Application Report
SNAA075A–August 2011–Revised May 2013

Microcontroller Firmware Considerations for the LMP90100 Sensor AFE

ABSTRACT

This application report provides guidance to the firmware designer on how to communicate with the LMP90100 across the SPI interface as implemented on this device.

Contents

List of Figures

1 Introduction

The LMP90100 is an analog front-end (AFE) that is ideally suited for low-noise, precision sensing applications such as industrial temperature transmitters. This product is based on a 24-bit Sigma-Delta ADC and uses the SPI interface to communicate with a local processing IC – most often a microcontroller. The SPI interface is used to read/write the device's control and status registers as well as to retrieve the sensor data. This document provides guidance to the firmware designer on how to communicate with the LMP90100 across the SPI interface as implemented on this device.

This document includes high-level example code. Note that National's software license applies to the included code. The policy appears at the end of this document.

2 SPI Protocol of the LM90100

At the electrical level, the SPI interface of the LMP90100 is very conventional: four wires (CSB, SCLK, SDI, SDO) with typical polarity and phases for the SCLK. For a full description of the SPI interface, see the *LMP90100/LMP90099/LMP90098/LMP90097 Sensor AFE System: Multi-Channel, Low Power 24-Bit Sensor AFE with True Continuous Background Calibration Data Sheet* (SNAS510). The LMP90100, however, provides additional capabilities on its serial interface:

- Since the LMP90100 has its own sampling clock, a Data Ready (DRDYB) signal is provided to the microcontroller as each new sample value becomes available from the device. (The SPI interface clock, SCLK, is used only for the data transfers across the interface).
- The LMP90100 has powerful calibration capabilities, and several other features as well, that necessitate close to 64 control registers. To support the large register set, the device divides the register space into multiple 16-register segments and the SPI protocol requires a segment address to be written when the segment boundary is crossed. As long as you stay within the current segment, you can keep doing conventional SPI accesses – for example, as you do repetitive data reads. Figure 1 illustrates the concept of segment addresses. For more details, see the device-specific data sheet.
- A number of other features are also offered:
  - The data-only mode where the instruction byte is not required
  - The streaming mode where more than three consecutive registers can be read or written with one SPI transaction only
  - CRC check for the data

You, however, may choose to avoid the additional complexity and access the device with SPI transactions that are close to the generic SPI protocol that he or she may already be familiar with. This application report provides a pseudo-code example to illustrate the basic read/write access of control and status registers as well as the data registers for typical applications. The more complex additional features are not used in this example code. Those items are referenced in the third bulleted item in Section 2.

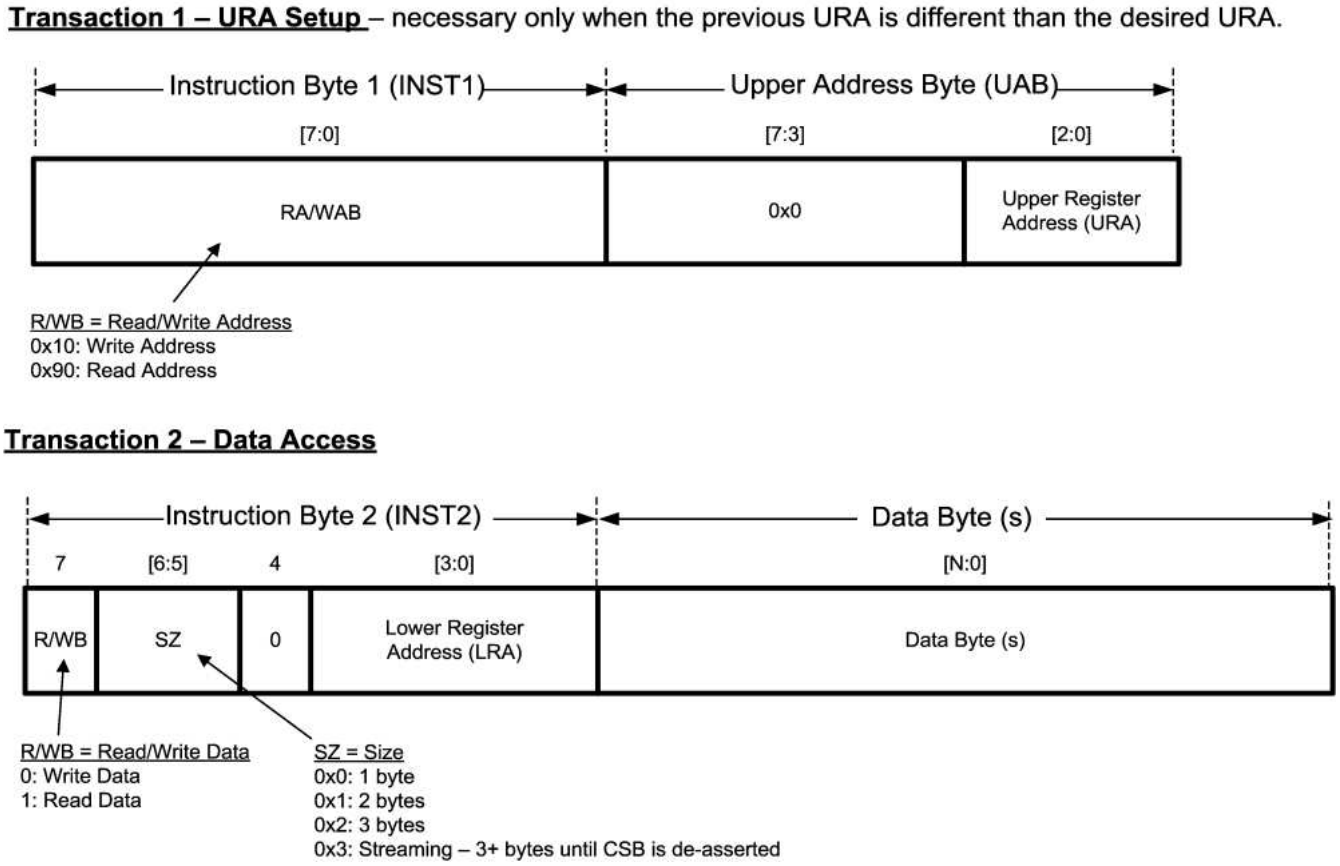

Figure 1. Segment Address Setup and Conventional Data Access

3 Assumed Application

This example assumes that multiple sensor channels are being read in a round-robin fashion. Furthermore, the example only deals with acquiring the data from the device and storing it in microcontroller buffers as dictated by upper level firmware. Processing of the received data is not covered. It is also assumed that the microcontroller being used has an integrated SPI Master block. However, micro-controller based SPI Masters vary a great deal in their capabilities. Here, a rather simple SPI Master is assumed. For example, it is assumed that the SPI Master is capable of doing byte-long SPI transactions at-a-time and its transmit and receive data registers are not buffered. This causes there to be multiple "gaps" within an SPI transaction. The gaps would affect the data throughput on the SPI interface, but this is considered not to be an issue in this very slow sensor application. A simple GPIO bit-banging code is also included for those application where the micro may not have an SPI Master at all.

4 Overall Considerations

The LMP90100 is an AFE designed for measurements of very slow physical properties and, as such, operates at output data rates (ODR) of only up to 214.65 samples per second. This equates to a DRDYB signal every 4.66 msec or slower; which leaves a lot of time for each sample to be processed within each DRDYB cycle even with a relatively slow microcontroller. For example, the rather slow pseudo-code example shown here should be able to deliver each new sample to the controller memory in roughly 100 - 150 usec on a microcontroller running with a 4MHz CPU clock. If the user needs to speed up the data access, the code can be made more streamlined and the device's data-only mode may be employed as well.

With the LMP90100 the user has the option of either using the DRDYB signal or a Data Available bit in a control register to find out when the next sample data is available. Further, the DRDYB signal can be combined with the SDO pin (chip's default) or made a separate signal by assigning it to the D6 pin – one of the GPIO pins. This example code uses the latter method – that is, a separate DRDYB signal on the D6 pin. Furthermore, this example code uses the DRDYB signal as an interrupt signal. An alternative would be to poll this signal and the user may do this if the overall firmware architecture so dictates.

Since the customer's actual microcontroller may be one of several typical controllers used in sensor applications, we have taken the approach of providing pseudo-code in this example. The intention is for it to serve as an example for the user to write his own code as suitable for the chosen microcontroller. National's evaluation system for this device uses the Atmel SAM3U microcontroller that is based on the ARM® Cortex™ M3 core. The code makes reference to the registers of this device, but the customer is expected to replace these hardware dependencies with his own code.

5 High-Level Description

The example code consists of several routines. The two main routines, callable by the higher level code, are: **spi_reg_read** and **spi_reg_write**. These are general purpose SPI routines that will transfer as many bytes of data between the device and the controller's memory as instructed. These routines then call one of the following routines to carry out the actual SPI transaction:

write_spi16	used to write the upper address byte (only when a segment change is occurring).
write_spi8 read_spi8	used to read or write 8 bits of data across SPI.

(Two types of **read_spi8/write_spi8** are provided: simple GPIO bit-banging types or ones using the microcontroller's SPI Master).

spi_reg_read/spi_reg_write thus use one instance of **write_spi16**, if necessary, and as many instances of read_spi_8 to write_spi8 to construct a complete transaction as necessary to abide by the user's command. The higher-level code expected to indicate when a segment change is occurring. There will be gaps between these instances within the single SPI transaction – where a transaction is defined as all the while the CSB signal is held asserted. The code directly controls the CSB signal with a GPIO pin (the SPI Master is not used to automatically drive CSB).

The higher-level code would have to perform the following functions:

1. Configure the microcontroller.
2. Initialize the LMP90100.
   - Build the device register image in memory.
   - Call **spi_reg_write** once for each segment of the device register space where at least one register has to be modified from its default values. Indicate to the routine the starting register address, the number of bytes to write, where in the memory to get the data from, and whether or not this is a new segment. See the actual routine code for the precise description. (It s a good practice to assume you need to explicity set the segment address with the first device access).

     For this code's purposes, relating only to the SPI operation, the following device register settings are necessary. (Register_Name:Bit_Name = value).

     SPI_DRDYBCN:SDO_DRDYB_D6 = 1

     SPI_HANDSHAKECN:SDO_DRDYB_DRIVER = 0x4

     The example assumes that the multiple sensor channels are being scanned continuously. This means that CH_SCAN register should be appropriately set and the CHx_INPUTCN registers should be configured to set the front-end MUX as appropriate for the application.
3. Verify the device is initialized correctly
   - Call spi_reg_read once for each segment
   - Compare read-back data with device register image in the memory.
4. Set up memory buffers for the sensor data – one buffer for each sensor channel.
5. Initialize the SPI Master with the appropriate SCLK frequency, polarity and phase settings. Also set the

number of bits per transfer to 8.

6. Set the device register segment to where the data registers are located.

   Call **spi_reg_write** (reg_addr=0x1A, segment_change=1, ptr_data=null, data_length=0).

7. Call the Data Handling Loop.
8. Data Handling Loop: (This loop maintains a set of pointers that point to the memory locations where the sensor data is to be delivered. A channel number count is used to scan through the channels the same way the device has been configured to do. The actual memory pointers are passed to the **spi_reg_read** routine to inform it of where the sensor data is to be written).
   a. Enable the interrupts for the pin where DRDYB signal is connected. (The ISR should be very simple: just post the data_ready message and return. This message is what the Data Handling Loop waits for in this example. Other methods are possible).
   b. When the message data_ready is received:
      - Call **spi_reg_read** (reg_addr=0x1A, segment_change=0, ptr-data=memory pointer for this channel at this time, data_length=3).
      - Check routine return value. It should be 3.
      - Increment memory pointer for this channel and check if it has reached the max. limit.
      - If not, obtain the next channel's pointer and loop back.
      - If yes, check if all channels' data counts have reached the max. values.
      - If not, obtain the next channel's pointer and loop back.
      - If yes, exit (all channels' data have been captured).

Note that this routine assumes that a finite amount of data have to be captured from the sensor channels. It is also possible to continuously receive data in the above manner and process each data sample in an ongoing fashion as it is received.

6 Example Code Routines

The actual code routines, called by the high-level function described in Section 5, are as follows:

```
//////////////////////////////////////////////////////////////////////
// Generic National SPI Example Code                                //
// Filename: spi_rw.c                                               //
// Author: Strategic Apps Group                                     //
// Revision: 1.1                                                    //
// Date: Mar. 7, 2011                                               //
//                                                                  //
// Exported API for this code: spi_reg_read and spi_reg_write       //
//                                                                  //
// This is an example code provided to National's customers to      //
// assist them in writing microcontroller firmware routines that    //
// can access National's Sensor AFE chips that use the SPI          //
// interface but with a 2-step, address/data protocol that          //
// supports large, segmented address spaces.                        //
// Please Note: This code is part of an Application Note that       //
//              should be read for some important considerations    //
//              beyond what is in this code.                        //
//                                                                  //
// The code illustrates two types of hardware-dependent routines:   //
//   - a simple bit-banging type method that utilizes GPIO; and     //
//   - a higher-level method that relies on the microcontroller's   //
//   SPI Master Controller to carry out the transactions.           //
//   The latter method would have to change a good deal when this   //
//   code is targetted to the user's specific microcontroller.      //
//   This example code is based on National's evaluation board      //
//   firmware that runs on Atmel ATSAM3U ARM M3-based controller.   //
//                                                                  //
// National's EULA policy applies.                                  //
// (c) National Semiconductor Corp, 2011                            //
//////////////////////////////////////////////////////////////////////
typedef unsigned char           UINT8;
```

```
typedef unsigned short int      UINT16;
typedef unsigned int            UINT32;
#define CSB          GPIO_PIN_CSB         //hardware dependent
#define MISO         GPIO_PIN_MISO        //hardware dependent
#define MOSI         GPIO_PIN_MOSI        //hardware dependent
#define SCLK         GPIO_PIN_SCLK        //hardware dependent
#define SET_HIGH       1
#define SET_LOW        0


//hardware dependent API

void delay()
{
  //delay some time
}
void set_gpio(UINT32 pin, UINT8 value)

{

//set GPIO pin value
}

void get_gpio(UINT32 pin)
{
//get GPIO pin value
}

//The following code is based on using GPIO to control SPI

//Read 8 bits of data through SPI
//Only manipulates SDI and SCLK, not CS
#if USE_GPIO
void read_spi8(UINT8 *data)
{
UINT8 bits;

  //clear the data memory
*data = 0;

for (bits = 7; bits >= 0 ; bits--) {
  set_gpio(SCLK,SET_LOW);
  delay();
  set_gpio(SCLK,SET_HIGH);
  *data |= ((get_gpio(MISO) & 1 ) << bits);
}

}

//Write 8 bits of data through SPI
//Only manipulates SDO and SCLK, not CS

void write_spi8(UINT8 data)
{

UINT8 bits;

for (bits = 7; bits > = 0 ; bits--) {
  set_gpio(MOSI, (data >> bits) & 1) ; //shift the needed bit to the LSB position
  set_gpio(SCLK,SET_LOW) ;
  delay();
  set_gpio(SCLK,SET_HIGH);
}

}
```

```
#elseif ATMEL_SAM3U4E //use SPI_MASTER
//the following code is based on Atmel microcontroller SAM3U4E's SPI controller

typedef struct _AT91S_SPI {
  AT91_REG SPI_CR; // Control Register
  AT91_REG SPI_MR; // Mode Register
  AT91_REG SPI_RDR; // Receive Data Register
  AT91_REG SPI_TDR; // Transmit Data Register
  AT91_REG SPI_SR; // Status Register
  AT91_REG SPI_IER; // Interrupt Enable Register
  AT91_REG SPI_IDR; // Interrupt Disable Register
  AT91_REG SPI_IMR; // Interrupt Mask Register
  AT91_REG Reserved0[4]; //
  AT91_REG SPI_CSR[4]; // Chip Select Register
  AT91_REG Reserved1[43]; //
  AT91_REG SPI_ADDRSIZE; // SPI ADDRSIZE REGISTER
  AT91_REG SPI_IPNAME1; // SPI IPNAME1 REGISTER
  AT91_REG SPI_IPNAME2; // SPI IPNAME2 REGISTER
  AT91_REG SPI_FEATURES; // SPI FEATURES REGISTER
  AT91_REG SPI_VER; // Version Register
  AT91_REG SPI_RPR; // Receive Pointer Register
  AT91_REG SPI_RCR; // Receive Counter Register
  AT91_REG SPI_TPR; // Transmit Pointer Register
  AT91_REG SPI_TCR; // Transmit Counter Register
  AT91_REG SPI_RNPR; // Receive Next Pointer Register
  AT91_REG SPI_RNCR; // Receive Next Counter Register
  AT91_REG SPI_TNPR; // Transmit Next Pointer Register
  AT91_REG SPI_TNCR; // Transmit Next Counter Register
  AT91_REG SPI_PTCR; // PDC Transfer Control Register
  AT91_REG SPI_PTSR; // PDC Transfer Status Register
} AT91S_SPI, *AT91PS_SPI;

//assigning the SPI Master Controller address to the data structure
AT91PS_SPI *spiA = (AT91PS_SPI)SPI_MASTER_CONTROOLLER_A_ADDR; // SPI_MASTER_CONTROOLLER_A_ADDR ↵
= 0x40008000

#define TRANSFER_DONE (0x1 << 1) // (SPI) Transmit Data Register Empty

void SPI_Base_Initialize(void)
{
  //User to insert his controller-specific code here to do:
  //initialize SPI master controller
  //set SPI controller register, intilaize SCLK rate, Polarity etc.
  //set the number of SCLKs per transfer
  // spiA->SPI_CSR[0] = (spi->SPI_CSR[0] & ~AT91C_SPI_BITS) | AT91C_SPI_BITS_8;
  //enable SPI

}

void write_spi8(UINT8 data)
{
spiA->SPI_TDR =data; // Send data - Starts as soon as this completes
while ((spiA-> SPI_SR & TRANSFER_DONE) == 0); // Wait for transfer done
}

//read 8 bit data through spi

void read_spi8(UINT8 *data)
{
spiA->SPI_TDR =0xFF; // Send data - Starts as soon as this completes
while ((spiA->SPI_SR & TRANSFER_DONE) == 0); // Wait for transfer done
*data = (spiA->SPI_RDR & 0x000000FF);
}

#endif
```

```
//general API

#define SPI_READ 0x80
#define SPI_SEGMENT 0x1000
#define SIZE_MASK 0x60
#define SIZE_SHIFT 5
#define SIZE_FLAG 3 //1 or more bytes, until CSB is de-asserted

//write 16 bit data through spi

void write_spi16(UINT16 data)
{
UINT8 temp;

  temp = (data>>8)&0xFF;
  //write the upper 8bit data
  write_spi8(temp);
  temp = data & 0xFF;

  //write low 8 bit data
  write_spi8(temp);
  }

//read 16 bit data through spi

void read_spi16(UINT16 *data)
{
UINT8 temp;

//get the upper 8bit data
read_spi8(&temp);
*data = temp;
*data <<= 8;
}

//set csb pin to high
void set_csb_hi()
{
set_gpio(CSB, SET_HIGH);
}

//set csb pin to low
void set_csb_lo()
{
set_gpio(CSB, SET_LOW);
}

/*
function: write the data in controller memory to the device
paramters:
reg_addr: device register address
segment_change: TRUE if the adress is in a new segment
ptr_data: starting location in proc. memory where data resides
data_length: number of bytes of data to write, range 0-16, 0:just change segment
(note: SPI transaction = data_length + 1)
returns : write data length, should be the same as what was requested
*/
int spi_reg_write(UINT16 reg_addr,bool segment_change, UINT8 *ptr_data, UINT16 data_length )
{
UINT16 instruction;
UINT8 low_addr;
UINT16 w_count;

//set chip select to low
set_csb_lo();
```

```
if (segment_change) {
//write the high address

  //get the register upper 8bit address
  instruction = (reg_addr >> 4) & 0x7;
  instruction |= SPI_SEGMENT;

  write_spi16(instruction);

  //set chip select high
  set_csb_hi();

  //set chip select low
  set_csb_lo();
}

    if(data_length == 0) // just change the segment
{

  set_csb_hi();

    return 0;

}

    if (ptr_data ==NULL)
  return 0;

low_addr = (reg_addr & 0XF);

low_addr |= (SIZE_FLAG << SIZE_SHIFT);

// send the low address
write_spi8(low_addr);

w_count = 0;

while (w_count < data_length) {

write_spi8(ptr_data[w_count++]);
}

//set chip select high
set_csb_hi();

return data_length;
}
/*
  function: read data from the device to the controller's memory
  paramters:
reg_addr: device register address
segment_change: TRUE if this access is to a new segment
ptr_data: starting address in proc. memory where data will go
data_length: number of bytes of data to read, range 1-16
  (note: SPI transaction = data_length + 1)
  returns : read data length, should be the same as what was requested
*/
int spi_reg_read(UINT16 reg_addr,bool segment_change, UINT8 *ptr_data, UINT16 data_length )
{

UINT16 instruction;
UINT8 low_addr;
UINT16 r_count;

if (data_length == 0 || ptr_data ==NULL)
```

```
return 0;

//set chip select to low
set_csb_lo();

//write the high address

if (segment_change) {
  //get the register upper 8bit address
  instruction = (reg_addr >> 4) & 0x7;
  instruction |= SPI_SEGMENT;

  write_spi16(instruction);

  //set chip select high
  set_csb_hi();

  //set chip select low
  set_csb_lo();
  }

  low_addr = (reg_addr & 0XF);

  low_addr |=SPI_READ;

  low_addr |= (SIZE_FLAG << SIZE_SHIFT);

  // send the low address
  write_spi8(low_addr);

  r_count = 0;

  while (r_count < data_length) {

  read_spi8(&ptr;_data[r_count++]);
  }

  //set chip select high
  set_csb_hi();

  return data_length;
 }
```

7 Conclusion

Though the LMP90100 offers many advanced functions with its SPI interface, an example has been provided for carrying out basic device configuration and data read operations.

8 References

LMP90100/LMP90099/LMP90098/LMP90097 Sensor AFE System: Multi-Channel, Low Power 24-Bit Sensor AFE with True Continuous Background Calibration Data Sheet (SNAS510)

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

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