Application Report
SPMA057C–August 2013–Revised January 2018

1-Wire Enumeration

ABSTRACT

1-Wire can be used in systems that have low speed and low power communication requirements. This application report describes the 1-Wire communication protocol, available TivaWare™ for C Series APIs for the 1-Wire module in Tiva™ C Series microcontrollers and an example enumeration algorithm using binary tree search.

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/spma057.

Contents

Trademarks

TivaWare, Tiva are trademarks of Texas Instruments.
All other trademarks are the property of their respective owners.

1 Introduction to 1-Wire

1-Wire is a communication system designed to interface simple sensors and devices with a single wire interface. This system is used for low-speed and low-power communication devices. There are two modes of operation available: standard speed and overdrive speed. The achievable data rate with standard speed is 16.3 kbit/s, while overdrive mode communication is done at 10 times the standard speed.

This protocol uses a single data line for data transmission from one device to another device. The bus is half duplex so that data can move in both directions, but not at the same time. When required, an extra wire can also be used to power up the slave devices.

The protocol supports one slave (single drop) or many slaves (multi drop) on the bus. There is also a single master on the bus that controls the transfer of information on the bus. The master initiates all transfers on the data line. Transfer of data is only possible between master and slaves, so data cannot be transferred between slaves.

A clock is not required for this protocol as each slave is clocked by an internal oscillator synchronized to the falling edge of the bus.

When transferring a byte, the least significant bit is transferred first.

Copyright © 2013–2018, Texas Instruments Incorporated

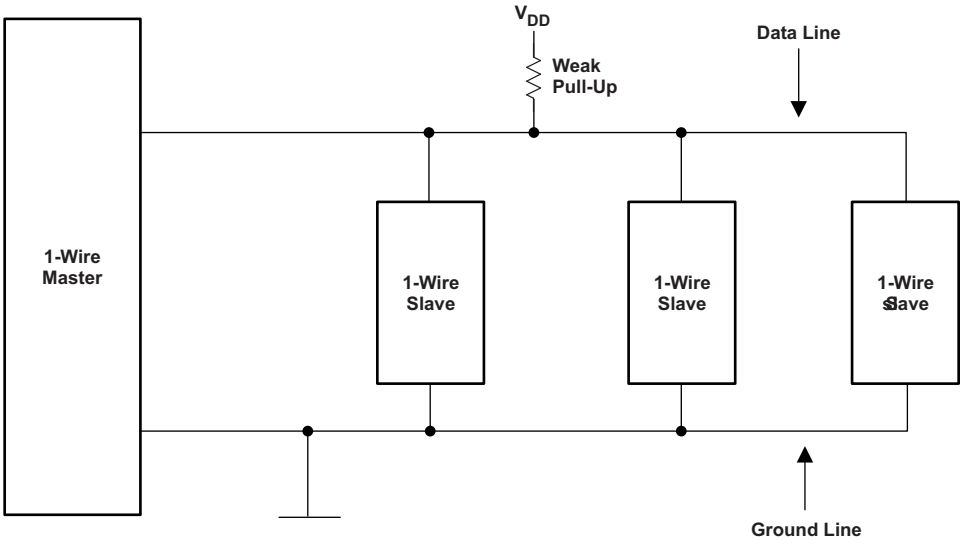

Figure 1. Bus Topology

1.1 Bus Requirements

Every output pin must be open-drain, and a weak pull up must be attached to the signal so that, the bus is driven low if at least one device drives the bus low. This protocol enables the transfer of data between two devices on the bus while the other devices are in the idle state. The strength of the pull up can be decided by the user based on the following factors:

- The device being externally powered may have pull up values in the range of 10K or more if data rate is not required to be high and trace length on the system is not high.
- The device being externally powered may have pull up values less than 1K if the data rate is high especially in the overdrive mode or the trace lengths on the system are longer.
- The device being parasitically powered may require active drivers after being selected so that the end device can have sufficient energy to perform the end operation.

1.2 Powering

Slaves can be powered in two modes:

- Externally Powered: A power pin on the slave device is used to power up the slave on the bus. This topology is used when a slave has high power needs.
- Parasitically Powered: The slave is powered by the data line. Slave devices have an internal capacitor that stores this energy when the bus is idle and pulled up by the weak pull-up.

Copyright © 2013–2018, Texas Instruments Incorporated

2 Functional Description

2.1 Signaling on 1-Wire

The four types of signaling that are possible on the data line are:

- Reset Sequence with Reset Pulse and Answer to Reset (ATR): A reset pulse is used to put all the devices in a known state. Slaves confirm their presence by sending an ATR signal, which is done by holding the line low. The master samples the bus, and if the bus reads low, then at least one slave device is present.

Table 1. Reset Signaling Description and Implementation

Operation	Description	Implementation
Reset	Reset the 1-Wire bus slave devices and prepare them for a command.	Drive the bus low for 480 µs to reset all the slaves. The master then samples the bus for the next 240 µs while the slaves Answer to Reset (ATR).

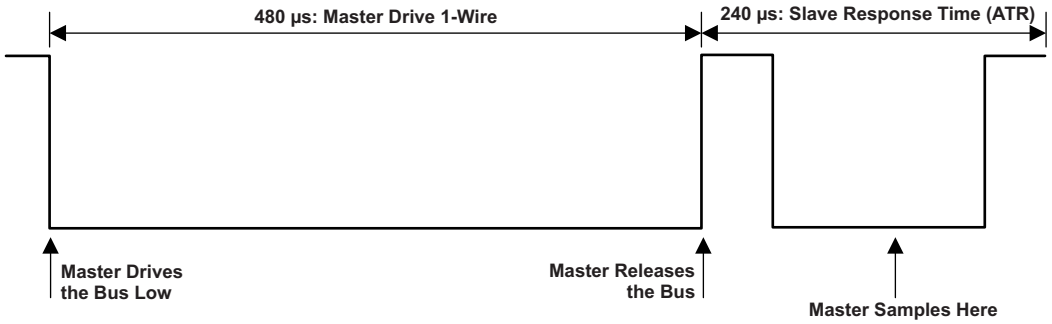

Figure 2. Reset Sequence Bus Timing When There is at Least 1 Slave on the Bus

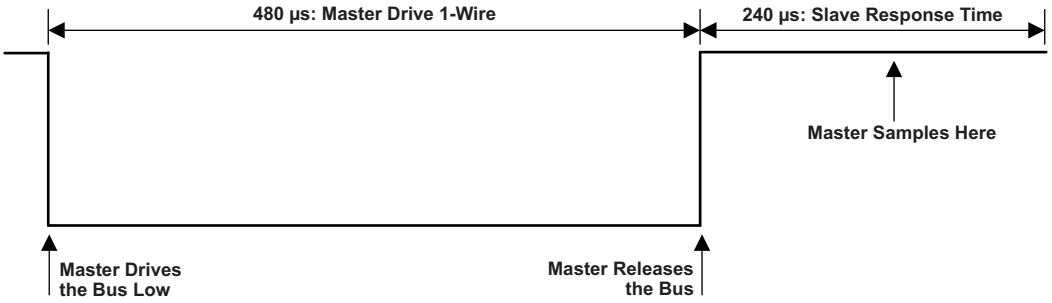

Figure 3. Reset Sequence Bus Timing When There are no Slaves on the Bus

- Write 0 bit onto the bus

Table 2. Write 0 Bit Signaling Description and Implementation

Operation	Description	Implementation
Write 0 bit	Send 0 bit to the 1-Wire slaves	Drive the bus low for 60 µs

Copyright © 2013–2018, Texas Instruments Incorporated

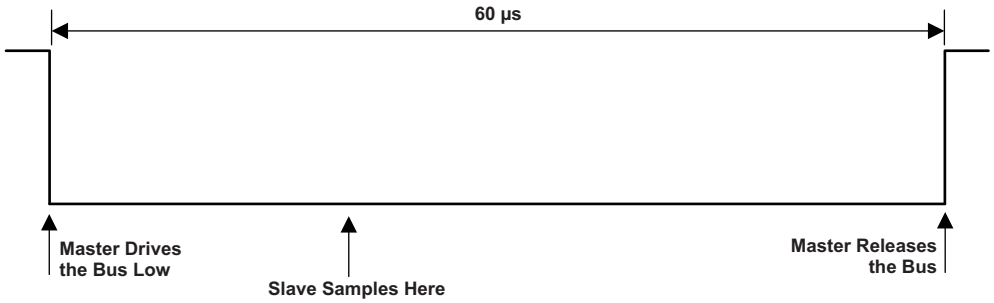

Figure 4. Write 0 Bus Timing

- Write 1 bit onto the bus

Table 3. Write 1 Bit Signaling Description and Implementation

Operation	Description	Implementation
Write 1 bit	Send 1 bit to the 1-Wire slaves	Drive the bus low for < 15 µs. Typical times are about 6 µs. Release the bus until 60 µs after the falling edge.

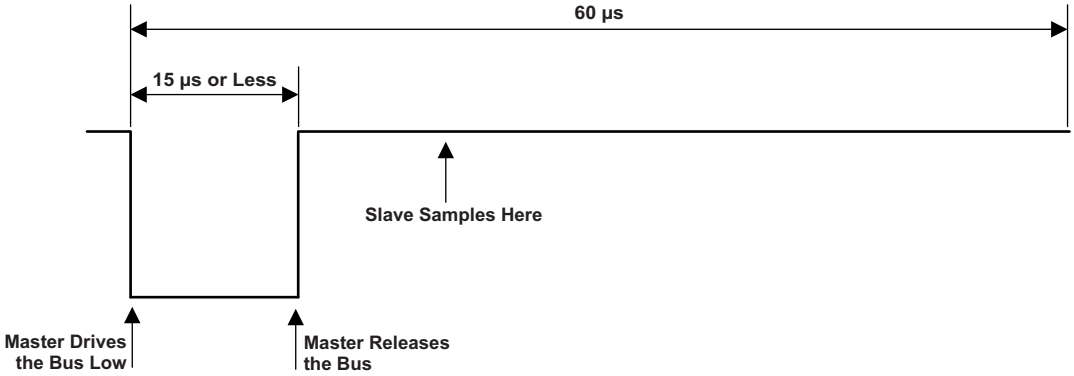

Figure 5. Write 1 Bus Timing

- Read bit: Reads one bit from the slaves. Read bit signaling is similar to write “1” signaling, except that the master reads instead of writes.

Table 4. Read Bit Signaling Description and Implementation

Operation	Description	Implementation
Read bit	Read a bit from the 1-Wire slave	Drive the bus low from 1 µs to 15 µs. Sample the bus at 15 µs after the falling edge to read the bit from the slave.

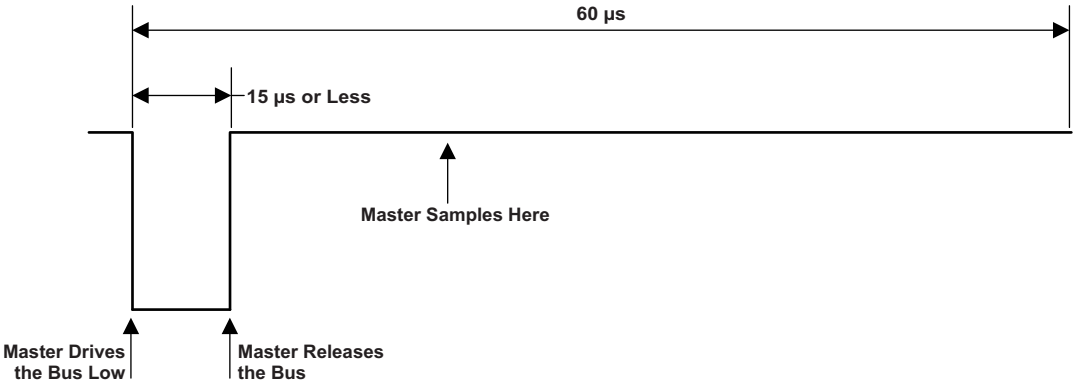

Figure 6. Read 1 Bus Timing

Copyright © 2013–2018, Texas Instruments Incorporated

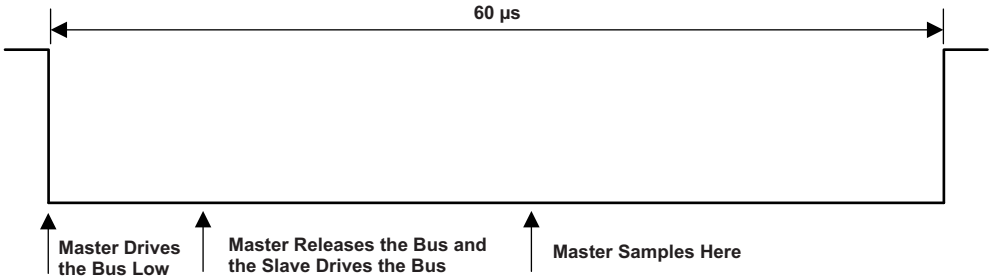

Figure 7. Read 0 Bus Timing

2.2 Address Format of the 1-Wire Device

1-Wire slave devices from manufacturers have a unique 64-bit address stored in it, which is also called the ROM number. The least significant 8 bits of the address give the family code of the device. The next least 48 bits give the serial number of the device. The most significant 8 bits give the CRC generated from the least 56 bits.

Table 5. Address Format of the 1-Wire Slave

	64-Bit ROM Number	
8-Bit CRC	48-Bit Serial Number	8-Bit Family Code
63:56 Bits	55:8 Bits	7:0 Bits

NOTE: The CRC is used to check if the data is received correctly. Tiva C Series devices do not implement the CRC in hardware, so a software implementation is required.

2.3 Typical Communication Flow on the 1-Wire Bus

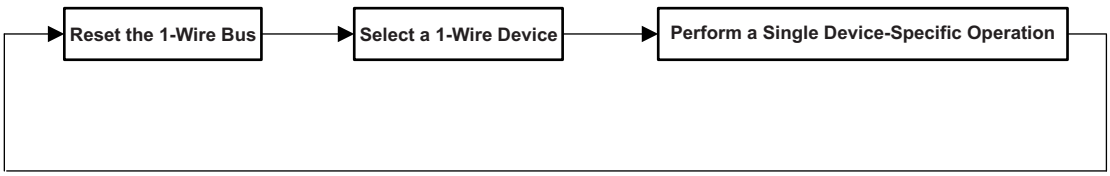

Figure 8. Typical Communication Flow on a 1-Wire Bus

- Start with the Reset Sequence
- If the master must determine which slave devices are present on the bus, the master should perform a search to detect the ROM numbers of the slave devices.
- Before performing an operation on a device, the device must be configured and/or selected using the ROM commands. Some of the available functional ROM commands are:
  - Read ROM [0x33]: Only used when there is a single slave on the bus. This command reads the ROM number of the only slave present on the bus.
  - Match ROM [0x55]: This command followed by a 64-bit ROM number selects the slave with the matching ROM number. All other devices wait until the next reset pulse.
  - Search ROM [0xF0]: This command is required to obtain the ROM numbers of multiple devices, and it informs slave devices that a search is going to be conducted by the master. The Search is then conducted by reading a bit and its complement of the ROM numbers from the slaves and sending an appropriate bit back. For more details, see the Section 4. Slave devices that have the same bit as the one sent by the master remain active while others wait for the next reset
  - Skip ROM [0xCC]: Devices can be addressed without the master knowing the ROM numbers. This command is helpful when giving a common command to all the devices.

Copyright © 2013–2018, Texas Instruments Incorporated

  - Overdrive Skip ROM [0x3C]: This command is used only in single drop. This command is the same as the Skip ROM command except that only devices that can run in overdrive mode remain active and go into overdrive mode. Devices that cannot run in overdrive mode wait for the next reset.
  - Overdrive Match ROM [0x69]: This command is the same as the Match ROM command except that the device is only matched if it can run in overdrive mode. All other devices wait for the next reset.
- After selecting the required devices, a device-specific command can be issued to perform the required operation.
- Typically after each operation, a reset pulse is issued.

3 Functions Available in TivaWare for C Series for 1-Wire Module

NOTE: ui32Base should contain the base address of the 1-Wire module.

OneWireBusReset(uint32_t ui32Base); This function issues a reset on the 1-Wire bus; it does not wait for the completion of the reset.

OneWireBusStatus(uint32_t ui32Base); This function retrieves the bus condition status, which is used to determine if the bus is ready to perform an operation. This status is busy if the master is performing any operation or idle if the master is not performing an operation.

OneWireDataGet(uint32_t u3i2Base, uint32_t *pui32Data);This function waits for the transaction, if any, to complete and retrieves data from the 1-Wire interface. The data is stored in the address given by pui32Data.

OneWireDataGetNonBlocking(uint32_t ui32Base, uint32_t *pui32Data); This function retrieves data from the 1-Wire interface. If there is an active transaction, then 0xffffffff is returned. The data is stored in the address given by pui32Data.

OneWireInit(uint32_t ui32Base, uint32_t ui32InitFlags); This function initializes the 1-Wire module. The ui32InitFlags parameter contains the initialization flags.

OneWireIntClear(uint32_t ui32Base, uint32_t ui32IntFlags); This function clears the required interrupt sources in the 1-Wire module.

OneWireIntDisable(uint32_t ui32Base, uint32_t ui32IntFlags); This function disables the required interrupt sources in the 1-Wire module.

OneWireIntEnable(uint32_t ui32Base, uint32_t ui32IntFlags); This function enables the required interrupt sources in the 1-Wire module.

OneWireIntRegister(uint32_t ui32Base, void(*pfnHandler)(void)); This function registers an interrupt handler for the 1-Wire module.

OneWireIntUnregister(uint32_t ui32Base); This function unregisters an interrupt handler for the 1-Wire module.

OneWireIntStatus(uint32_t ui32Base, bool bMasked); This function gets the current interrupt status. If bMasked is true, then the masked interrupt status is obtained. If bMasked is false then the raw interrupt status is obtained.

OneWireTransaction(uint32_t ui32Base, uint32_t ui32OpFlags,uint32_t ui32Data, uint32_t ui32BitCnt); This function is used to perform a 1-Wire protocol transfer on the bus. ui32BitCnt is used to configure the number of bits to be sent or received. Written data is specified by ui32Data. The ui32OpFlags parameter is used to define which operation (reset, read, write) is to be performed.

NOTE: More information about the APIs can be found in the *TivaWare™ Peripheral Driver Library User's Guide* (SPMU298).

Copyright © 2013–2018, Texas Instruments Incorporated

4 Enumeration

When multiple slave devices are present on the bus, the master must know the ROM numbers to match the required devices. Enumeration is performed to obtain the ROM numbers of the slave devices on the bus and therefore what classes of devices are available.

The search operation is performed by executing two steps, iteratively:

1. Read two bits from the slaves (the actual bit value and its complement)
2. Write the appropriate bit that defines the search path in the enumeration algorithm. Devices with the corresponding bit equal to the written bit remain active, while the rest go to the idle state and wait for the next reset command.

This cycle is done for all 64 bits and the 64-bit ROM number is assembled at the end of the iteration.

4.1 Algorithm

The Search algorithm uses a binary search tree. At every node, the algorithm can take the path dictated by either a “0” or “1.” The relationship between the two bits obtained in step 1 and the path to be taken in step 2 are given in Table 6.

Table 6. Search Algorithm

Actual Bit Read Value	Complement Bit Read Value	Conclusions	Path to be taken
0	0	Multiple devices have a corresponding 0 bit and a corresponding 1 bit	This is a conflict situation and requires a decision on which path to take
1	0	Only one device has a 1 in the corresponding bit location	The 1 path is taken
0	1	Only one device has a 0 in the corresponding bit location	The 0 path is taken
1	1	No devices are on the bus	End the search

The only time that a decision must be made is when there is a conflict. In the other three cases, the path to be taken is already defined. Figure 9 shows the algorithm flow for a 4-bit search. The following variables in software are key to the search.

- The variable *i32LastConflictZeroBitNumber* stores the iteration number of the last conflict node where the path taken is “0” while finding the current ROM number.
- The variable *li32ConflictBitNumber* stores the value of last_conflict_zero for the last ROM number.
- The variable *ui32BitNumber* gives the position of the bit under consideration in the ROM number.

Copyright © 2013–2018, Texas Instruments Incorporated

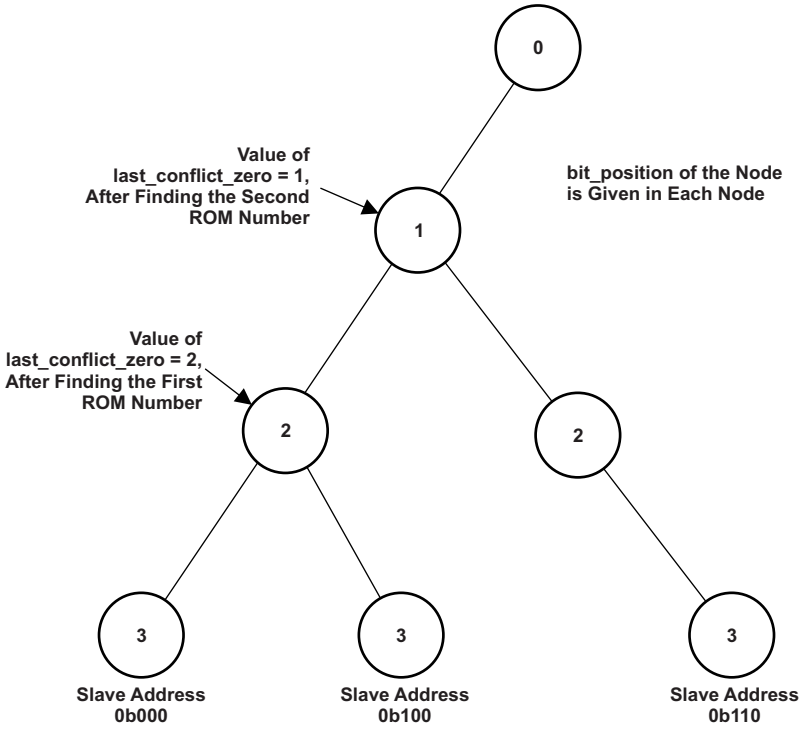

Figure 9. The *i32LastConflictZeroBitNumber* and *ui32BitNumber* Variables

4.2 Steps of the 4-Bit Search Algorithm

1. Reset the bus and look for ATR responses. End the process if there are no devices on the bus.
2. Send the Search ROM command if an ATR response is received.
3. Read a bit from the slaves.
4. Read the complement of the bit in step 3 from the slaves.
5. Check if both the bits are 1. If 'yes,' end the process. If 'no', continue.
6. Check if the first read bit is a 0 and the second read bit is a '1.' If 'yes,' write 0 onto the bus and go to step 14. If 'no', continue.
7. Check if the first read bit is a 1 and the second read bit is a zero. If 'yes', go to step 9. If 'no', continue.
8. Check whether or not *ui32BitNumber* is equal to *i32ConflictBitNumber*. If 'yes', continue. If 'no', go to step 10
9. Write 1 onto the bus. Go to step 14.
10. Check whether or not *ui32BitNumber* is less than *i32ConflictBitNumber*. If 'yes', continue. If 'no', go to step 12.
11. Check whether or not the bit in the *ui32BitNumber* of the last ROM number is equal to 1. If 'yes', go to step 9. If 'no', continue.
12. Write 0 onto the bus.
13. Update the value of *i32LastConflictZeroBitNumber* with the *ui32BitNumber*.
14. Check whether or not *ui32BitNumber* is equal to 63. If 'yes', go to step 1. If 'no', go to step 3.

5 Conclusion

This application report provides an overview of the 1-Wire protocol and presents an example of how the master identifies the slave devices on the bus using binary tree search.

Copyright © 2013–2018, Texas Instruments Incorporated

6 References

- *TivaWare™ Peripheral Driver Library User's Guide* (SPMU298)
- TivaWare™ Peripheral Driver Library for C Series

Copyright © 2013–2018, Texas Instruments Incorporated

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from B Revision (April 2016) to C Revision	Page
• Update was made in Section 4.1.	7

Copyright © 2013–2018, Texas Instruments Incorporated

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
Copyright © 2018, Texas Instruments Incorporated