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        "    NOP\n"
        "    STRB %[value],[%[addr]]\n"
        "    LDR %[scratch],[sp]\n"
         :  [scratch] "=r" (ui32Scratch)
         :  [addr] "r" (pui8Addr), [value] "r" (ui8Value)
     );

    //
    // Keep the compiler from generating a warning.
    //
    ui32Scratch = ui32Scratch;
}
```

```
inline uint8_t
EPIWorkaroundByteRead(uint8_t *pui8Addr)
{
    register uint8_t ui8Data;
    register uint32_t ui32Scratch;

    //
    // ui32Scratch is not used other than to add a padding read following the
    // "real" read.
    //

    __asm volatile(
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        "    NOP\n"
        "    LDRB %[ret],[%[addr]]\n"
        "    LDR %[scratch],[sp]\n"
         : [ret] "=r" (ui8Data),
           [scratch] "=r" (ui32Scratch)
         : [addr] "r" (pui8Addr)
    );

    //
    // Keep the compiler from generating a warning.
    //
    ui32Scratch = ui32Scratch;

    return(ui8Data);
}

#endif
```

In addition, if using CCS, the following code should be saved as a file entitled epi_workaround_ccs.s in the C:\ti\TivaWare_C_Series-2.0\driverlib directory and included in the project:

```
;*******************************************************************************
;
; epi_workaround_ccs.s - EPI memory access functions.
;
; Copyright (c) 2013 Texas Instruments Incorporated.  All rights reserved.
; TI Information - Selective Disclosure
;
;*******************************************************************************

;*******************************************************************************
;
; void EPIWorkaroundWordWrite(uint32_t *pui32Addr, uint32_t ui32Value)
;
;*******************************************************************************
    .sect ".text:EPIWorkaroundWordWrite"
    .global EPIWorkaroundWordWrite
EPIWorkaroundWordWrite:
    ;
    ; Include a no-op to ensure that we don't have a flash data access
    ; immediately before the EPI access.
    ;
    nop

    ;
    ; Store the word in EPI memory.
    ;
    str r1, [r0]

    ;
```

```
    ; Make a dummy read from the stack to ensure that we don't have a flash
    ; data access immediately after the EPI access.
    ;
    ldr r1, [sp]

    ;
    ; Return to the caller.
    ;
    bx lr

    .align 4

;*******************************************************************************
;
; uint32_t EPIWorkaroundWordRead(uint32_t *pui32Addr)
;
;*******************************************************************************
    .sect ".text:EPIWorkaroundWordRead"
    .global EPIWorkaroundWordRead
EPIWorkaroundWordRead:
    ;
    ; Include a no-op to ensure that we don't have a flash data access
    ; immediately before the EPI access.
    ;
    nop

    ;
    ; Read the word from EPI memory.
    ;
    ldr r0, [r0]

    ;
    ; Make a dummy read from the stack to ensure that we don't have a flash
    ; data access immediately after the EPI access.
    ;
    ldr r1, [r13]

    ;
    ; Return to the caller.
    ;
    bx lr

    .align 4


;*******************************************************************************
;
; void EPIWorkaroundHWordWrite(uint16_t *pui16Addr, uint16_t ui16Value)
;
;*******************************************************************************
    .sect ".text:EPIWorkaroundHWordWrite"
    .global EPIWorkaroundHWordWrite
EPIWorkaroundHWordWrite:
    ;
    ; Include a no-op to ensure that we don't have a flash data access
    ; immediately before the EPI access.
    ;
    nop

    ;
    ; Store the word in EPI memory.
    ;
    strh r1, [r0]

    ;
    ; Make a dummy read from the stack to ensure that we don't have a flash
```

```
    ; data access immediately after the EPI access.
    ;
    ldr r1, [sp]

    ;
    ; Return to the caller.
    ;
    bx lr

    .align 4

;*******************************************************************************
;
; uint16_t EPIWorkaroundHWordRead(uint16_t *pui16Addr)
;
;*******************************************************************************
    .sect ".text:EPIWorkaroundHWordRead"
    .global EPIWorkaroundHWordRead
EPIWorkaroundHWordRead:
    ;
    ; Include a no-op to ensure that we don't have a flash data access
    ; immediately before the EPI access.
    ;
    nop

    ;
    ; Read the half word from EPI memory.
    ;
    ldrh r0, [r0]

    ;
    ; Make a dummy read from the stack to ensure that we don't have a flash
    ; data access immediately after the EPI access.
    ;
    ldr r1, [r13]

    ;
    ; Return to the caller.
    ;
    bx lr

    .align 4

;*******************************************************************************
;
; void EPIWorkaroundByteWrite(uint8_t *pui8Addr, uint8_t ui8Value)
;
;*******************************************************************************
    .sect ".text:EPIWorkaroundByteWrite"
    .global EPIWorkaroundByteWrite
EPIWorkaroundByteWrite:
    ;
    ; Include a no-op to ensure that we don't have a flash data access
    ; immediately before the EPI access.
    ;
    nop

    ;
    ; Store the byte in EPI memory.
    ;
    strb r1, [r0]

    ;
    ; Make a dummy read from the stack to ensure that we don't have a flash
    ; data access immediately after the EPI access.
    ;
```

```
    ldr r1, [sp]

    ;
    ; Return to the caller.
    ;
    bx lr

    .align 4

;*******************************************************************************
;
; uint8_t EPIWorkaroundByteRead(uint8_t *pui8Addr)
;
;*******************************************************************************
    .sect ".text:EPIWorkaroundByteRead"
    .global EPIWorkaroundByteRead
EPIWorkaroundByteRead:
    ;
    ; Include a no-op to ensure that we don't have a flash data access
    ; immediately before the EPI access.
    ;
    nop

    ;
    ; Read the byte from EPI memory.
    ;
    ldrb r0, [r0]

    ;
    ; Make a dummy read from the stack to ensure that we don't have a flash
    ; data access immediately after the EPI access.
    ;
    ldr r1, [r13]

    ;
    ; Return to the caller.
    ;
    bx lr

    .align 4

.end
```

Revision History

This silicon errata history highlights the technical changes made to TM4C129x Errata to make it the new document SPMZ850, as well as the technical changes made to the SPMZ850 revision to make it an SPMZ850A revision.

SEE	ADDITIONS/MODIFICATIONS/DELETIONS
Revision A (October 2013) Changes Below:	
Global	Corrected GPIO#07 and moved to the data sheet.
Section 5 Known Design Exceptions to Functional Specifications	Updated/Changed SYSCTL#14 Description by changing power consumption from "3mA" to "up to 3.5mA"
Revision * (October 2013) Changes Below:	
Global	Updated/Changed all instances of "X" to "XM4C" and all instances of "T" to "TM4C" (with regard to part numbers).
Section 2 Device Nomenclature	Updated/Changed title of section from "Device and Development Support-Tool Nomenclature" to "Device Nomenclature". Removed all information pertaining to tool development or development-support tools.
Section 3 Device Markings	Figure 1, Example of Device Part Markings: • Added TI logo. • Removed "980" from first row of text. • Moved the "C" from the end of line 2 to the beginning of line 3. • Updated/Changed "G4" to "G1".
Section 5 Known Design Exceptions to Functional Specifications	Added the following advisories: • ADC#15: ADC Global Synchronization Does not Function • CRC#01: Any Data Read From a Non-CRC Register After Accessing the CRCRSLTPP Register is Incorrect • ETH#01: The LED Polarity Bit in the Ethernet MAC Clock Configuration Register Does Not Function • EPI#01: Data Reads can be Corrupted when the Code Address Space in the EPI Module is Used • GPTM#16: Special Configuration is Required when Operating the GPTM in 32-bit Mode with the Alternate Clock Source • GPTM#17: The GPTMSYNC Register Bits Must be Manually Cleared when Using an Alternate Clock Source • HIB#15: The VDDFAIL Interrupt bit in the HIBIC Register is not Properly Cleared • HIB#16: Application Code May Miss New Tamper Event During Clear • HIB#17: $\overline{\text{WAKE}}$ Cannot be Used to Wake From Hibernate Mode • HIB#18: Can get two Matches per day in Calendar Mode • MEM#09: ROM_SysCtlClockFreqSet() Does not Properly Configure MOSC • PWM#01: Under Certain Circumstances, the PWM Load Interrupt is Triggered as Soon as the PWM is Enabled • PWM#02: Setting the PWMSYNC Bits May Not Synchronize the PWM Counters if PWMDIV is Used • PWM#03: The PWM Generators May Not Generate Interrupts or ADC Triggers • SSI#04: The First Byte Sent by the SSI in Master Mode is Incorrect when Using the Alternate Clock • SSI#05: Bus Contention in Bi- and Quad-Mode of SSI • SYSCTL#12: MOSC Does not Power Down in Deep-Sleep when it is not the Deep-Sleep Clock Source • SYSCTL#13: The NMIC Register Does not Indicate NMI Sources when Read • SYSCTL#14: Power Consumption is Higher When MOSC is Used in Single-Ended Mode • SYSCTL#15: Watchdog Reset Improperly Updates the NMIC Register • USB#03: Any Data Read From a Non-USB Register After Accessing the USBPP, USBPC, or the USBCC Register is Incorrect Updated/Changed LCD#01 Description introductory paragraph. Updated/Changed LCD#02 Workaround last sentence by removing "unless a high-throughput condition is needed." Updated/Changed SYSCTL#09 Description and Workaround to current text.
Section 6 Appendix 1	Added **NEW** section.
Section 7 Appendix 2	Added **NEW** section.

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

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information 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.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

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

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