Errata

MSP430G2955 Device Erratasheet

1 Functional Errata Revision History

Errata impacting device's operation, function or parametrics.
✓ The check mark indicates that the issue is present in the specified revision.

<table>
<thead>
<tr>
<th>Errata Number</th>
<th>Rev A</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCL12</td>
<td>✓</td>
</tr>
<tr>
<td>SYS15</td>
<td>✓</td>
</tr>
<tr>
<td>TA12</td>
<td>✓</td>
</tr>
<tr>
<td>TA16</td>
<td>✓</td>
</tr>
<tr>
<td>TA21</td>
<td>✓</td>
</tr>
<tr>
<td>TAB22</td>
<td>✓</td>
</tr>
<tr>
<td>TAB26</td>
<td>✓</td>
</tr>
<tr>
<td>TB2</td>
<td>✓</td>
</tr>
<tr>
<td>TB16</td>
<td>✓</td>
</tr>
<tr>
<td>TB24</td>
<td>✓</td>
</tr>
<tr>
<td>USCI20</td>
<td>✓</td>
</tr>
<tr>
<td>USCI22</td>
<td>✓</td>
</tr>
<tr>
<td>USCI23</td>
<td>✓</td>
</tr>
<tr>
<td>USCI24</td>
<td>✓</td>
</tr>
<tr>
<td>USCI25</td>
<td>✓</td>
</tr>
<tr>
<td>USCI26</td>
<td>✓</td>
</tr>
<tr>
<td>USCI29</td>
<td>✓</td>
</tr>
<tr>
<td>USCI30</td>
<td>✓</td>
</tr>
<tr>
<td>USCI34</td>
<td>✓</td>
</tr>
<tr>
<td>USCI35</td>
<td>✓</td>
</tr>
<tr>
<td>USCI40</td>
<td>✓</td>
</tr>
<tr>
<td>XOSC5</td>
<td>✓</td>
</tr>
</tbody>
</table>

2 Preprogrammed Software Errata Revision History

Errata impacting pre-programmed software into the silicon by Texas Instruments.
✓ The check mark indicates that the issue is present in the specified revision.
The device doesn't have Software in ROM errata.

3 Debug only Errata Revision History

Errata only impacting debug operation.
✓ The check mark indicates that the issue is present in the specified revision.
The device doesn't have Debug errata.
4 Fixed by Compiler Errata Revision History

Errata completely resolved by compiler workaround. Refer to specific erratum for IDE and compiler versions with workaround.

✓ The check mark indicates that the issue is present in the specified revision.

<table>
<thead>
<tr>
<th>Errata Number</th>
<th>Rev A</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU4</td>
<td>✓</td>
</tr>
</tbody>
</table>

Refer to the following MSP430 compiler documentation for more details about the CPU bugs workarounds.

**TI MSP430 Compiler Tools (Code Composer Studio IDE)**
- MSP430 Optimizing C/C++ Compiler: Check the --silicon_errata option
- MSP430 Assembly Language Tools

**MSP430 GNU Compiler (MSP430-GCC)**
- MSP430 GCC Options: Check -msilicon-errata= and -msilicon-errata-warn= options
- MSP430 GCC User's Guide

**IAR Embedded Workbench**
- IAR workarounds for msp430 hardware issues
## 5 Package markings

### DA38

**TSSOP (DA), 38 Pin**

```
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNNNNNG4</td>
<td>Die revision</td>
</tr>
<tr>
<td>MSP430™ REV #</td>
<td>Pin 1 location</td>
</tr>
<tr>
<td>Gxxxx</td>
<td>Lot trace code</td>
</tr>
</tbody>
</table>
```

### RHA40

**QFN (RHA), 40 Pin**

```
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ø</td>
<td>Die revision</td>
</tr>
<tr>
<td>MSP430™</td>
<td>Pin 1 location</td>
</tr>
<tr>
<td>Gxxxx</td>
<td>Lot trace code</td>
</tr>
<tr>
<td>TI #NNN</td>
<td>Lot trace code</td>
</tr>
<tr>
<td>NNNN G4</td>
<td>Lot trace code</td>
</tr>
</tbody>
</table>
```
Detailed Bug Description

BCL12  BCS Module

Category  Functional
Function  Switching RSELx or modifying DCOCCTL can cause DCO dead time or a complete DCO stop
Description  After switching RSELx bits (located in register BCSCTL1) from a value of >13 to a value of <12 OR from a value of <12 to a value of >13, the resulting clock delivered by the DCO can stop before the new clock frequency is applied. This dead time is approximately 20 us. In some instances, the DCO may completely stop, requiring a power cycle.

Furthermore, if all of the RSELx bits in the BSCTL1 register are set, modifying the DCOCCTL register to change the DCOx or the MODx bits could also result in DCO dead time or DCO hang up.

Workaround
- When switching RSEL from >13 to <12, use an intermediate frequency step. The intermediate RSEL value should be 13.

<table>
<thead>
<tr>
<th>Current RSEL</th>
<th>Target RSEL</th>
<th>Recommended Transition Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>Switch directly to target RSEL</td>
</tr>
<tr>
<td>14 or 15</td>
<td>13</td>
<td>Switch directly to target RSEL</td>
</tr>
<tr>
<td>14 or 15</td>
<td>0 to 12</td>
<td>Switch to 13 first, and then to target RSEL (two step sequence)</td>
</tr>
<tr>
<td>0 to 13</td>
<td>0 to 12</td>
<td>Switch directly to target RSEL</td>
</tr>
</tbody>
</table>

AND
- When switching RSEL from <12 to >13 it's recommended to set RSEL to its default value first (RSEL = 7) before switching to the desired target frequency.

AND
- In case RSEL is at 15 (highest setting) it's recommended to set RSEL to its default value first (RSEL = 7) before accessing DCOCCTL to modify the DCOx and MODx bits. After the DCOCCTL register modification the RSEL bits can be manipulated in an additional step.

In the majority of cases switching directly to intermediate RSEL steps as described above will prevent the occurrence of BCL12. However, a more reliable method can be implemented by changing the RSEL bits step by step in order to guarantee safe function without any dead time of the DCO.

Note that the 3-step clock startup sequence consisting of clearing DCOCCTL, loading the BCSCTL1 target value, and finally loading the DCOCCTL target value as suggested in the in the "TLV Structure" chapter of the MSP430x2xx Family User's Guide is not affected by BCL12 if (and only if) it is executed after a device reset (PUC) prior to any other modifications being made to BCSCTL1 since in this case RSEL still is at its default value of 7. However any further changes to the DCOx and MODx bits will require the consideration of the workaround outlined above.

CPU4  CPU Module

Category  Compiler-Fixed
Function  PUSH #4, PUSH #8CPU4 - Bug
Description  The single operand instruction PUSH cannot use the internal constants (CG) 4 and 8.
The other internal constants (0, 1, 2, -1) can be used. The number of clock cycles is different:

- **PUSH #CG** uses address mode 00, requiring 3 cycles, 1 word instruction
- **PUSH #4/#8** uses address mode 11, requiring 5 cycles, 2 word instruction

### Workaround

Refer to the table below for compiler-specific fix implementation information.

<table>
<thead>
<tr>
<th>IDE/Compiler</th>
<th>Version Number</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAR Embedded Workbench</td>
<td>IAR EW430 v2.x until v6.20</td>
<td>User is required to add the compiler flag option below: --hw_workaround=CPU4</td>
</tr>
<tr>
<td>IAR Embedded Workbench</td>
<td>IAR EW430 v6.20 or later</td>
<td>Workaround is automatically enabled</td>
</tr>
<tr>
<td>TI MSP430 Compiler Tools (Code Composer Studio)</td>
<td>v1.1 or later</td>
<td></td>
</tr>
<tr>
<td>MSP430 GNU Compiler (MSP430-GCC)</td>
<td>MSP430-GCC 4.9 build 167 or later</td>
<td></td>
</tr>
</tbody>
</table>

### SYS15

**SYS Module**

**Category**

Functional

**Function**

LPM3 and LPM4 currents exceed specified limits

**Description**

LPM3 and LPM4 currents may exceed specified limits if the SMCLK source is switched from DCO to VLO or LFXT1 just before the instruction to enter LPM3 or LPM4 mode.

**Workaround**

After clock switching, a delay of at least four new clock cycles (VLO or LFXT1) must be implemented to complete the clock synchronization before going into LPM3 or LPM4.

### TA12

**TIMER_A Module**

**Category**

Functional

**Function**

Interrupt is lost (slow ACLK)

**Description**

Timer_A counter is running with slow clock (external TACLK or ACLK) compared to MCLK. The compare mode is selected for the capture/compare channel and the CCRx register is incremented by one with the occurring compare interrupt (if TAR = CCRx). Due to the fast MCLK the CCRx register increment (CCRx = CCRx+1) happens before the Timer_A counter has incremented again. Therefore the next compare interrupt should happen at once with the next Timer_A counter increment (if TAR = CCRx + 1). This interrupt gets lost.

**Workaround**

Switch capture/compare mode to capture mode before the CCRx register increment. Switch back to compare mode afterwards.

### TA16

**TIMER_A Module**

**Category**

Functional

**Function**

First increment of TAR erroneous when IDx > 00

**Description**

The first increment of TAR after any timer clear event (POR/TACLR) happens immediately following the first positive edge of the selected clock source (INCLK, DCO, VLO).
SMCLK, ACLK or TACLK). This is independent of the clock input divider settings (ID0, ID1). All following TAR increments are performed correctly with the selected IDx settings.

Workaround
None

**TA21**  
**TIMER_A Module**

Category  
Functional

Function  
TAIFG Flag is erroneously set after Timer A restarts in Up Mode

Description
In Up Mode, the TAIFG flag should only be set when the timer counts from TACCR0 to zero. However, if the Timer A is stopped at TAR = TACCR0, then cleared (TAR=0) by setting the TACLK bit, and finally restarted in Up Mode, the next rising edge of the TACLK will erroneously set the TAIFG flag.

Workaround
None.

**TAB22**  
**TIMER_A/TIMER_B Module**

Category  
Functional

Function  
Timer_A/Timer_B register modification after Watchdog Timer PUC

Description
Unwanted modification of the Timer_A/Timer_B registers TACTL/TBCTL and TAIV/TBIV can occur when a PUC is generated by the Watchdog Timer(WDT) in Watchdog mode and any Timer_A/Timer_B counter register TACCRx/TBCCRx is incremented/decremented (Timer_A/Timer_B does not need to be running).

Workaround
Initialize TACTL/TBCTL register after the reset occurs using a MOV instruction (BIS/BIC may not fully initialize the register). TAIV/TBIV is automatically cleared following this initialization.

Example code:
MOV.W #VAL, &TACTL
or
MOV.W #VAL, &TBCTL
Where, VAL=0, if Timer is not used in application otherwise, user defined per desired function.
TAB26  
**TIMER_A/TIMER_B Module**

**Category**  Functional

**Function**  Reading TxIV register using a CMP instruction could result in a missed interrupt

**Description**  If more than one capture/compare interrupt flags (TxCCTLx.CCIFG) are set and if a CMP instruction is used to read the TxIV register when servicing the interrupt, all flags may be erroneously cleared instead of just the highest priority pending CCIFG. Following this the TxIV register reads a value of 0 and the application could miss a valid interrupt.

**Workaround**  Do not use the CMP instruction to read the TxIV register. Instead use the ADD instruction. For example add.w &TAIV,PC

---

TB2  
**TIMER_B Module**

**Category**  Functional

**Function**  Interrupt is lost (slow ACLK)

**Description**  Timer_B counter is running with slow clock (external TBCLK or ACLK) compared to MCLK. The compare mode is selected for the capture/compare channel and the CCRx register is incremented by 1 with the occurring compare interrupt (if TBR = CCRx). Due to the fast MCLK, the CCRx register increment (CCRx = CCRx + 1) happens before the Timer_B counter has incremented again. Therefore, the next compare interrupt should happen at once with the next Timer_B counter increment (if TBR = CCRx + 1). This interrupt is lost.

**Workaround**  Switch capture/compare mode to capture mode before the CCRx register increment. Switch back to compare mode afterward.

---

TB16  
**TIMER_B Module**

**Category**  Functional

**Function**  First increment of TBR erroneous when IDx > 00

**Description**  The first increment of TBR after any timer clear event (POR/TBCLR) happens immediately following the first positive edge of the selected clock source (INCLK, SMCLK, ACLK, or TBCLK). This is independent of the clock input divider settings (ID0, ID1). All following TBR increments are performed correctly with the selected IDx settings.

**Workaround**  None

---

TB24  
**TIMER_B Module**

**Category**  Functional

**Function**  TBIFG Flag is erroneously set after Timer B restarts in Up Mode

**Description**  In Up Mode, the TBIFG flag should only be set when the timer resets from TBCCR0 to zero. However, if the Timer B is stopped at TBR = TBCCR0, then cleared (TBR=0) by setting the TBCLR bit, and finally restarted in Up Mode, the next rising edge of the TBCLK will erroneously set the TBIFG flag.
USCI20  
**USCI Module**  
**Category** Functional  
**Function** I2C Mode Multi-master transmitter issue  
**Description** When configured for I2C master-transmitter mode, and used in a multi-master environment, the USCI module can cause unpredictable bus behavior if all of the following four conditions are true:  
1 - Two masters are generating SCL  
And  
2 - The slave is stretching the SCL low phase of an ACK period while outputting NACK on SDA  
And  
3 - The slave drives ACK on SDA after the USCI has already released SCL, and then the SCL bus line gets released  
And  
4 - The transmit buffer has not been loaded before the other master continues communication by driving SCL low  
The USCI will remain in the SCL high phase until the transmit buffer is written. After the transmit buffer has been written, the USCI will interfere with the current bus activity and may cause unpredictable bus behavior.  
**Workaround**  
1 - Ensure that slave doesn't stretch the SCL low phase of an ACK period  
Or  
2 - Ensure that the transmit buffer is loaded in time  
Or  
3 - Do not use the multi-master transmitter mode

USCI22  
**USCI Module**  
**Category** Functional  
**Function** I2C Master Receiver with 10-bit slave addressing  
**Description** Unexpected behavior of the USCI_B can occur when configured in I2C master receive
mode with 10-bit slave addressing under the following conditions:
1) The USCI sends first byte of slave address, the slave sends an ACK and when
second address byte is sent, the slave sends a NACK.
2) Master sends a repeat start condition (If UCTXSTT=1).
3) The first address byte following the repeated start is acknowledged.

However, the second address byte is not sent, instead the Master incorrectly starts to
receive data and sets UCBxRXIFG=1.

**Workaround**

Do not use repeated start condition instead set the stop condition UCTXSTP=1 in the
NACK ISR prior to the following start condition (USTXSTT=1).

---

**USCI23**

**USCI Module**

**Category**
Functional

**Function**
UART transmit mode with automatic baud rate detection

**Description**
Erroneous behavior of the USCI_A can occur when configured in UART transmit mode
with automatic baud rate detection. During transmission if a “Transmit break” is initiated
(UCTXBRK=1), the USCI_A will not deliver a stop bit of logic high, instead, it will send a
logic low during the subsequent synch period.

**Workaround**

1) Follow User's Guide instructions for transmitting a break/synch field following
UCSWRST=1.
Or,
2) Set UCTXBRK=1 before an active transmission, i.e. check for bit UCBUSY=0 and
then set UCTXBRK=1.

---

**USCI24**

**USCI Module**

**Category**
Functional

**Function**
Incorrect baud rate information during UART automatic baud rate detection mode

**Description**
Erroneous behavior of the USCI_A can occur when configured in UART mode with
automatic baud rate detection. After automatic baud rate measurement is complete, the
UART updates UCAxBR0 and UCAxBR1. Under Oversampling mode (UCOS16=1), for
baud rates that should result in UCAxBRx=0x0002, the UART incorrectly reports it as
UCAxBRx=0x5555.

**Workaround**

When break/synch is detected following the automatic baud rate detection, the flag
UCBRK flag is set to 1. Check if UCAxBRx=0x55555 and correct it to 0x0002.

---

**USCI25**

**USCI Module**

**Category**
Functional

**Function**
TXIFG is not reset when NACK is received in I2C mode

**Description**
When the USCI_B module is configured as an I2C master transmitter the TXIFG is not
reset after a NACK is received if the master is configured to send a restart (UCTXSTT=1
& UCTXSTP=0).
**USCI26**  
**USCI Module**

**Category**  
Functional

**Function**  
Tbuf parameter violation in I2C multi-master mode

**Description**  
In multi-master I2C systems the timing parameter Tbuf (bus free time between a stop condition and the following start) is not guaranteed to match the I2C specification of 4.7us in standard mode and 1.3us in fast mode. If the UCTXSTT bit is set during a running I2C 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**  
Reset TXIFG in software within the NACKIFG interrupt service routine

---

**USCI29**  
**USCI Module**

**Category**  
Functional

**Function**  
Timing of USCI I2C interrupts may result in call to a reserved ISR location

**Description**  
When certain USCI I2C interrupt flags (IFG) are set and an automatic flag-clearing event on the I2C bus occurs, the device makes a call to the TRAPINT interrupt vector. This will only happen if 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:

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

**Workaround**  
1. Poll the affected flags instead of enabling the interrupts.
2. Define an ISR for the interrupt vector TRAPINT. If the failure condition occurs; a call to the TRAPINT ISR is made. After the interrupt is serviced, the device returns to the application code and continues execution.

Include the following ISR definition in the application code.

```c
#pragma vector= TRAPINT_VECTOR
__interrupt void TRAPINT_ISR(void)
{
   __no_operation();
}
```

---

**USCI30**  
**USCI Module**

**Category**  
Functional

**Function**  
I2C mode master receiver / slave receiver
When the USCI I2C module is configured as a receiver (master or slave), it performs a double-buffered receive operation. In a transaction of two bytes, once the first byte is moved from the receive shift register to the receive buffer the byte is acknowledged and the state machine allows the reception of the next byte.

If the receive buffer has not been cleared of its contents by reading the UCBxRXBUF register while the 7th bit of the following data byte is being received, an error condition may occur on the I2C bus. Depending on the USCI configuration the following may occur:

1) If the USCI is configured as an I2C master receiver, an unintentional repeated start condition can be triggered or the master switches into an idle state (I2C communication aborted). The reception of the current data byte is not successful in this case.

2) If the USCI is configured as I2C slave receiver, the slave can switch to an idle state stalling I2C communication. The reception of the current data byte is not successful in this case. The USCI I2C state machine will notify the master of the aborted reception with a NACK.

Note that the error condition described above occurs only within a limited window of the 7th bit of the current byte being received. If the receive buffer is read outside of this window (before or after), then the error condition will not occur.

Workaround

a) The error condition can be avoided altogether by servicing the UCBxRXIFG in a timely manner. This can be done by (a) servicing the interrupt and ensuring UCBxRXBUF is read promptly or (b) Using the DMA to automatically read bytes from receive buffer upon UCBxRXIFG being set.

OR

b) In case the receive buffer cannot be read out in time, test the I2C clock line before the UCBxRXBUF is read out to ensure that the critical window has elapsed. This is done by checking if the clock line low status indicator bit UCSCLLOW is set for atleast three USCI bit clock cycles i.e. 3 X t(BitClock).

Note that the last byte of the transaction must be read directly from UCBxRXBUF. For all other bytes follow the workaround:

Code flow for workaround

1) Enter RX ISR for reading receiving bytes
2) Check if UCSCLLOW.UCBxSTAT == 1
3) If no, repeat step 2 until set
4) If yes, repeat step 2 for a time period > 3 x t (BitClock) where t (BitClock) = 1/ f (BitClock)
5) If window of 3 x t(BitClock) cycles has elapsed, it is safe to read UCBxRXBUF

USCI34

USCI Module

Category  Functional
Function  I2C multi-master transmit may lose first few bytes.
Description  In an I2C multi-master system (UCMM =1), under the following conditions:
(1) the master is configured as a transmitter (UCTR =1) AND
(2) the start bit is set (UCTXSTT =1);
if the I2C bus is unavailable, then the USCI module enters an idle state where it waits
and checks for bus release. While in the idle state it is possible that the USCI master updates its TXIFG based on clock line activity due to other master/slave communication on the bus. The data byte(s) loaded in TXBUF while in idle state are lost and transmit pointers initialized by the user in the transmit ISR are updated incorrectly.

**Workaround**

Verify that the START condition has been sent (UCTXSTT =0) before loading TXBUF with data.

Example:

```
#pragma vector = USCIAB0TX_VECTOR
__interrupt void USCIAB0TX_ISR(void)
{
    // Workaround for USCI34
    if(UCB0CTL1&UCTXSTT)
    {
        // TXData = pointer to the transmit buffer start
        // PTxData = pointer to transmit in the ISR
        PTxData = TXData; // restore the transmit buffer pointer if the Start bit is set
    }
    //
    if(IFG2&UCB0TXIFG)
    {
        if (PTxData<=PTxDataEnd) // Check TX byte counter
        {
            UCB0TXBUF = *PTxData++; // Load TX buffer
        }
        else
        {
            UCB0CTL1 |= UCTXSTP; // I2C stop condition
            IFG2 &= ~UCB0TXIFG; // Clear USCI_B0 TX int flag
            __bic_SR_register_on_exit(CPUOFF); // Exit LPM0
        }
    }
}
```

**USCI35**

**USCI Module**

**Category**

Functional

**Function**

Violation of setup and hold times for (repeated) start in I2C master mode

**Description**

In I2C master mode, the setup and hold times for a (repeated) START, \( t_{SU,STA} \) and \( t_{HD,STA} \) respectively, can be violated if SCL clock frequency is greater than 50kHz in standard mode (100kbps). As a result, a slave can receive incorrect data or the I2C bus can be stalled due to clock stretching by the slave.
<table>
<thead>
<tr>
<th><strong>USCI40</strong> USCI Module</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Category</strong></td>
</tr>
<tr>
<td><strong>Function</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Workaround</strong></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>XOSC5</strong> XOSC Module</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Category</strong></td>
</tr>
<tr>
<td><strong>Function</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Workaround</strong></td>
</tr>
</tbody>
</table>
7 Document Revision History

Changes from device specific erratasheet to document Revision A.
1. Errata TB24 was added to the errata documentation.
2. USCI29 Workaround was updated.
3. USCI29 Function was updated.
4. USCI29 Description was updated.

Changes from document Revision A to Revision B.
1. Errata USCI35 was added to the errata documentation.

Changes from document Revision B to Revision C.
1. Package Markings section was updated.

Changes from document Revision C to Revision D.
1. Errata USCI40 was added to the errata documentation.

Changes from document Revision D to Revision E.
1. TA21 Description was updated.

Changes from document Revision E to Revision F.
1. Errata TAB26 was added to the errata documentation.

Changes from document Revision F to Revision G.
1. Function for CPU4 was updated.
2. Workaround for CPU4 was updated.

Changes from document Revision G to Revision H.
1. Erratasheet format update.
2. Added errata category field to "Detailed bug description" section

Changes from document Revision H to Revision I.
1. USCI34 was added to the errata documentation.

Changes from document Revision I to Revision J.
1. Description for TB24 was updated.
IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to TI's Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2019, Texas Instruments Incorporated