Application Report
SPNA216–January 2016

TMS570LC4357 and RM57L843 On-Chip Temperature Sensor Measurements

David Livingston
Bob Crosby

ABSTRACT

This application report demonstrates how to use the on-chip temperature sensors for junction temperature monitoring. Junction temperature monitoring is the only recommended use for these on-chip temperature sensors.

Project collateral and source code discussed in this application report were generated with HalCoGen 4.05.01 for the TMS570LC43x launchpad and can be downloaded from the following URL: http://www.ti.com/lit/zip/spna216.

Contents

1 Introduction 2
2 Temperature Sensor Information 2
3 Example 5
4 References 9

List of Figures

1 TI OTP Bank 0 Temperature Sensor 1 Calibration Information (F008 0310h-F008 031Fh) 3
2 TI OTP Bank 0 Temperature Sensor 2 Calibration Information (F008 0320h-F008 032Fh) 3
3 TI OTP Bank 0 Temperature Sensor 3 Calibration Information (F008 0330h-F008 033Fh) 3
4 TMS570LC4357B Power Temperature Cycling Results 4

List of Tables

1 Temperature Sensor Selection 2
2 TI OTP Bank 0 Temperature Sensor Calibration Information Field Descriptions 3

1 Introduction

The TMS570LC4357 and RM57L843 have three on-chip temperature sensors for monitoring the junction temperature, allowing the application to pro-actively manage device utilization so as not to violate the maximum junction temperature specification.

The measurements of the three on-die temperature sensors are routed to a MibADC for conversion into digital values. The CPU can read out the digital values and compare with the calibrated temperature value stored in the device's one-time programmable memory (OTP). If the measured temperature exceeds the specification, the application can choose to take actions such as log the event or reduce the junction temperature by turning off peripherals or slowing down some clocks.

2 Temperature Sensor Information

2.1 Sensor Connection

There are three instances of temperature sensors in the device. The temperature sensors are connected to a on-chip analog-to-digital converter (ADC) through the Input/Output Multiplexing Module (IOMM). Temperature sensor 1 is multiplexed with AD1IN[31]. Temperature sensor 2 is connected to AD2 channel 31. Temperature sensor 3 is connected to AD2 channel 30. By default, the three temperature sensors are disabled. The three temperature sensors are enabled by clearing PINMMR174[24] to low. By default pin multiplexing is locked and can be unlocked by writing the unlock values to the kicker0 and kicker1 registers as demonstrated in the example code.

Table 1. Temperature Sensor Selection

Decode (1)	Select
AD1CHNSEL(31) = 1 PINMMR173(16) = 0 PINMMR173(17) = 1	Temp Sensor 1
AD1CHNSEL(31) = 1 PINMMR173(16) = 1 PINMMR173(17) = 0	ADC1IN[31]
AD2CHNSEL(31) = 1 PINMMR173(24) = 0 PINMMR173(25) = 1	Temp Sensor 2
AD2CHNSEL(31) = 1 PINMMR173(24) = 1 PINMMR173(25) = 0	No Connection
AD2CHNSEL(30) = 1 PINMMR174(0) = 0 PINMMR174(1) = 1	Temp Sensor 3
AD2CHNSEL(30) = 1 PINMMR174(0) = 1 PINMMR174(1) = 0	No Connection

(1) AD1CHNSEL is configured inside the MibADC1 module. AD2CHNSEL is configured inside the MibADC2 module.

2.2 Temperature Sensor Location

Temperature sensors 1 and 2 are near the ADCs, opposite the pin A1 corner. Temperature sensor 3 is near the center of the device. In this example, the three sensors gave close to the same results. This is as expected because the thermal conductivity of silicon is high.

2.3 Temperature Sensor Calibration

The temperature sensors are designed to be used with software calibration. To support software calibration of the temperature sensors, TI has recorded the value returned by recording the reading of each sensor at different junction temperatures. The measurements were done while the device is still in wafer form and the junction temperature can be well controlled by the wafer prober. The values are recorded in the TI OTP starting at location F008 0310h as shown in Figure 1 and described in Table 2. The values recorded were measured with ADREFHI equal to 3.30 V. It is important to have ADREFHI at 3.30V and ADREFLO at 0V, or to adjust the results from reading the temp sensor to compensate for the difference. A difference on ADREF of 5% can result in the calculations being off by 22°C at 125°C junction temperature.

Figure 1. TI OTP Bank 0 Temperature Sensor 1 Calibration Information (F008 0310h-F008 031Fh)

0x00		0x04		0x08		0x0C	
S1TEMP1VAL	S1TEMP1	S1TEMP2VAL	S1TEMP2	S1TEMP3VAL	S1TEMP3	0xFFFF	0xFFFF
R	R	R	R	R	R	R	R

LEGEND: R = Read only

Figure 2. TI OTP Bank 0 Temperature Sensor 2 Calibration Information (F008 0320h-F008 032Fh)

0x00		0x04		0x08		0x0C	
S2TEMP1VAL	S2TEMP1	S2TEMP2VAL	S2TEMP2	S2TEMP3VAL	S2TEMP3	0xFFFF	0xFFFF
R	R	R	R	R	R	R	R

LEGEND: R = Read only

Figure 3. TI OTP Bank 0 Temperature Sensor 3 Calibration Information (F008 0330h-F008 033Fh)

0x00		0x04		0x08		0x0C	
S3TEMP1VAL	S3TEMP1	S3TEMP2VAL	S3TEMP2	S3TEMP3VAL	S3TEMP3	0xFFFF	0xFFFF
R	R	R	R	R	R	R	R

LEGEND: R = Read only

Table 2. TI OTP Bank 0 Temperature Sensor Calibration Information Field Descriptions

Address	Width	Field	Description
F008 03x0h [31:16]	16 bits	SxTEMP1VAL	The value read from the ADC for this sensor at the first calibration temperature
F008 03x0h [15:0]	16 bits	SxTEMP1	The temperature in degrees Kelvin
F008 03x4h [31:16]	16 bits	SxTEMP2VAL	The value read from the ADC for this sensor at the second calibration temperature
F008 03x4h [15:0]	16 bits	SxTEMP2	The temperature in degrees Kelvin
F008 03x8h [31:16]	16 bits	SxTEMP3VAL	The value read from the ADC for this sensor at the third calibration temperature
F008 03x8h [15:0]	16 bits	SxTEMP3	The temperature in degrees Kelvin
F008 03xCh [31:16]	16 bits	0xFFFF	Reserved
F008 03xCh [15:0]	16 bits	0xFFFF	Reserved

2.4 Junction Temperature Monitor Example: Results

To estimate the accuracy of the calibrated temperature sensors, TI ran an experiment with 24 TMS570LC4357BZWT devices. The devices were in a temperature chamber where they soaked at -45°C for five minutes. Then the devices powered up and quickly did a temperature read using an ADC discharge period of 500 ns and a sample period of 1 µs. The results were returned to a PC on a CAN bus. The devices powered down and the temperature chamber incremented by one degree Celcius. The process of five minutes of soak time and then a reading repeated until the last reading was taken with the chamber at 130°C. It was important to do the ADC reading early in the code, even before the PLL initialization, to minimize the impact of the device's self heating. The results of the experimient are shown in Figure 4 below. This example achieved a standard deviation of less than 2.5°C from the temperature chamber.

NOTE: The use of ambient temperatures beyond the specified operating range was done to validate the algorithms used in the calibration routines. TI does not guarantee nor imply proper operation of the parts beyond the specified operating temperature range.

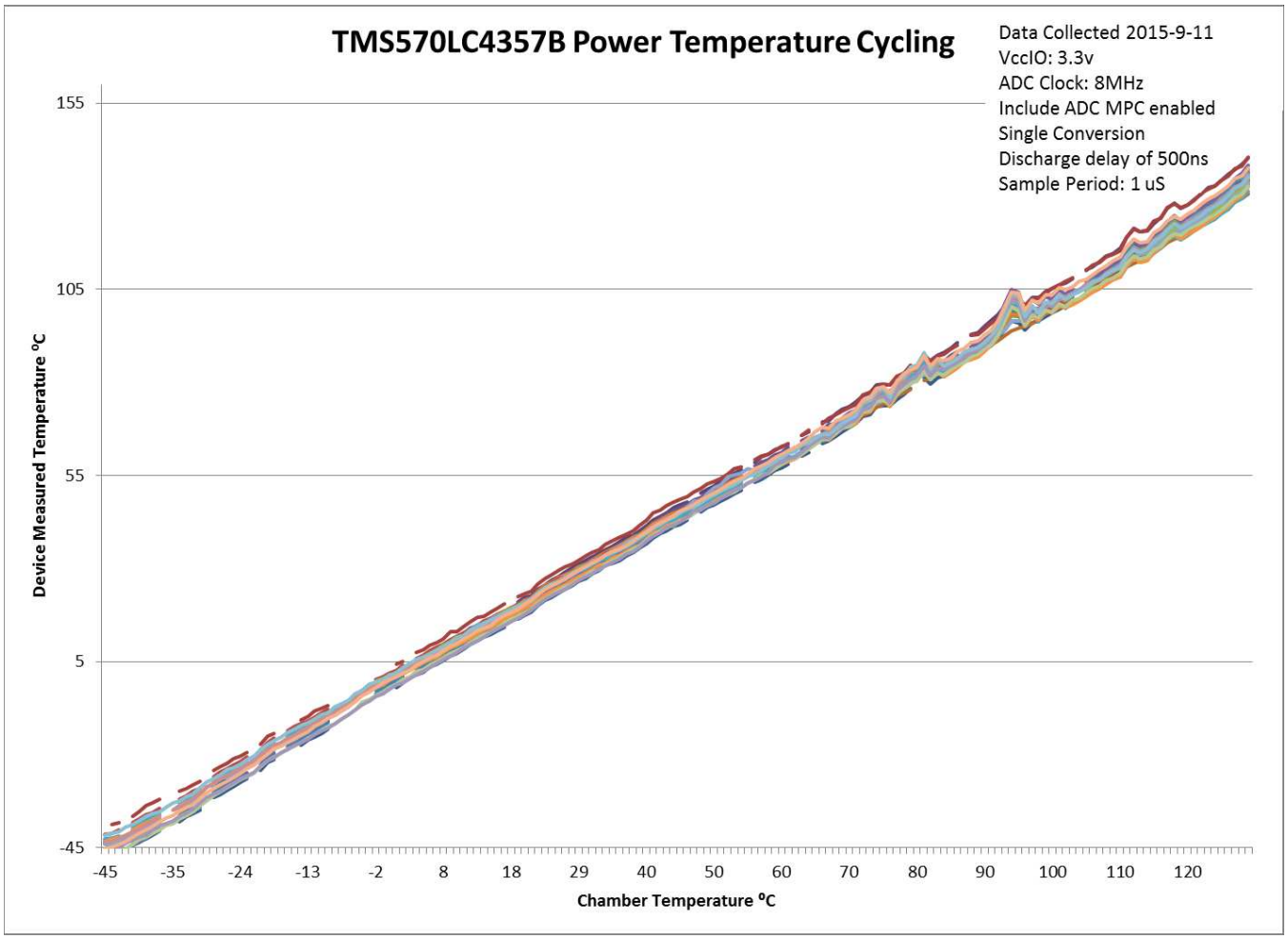

Figure 4. TMS570LC4357B Power Temperature Cycling Results

3 Example

This example is designed to demonstrate how to use the on-chip temperature sensors of the TMS570LC4357 in the TMS570LC43x launchpad. This example was built using HalCoGen version 04.05.01, Code Composer Studio version 6.1.1.00022 and ARM Compiler Tools version 5.2.6.

3.1 Example Code

The entire Code Composer Studio project is available in the ZIP file http://www.ti.com/lit/zip/spna216. All of the files were created by HalCoGen and were not hand modified except for three files. The file HL_sys_main.c was started by HalCoGen but then user code was added to create this example. The files TempSensor.h and TempSensor.c contain the code for the temperature sensor routines.

3.1.1 HL_sys_main.c

This file is provided as an example of how to call the thermistor calibration routine and the thermistor read routine. This example uses temperature sensor 3. Therefore ADC2 group one is used. The group one discharge time and sample time are set in the adcInit() routine. The 500 nS discharge time and 1 µs sample time were setup using the HalCoGen graphical interface.

```
/* Include Files */

#include "HL_sys_common.h"

/* USER CODE BEGIN (1) */
#include "TempSensor.h"
#include "HL_adc.h"
/* USER CODE END */

/** @fn void main(void)
*   @brief Application main function
*   @note This function is empty by default.
*
*   This function is called after startup.
*   The user can use this function to implement the application.
*/

/* USER CODE BEGIN (2) */
#define MAX_JUNCTION_TEMP 150    // Check datasheet for absolute maximum junction temperature
unsigned int ErrorCode;
float MaxTemp=0, CurrentTemp=0;
/* USER CODE END */

void main(void)
{
/* USER CODE BEGIN (3) */
    float JunctionTemp, AverageTemp;
    unsigned int i;

    /*  ADC group 1 was set to 500ns discharge and 1us sample time using HalCoGen */
    /*      these values are not the default values                                */
    adcInit();
    adcMidPointCalibration(adcREG2);

    /* Calculate slope and intercept for sensor 3 (only need to do once) */
    if(thermistor_calibration() == FALSE)
    {
        // Put warning about no temperature calibration data in OTP
        ErrorCode = 1;
    }
    else
    {
        for(;;)
        {
```

```
                if((JunctionTemp=thermistor_read()) < 0.0)
                {
                    /* Put warning about failure                                */
                    /*   Return = -1; ADC2 Group 1 in use                       */
                    /*   Return = -2; Missing calibration values                */
                    ErrorCode = 2;
                    break;
                }
                else
                {
                    // Assumes ADC reference voltage is 3.30V so no scaling is required
                    JunctionTemp = JunctionTemp - 273.15;  // Convert from Kelvin to Celcius
                    CurrentTemp = JunctionTemp; // Saving in static to make it easy to watch
                    if(JunctionTemp > MAX_JUNCTION_TEMP)
                    {
                        // Do four readings to be sure this was not just noise on the ADC voltage
                        AverageTemp = 0.0;
                        for(i = 0; i < 4; i++)
                            AverageTemp += thermistor_read();
                        JunctionTemp = (AverageTemp / 4.0) - 273.15;
                        if(JunctionTemp > MAX_JUNCTION_TEMP)
                        {
                            // Put warning "junction temperature too high" routine here
                            ErrorCode = 3;
                            break;
                        }
                    }
                    if(JunctionTemp > MaxTemp)
                        MaxTemp = JunctionTemp;
                }
            }
        }

    /* USER CODE END */
    }

    /* USER CODE BEGIN (4) */
    /* USER CODE END */
```

3.1.2 TempSensor.h

```
#ifndef __TEMPSENSOR_H__
#define __TEMPSENSOR_H__

extern float thermistor_read(void);
extern bool thermistor_calibration(void);

#endif
```

3.1.3 TempSensor.c

```
#include "HL_sys_common.h"
#include "HL_adc.h"
#include "HL_pinmux.h"
#include "TempSensor.h"

/* Thermistor */
#ifndef __little_endian__
#define __little_endian__  0
#endif

#if __little_endian__
    typedef struct OTP_temperature_calibration_data
    {
        uint16_t Temperature;
        uint16_t AdcValue;
    } OTP_temperature_calibration_data_t;
#else
    typedef struct OTP_temperature_calibration_data
    {
        uint16_t AdcValue;
        uint16_t Temperature;
    } OTP_temperature_calibration_data_t;
#endif

#define THERMISTOR_CAL_DATA        0xF0080310 /* OTP Temperature Sensor Data Location */

typedef struct Thermistor_Calibration
{
    float slope;
    float offset;
    float rsquared;
} Thermistor_CAL_t;


static Thermistor_CAL_t Thermistor_Fit = {0.0, 0.0, 0.0};

/*************************************************************/
/**** Start of Temp Sensor functions                      ***/
/*************************************************************/

/** @fn float thermistor_read(void)
*   @brief read on-chip thermistor 3
*   @note This will return the temperature of thermistor 3 in Kelvin
*
*   This requires adcInit() to be called before to setup ADC2.
*
*   This function will modify the Pin Muxing and ADC2 to read the thermistor,
*   care has been taken to restore modified configurations however the user is
*   responsible for verifying both the Pin Muxing and ADC are configured
*   as desired.
*
*   The returned temperature will need to be scaled by the reference voltage difference
*   Calibration values are taken at nominal voltage 3.30V.
*
*   Kelvin = ReturnValue * (VccADrefHi - VccADrefLow)/3.30V
*    Celsius = Kelvin - 273.15;
*    Fahrenheit = (Kelvin - 273.15) * 1.8 + 32;
*
*/
float thermistor_read()
{    unsigned int value, pinmux_restore, GxSEL_restore;
    float JunctionTempK;
    adcBASE_t *adcreg;
```

```
    /* Select the ADC */
    adcreg = adcREG2;

    if(adcreg->G1SR != 0x00000008 )
        return (-1.0); // Group 1 is being used

    /* Check that we have valid calibration data */
    if(Thermistor_Fit.rsquared == 0.0)
        return (-2.0); //Calibration data missing, must run thermistor_calibration() first

    /* Enable Temperature Sensors in Pin Muxing */

    /* Enable Pin Muxing */
    pinMuxReg->KICKER0 = 0x83E70B13U;
    pinMuxReg->KICKER1 = 0x95A4F1E0U;

    /* Enable Temp Sensor */
    pinMuxReg->PINMUX[174] &= 0xFEFFFFFF;

    /* Connect Sensor 3 - Temperature sensor 3's output is connected to AD2IN[30] */
    pinmux_restore = pinMuxReg->PINMUX[174];
    pinMuxReg->PINMUX[174] = (pinMuxReg->PINMUX[174] & 0xfffffffe) | 0x00000002;

    /* Start Converting, Choose Channel */
    GxSEL_restore = adcreg->GxSEL[1U]; // Save the original value in the channel select register
    adcreg->GxSEL[1U] = 0x40000000;

    /* Poll for end of Conversion */
    while(!(adcreg->G1SR & 1));

    /* Read adc value */
    value = adcreg->GxBUF[1U].BUF0;

    /* Disable Temperature Sensor */
    pinMuxReg->PINMUX[174] |= 0x01000000;

    /* Restore Sensor 3 Pin Muxing */
    pinMuxReg->PINMUX[174] = pinmux_restore;

    /* Disable Pin Muxing */
    pinMuxReg->KICKER0 = 0x00000000U;
    pinMuxReg->KICKER1 = 0x00000000U;

    /* Restore Channel Select Register */
    adcreg->GxSEL[1U] = GxSEL_restore;

    /* Convert ADC value into floating point temp Kelvin */
    JunctionTempK = (((float)value) - Thermistor_Fit.offset) *
            Thermistor_Fit.slope;

    return JunctionTempK;
} // thermistor_read

/** @fn void thermistor_calibration(void)
*   @brief Load the thermister calibration information
*   @note
*/
bool thermistor_calibration(void)
{
    OTP_temperature_calibration_data_t *OTPdataptr;
    int i, cal_data_count=0;
    float slope,offset,sumtemp=0,sumconv=0,sumtempxconv=0,sumtempxtemp=0;
    float cal_adc_code_array[4],avgconv,cal_temperature_array[4],yx=0.0,ya=0.0,ym,yn;

    /* Create pointer to temp sensor 3 calibration data */
    OTPdataptr = (OTP_temperature_calibration_data_t *)(THERMISTOR_CAL_DATA + (2 * 0x10));
```

```
    /* Check for valid calibration data */
    /* Valid codes are 0 to 0xFFF, valid temperatures are 0 to 400 kelvin */
    for(i = 0; i  < 4; i++)
    {

        /* Calculate Slope and Offset for the 4 possible value pairs */
        if((OTPdataptr[i].AdcValue   < 0xFFF) && (OTPdataptr[i].Temperature < 401))
        {
            /* Load valid calibration information */
            cal_temperature_array[i] = (float)(OTPdataptr[i].Temperature);
            cal_adc_code_array[i] = (float)OTPdataptr[i].AdcValue;
            sumtemp += cal_temperature_array[i];
            sumconv += cal_adc_code_array[i];
            sumtempxconv += cal_temperature_array[i] * cal_adc_code_array[i];
            sumtempxtemp += cal_temperature_array[i] * cal_temperature_array[i];
            cal_data_count++;
        }
    }

    /* Calculate slope and Offset for the 4 possible value pairs */
    if(cal_data_count < 2)
        return FALSE;
    else
    {
        slope = (sumtempxtemp * cal_data_count - sumtemp * sumtemp) /
                (sumtempxconv * cal_data_count - sumtemp * sumconv);
        offset = (sumconv - sumtemp / slope) / cal_data_count;
        Thermistor_Fit.slope = slope;
        Thermistor_Fit.offset = offset;
        avgconv = sumconv / cal_data_count;
    }

    /* Calculate R-Squared Value */
    for(i = 0; i < cal_data_count; i++)
    {
        yn = (((cal_temperature_array[i] / slope) + offset) - cal_adc_code_array[i]);
        yx = yx + (yn * yn);
        ym = (avgconv - cal_adc_code_array[i]);
        ya = ya + (ym * ym);
    }
    Thermistor_Fit.rsquared = 1.0 - (yx / ya);

    return TRUE;

} // thermistor_calibrate
```

4 References

- *TMS570LC43x 16/32-Bit RISC Flash Microcontroller Technical Reference Manual* (SPNU563)
- *TMS570LC4357 Hercules™ Microcontroller Data Manual* (SPNS195)
- *RM57L843 16/32-Bit RISC Flash Microcontroller Technical Reference Manual* (SPNU563)
- *RM57L843 Hercules Microcontroller Data Manual* (SPNS195)

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