Application Report
SLVA704–June 2015

Understanding the I²C Bus

Jonathan Valdez, Jared Becker

ABSTRACT

The I²C bus is a very popular and powerful bus used for communication between a master (or multiple masters) and a single or multiple slave devices. Figure 1 illustrates how many different peripherals may share a bus which is connected to a processor through only 2 wires, which is one of the largest benefits that the I²C bus can give when compared to other interfaces.

This application note is aimed at helping users understand how the I²C bus works.

Figure 1 shows a typical I²C bus for an embedded system, where multiple slave devices are used. The microcontroller represents the I²C master, and controls the IO expanders, various sensors, EEPROM, ADCs/DACs, and much more. All of which are controlled with only 2 pins from the master.

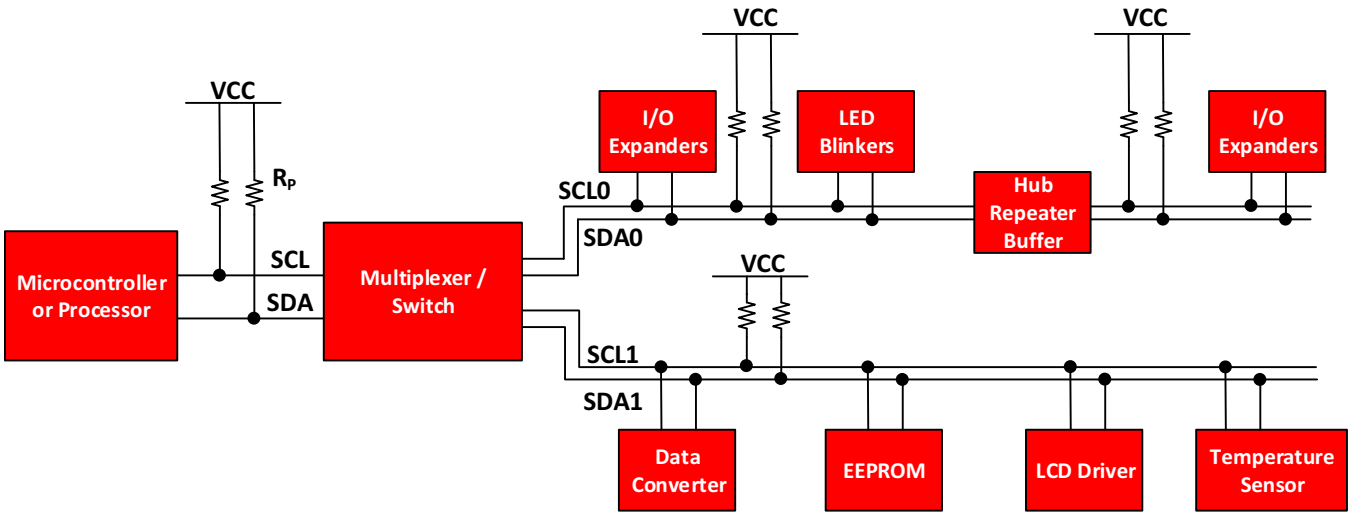

Figure 1. Example I²C Bus

Trademarks

1 Electrical Characteristics

I²C uses an open-drain/open-collector with an input buffer on the same line, which allows a single data line to be used for bidirectional data flow.

1.1 Open-Drain for Bidirectional Communication

Open-drain refers to a type of output which can either pull the bus down to a voltage (ground, in most cases), or "release" the bus and let it be pulled up by a pull-up resistor. In the event of the bus being released by the master or a slave, the pull-up resistor (R_{PU}) on the line is responsible for pulling the bus voltage up to the power rail. Since no device may force a high on a line, this means that the bus will never run into a communication issue where one device may try to transmit a high, and another transmits a low, causing a short (power rail to ground). I²C requires that if a master in a multi-master environment transmits a high, but see's that the line is low (another device is pulling it down), to halt communications because another device is using the bus. Push-pull interfaces do not allow for this type of freedom, which is a benefit of I²C.

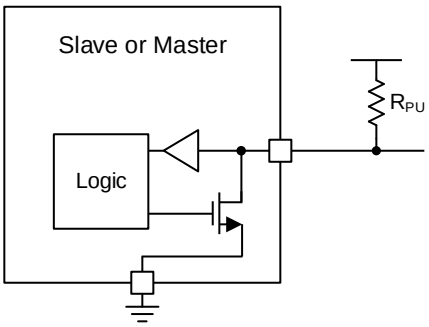

Figure 2. Basic Internal Structure of SDA/SCL Line

Figure 2 shows a simplified view of the internal structure of the slave or master device on the SDA/SCL lines, consisting of a buffer to read input data, and a pull-down FET to transmit data. A device is only able to pull the bus line low (provide short to ground) or release the bus line (high impedance to ground) and allow the pull-up resistor to raise the voltage. This is an important concept to realize when dealing with I²C devices, since no device may hold the bus high. This property is what allows bidirectional communication to take place.

1.1.1 Open-Drain Pulling Low

As described in the previous section, the Open-Drain setup may only pull a bus low, or "release" it and let a resistor pull it high. Figure 3 shows the flow of current to pull the bus low. The logic wanting to transmit a low will activate the pull-down FET, which will provide a short to ground, pulling the line low.

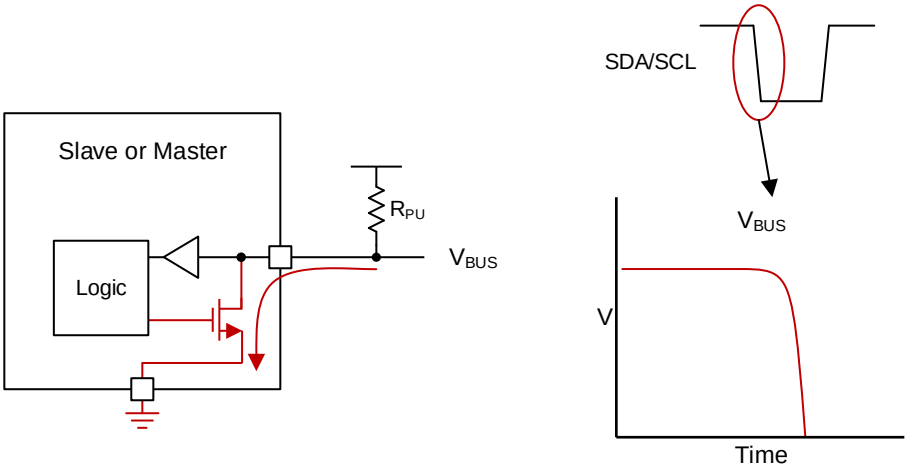

Figure 3. Pulling the Bus Low With An Open-Drain Interface

1.1.2 Open-Drain Releasing Bus

When the slave or master wishes to transmit a logic high, it may only release the bus by turning off the pull-down FET. This leaves the bus floating, and the pull-up resistor will pull the voltage up to the voltage rail, which will be interpreted as a high. Figure 4 shows the flow of current through the pull-up resistor, which pulls the bus high.

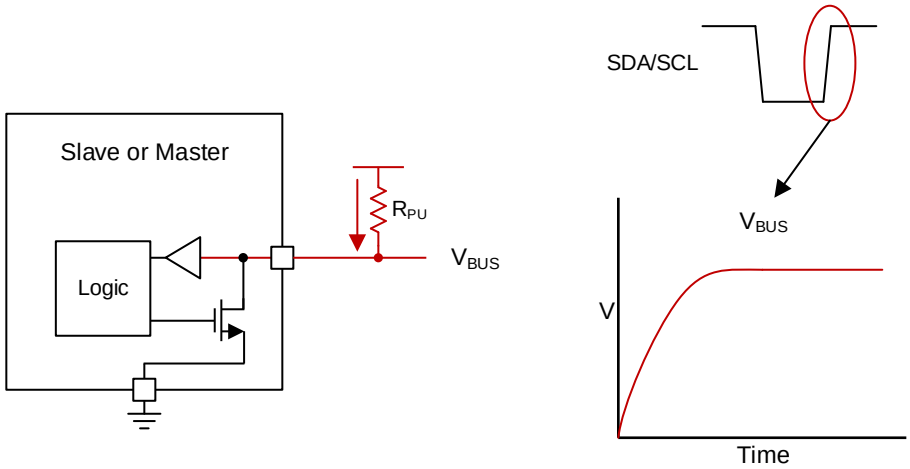

Figure 4. Releasing the Bus With An Open-Drain Interface

2 I²C Interface

2.1 General I²C Operation

The I²C bus is a standard bidirectional interface that uses a controller, known as the master, to communicate with slave devices. A slave may not transmit data unless it has been addressed by the master. Each device on the I²C bus has a specific device address to differentiate between other devices that are on the same I²C bus. Many slave devices will require configuration upon startup to set the behavior of the device. This is typically done when the master accesses the slave's internal register maps, which have unique register addresses. A device can have one or multiple registers where data is stored, written, or read.

The physical I²C interface consists of the serial clock (SCL) and serial data (SDA) lines. Both SDA and SCL lines must be connected to V_{CC} through a pull-up resistor. The size of the pull-up resistor is determined by the amount of capacitance on the I²C lines (for further details, refer to *I²C Pull-up Resistor Calculation* (SLVA689). Data transfer may be initiated only when the bus is idle. A bus is considered idle if both SDA and SCL lines are high after a STOP condition.

The general procedure for a master to access a slave device is the following:

1. Suppose a master wants to send data to a slave:
   - Master-transmitter sends a START condition and addresses the slave-receiver
   - Master-transmitter sends data to slave-receiver
   - Master-transmitter terminates the transfer with a STOP condition
2. If a master wants to receive/read data from a slave:
   - Master-receiver sends a START condition and addresses the slave-transmitter
   - Master-receiver sends the requested register to read to slave-transmitter
   - Master-receiver receives data from the slave-transmitter
   - Master-receiver terminates the transfer with a STOP condition

2.1.1 START and STOP Conditions

I²C communication with this device is initiated by the master sending a START condition and terminated by the master sending a STOP condition. A high-to-low transition on the SDA line while the SCL is high defines a START condition. A low-to-high transition on the SDA line while the SCL is high defines a STOP condition.

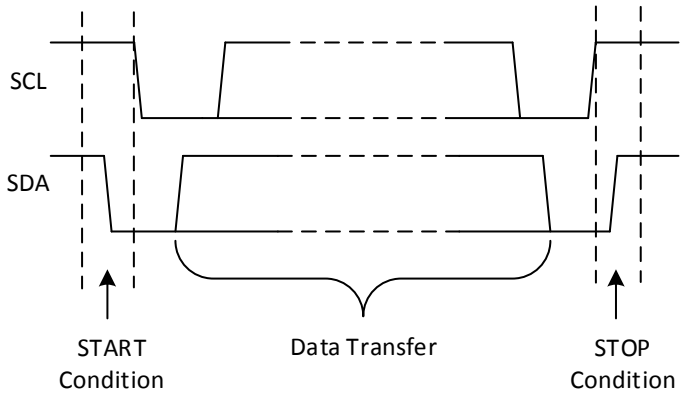

Figure 5. Example of START and STOP Condition

2.1.2 Repeated START Condition

A repeated START condition is similar to a START condition and is used in place of a back-to-back STOP then START condition. It looks identical to a START condition, but differs from a START condition because it happens before a STOP condition (when the bus is not idle). This is useful for when the master wishes to start a new communication, but does not wish to let the bus go idle with the STOP condition, which has the chance of the master losing control of the bus to another master (in multi-master environments).

2.2 Data Validity and Byte Format

One data bit is transferred during each clock pulse of the SCL. One byte is comprised of eight bits on the SDA line. A byte may either be a device address, register address, or data written to or read from a slave. Data is transferred Most Significant Bit (MSB) first. Any number of data bytes can be transferred from the master to slave between the START and STOP conditions. Data on the SDA line must remain stable during the high phase of the clock period, as changes in the data line when the SCL is high are interpreted as control commands (START or STOP).

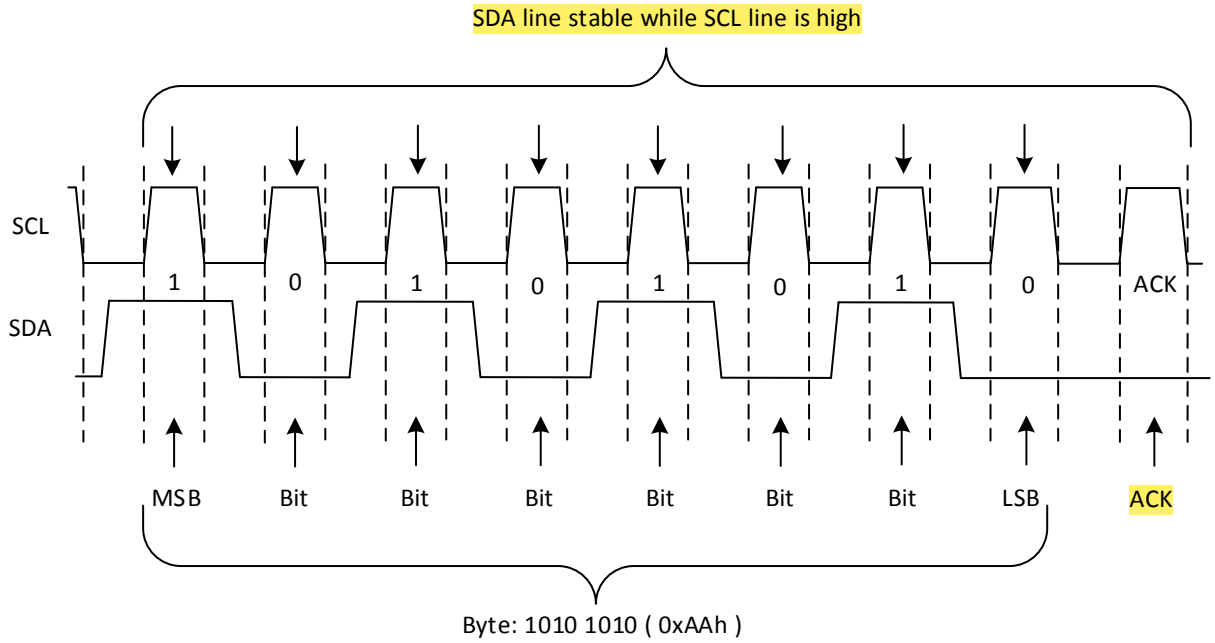

Figure 6. Example of Single Byte Data Transfer

2.3 Acknowledge (ACK) and Not Acknowledge (NACK)

Each byte of data (including the address byte) is followed by one ACK bit from the receiver. The ACK bit allows the receiver to communicate to the transmitter that the byte was successfully received and another byte may be sent.

Before the receiver can send an ACK, the transmitter must release the SDA line. To send an ACK bit, the receiver shall pull down the SDA line during the low phase of the ACK/NACK-related clock period (period 9), so that the SDA line is stable low during the high phase of the ACK/NACK-related clock period. Setup and hold times must be taken into account.

When the SDA line remains high during the ACK/NACK-related clock period, this is interpreted as a NACK. There are several conditions that lead to the generation of a NACK:

1. The receiver is unable to receive or transmit because it is performing some real-time function and is not ready to start communication with the master.
2. During the transfer, the receiver gets data or commands that it does not understand.
3. During the transfer, the receiver cannot receive any more data bytes.
4. A master-receiver is done reading data and indicates this to the slave through a NACK.

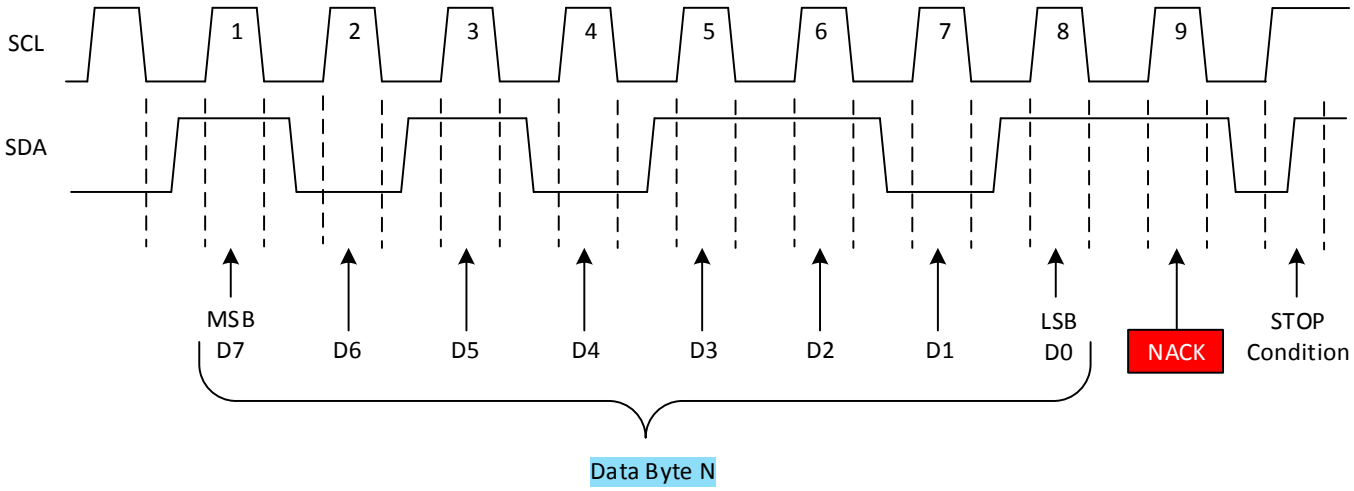

Figure 7. Example NACK Waveform

3 I²C Data

Data must be sent and received to or from the slave devices, but the way that this is accomplished is by reading or writing to or from registers in the slave device.

Registers are locations in the slave's memory which contain information, whether it be the configuration information, or some sampled data to send back to the master. The master must write information into these registers in order to instruct the slave device to perform a task.

While it is common to have registers in I²C slaves, please note that not all slave devices will have registers. Some devices are simple and contain only 1 register, which may be written directly to by sending the register data immediately after the slave address, instead of addressing a register. An example of a single-register device would be an 8-bit I²C switch, which is controlled via I²C commands. Since it has 1 bit to enable or disable a channel, there is only 1 register needed, and the master merely writes the register data after the slave address, skipping the register number.

3.1 Writing to a Slave On The I²C Bus

To write on the I²C bus, the master will send a start condition on the bus with the slave's address, as well as the last bit (the R/$\overline{W}$ bit) set to 0, which signifies a write. After the slave sends the acknowledge bit, the master will then send the register address of the register it wishes to write to. The slave will acknowledge again, letting the master know it is ready. After this, the master will start sending the register data to the slave, until the master has sent all the data it needs to (sometimes this is only a single byte), and the master will terminate the transmission with a STOP condition.

Figure 8 shows an example of writing a single byte to a slave register.

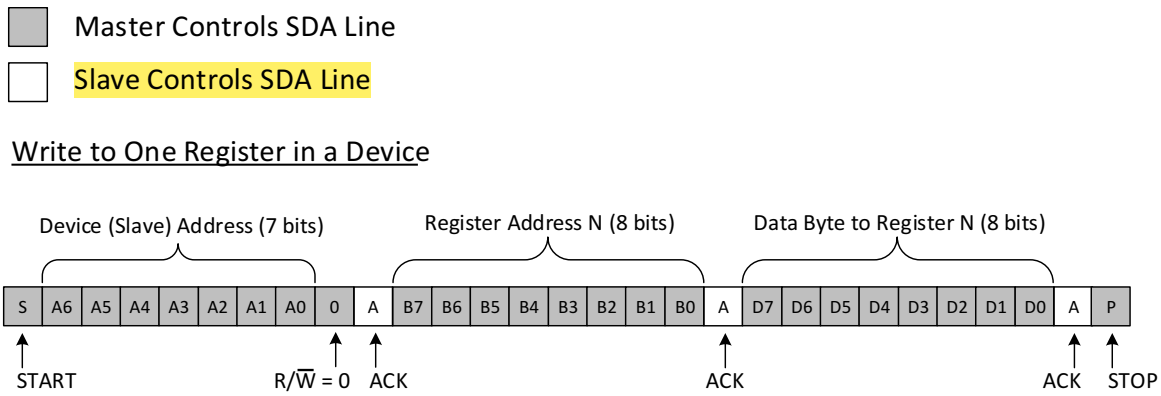

Figure 8. Example I²C Write to Slave Device's Register

3.2 Reading From a Slave On The I²C Bus

Reading from a slave is very similar to writing, but with some extra steps. In order to read from a slave, the master must first instruct the slave which register it wishes to read from. This is done by the master starting off the transmission in a similar fashion as the write, by sending the address with the R/$\overline{W}$ bit equal to 0 (signifying a write), followed by the register address it wishes to read from. Once the slave acknowledges this register address, the master will send a START condition again, followed by the slave address with the R/$\overline{W}$ bit set to 1 (signifying a read). This time, the slave will acknowledge the read request, and the master releases the SDA bus, but will continue supplying the clock to the slave. During this part of the transaction, the master will become the master-receiver, and the slave will become the slave-transmitter.

The master will continue sending out the clock pulses, but will release the SDA line, so that the slave can transmit data. At the end of every byte of data, the master will send an ACK to the slave, letting the slave know that it is ready for more data. Once the master has received the number of bytes it is expecting, it will send a NACK, signaling to the slave to halt communications and release the bus. The master will follow this up with a STOP condition.

Figure 9 shows an example of reading a single byte from a slave register.

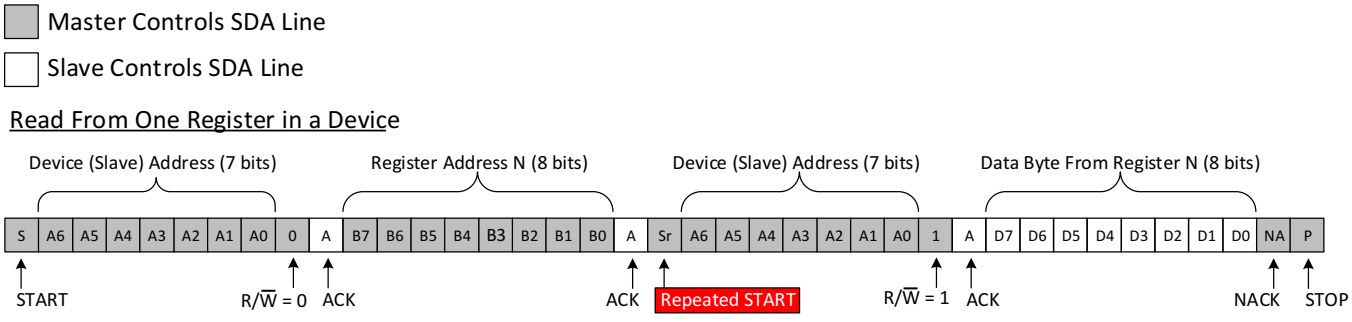

Figure 9. Example I²C Read from Slave Device's Register

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