MSP430F541x/543x Device Erratasheet

1 Current Version

See Appendix for prior revisions.
✓ indicates that the bug is present in the specified revision.

Devices	Rev:	ADC20	ADC21	ADC24	ADC25	CPU15	CPU16	CPU18	CPU20	CPU24	CPU25	CPU26	CPU27	CPU28	CPU29	CPU30	CPU31	CPU32	CPU33	CPU34	CPU35	CPU36
MSP430F5418	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5419	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5435	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5436	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5437	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5438	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

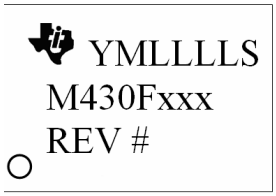
Devices	Rev:	DMA4	DMA5	DMA6	DMA7	DMA8	EEM4	EEM5	EEM6	EEM8	EEM9	EEM12	EEM13	EEM14	FLASH28	FLASH29	FLASH30	FLASH31	FLASH32	FLASH33	JTAG17	JTAG19
MSP430F5418	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5419	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5435	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5436	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5437	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5438	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Devices	Rev:	JTAG20	PMM1	PMM2	PMM3	PMM5	PMM6	PMM7	PMM8	PMM9	PMM10	PORT13	PORT14	RTC2	RTC3	RTC4	SYS1	SYS2	SYS3	SYS4	SYS5	SYS6
MSP430F5418	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5419	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5435	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5436	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5437	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5438	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Devices	Rev:	SYS7	SYS8	SYS9	TA23	TB20	TB21	UCS1	UCS2	UCS4	UCS5	UCS6	UCS7	USCI26	WDG4
MSP430F5418	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5419	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5435	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5436	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5437	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430F5438	L	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

2 Package Markings

PN80 ***LQFP (PN), 80 Pin***



YM = Year and Month Date Code
LLLL = LOT Trace Code
S = Assembly Site Code
# = DIE Revision
o = PIN 1

PZ100 ***LQFP (PZ) 100 Pin***

YM = Year and Month Date Code
LLLL = LOT Trace Code
S = Assembly Site Code
# = DIE Revision
o = PIN 1

3 Detailed Bug Description

ADC20 *ADC12 Module*

Function ADC12DF does not work as described in the user's guide

Description The conversion data format (ADC12DF bit setting) must be selected before a conversion. Switching the conversion data format bit after a conversion has begun does not deliver results in the newly selected format.

Workaround Change the conversion data format before starting the conversion.

ADC21 *ADC12 Module*

Function ADC12ENC is not working correctly

Description Resetting the ADC12ENC bit during an active conversion or within a sequence can lead to an invalid result for both the current and next ADC12 conversions. Subsequent ADC12 conversions are valid.

Workaround Resetting the ADC12ENC bit during an active conversion or within a sequence should be avoided. If the ADC12ENC bit is reset during an active conversion, this and the next conversion result should be discarded.

ADC24 *ADC12 Module*

Function Unexpected ADC12 current draw when ADC12ENC = 1

Description When set, the ADC12ENC bit issues a clock request to the selected source clock, even before the conversion trigger. This causes some extra current consumption, depending on the selected clock.

Workaround None

ADC25 *ADC12 Module*

Function Write to ADC12CTL0 triggers ADC12 when CONSEQ = 00

Description If ADC conversions are triggered by the Timer_B module and the ADC12 is in single-channel single-conversion mode (CONSEQ = 00), ADC sampling is enabled by write access to any bit(s) in the ADC12CTL0 register. This is contrary to the expected behavior that only the ADC12 enable conversion bit (ADC12ENC) triggers a new ADC12 sample.

Workaround When operating the ADC12 in CONSEQ = 00 and a Timer_B output is selected as the sample and hold source, temporarily clear the ADC12ENC bit before writing to other bits in the ADC12CTL0 register. The following capture trigger can then be re-enabled by setting ADC12ENC = 1.

CPU15 *CPU Module*

Function Modifying the Program Counter (PC) behaves differently than in previous devices

Description When using instructions with immediate or indirect addressing mode to modify the PC, a different value compared to previous devices must be added to get to the same destination.

Example Previous device (MSP430F4619)

```
label_1 ADD.W #Branch1-label_1-4h,PC
```

MSP430F5438

```
label_1 ADD.W #Branch1-label_1-2h,PC
```

NOTE: The MOV instruction is not affected

Workaround

- Additional NOP after the PC-modifying instruction
  or
- Change the offset value in software

CPU16 *CPU Module*

Function Indexed addressing with instructions `calla`, `mova`, and `bra`.

Description With indexed addressing mode and instructions `calla`, `mova`, and `bra`, it is not possible to reach memory above 64k if the register content is < 64k.
For example, `calla idx16` = `0004h` (R5 = FFFEh) results in a 20-bit call of address 0002h instead of 10002h.

Workaround

- Use different addressing mode to reach memory above 64k.
- Use `adda index, Reg` to calculate address in upper memory, and then use `calla Reg`.

CPU18 *CPU Module*

Function LPM instruction can corrupt PC/SR registers

Description The PC and SR registers have the potential to be corrupted when:

- An instruction using register, absolute, indexed, indirect, indirect auto-increment, or symbolic mode is used to set the LPM bits (e.g., `BIS &xyh, SR`).
  and
- This instruction is followed by a CALL or CALLA instruction.

Upon servicing an interrupt service routine, the program counter (PC) is pushed twice onto the stack instead of the correct operation where the PC, then the SR registers are pushed onto the stack. This corrupts the SR and possibly the PC on RETI from the ISR.

Workaround Insert a NOP or __no_operation() intrinsic function between the instruction to enter low-power mode and the CALL or CALLA instruction.

CPU20 *CPU Module*

Function An unexpected Vacant Memory Access Flag (VMAIFG) can be triggered due to the CPU autoincrement of the MAB+2 outside the range of a valid memory block.

Description The VMAIFG is triggered if a PC-modifying instruction (e.g., ret, push, call, pop, jmp, br) is fetched from the last address of a section of memory (e.g., flash, RAM) that is not contiguous to a higher valid section on the memory map.

Workaround If code is affected, edit the linker command file to make the last four bytes of affected memory sections unavailable.

CPU24 *CPU Module*

Function Program counter corruption following entry into low-power mode

Description The program counter is corrupted when an interrupt event occurs in the time between (and including) one cycle before and one cycle after the CPUOFF bit is set in the status register. This failure occurs when the BIS instruction is followed by a CALL or CALLA instruction using the following addressing modes:

```
BIS &, SR
CALLA indir, indir autoinc, reg
BIS INDEX, SR
CALLA indir, indir autoinc, reg
BIS reg, SR
CALLA reg, indir, indir autoinc
```

NOTE: Due to the instruction emulation, the EINT instruction, as well as the __enable_interrupts() and possibly the __bis_SR_register() intrinsic functions are affected.

Workaround Insert a NOP instruction or __no_operation() intrinsic function call between the BIS and CALL or CALLA instructions.

CPU25 *CPU Module*

Function DMA transfer does not execute during low-power mode

Description If the following instruction sequence is used ([] denotes an addressing mode) to enter a low-power mode, and the DMARMWDIS bit is set, then DMA transfers are blocked for the duration of the low-power mode.

```
BIS [register|index|absolute|symbolic],SR
CALLA [register]
```

Workaround

- Insert a NOP instruction or __no_operation() intrinsic function call between the BIS and CALLA instructions

  or
- Temporarily disable the DMARMWDIS bit when entering low-power mode.

CPU26 *CPU Module*

Function CALL SP does not behave as expected

Description When the intention is to execute code from the stack, a CALL SP instruction skips the first piece of data (instruction) on the stack. The second piece of data at SP+2 is used as the first executable instruction.

Workaround Write the op code for a NOP as the first instruction on the stack. Begin the intended subroutine at address SP + 2.

NOTE: The compiler will not generate a CALL SP instruction

CPU27 *CPU Module*

Function Program Counter (PC) is corrupted during the context save of a nested interrupt

Description When a low-power mode is entered within an interrupt service routine that has enabled nested interrupts (by setting the GIE bit), and the instruction that sets the low-power mode is directly followed by a RETI instruction, an incorrect value of PC + 2 is pushed to the stack during the context save. Hence, the RETI instruction is not executed on return from the nested interrupt and the PC becomes corrupted.

Workaround Insert a NOP or __no_operation() intrinsic function between the instruction that sets the lower power mode and the RETI instruction.

CPU28 *CPU Module*

Function PC is corrupted when using certain extended addressing mode combinations

Description An extended memory instruction that modifies the program counter executes incorrectly when preceded by an extended memory write-back instruction under the following conditions:

First instruction:

2-operand instruction, extended mode using (register,index), (register,absolute), or (register,symbolic) addressing modes

Second instruction:

2-operand instruction, extended mode using the (indirect,PC), (indirect auto-increment,PC), or (indexed [with ind 0], PC) addressing modes

Example

```
BISX.A R6,&AABCD
ANDX.A @R4+,PC
```

Workaround

- Insert a NOP or a __no_operation() intrinsic function between the two instructions.
  or
- Do not use an extended memory instruction to modify the PC.

CPU29 *CPU Module*

Function Using a certain instruction sequence to enter low-power mode(s) affects the instruction width of the first instruction in an NMI ISR

Description If there is a pending NMI request when the CPU enters a low-power mode (LPMx) using an instruction of Indexed source addressing mode, and that instruction is followed by a 20-bit wide instruction of Register source and destination addressing modes, the first instruction of the ISR is executed as a 20-bit wide instruction.

Example

```
main:
    ...
    MOV.W [indexed],SR ; Enter LPMx
    MOVX.A [register],[register] ; 20-bit wide instruction
    ...
ISR_start:
    MOV.B [indexed],[register] ; ERROR - Executed as a 20-bit instruction!
```

Note: [] indicates addressing mode

Workaround

- Insert a NOP or a __no_operation() intrinsic function following the instruction that enters the LPMx using indexed addressing mode.
  or
- Use a NOP or a __no_operation() intrinsic function as first instruction in the ISR.
  or
- Do not use the indexed mode to enter LPMx.

CPU30 *CPU Module*

Function ADDA, SUBA, CMPA [immediate],PC behave as if immediate value were offset by -2

Description The extended address instructions ADDA, SUBA, CMPA in immediate addressing mode are represented by 4-bytes of opcode. In cases where the program counter (PC) is used as the destination register only 2 bytes of the current instruction's 4-byte opcode are accounted for in the PC value. The resulting operation executes as if the immediate value were offset by a value of -2.

NOTE: The MOV instruction is not affected.

Workaround

- Modify immediate value in software to account for the offset of 2.
  or
- Use extended 20-bit instructions (addx.a, subx.a, cmpx.a) instead.

CPU31 *CPU Module*

Function PUSHX.A @SP+ corrupts the stack pointer

Description When the instruction PUSHX.A is executed using the indirect auto-increment mode with the stack pointer (SP) as the source register [PUSHX.A @SP+] the SP is consequently corrupted. Instead of decrementing the value of the SP by four, the value of the SP is replaced with the data pointed to by the SP previous to the PUSHX.A instruction execution.

Workaround None. The compiler does not generate a PUSHX.A instruction that involves the SP.

CPU32 *CPU Module*

Function CALLA PC executes incorrectly

Description When the instruction CALLA PC is executed, the program counter (PC) that is pushed onto the stack during the context save is incorrectly offset by a value of -2.

Workaround None. The compiler does not generate a CALLA PC instruction.

CPU33 *CPU Module*

Function CALLA [indexed] may corrupt the program counter

Description When the Stack Pointer (SP) is used as the destination register in the CALLA index(Rdst) instruction and is preceded by a PUSH or PUSHX instruction in any of the following addressing modes: Absolute, Symbolic, Indexed, Indirect register or Indirect auto increment, the "index" of the CALLA instruction is not sign extended to 20-bits and is always treated as a positive value. This causes the Program Counter to be set to a wrong address location when the index of the CALLA instruction represents a negative offset.

NOTE: This erratum applies only when the instruction sequence is: PUSH or PUSHX followed by CALLA index(SP).

This erratum does not apply if the PUSH or PUSHX instruction is used in the Register or Immediate addressing mode.

This erratum applies only when SP is used as the destination register in the CALLA index(Rdst) instruction.

Workaround Place a NOP instruction in between the PUSH or PUSHX and the CALLA index(SP) instructions.

NOTE: This bug has no compiler impact as the compiler does not generate a CALLA instruction that uses indexed addressing mode with the SP.

CPU34 *CPU Module*

Function CPU may be halted if a conditional jump is followed by a rotate PC instruction

Description If a conditional jump instruction (JZ, JNZ, JC, JNC, JN, JGE, JL) is followed by an Address Rotate instruction on the PC (RRCM, RRAM, RLAM, RRUM) and the jump is not performed, the CPU is halted.

Workaround Insert a NOP between the conditional jump and the rotate PC instructions.

CPU35 *CPU Module*

Function Instruction BIT.B @Rx,PC uses the wrong PC value

Description The BIT(.B/.W) instruction in indirect register addressing mode is represented by 2 bytes of opcode. And if Program Counter (PC) is used as the destination register, the 2 opcode bytes of the current BIT instruction are not accounted for. The resulting operation executes the instruction using the wrong PC value and this affects the results in the Status Register (SR).

Workaround None

NOTE: The compiler does not generate a BIT instruction that uses the PC as an operand.

CPU36 *CPU Module*

Function PC corruption when single-stepping through flash erase

Description When single-stepping over a line code line that initiates an INFOD Flash memory erase, the program counter is corrupted.

Workaround None

NOTE: This erratum applies to debug mode only.

DMA4 *DMA Module*

Function Corrupted write access to 20-bit DMA registers

Description When a 20-bit wide write to a DMA address register (DMAxSA or DMAxDA) is interrupted by a DMA transfer, the register contents may be unpredictable.

Workaround

- Design the application to ensure that no DMA access interrupts 20-bit wide accesses to the DMA address registers.
  or
- When accessing the DMA address registers, enable the Read Modify Write disable bit (DMARMWDIS = 1) or temporarily disable all active DMA channels (DMAEN = 0).
  or
- Use word access for accessing the DMA address registers. Note that this limits the values that can be written to the address registers to 16-bit values (lower 64K of Flash).

DMA5 *DMA Module*

Function DMAxSZ is not updated correctly

Description The total number of DMA transactions to be executed per transfer (stored initially in DMAxSZ) is updated under the following conditions:

1. A DMA transfer has been completed.
   - DMAxSZ is refreshed by the value in the temporary register, T_Size, because it has been decremented to zero (any mode).
   - DMAxSZ is refreshed by the value in the temporary register, T_Size, because the DMA enable bit has transitioned from Hi -> Lo (any non-repeat mode).
2. A value is written into the DMAxSZ register by the application when the DMA state machine is in the Reset state (DMAEN = 0).

For all transfer modes the second condition is false. Writing to the DMAxSZ register (in an NMI service routine, for instance) immediately affects the DMAxSZ counter and the DMA will execute transfers until the DMAxSZ value decrements to zero, regardless of the current DMA state or DMAxSZ value.

Workaround The application should ensure that the DMAxSZ register is only written when the DMA is in the Reset state (DMAEN = 0).

DMA6 *DMA Module*

Function DMA cannot write to DMA

Description One DMA channel cannot modify the registers of another DMA channel. DMA register modifications must be done by JTAG or by the CPU.

Workaround None

DMA7 *DMA Module*

Function DMA interrupt lost

Description If a DMA request is received during the time when the read address of a read modify write instruction is on the Memory Address Bus (MAB), application interrupts could be lost.

Workaround Set the DMARMWDIS bit when using the DMA.

DMA8 *DMA Module*

Function DMA can corrupt values on write-access to program stack

Description If the DMA controller makes a write access to the stack while executing one of the following instructions, the data that is written may be corrupted.

CALLA [REG | IDX | SYM | ABS | IND | INA | IMM]
PUSHX.A [IDX | SYM | ABS | IND | IMM | INA]
PUSHX.A [REG]
PUSHM.A [REG]
POPM.A [REG]

Note: [ ] denotes an addressing mode

Workaround Do not declare function-scope variables. Declare all variables that are intended to be modified by the DMA as global- or file-scope such that they are allocated in the data section of RAM and not on the program stack.

EEM4 *Enhanced Emulation Module*

Function No breakpoint stop reaction at DMA/CPU switch

Description If an address-related trigger, such as the general IDE breakpoint, is triggered for a CPU instruction one cycle after a DMA transaction, the stop reaction will not be accepted by the EEM (will not halt program execution). A data related trigger set at a DMA transaction one cycle after a CPU instruction results in the same behavior.

Workaround Use trigger configurations that are not dependent on DMA/not-DMA transactions (Read, Write, or Don't Care). The State Storage Block can then be used at program stop to determine:

- Whether the trigger-causing event was a DMA or a non-DMA transaction
- The state of the program at the time of the event

EEM5 *Enhanced Emulation Module*

Function Individual timer enable/disable control unavailable

Description The timer control bits work such that the timers cannot be individually enabled or disabled for debug operation. All three timers are sourced by the selected global debug clock.

Workaround None

EEM6 *Enhanced Emulation Module*

Function Return from breakpoint with wrong PC value

Description If a breakpoint occurs on an instruction which has a 20-bit write back to memory, the address offset does not match the debug flow. The PC is offset by 6 rather than 4, meaning it points to the 16-bit word after the op-code of the instruction at which the program was to continue execution.

Workaround Manually change the PC value to [(displayed_value) – 2] (i.e., via the debugger's CPU register window) before the next single step or run operation is issued.

EEM8 *Enhanced Emulation Module*

Function Debugger stops responding when using the DMA

Description In repeated transfer mode, the DMA automatically reloads the size counter (DMAxSZ) once a transfer is complete and immediately continues to execute the next transfer unless the DMA Enable bit (DMAEN) has been previously cleared. In burst-block transfer mode, DMA block transfers are interleaved with CPU activity 80/20% – of ten CPU cycles, eight are allocated to a block transfer and two are allocated for the CPU.

Since the JTAG system must wait for the CPU bus to be clear in order to halt the device, it can only do so when two conditions are met:

- If three clock cycles after any DMA transfer, the DMA is no longer requesting the bus and
- If the CPU is not requesting the bus

Therefore, if the DMA is configured to operate in the repeat burst-block transfer mode and a breakpoint is set between the line of code that triggers the DMA transfers and the line that clears the DMAEN bit the DMA will always request the bus and the JTAG system will never gain control of the device.

Workaround When operating the DMA in repeat burst-block transfer mode, set breakpoint(s) only when the DMA transfers are not active (before the start or after the end of the DMA transfers).

EEM9 *Enhanced Emulation Module*

Function Combined breakpoint triggers may be missed

Description When debugging, if combined breakpoint triggers are used with certain combinations of instructions (examples below) the trigger may be missed.

Examples

```
1)  PUSH [IDX|SYM|ABS|IND|INA]
    MOV [REG],[REG]
2)  BIS [REG],SR
    CALLA [REG]
3)  ADD [IMM],[ABS] || ADD [REG],[IDX] || ADD [REG],[SYM] || ADD [IND],[SYM]
    ADD [IND],[REG]
4)  PUSHX.A [REG] with rep_cnt = 4
    SXTX.W [ABS]
    PUSHX [INA]
```

Note: [ ] denotes an addressing mode

Workaround None

EEM12 *Enhanced Emulation Module*

Function Debugger unexpectedly halts at ISR entry

Description When using breakpoints or single stepping in the presence of frequent system interrupts, the debugger may not halt at the breakpoint location and will instead halt at the entry to an interrupt service routine.

Workaround In certain instances inserting a NOP or __no_operation() intrinsic before and after the affected statement or moving the breakpoint can be used as a workaround.

NOTE: This erratum applies to debug mode only.

EEM13 *Enhanced Emulation Module*

Function Halting the debugger does not return correct PC value when in LPM

Description When debugging, if the device is in any low-power mode and the debugger is halted, the program counter update by the debugger is corrupted. The debugger is unable to halt at the correct location.

Workaround None

NOTE: This erratum applies to debug mode only.

EEM14 *Enhanced Emulation Module*

Function Single-step or breakpoint on module register with WAIT capability may not work

Description In debug mode the CPU clock is driven independently from the wait inputs of device modules (i.e., MULT, USB, RF1A, CRC). As a result, an EEM halt on an access to the module data registers (breakpoint or single-step) may show incorrect results due to incomplete execution.

Workaround Do not single-step through a data register access that holds the CPU to provide a valid result. Place breakpoints after the affected register access has been provided sufficient clock cycles to return a valid result.

NOTE: This erratum applies to debug mode only.

FLASH28 *Flash Module*

Function Read disturb due to PMM level < 2

Description Flash access with default PMMCOREV settings (PMMCOREVx = 00) or PMM level 1 (PMMCOREVx = 01) is not reliable over the full operating range of the device and is not recommended.

Workaround None.

NOTE: The default PMM setting has been modified to PMM level 2. The affected registers and the new default register settings are shown below. The PMM level setting should not be changed in application code.

PMMCTL0 = 0x02
SVSMHCTL = 0x4602
SVSMLCTL = 0x4602

FLASH29 *Flash Module*

Function Read disturb due to emergency exit from write/erase Flash operation

Description When a Flash write or erase is abruptly terminated, any further reliable reads from Flash are not ensured. The abrupt termination can occur as a result of the Emergency Exit bit (EMEX in FCTL3) being set. This forces a write or an erase operation to be terminated before normal completion.

Workaround After setting EMEX = 1, wait for at least 100 µs after a bank or mass erase and at least 6 µs after a segment erase before Flash is accessed again.

FLASH30 *Flash Module*

Function Read disturb

Description There is a problem that affects how addresses are decoded for INFO memory, creating read disturb issues for code executed out of information memory.

Workaround No code should be executed from INFO memory. In addition, data stored in INFO memory can be accessed only using code executed from the lower 64K of the memory map. Such accesses to INFO memory are unaffected by the problem.

FLASH31 *Flash Module*

Function Interrupts not disabled during flash operation

Description When a flash operation is in progress, interrupts are not automatically disabled. The CPU will always attempt to service the interrupt request, whether or not the flash is busy.

Workaround Disable interrupts using the GIE bit before erasing flash in another bank of memory. Note that all interrupts during this period of time remain pending until GIE = 1.

FLASH32 *Flash Module*

Function Bank erase problems

Description A Bank Erase operation can also lead to the erase of the information memory of the selected bank, when there was a read from any information memory address before the dummy write.

Workaround Insert a dummy read from any MAIN memory address before the dummy write, which triggers the Bank Erase operation.

FLASH33 *Flash Module*

Function Flash erase/program with fsystem <125 kHz causes code execution to fail

Description A flash erase or flash program operation with the system frequency (f_{system}) < 125 kHz causes the program execution (executing out of main or info memory) that follows to fail.

Workaround Make sure the f_{system} >125 kHz before doing a flash erase or program operation.

JTAG17 *JTAG Module*

Function JTAG mailbox handshake mechanism can go out of sync.

Description The JTAG mailbox handshake mechanism can go out of sync, if the CPU accesses the Mailbox (read or write on JMB data registers) while the JMBOUTCTL/JMBIBCTL register is accessed via the JTAG interface.

Workaround

- Avoid reading or writing on JMB data registers by CPU while the JMBOUTCTL/JMBIBCTL register is accessed via the JTAG interface.
  or
- Send a request (set OUTREQ or INPREQ bit in JMBINCTL register) only when it can be served immediately. This means in detail:
  INPUT: Wait until the the input registers are empty (IN0RDY = IN1RDY = 1) before sending an input request (INPREQ = 1) to the JMBINCTL register.
  OUTPUT 16 Bit: Wait until the CPU has written Register JMBOUT0 (OUT0RDY = 1) before sending an output request (OUTREQ = 1) to the JMBINCTL register.
  OUTPUT 32 Bit: Wait until the CPU has written Registers JMBOUT0 and JMBOUT1 (OUT0RDY = OUT1RDY = 1) before sending an output request (OUTREQ = 1) to the JMBINCTL register.

JTAG19 *JTAG Module*

Function 'Attach to running target' can cause a device reset

Description When a device is in LPM2,3,4 and both SVSx are disabled (see SVSxMD bit description in SVSMxCTL register) it is not possible to access the device via a 4-wire JTAG or Spi-Bi-Wire entry sequence without causing a reset of the device.

Workaround None

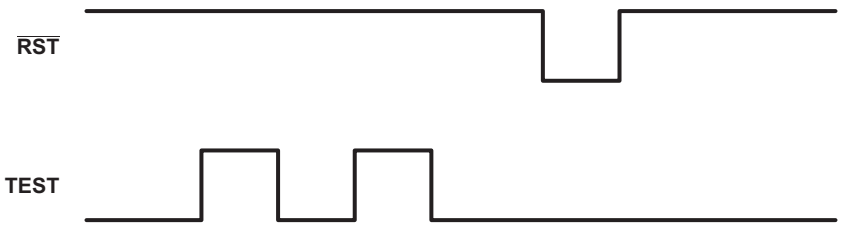
JTAG20 *SYS Module*

Function BSL does not exit to application code

Description The methods used to exit the BSL per the MSP430 Memory Programming User's Guide (SLAU265) are invalid.

Workaround To exit the BSL one of the following methods must be used.

- A Power cycle
- or
- Toggle the TEST pin twice when $\overline{\text{RST}}$ is high, and then pull $\overline{\text{RST}}$ low.



NOTE: This sequence is not subject to timing constraints and the appropriate level transitions are sufficient to trigger an exit from BSL mode.

PMM1 *Power Management Module*

Function Interrupt or POR events from SVSx or SVMx are delayed until end of DMA transfer.

Description If a DMA transfer is active, any interrupt or POR event triggered by SVSx or SVMx is delayed until end of DMA transfer. No event gets lost, but it is delayed.

Workaround Avoid long durations of DMA transfers.

PMM2 *Power Management Module*

Function Manual change of LDO voltage reference from 'switched' to 'static' mode may trigger an incorrect set event for SVSxIFG's (where 'x' stands for 'H' or 'L')

Description The incorrect set event can occur under the following PMM configuration and conditions (please refer to PMM figures titled "High-Side/Low-Side SVS and SVM"):

- SVSxMD is reset to 0, disabling the set of SVSxIFG when LDO is in 'switched' mode
  - LDO voltage reference automatically set to switched mode if:
    - Device enters LPM2, LPM3, LPM4
    - PMMCTL1 register bits PMMCMDx = 10b
- While LDO voltage reference is in 'switched' mode, the supervised voltage level is violated
- LDO voltage reference is changed back to 'static' mode (either by return from LPM mode or through the use of PMMCMDx bits) and the supervised voltage level is still being violated

Workaround SVSx events are not triggered when the LDO voltage reference is in 'switched' mode and the SVSx high-side mode is disabled (SVSHMD = 0). Under these conditions, the SVSx (SVSxE = 0) should be temporarily disabled for any entry into 'switched' mode.

PMM3 *Power Management Module*

Function Bit PMMLPM5IFG in PMMIFG register is not readable

Description Bit PMMLPM5IFG in PMMIFG register is not readable. User software read value is always 0; however, the bit functions as expected. Write access is working, and the reset of the bit by reading the respective SYSRSTIV vector word operates as described in the device user's guide.

Workaround Check the status of PMMLPM5IFG flag by reading the register SYSRSTIV vector word.

PMM5 *Power Management Module*

Function Unexpected SVMxIFG flag

Description The SVMxIFG flags can be unexpectedly set if the following conditions are met:

SVMxE = 1 ; SVM is enabled

SVMxOVPE = 1 ; Overvoltage protection is on

SVSMxACE = 1 ; Automatic performance mode selection is on

SVMLFP = 0 ; Performance mode set to 'normal'

Workaround None

PMM6 *Power Management Module*

Function Brownout startup problem

Description At low temperature (below -20°C) the device is likely to not work correctly after a brownout condition occurs. A brownout condition is defined as a drop on the DVDD supply to DVSS level that is less than 2 seconds in duration.

Workaround None

PMM7 *Power Management Module*

Function PMMRIE default conditions different from defaults shown in the device user's guide

Description The user's guide specifies that after a BOR reset condition the SVS is not configured to trigger a POR signal if the monitored voltages fall below the SVS level(s). This is not true for this device. The SVS Low and SVS High Side POR Enable bits (SVSLPE/SVSHPE) in the Power Management System Reset Enable and Interrupt Enable register are set by default (PMMRIE = 0x1100).

Workaround If this behavior is not desired, reset the SVSLPE/SVSHPE bits in the PMMRIE register at the beginning of the application.

PMM8 *Power Management Module*

Function Supply current in LPM4.5 is unpredictable

Description Due to an unpredictable value of the supply current in LPM4.5, the mode should not be used.

Workaround None

PMM9 *Power Management Module*

Function False SVSxIFG events when SVSxMD = 0 and SVSxFP = 1

Description When SVSxMD = 0 (default setting), the SVSx comparators are disabled automatically in LPM2/3/4 (disabling the respective SVSx events) and are then re-enabled on return to active mode. The comparators of the SVS require a certain amount of time to stabilize and output a correct result once re-enabled; this time is different for the Full Performance mode versus the Normal mode. This time to stabilize the SVS comparators is intended to be accounted for by a built-in event-masking delay of 2 µs when Full Performance mode is enabled. However, the comparators of the SVS in Full Performance mode take longer than 2 µs to stabilize, so the possibility exists that a false positive will be triggered on the SVSH or SVSL.

Workaround If the Full Performance mode is to be enabled for either the high- or low-side SVS comparators, the respective SVSxMD bits must be set (SVSxMD = 1) such that the SVS comparators are not temporarily shut off in LPM2/3/4. Note that this is equivalent to a 2 µA (typical) adder to the low-power mode current, per the device-specific data sheet, for each SVSx that remains enabled.

PMM10 *Power Management Module*

Function SVS/SVM flags disabled after Power Up Clear reset

Description SVS/SVM interrupt flags are not set after a Power Up Clear (PUC) Reset if the SVS was disabled before the PUC reset was applied.

Workaround A write access to the intended SVSx register after PUC re-enables the SVS and SVM interrupt flags.

PORT13 *Digital I/O Module, Port 1 and 2*

Function PxIV register is cleared by debugger, affecting debug program flow

Description When the PxIV register is read via the debugger the register gets cleared. This can cause lost port interrupts during the debug sessions.

Workaround The wrong clearing of the register during debugging itself cannot be avoided. The debug software should write back by the interrupt flag after it has been read and cleared.

PORT14 *Digital I/O Module, Port 1 and 2*

Function GPIO pins set to high impedance on exit from LPM2/3/4

Description When automatic SVS control is enabled (SVSMHACE or SVSMLACE are set), all I/Os are set to high-impedance state after wakeup from LPM 2/3/4. The duration of this high impedance state is between 10 µs and 100 µs. All input, output, and pull resistor settings are not applied during this time.

Workaround None

RTC2 *RTC Module*

Function RTC base address does not match datasheet specification

Description The range 0x0480 to 0x049F should be vacant space in the memory map, and any attempts to access it should create an NMI interrupt with a "vacant memory access" violation. However, the RTC module, which has a base address of 0x4A0, has been mirrored to this range instead and, therefore, no violation is generated if code attempts to access the space.

Workaround None

RTC3 *RTC Module*

Function Unreliable write to RTC register

Description A write access to the RTC registers (SEC, MIN, HOUR, DATE, MON, YEAR, DOW) may result in unexpected results. As a consequence the addressed register might not contain the written data, or some data can be accidentally written to other RTC registers.

Workaround Use the RTC library routines, available as F541x/F543x code examples on the MSP430 Code Examples page (www.ti.com/msp430 > Code Examples), which use carefully aligned MOV instructions. Library is listed as RTC_Workaround.zip and includes both CCE and IAR example projects that show proper usage. Using this library, full access to RTC registers is possible.

RTC4 *RTC Module*

Function RTC clock calibration maybe offset by +2 ppm

Description When using the RTC clock calibration feature, the RTCCLK frequency maybe offset by +2 ppm after the calibration procedure is completed. This affects both up and down calibration.

Workaround None

SYS1 *System Module*

Function User NMI can disturb the higher priority system NMI (priority inversion)

Description A lower priority user NMI (UNMI) can disturb a higher priority system NMI (SNMI), leading to a priority inversion. Note that an interrupt request will still not disturb a higher priority NMI.

Workaround None

SYS2 *System Module*

Function Protection weakness in BSL flash address range

Description If the BSL Protection is enabled, just the lowest 16 bytes of the secured BSL part should be usable as entry area (jump table). Instead, this entry window (not secured from readout by CPU) can also be seen on other addresses - address locations affected are dependent on the selected size of the BSL.

Example: for BSLSIZE = 0, memory is reflected into 0x01600 to 0x0160F and 0x01700 to 0x0170F.

Similarly, if RAM space is assigned as BSL area and secured, the mechanism can be observed. Only the area 0x01C00 to 0x01C0F should be protected, but the following RAM areas are also secured and no longer accessible: 0x01D00 to 0x01D0F, 0x01E00 to 0x01E0F, 0x01F00 to 0x01F0F, 0x02000 to 0x0200F...0x02B00 to 0x02B0F.

Any access to locations affected by errata leads to a security reset.

Workaround

- Do not use RAM as secured area for BSL.
  or
- Do not put sensitive user code/data in the additional BSL entry address windows.

SYS3 *System Module*

Function OFIFG is wrongly cleared by a SYSUNIV read

Description In a situation where more than one NMI flag is set, and software reads the SYSUNIV register, and then the very next memory access is also to the SYS module, then it is possible that an additional, lower priority NMI flag will also be cleared.

Workaround After reading the SYSUNIV register, insert an access to an address outside of the SYS module address space before making another access to the SYS module.

SYS4 *System Module*

Function BSL is not programmable

Description Due to other errata negatively affecting the expected behavior of code executed from non-MAIN segments of Flash, the BSL is not programmable for this device.

Workaround None

SYS5 *System Module*

Function Higher priority NMI interrupt lost during execution of lower priority NMI ISR

Description If a higher prioritized NMI occurs during the execution of a lower priority NMI ISR (read out of NI vector word) the higher IFG can be lost (but the lower interrupt is executed), which leads to a second execution of the lower NMI. The bug only occurs if the NMIs are of the same type (UNMI or SNMI).

Workaround None

SYS6 *System Module*

Function CPU skips execution of one additional instruction after returning from an SNMI ISR and before servicing a pending IRQ.

Description If an IRQ is activated during a SNMI ISR, after leaving the SNMI ISR, the CPU services the IRQ ISR immediately without first returning to the main program or the previously running ISR to execute one additional instruction (as defined in the SYS chapter of the device user's guide).

Workaround None

SYS7 *System Module*

Function Timing of USCI interrupts may cause PC corruption due to automatic clear of an IFG

Description When certain USCI I2C interrupt flags (IFG) are set and an automatic flag-clearing event on the I2C bus occurs, the address bus will default to FFFEh (Reset Vector) and the data bus to the opcode for the RETI instruction. This means the program counter defaults to FFFEh (Reset Vector) during this period of time. This will only happen when the IFG is cleared within a critical time window (~6 CPU clock cycles) after a USCI interrupt request occurs and before the interrupt servicing is initiated. The affected interrupts are UCBxTXIFG, UCSTPIFG, UCSTTIFG and UCNACKIFG.

The automatic flag-clearing scenarios are described in the following situations:

- A pending UCBxTXIFG interrupt request is cleared on the falling SCL clock edge following a NACK.
- A pending UCSTPIFG, UCSTTIFG, or UCNACKIFG interrupt request is cleared by a following Start condition.

If an NMI request occurs during the time that the PC holds the default value FFFEh before the default RETI instruction is executed, the PC will be corrupted on return from the NMI ISR. This is because the previous PC contents of FFFEh were stored onto the stack as the address of the next instruction to be executed. The CPU consequently interprets the address in FFFEh as an instruction (errant behavior) and because the default RETI was never executed, the stack pointer remains decreased by four (errant behavior).

Workaround Prevent the flag-clearing event from interrupting the servicing of the affected USCI IFGs:

- Poll the affected USCI flags instead of enabling the interrupts.

  or
- Ensure the above mentioned flag-clearing events occur after a time delay of 6 CPU clock cycles after the interrupt requests occur and are accepted.

SYS8 *System Module*

Function Sporadic device crash on LPMx

Description If an interrupt occurs within 5 ns of the last rising edge of the MCLK prior to entering a low-power mode (LPMx) the flash controller experiences a glitch that latches the last address on the address bus. This is the address that contains the opcode following the entry to LPMx.

When the core then requests the address for the interrupt handler routine, the Flash instead returns the the data at the address that was latched. This data is interpreted as the address for the interrupt handler, resulting in errant code execution and an eventual reset condition.

Workaround None

SYS9 *System Module*

Function UNMI ISR executes twice

Description If a running User NMI interrupt service routine is interrupted by a SYSNMI interrupt before the respective USERNMI flag is cleared (NMIIFG, OFIFG, or ACCVIFG) the SYSNMI ISR will be executed, the UNMI ISR will complete, then the UNMI ISR will be executed a second time.

Workaround A common NMI handler can be written where the SNMI and UNMI Interrupt Vectors point to the same function. This common NMI handler must manually clear both types of pending NMI sources (User and System) to avoid double execution of the UNMI ISR.

TA23 *Timer_A Module*

Function TAxR read can be corrupted when TAxR = TAxCCR0

Description When a timer in Up mode is stopped and the counter register (TAxR) is equal to the TAxCCR0 value, a read of the TAR register may return an unexpected result.

Workaround

- Use Up/Down mode instead of Up mode.
  or
- In Up mode, use the timer interrupt instead of halting the counter and reading out the value in TAxR

TB20 *Timer_B Module*

Function ACLK cannot be used as an internal capture input

Description The capture compare input of Timer_B7, CCI6B, is not connected internally to ACLK, as in previous device families.

Workaround ACLK can be used indirectly by exporting the ACLK signal through its respective output port pin and feeding it back into TB6.

TB21 *Timer_B Module*

Function Timer_B7 outputs not set to Hi-Z state

Description The Timer_B7 TBOUTH pin function does not work as described in the user's guide. When the TBOUTH pin is pulled high, the Timer_B7 outputs do not enter a Hi-Z state.

Workaround None

UCS1 *Universal Clock System Module*

Function WDT sourced by VLO is not halted at breakpoint condition

Description When using VLO Clock as WDT source clock, the WDT does not stop in a Breakpoint condition. The VLO clock is not controllable in Debug mode (no standard MSP430 module source clock), resulting in unexpected resets or counter values from the WDT.

Workaround Do not use the VLO clock as the WDT source for debug.

UCS2 *Universal Clock System Module*

Function LPM modes not entered properly when MCLK < ACLK

Description When MCLK is sourced by XT1, XT2, or REFO clock sources and MCLK is slower than ACLK, the low-power modes are not entered properly.

This applies when the MCLK clock divider is greater than the ACLK clock divider setting.

Workaround None

UCS4 *Universal Clock System Module*

Function Non-monotonic behavior of DCO

Description When FLL modulation is enabled, the modification of DCO & MOD bits in the UCSCTL0 register can result in an unexpected increment or decrement of the DCO tap. This leads to the non-monotonic behavior of the DCO.

Workaround Write the DCO and MOD values when modulation is disabled (DISMOD = 1).

```
BIS.W  #DISMOD, &UCSCTL1
MOV.W  #-YOUR_DCO_MOD_SETTING-, &UCSCTL0
BIC.W  #DISMOD, &UCSCTL1
```

UCS5 *Universal Clock System Module*

Function Results are not updated for two writes in one clock cycle

Description During FLL operation with REFO as the reference clock, if two subsequent writes occur to the UCSCTL0-2 registers within one cycle of the DCO, the results are not updated within the FLL. Similarly, if two subsequent writes are performed to the UCSCTL3 register within one cycle of FLLREFCLK, the results are not updated.

Workaround When writing to the UCSCTL0-2 registers, drive the CPU from a clock derived from the DCO. Avoid making subsequent writes, or insert a delay loop between the accesses to prevent them from occurring within the same FLLREFCLK cycle.

UCS6 *Universal Clock System Module*

Function USCI source clock does not turn off in LPM3/4 when UART is idle

Description The USCI clock source (ACLK/SMCLK) remains enabled in LPM3 and LPM4 when the USCI is configured in UART mode and the communication is idle (UCSWRST = 0 but no TX or RX currently executing). This is contrary to the expected automatic clock activation described in the User Guide and can lead to higher current consumption in low-power modes, depending on the oscillator that feeds ACLK / SMCLK.

Workaround Use the oscillator that is already active in LPM3 (ACLK) to source the USCI and utilize the low-power baud rate generator (UCOS16 = 0). For UART baud rates where a fast SMCLK sourced by the internal DCO is required use LPM0 instead of LPM3.

UCS7 *Universal Clock System Module*

Function DCO drifts for short periods of time when exiting active from ISRs

Description The FLL uses two rising edges of the reference clock to compare against the DCO frequency and decide on the required modifications to the DCOx and MODx bits. If a program flow exits from LPM2, LPM3, or LPM4 into the main loop for a period of time shorter than 3 x reference clock periods the FLL may drift.

This occurs because the FLL immediately begins comparing an active DCO with its reference clock and making the respective modifications to the DCOx and MODx bits. If the FLL is not given enough time to capture a full reference clock cycle (2 x reference clock periods) and adjust accordingly (1 x reference clock period), the main loop will slowly cause the DCO to drift.

Workaround If regulation is enabled insert a wait loop that is 3 x reference clock periods in length after wake-up from LPM2, LPM3, or LPM4. To save on power budget, the 3 x reference clock periods could also be spent in LPM0 with a TimerA or TimerB wake-up. The FLL and DCO are still active in LPM0.

Note that the FLL is not active in interrupt service routines. The drift may only occur in the main loop.

USCI26 *USCI Module*

Function t_{buf} parameter violation in I^2C multi-master mode

Description In multi-master I^2C systems, the timing parameter t_{buf} (bus free time between a stop condition and the following start) is not ensured to match the I^2C specification of 4.7 µs in standard mode and 1.3 µs in fast mode. If the UCTXSTT bit is set during a running I^2C transaction, the USCI module waits and issues the start condition on bus release, causing the violation to occur.

NOTE: It is recommended to check if UCBBUSY bit is cleared before setting UCTXSTT = 1.

Workaround None

WDG4 *Watchdog Module*

Function The WDT failsafe can be disabled

Description The UCS is capable of masking clock requests (ACLK, SMCLK, MCLK) from peripheral modules; see request enable (REQEN) bits in the UCS control register, UCSCTL8.

The clock request logic of the UCS is used by the WDT module to ensure a fail-safe clock source in all low-power modes. Therefore, de-asserting the request enable bit of the watchdog clock source (xCLKREQEN = 0) allows the respective clock to be disabled upon entry into a low-power mode. Without an active clock source, the WDT timer stops incrementing and a watchdog event does not occur.

Workaround None

Appendix A Prior Versions

None

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

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

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI 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 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. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Broadband	www.ti.com/broadband
DSP	dsp.ti.com	Digital Control	www.ti.com/digitalcontrol
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Military	www.ti.com/military
Logic	logic.ti.com	Optical Networking	www.ti.com/opticalnetwork
Power Mgmt	power.ti.com	Security	www.ti.com/security
Microcontrollers	microcontroller.ti.com	Telephony	www.ti.com/telephony
RFID	www.ti-rfid.com	Video & Imaging	www.ti.com/video
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Wireless	www.ti.com/wireless

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