ABSTRACT

In industrial applications, it is often necessary for multiple devices to communicate with each other. A new communication peripheral created for C2000™ Microcontrollers (MCU), the Fast Serial Interface (FSI) can expand its reliable high-speed communication feature to multiple devices in a system. This application report demonstrates how to implement FSI using a daisy-chain connection. Test results are provided to validate the high-speed communication capability of FSI with different configuration methods. A user can quickly verify and design FSI in different applications using the provided algorithm and source code which can be downloaded from C2000ware.

The target processor for the corresponding software is the TMS320F28004x. The implementation methods and software can be applied and ported to future C2000 processors that include FSI. Example code discussed in this document can be found in the latest C2000ware release, located within the following local directory after installation:

C:\ti\c2000\C2000Ware_<version_number>\driverlib\f28004x\examples\fsi

The available example projects are:
- fsi_ex16_daisy_handshake_lead
- fsi_ex16_daisy_handshake_node

Contents
1 Introduction to the FSI Module ........................................................................................................ 2
2 FSI Applications .................................................................................................................................. 3
3 Daisy-chain Connection Using FSI ..................................................................................................... 3
4 References ........................................................................................................................................ 13

List of Figures
1 FSITX and FSIRX CPU Interface ........................................................................................................ 2
2 Daisy-Chain Connection Example ................................................................................................... 3
3 Handshake Data Flow ...................................................................................................................... 4
4 Software Flow Chart With Different Project Settings ..................................................................... 6
5 Test Platform for Two Devices Communication ............................................................................. 7
6 Data Transmission Test Using CPU Control .................................................................................. 8
7 FSI Communication Using DMA Control ....................................................................................... 9
8 Time Needed From Device 1 to Device 2 ....................................................................................... 9
9 Test Platform for Three Devices Communication ......................................................................... 11
10 FSI Communication With DMA Control Among Three Devices .................................................. 11
11 Time of Data Going Through One Device ................................................................................... 12
12 FSI Communication With CPU Control among Three Devices ................................................... 12

List of Tables
1 Example Projects Description .......................................................................................................... 5
2 Data Frame Structure ..................................................................................................................... 8
1 Introduction to the FSI Module

The FSI module in TMS320F28004x is a serial communication peripheral capable of reliable and robust high-speed communications up to 200 Mbps. There are many features available for FSI, including programmable data length, CRC, ECC support, and so forth. A PING watchdog and Frame watchdog can enable automatic communication-break detection. Most importantly, delay line control can adjust signal delays introduced by trace-length mismatch or an isolation chip on the receiver, enabling FSI to achieve high-speed and robust communication. Thus, FSI can enable innovative, high-performance algorithms in industrial applications, where real-time control with critical communication speed is required.

Generally, FSI can be implemented in two kinds of system conditions:
- Direct connection for chip-to-chip communications when MCUs exist on the same voltage plane.
- Chip-to-chip or board-to-board communications across an isolation barrier (like ISO7842), commonly for MCUs placed on hot-side and cold-side, or boards with different voltage planes.

The FSI consists of independent transmitter (FSITX) and receiver (FSIRX) cores, which are configured and operated independently. Because of this the FSI protocol does not have a notion of master and slave, unlike some other synchronous communication protocols. Figure 1 shows the CPU interface of each FSI module. Each module owns up to three signal lines: one clock and two data signals, where the second data lines, FSITXyD1 and FSIRXyD1, are optional, and can be enabled for multi-lane transmission and double the speed for data bits. Thus, at least four signal lines are needed to create 2-way point-to-point communication. Considering the timing spec for FSITX (see the TMS320F28004x Piccolo™ Microcontrollers Data Sheet), the maximum data rate of 200 Mbps can be achieved with the maximum clock of 50 MHz, using two data lines, since the data is transmitted on both edges of the clock. For a full overview of FSI including all features and functions available, see the TMS320F28004x Piccolo Microcontrollers Technical Reference Manual.

![Figure 1. FSITX and FSIRX CPU Interface](image-url)
2 FSI Applications

In terms of the trend in power applications, increasing demand for higher power levels makes multiple power modules in parallel much more popular, such as in applications like telecom rectifiers, server power supplies, on-board chargers, and so forth. Meanwhile, to achieve a complex system with high performance, multiple MCUs are commonly used and must operate in a synchronized fashion. Thus, critical data, including protection signals, sampling parameters, and even control loop data, needs to be informed or transferred with the fastest speed and least latency among multiple devices/modules. FSI will be more suitable to handle this when compared to the traditional Controller Area Network (CAN), Serial Peripheral Interface (SPI) or Universal Asynchronous Receiver/Transmitter (UART).

There are a number of communication network topologies for connecting multiple devices. A ring topology can be created by connecting multiple devices with FSI communication in a daisy-chain fashion. Figure 2 shows a daisy-chain connection system for N (N≥2) devices, where each device (index i) connects with the FSITX of device i-1 and FSIRX of device i+1.

![Figure 2. Daisy-Chain Connection Example](image)

3 Daisy-chain Connection Using FSI

The basic steps for achieving FSI communication for daisy-chain connection is provided in the following sections.

3.1 Handshake Mechanism

Once the FSITX and FSIRX modules of each device have been configured, the handshake mechanism should be implemented to prepare the sender and receiver before actual data transmission, since devices may power up in an arbitrary order in a real scenario.

In order to simplify the data flow, one device is assigned as the lead, working as the driver of the handshake sequence, and the other N-1 devices, within the daisy-chain loop, assigned as nodes. Following the example in Figure 2, Device 1 will be the lead. It should be noted that the other N-1 node devices can share the same handshake configuration.

The handshake process can be described as follows:

1. For all devices, configure the Frame Type of FSITX as Ping Frame, and enable the receiver interrupts for Ping Frame Received event on the FSI INT1 vector to detect the incoming transmission.
2. Begin the ping loop 0:
   a. The lead device sends the flush sequence to the second device, then sends a ping frame with Tag0(0000); wait for some time. If the lead device receives a valid ping frame tag Tag0, continue to the second loop; or else iterate the loop 0 again.
   b. The node devices enter a wait loop for a receiver interrupt. If receiving a valid ping frame tag of Tag0, continue to the loop 1; or else iterate the loop 0 again.
3. Begin the ping loop 1:
   a. The lead device sends a ping frame with Tag1(0001); wait for some time. If the lead device receives a valid ping frame tag Tag1, continue; or else iterate the loop 1 again.
   b. The node devices send a ping frame Tag0 and waits for a receiver interrupt. If receiving a valid ping frame Tag1, continue to send a ping frame Tag1; or else iterate the loop 1 again.
4. Handshake completes.
The simplified data flow is shown in Figure 3. Two ping loops are necessary for the daisy-chain connection handshake mechanism. Ping loop 0 has the purpose of establishing the communication path along the chain of devices and ping loop 1 acts as the acknowledgment to the nodes that the communication path is good. In ping loop 0, the node devices wait to receive a Ping Tag0 from the previous device in the chain. Once a Ping Tag0 is successfully received, it will be forwarded on to the next device in the chain. The ping loop 0 will fail if a device in the chain has not powered up or is not ready for the reception. Once ping loop 0 has succeeded, in which ping tag0 has made its way back to the lead device, ping loop 1 is initiated to inform the node devices that the handshake sequence has completed and to begin expecting actual data.

The handshake function can be found in the tested projects, with handshake_lead() for the lead device and handshake_node() for the other N-1 devices in the daisy-chain loop.

### 3.2 FSI Data Frame Configuration APIs

Several configurations are needed for the data frames to be sent/received properly, including the frame type, frame tag/user data, word length, number of data lines, and writing to the data buffer. The configuration example code using API drivers for FSITX and FSIRX, which are defined in the fsi.h driverlib header file in C2000ware, are shown below. Note that the content of frame tag and user data is fully user-configurable, which can be used to differentiate which device the data received is sent from.

```c
// TX setting part
FSI_setTxFrameType(FSITXA_BASE, FSI_FRAME_TYPE_NWORD_DATA);
FSI_setTxSoftwareFrameSize(FSITXA_BASE, nWords);
FSI_setTxDataWidth(FSITXA_BASE, nLanes);
FSI_setTxUserDefinedData(FSITXA_BASE, txUserData);
FSI_setTxFrameTag(FSITXA_BASE, txDataFrameTag);

// RX setting part
FSI_setRxSoftwareFrameSize(FSIRXA_BASE, nWords);
FSI_setRxDataWidth (FSIRXA_BASE, nLanes);`
3.3 **Start Transmitting Data Frames**

There are three methods to trigger the data transmission, including software triggered, externally triggered (EPWMx-SOCA/B) and using the DMA. For the software triggered method, writing 1 to TX_FRAME_CTRL.START or “FSI_startTxTransmit()” will start the transmission, while if using an external trigger, like EPWM1-SOCA, once the external trigger signal occurs, the data will be sent.

Since the DMA trigger can be generated every time a data frame transmission or receiving is completed from FSITX or FSIRX module, it provides a convenient method to transferring and moving data, especially with a mass amount of data. Here a configuration example is given for the FSI communication using DMA.

TX_OPER_CTRL_LO.START.MODE must be set to 0x2, which means writing to frame tag/user data fields can trigger the transmission and then enables a DMA event on FSITX:

```c
FSI_setTxStartMode(FSITXA_BASE, FSI_TX_START_FRAME_CTRL_OR_UDATA_TAG);
FSI_enableTxDMAEvent(FSITXA_BASE);
```

Two consecutive DMA channels are needed to fill the transmit buffer and frame tag/user data fields, respectively. And the two in sequence channels make sure the data will be sent right after the frame tag and user data are set, aligned with the start mode. In the example code, DMA CH1 and DMA CH2 have been used. Another important point is that the wrap control must be enabled for data of more than 16 words, since the FSI transmit buffer is a 16-word circular buffer.

```c
DMA_configWrap(DMA_CH1_BASE, DMA_TRANSFER_SIZE_IN_BURSTS, 0, dest_WrapSize, 0);
```

Here, dest_WrapSize represents the number of bursts to be transferred before a wrap of the destination address, so dest_WrapSize should be 16/ nWords.

In this case, it is implemented such that the transmit buffer is continuously fed by the DMA, which is triggered by FSITX in return, with DMA Continuous Mode enabled.

The FSIRX is configured very similarly to the FSITX, except for that there is no order requirement for the DMA channels for RX buffer and tag and user data. In the attached example projects, DMA CH3 and DMA CH4 are used.

3.4 **Test Result**

In order to demonstrate the communication speed and different configurations for FSI, daisy-chain connections for two device and three device configurations has been tested and validated. The test hardware utilized is composed of multiple F280049C ControlCARD Evaluation Modules. The tested example projects can be found within the C2000WARE download. All test results were collected using optimization level 2, configured within Code Composer Studio™ (CCS). Changing the optimization level will yield different results. An overall description of the tested projects is shown in Table 1. For better understanding, a general software flow chart with different project settings is shown in Figure 4.

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsi_ex16_daisy_handshake_lead</td>
<td>Project for the lead device in the daisy-chain loop.</td>
<td>☑ [#define FSI_DMA_trigger 0] represents FSI communication using CPU control.</td>
</tr>
<tr>
<td>fsi_ex16_daisy_handshake_node</td>
<td>Project for the N-1 other devices (N=&gt;2) in the daisy-chain loop.</td>
<td>☑ [#define FSI_DMA_trigger 1 &amp; &amp; #define TX_DMA_enable 1] represents FSI communication using DMA control, and enabling FSITX to trigger DMA for the transmitted data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>☑ [#define FSI_DMA_trigger 1 &amp; &amp; #define TX_DMA_enable 0] represents FSI communication using DMA control, and using software to trigger DMA for the transmitted data (manually).</td>
</tr>
</tbody>
</table>

Table 1. Example Projects Description
Figure 4. Software Flow Chart With Different Project Settings
3.4.1 FSI Communication Between Two Devices

For a minimal daisy-chain connection test, a system of two F280049C ControlCARD Evaluation Modules are used as shown in Figure 5. The comparison of the communication speeds for both CPU control and DMA control are provided.

![Test Platform for Two Devices Communication](image)

**Figure 5. Test Platform for Two Devices Communication**

#### 3.4.1.1 CPU control

**Test condition**

Device 1 sends data -> Device 2 receives data and reads RX buffer -> Device 2 writes TX buffer and sends data to Device 1 -> Device1 receives data and reads RX buffer.

**Test case**

Data length of 16 words, two data lines, TXCLK = 50 MHz, with Setting (Table 1) enabled.

In the test, GPIOs are toggled within software when specific events occur during the communication and measured using an oscilloscope to obtain the respective timing data. In Figure 6, the yellow signal represents the GPIO toggling of Device 1 (Lead device) and the purple signal represents the GPIO toggling of Device 2 (Node device).
Using the Fast Serial Interface (FSI) With Multiple Devices in an Application

Figure 6. Data Transmission Test Using CPU Control

From the results shown in Figure 6, the time obtained for the data transmission is 2 µs. In order to calculate the actual transmission speed, the total data length should be calculated. Table 2 shows the general structure of a data frame, which can be divided into two parts: effective data bits and overhead bits.

- **Effective Data Bits:** Includes the 8-bit User Data, 16-bit Data Words, and 8-bit CRC fields
- **Overhead Bits:** Includes the Preamble, SOF, Frame Type, EOF, and Postamble fields

Therefore, the ideal transmission time for 16 words can be derived theoretically, as shown in Table 3.

It should be noted that since two data lines only work for effective data bits, one FSITXCLK cycle delivers 4 effective data bits, while one FSITXCLK cycle only delivers 2 overhead bits. Thus, with a total 80 FSITXCLK cycles, the data transmission time can be calculated as shown in Equation 1.

\[
\text{(FSITXCLK cycles) / (FSITXCLK frequency)} = \frac{80}{(50MHz)} = 1.6 \ \text{µS}
\] (1)

Therefore, the theoretical transmission speed is 185 Mbps (296/1.6 µs), while the test speed is 148 Mbps with 2 µs transmission time, due to the fact that the tested transmission time includes entering the ISR (to toggle an IO pin), delay introduced in connection cables, and so forth. If changing to one data line, the theoretical transmission speed is 100 Mbps, while the test speed is 87.1 Mbps with an actual transmission time of 3.4 µs.

Another finding from Figure 6 is that moving data into/from the FSITX/FSIRX buffer takes some time, approximately 6.5 µs for FSI_readRxBuffer or FSI_writeTxBuffer. This will be the key factor that distinguishes DMA control as shown in the next section.

<table>
<thead>
<tr>
<th>Table 2. Data Frame Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLE</td>
</tr>
<tr>
<td>1111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3. Calculated Transmission Time for 16 Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Effective Data Bits (bits)</td>
</tr>
<tr>
<td>272</td>
</tr>
</tbody>
</table>
3.4.1.2 DMA Control

- **Test condition**
  The two devices continuously send and receive data to/from each other, independently.

- **Test case**
  Data length of 16 words (16 words per burst, 1 burst per transfer), 1 data line, TXCLK = 50 MHz, with Setting ③ (Table 1) enabled.

In the test, DMA interrupts from CH2 and CH4 are enabled to trigger at the end of a transfer in each device, which means that interrupts occur every time data is copied into the FSITX buffer and data is moved out of the FSRX buffer. With GPIOs toggling in the interrupts, Figure 7 shows the test result of FSI communication using DMA control. In order to get the transmission information purely from Device 1 to Device 2, transmissions from Device 2 to Device 1 are disabled. The resulting timing measurement obtained is shown in Figure 8.

![Figure 7. FSI Communication Using DMA Control](image1)

![Figure 8. Time Needed From Device 1 to Device 2](image2)

It should be noted that the time of 3.92 µs includes the time for data transmission, data being moved out of the FSIRX buffer, entering the ISR, and so forth According to the DMA pipeline timing requirements (available in *TMS320F28004x Technical Reference Manual*), the time for moving data of 17 words (16 words data+1 word of user data and frame tag) using 2 channels can be calculated as shown in Equation 2.

\[
(17 \times 3 \text{ cycles/word} + 2 \text{ cycles}) \times (100 \text{MHz}) = 0.53 \mu s
\]
Thus, considering other delay times, the actual transmission time is almost aligned with the former test result (3.4 µs) using CPU control. Also, it should be highlighted that using DMA in this case saves the time for transferring received data to a large extent, especially for an application with mass data transmissions.

Further test results are given in Table 4, for the comparison of using CPU control and DMA control for FSI. Note that the actual effective data throughput represents the tested transmission speed of effective data bits. Since overhead bits are fixed and the necessary parts are transferred in one frame, it is recommended to use the data length that is as long as possible to maximize the effective data throughput, especially for a large amount of data. Thus, as shown in Table 4, the actual effective data throughput for a data length of 4 words is much lower than that of a data length of 16 words. Accordingly, for the DMA configuration, BURST_SIZE (number of words/burst) should be maximized.

<table>
<thead>
<tr>
<th>Data Line /words</th>
<th>Transmissio n Time/µs</th>
<th>Read Buffer Time/µs</th>
<th>Theoretical Transmissio n Speed/Mbps</th>
<th>Actual Transmissio n Speed/Mbps</th>
<th>Actual Effective Data Throughput /Mbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Control</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>16</td>
<td>3.4</td>
<td>6.5</td>
<td>100</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>16</td>
<td>2</td>
<td>6.5</td>
<td>185</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>4</td>
<td>1.4</td>
<td>2.3</td>
<td>100</td>
</tr>
<tr>
<td>DMA Control</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>16</td>
<td>3.9</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>16</td>
<td>2.6</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>16</td>
<td>15.7</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>16</td>
<td>8.9</td>
<td>/</td>
<td>/</td>
</tr>
</tbody>
</table>

A lower clock frequency has also been tested here, since Electromagnetic Interference (EMI) will distort the signal integrity in some cases due to the poor connection made with jumper wires between two ControlCard Experimenter kits. EMI will generally be worse when FSITX and FSIRX are both working at the same time. Thus, it is important to preserve signal integrity and noise immunity in the connection between different devices. Generally, it is better to use a twisted pair or shielded wire per line for off board connection, while on board connection needs to match different trace lengths and have care taken in the layout to enhance noise immunity. Otherwise, the clock frequency will need to be decreased.

Note that in the tests performed there are no isolation barriers or isolation devices between the different F280049 ControlCards. In a real world application that involves isolation barriers being crossed and/or varying signal trace lengths, the integrated skew compensation block within the FSI receiver module can be used to manage the skew that may occur between the clock and data signals. For more information on this topic, see Fast Serial Interface (FSI) Skew Compensation.

### 3.4.2 FSI Communication among Three Devices

Based on the test and comparison results given in Section 3.4.1, FSI communication for three devices has also been investigated. The test platform used is shown in Figure 9.

Due to the nature of a daisy-chain connection, data will need to pass through N devices for the transmission from the first device to reach the last device. Therefore, to reduce latency it is important to make the data handling and forwarding time in each device as short as possible, especially when there are a number of devices in a connection loop. From the conclusion drawn in Section 3.4.1, to avoid having the CPU spending too much bandwidth moving data, it is recommended to use the DMA to serve FSI communication.

Differing from the DMA configuration of independent transmissions in Section 3.4.1.2, for the three device configuration the DMA channels serving the FSI transmitter buffer will not be triggered by FSITX in this test, but triggered by a software trigger after the received data is moved out of the receive buffer instead. Therefore, the data will be forwarded to the next device after being received in the current one. Using a software trigger or other peripheral interrupt event triggers for the DMA can offer a more flexible configuration to decide when to send the data in an actual application, such as during control interrupts, after a specific defined event, and so forth.
• **Test condition**
  Device 1 sends data -> Device 2 receives data and reads RX buffer -> Device 2 writes TX buffer and sends data to Device 3 -> … -> Device 3 writes TX buffer and sends data to Device 1 -> Device 1 receives data and reads RX buffer.

• **Test case**
  Data length of 4 words (4 words per burst, 1 burst per transfer), 1 data line, TXCLK = 50 MHz, with Setting ③ (Table 1) enabled.

In the test, GPIOs are toggled within software when specific events occur during the communication and measured using an oscilloscope to obtain the respective timing data. In Figure 10, the yellow signal represents the GPIO toggling of Device 1 (Lead device), the blue signal represents the GPIO toggling of Device 2 (Node device), and the purple signal represents the GPIO toggling of Device 3 (Node device).
As shown in Figure 10, the time needed for the data transmission through the three device daisy-chain loop to complete is 5.42 µs, increasing by 2.10 µs for each device added to the daisy-chain connection system. The 2.10 µs increased time per device includes transmission, reading the RX buffer, and writing the TX buffer, as indicated in Figure 11. Meanwhile, Figure 12 shows a comparison test result for FSI communication using CPU control, where 13.44 µs is spent for the communication loop. Further test comparison results can be found in Table 5. Compared to the DMA control case, where the data length changes from 4 words to 8 words, the time increases a lot for the CPU control case, due to the CPU needing to spend a majority of the time moving data.
Table 5. Comparison of Using CPU Control and DMA Control in FSI Among Three Devices

<table>
<thead>
<tr>
<th></th>
<th>Clock/MHz</th>
<th>Data Line</th>
<th>Data Length /Words</th>
<th>Transmission and Read Buffer Time/µs</th>
<th>Time of Data Going through One Device/µs</th>
<th>Time Cost in the Connection Loop (3 devices)/µs</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU control</td>
<td>50</td>
<td>1</td>
<td>4</td>
<td>3.32</td>
<td>4.96</td>
<td>13.44</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>1</td>
<td>8</td>
<td>5.40</td>
<td>8.44</td>
<td>22.64</td>
</tr>
<tr>
<td>DMA control</td>
<td>50</td>
<td>1</td>
<td>4</td>
<td>1.26</td>
<td>2.10</td>
<td>5.42</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>1</td>
<td>8</td>
<td>2.18</td>
<td>3.06</td>
<td>8.24</td>
</tr>
</tbody>
</table>

4 References

- Texas Instruments: *TMS320F28004x Piccolo™ Microcontrollers Data Sheet*
- Texas Instruments: *TMS320F28004x Piccolo Microcontrollers Technical Reference Manual*
- Texas Instruments: *Fast Serial Interface (FSI) Skew Compensation*
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