Application Report
SLAA831–April 2018

Using the MSP430FR6047 Wireless M-Bus Serial Library for Metering Applications

Nathan Siegel *MSP430 Applications*

ABSTRACT

The Texas Instruments (TI) MSP430FR6047 ultrasonic sensing and measurement SoC is a powerful, highly integrated microcontroller (MCU) that is optimized for ultrasonic flow meters. By utilizing the serial interface provided with TI's wireless M-Bus software stack, the MSP430FR6047 can be used to create a wireless metering solution. This application report describes how to achieve such a solution and explains the serial library provided to assist in configuring and operating the serial interface used to construct a wireless meter.

Related software specific to this document can be downloaded from http://www.ti.com/lit/zip/slaa831.

Contents

List of Figures

List of Tables

Trademarks

MSP430, SimpleLink, Code Composer Studio, BoosterPack are trademarks of Texas Instruments.
Cortex is a registered trademark of Arm Limited.
IAR Embedded Workbench is a registered trademark of IAR Systems.
All other trademarks are the property of their respective owners.

Copyright © 2018, Texas Instruments Incorporated

1 Introduction

This application report explains how to use TI's MSP430FR6047 MCU as an application processor with TI's CC13x0 wireless MCU acting as a wireless network processor that is running TI's OMSv3.0.1 compatible Wireless M-Bus (wM-Bus or WMBus) software stack. Figure 1 shows the system block diagram. The configuration described in this document implements a wireless meter (the meter must be connected to a wM-Bus collector). The wM-Bus protocol stack used by TI's CC13x0 wM-Bus software stack, as defined by STACKFORCE GmbH, defines a serial interface that allows a host processor to control the wireless device running the wM-Bus software stack through UART. Regardless of the device that is running the stack or which wM-Bus mode is being used, the same serial command interface can be used. This interface and the corresponding MSP430FR6047 Wireless M-Bus Serial Library are described in this application report.

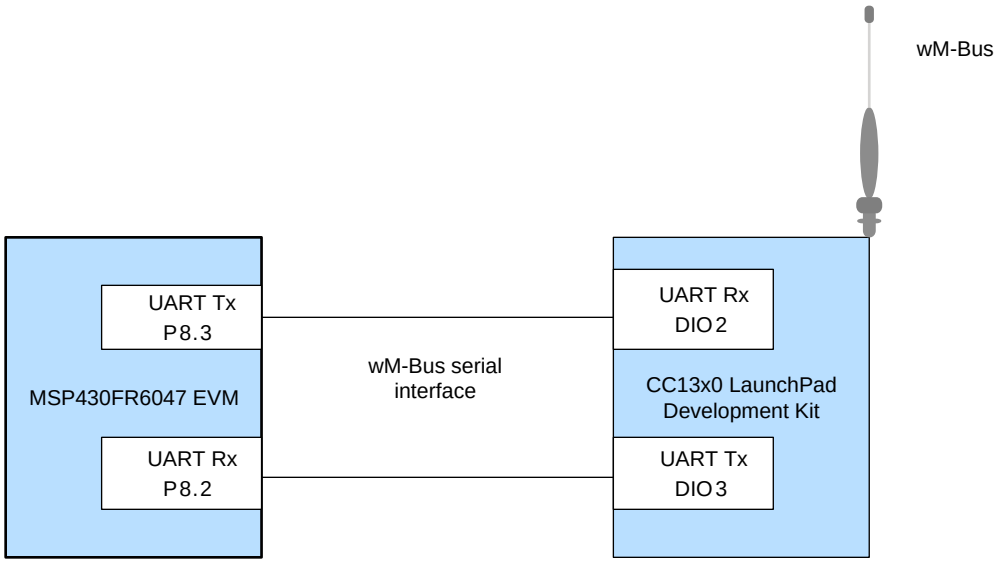

Figure 1. System Functional Block Diagram

1.1 Wireless M-Bus

The Metering Bus (M-Bus) is a field bus specialized for the transmission of metering data from meters (such as electricity, gas or water meters, or heat cost allocators) to a data collector. It is described by European Norms (EN 13757-1 to -7), which include specification of both wired and wireless M-Bus. Wired M-Bus is out of scope of this document.

The wM-Bus protocol stack discussed in this document is defined and supported by STACKFORCE GmbH. After downloading TI's wM-Bus software stack for the CC13x0, the serial application protocol layer (APL) interface documentation can be found in the *Documentation* folder. This fully describe the STACKFORCE protocol stack and its serial APL interface. The MSP430FR6047 Wireless M-Bus Serial Library follows this protocol and provides functions to allow users to easily create applications that use the serial commands defined by the protocol stack serial interface.

1.2 MSP430FR6047 Ultrasonic Sensing MCU

The MSP430FR6047 is a powerful and highly integrated ultrasonic sensing and measurement SoC that is optimized for Ultrasonic Flow Meters. Its integrated Ultrasonic Sensing Solution (USS) module provides high accuracy for a wide range of flow rates. This functionality, combined with its serial interfaces, allows the MSP430FR6047 to be combined with a wireless MCU to produce an ideal wireless metering solution. The MSP430FR6047 also contains TI's MSP430™ ultra-low power (ULP) FRAM platform that combines uniquely embedded FRAM and a holistic ultra-low-power system architecture, allowing system designers to increase performance while lowering energy consumption. FRAM technology combines the low-energy fast writes, flexibility, and endurance of RAM with the nonvolatility of Flash.

Copyright © 2018, Texas Instruments Incorporated

The MSP430FR6047 Ultrasonic Sensing Evaluation Module (MSP430FR6047 EVM) was used in the examples covered in this document. The serial interface used by the wM-Bus serial interface library was implemented to run on the evaluation module, though minor modifications can be made to use the library with a different board.

1.3 CC1350 SimpleLink™ Ultra-Low-Power Dual-Band Wireless MCU

The CC1350 is the first device in the CC13xx and CC26xx family of cost-effective, ultra-low-power wireless MCUs capable of handling both Sub-1 GHz and 2.4 GHz RF frequencies. The CC1350 device combines a flexible, very low-power RF transceiver with a powerful 48-MHz Cortex®-M3 MCU in a platform supporting multiple physical layers and RF standards. A dedicated radio controller (Cortex-M0) handles low-level RF protocol commands that are stored in ROM or RAM, thus ensuring ultra-low power and flexibility to handle both Sub-1 GHz and 2.4 GHz protocols.

The SimpleLink CC1350 wireless MCU LaunchPad development kit combines a Sub-1 GHz with a Bluetooth low energy radio for the ultimate combination of easy mobile phone integration with long-range connectivity including a 32-bit ARM Cortex-M3 processor on a single chip. For the wM-Bus communication system, the LAUNCHXL-CC1350EU kit is recommended because it is optimized for 868-MHz operation under ETSI and has been CE certified for operation in the EU. This is the only hardware for which the OMS3.0.1 stack has been developed and tested on.

Please note that the CC1310, which is a subset of the CC1350 that can only handle Sub-1 GHz frequencies, can be used as well. The corresponding CC1310 LaunchPad development kit (LAUNCHXL-CC1310) can be used instead of the CC1350 LaunchPad development kit without any modifications.

2 Wireless M-Bus Serial APL Interface

A detailed overview of the STACKFORCE wM-Bus protocol and the associated serial interface can be found in the *Documentation* folder of the wM-Bus software stack download. The following subsections give a brief overview of the protocol and the serial interface. See the detailed overview for more specifics.

2.1 STACKFORCE Wireless M-Bus Stack

The Wireless M-Bus protocol defines both meter and collector devices. A meter device takes measurements (for example, of water, gas, or electricity) and then sends them to a collector using the wireless M-Bus protocol. The collector receives this data and can use that data for whatever the application requires. A collector can be connected to multiple meters, but each meter can only be connected to one collector.

The Wireless M-Bus stack allows for two communication modes: unidirectional and bidirectional. Unidirectional wM-Bus modes only support data transmission from a meter device to a data collector. The advantage of those modes is the low overhead implementation of the single device - a meter device only needs to transmit data, and a data collector only needs to receive data. Bidirectional modes, on the other hand, support communication going both from a meter to a data collector as well as from a data collector to a meter. Data collectors are always bidirectional, and they can request information from a bidirectional meter.

The wM-Bus Stack supports two different operation models, with one being selected at compile-time:

- Operation model with nonvolatile memory
  - In this mode, the wM-Bus stack stores important runtime information in the Nonvolatile Memory (NVM). This gives the advantage of the stack being able to load the run-time information from the NVM at startup, in the case of a power failure, for example.
- Operation model without nonvolatile memory
  - In this mode, the wM-Bus stack stores all information in the random access memory (RAM). This gives the advantage of the stack not needing any NVM hardware resources, though all information is lost when the device performs a reset

Copyright © 2018, Texas Instruments Incorporated

The wM-Bus stack supports multiple communication modes that define the communication flow and configuration of the radio channel. The availability of these modes depends on the hardware that the specific software stack is design to run on. For example, some modes are specific to the 169 MHz range, but the hardware that the software stack is running on may not support frequencies this low. The following are the communication modes that are supported by the STACKFORCE wM-Bus stack (the number indicates whether the mode is bidirectional or unidirectional):

- Modes S1, S2 at 868.3 MHz
  - In Stationary mode, the metering devices send data several times a day. The data collector may save power as the metering devices send a wake-up signal before transmitting data.
- Modes T1, T2 at 868.3 MHz and 868.95 MHz
  - In Frequent Transmit mode, the metering devices periodically send data to collectors in range. The interval is configurable in terms of several seconds or minutes.
- Modes C1, C2 at 868.95 MHz and 868.525 MHz
  - Compact Mode is similar to Transmit Mode, but it allows for transmission of more data within the same energy budget and with the same duty cycle. It is suitable for walk-by or drive-by readout. The common reception of Transmit Mode and Compact Mode frames with a single receiver is possible.
- Modes N1 (a-f), N2(a-f), Ng
  - These are Narrowband communication Modes for long range transmissions in the 169 MHz ISM band in Europe.

2.2 STACKFORCE Serial Protocol

The serial command set provides access to the functions of the wM-Bus stack via a serial interface. This way, a PC or host controller can communicate with and control the wM-Bus device without having local access to the device firmware. This document describes the use of a host controller. This serial interface uses UART for communication between the wM-Bus device and the host device, with the following parameters:

- Baud rate of 115200 bps
- No flow control
- Data format: 8 data bits, no parity, 1 stop bit, LSB first

2.2.1 Binary Command Flow and Format

The serial interface was designed with the consideration that communication is most often started by the host controller. There are some cases where communication can be initiated by the device running the wM-Bus stack (these are called events and are described in more detail later in this document). Most commands transmitted by the host controller also require a response from the wM-Bus device.

The binary command format used by the serial interface consists of 5 parts: the start frame delimiter (SFD), the length, the CMD/type, the data and the cyclic redundancy check (CRC). These fields are defined as follows, and can be seen in Figure 2:

Start Frame Delimiter: This field is 1 byte and is always 0xA5. This signals that a serial command is being started.

Length: This field is 2 bytes (MSB first) and represents the length of the command in bytes. The SFD, length bytes and CRC bytes are not included in this value.

CMD/type: This field is 1 byte and represents the CMD field for that command type. This determines what the data looks like and what response commands could be returned.

Data: This field is the data that is being sent. The CMD/type of the command determines what required and optional bytes are contained in this field. It is therefore of a varying length of bytes (this length is specified by the length field).

Copyright © 2018, Texas Instruments Incorporated

CRC: This field is 2 bytes and is the checksum that can be used to determine if the serial data has been corrupted during transmission. It is created according to the wM-Bus block checksums described in EN-13757-4. The CRC is calculated using all bytes except for the SFD and length bytes. Figure 2 shows the fields used for this calculation in orange.

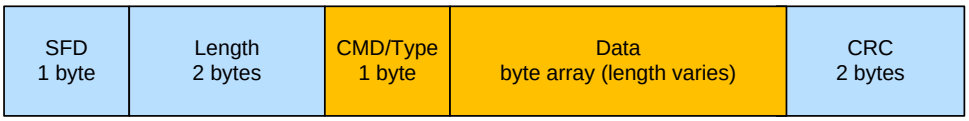

Figure 2. Serial Command Frame Format

If a response is not received within a short time of a command being sent (with the exception of a command that resets the wM-Bus device), or the response indicates that there was an error, then that packet needs to be resent.

2.2.2 Static Settings

The serial commands can be used for, among other things, setting the configuration options of the wM-Bus device, such as the device's address, encryption key or accessibility. Some of these configuration options, however, cannot be changed because they are write-protected in order to protect the wM-Bus device from failure or damage upon failure of the host device. The following settings are important ones that cannot be changed and therefore need to be supported by the host application:

- The serial interface is predefined in the hardware settings and changes are not supported
- The UART baud rate of the serial interface
- The flow control of the UART interface
- The data format of the UART interface

3 MSP430FR6047 Wireless M-Bus Serial Library

This section describes the library that provides functions and data structures to provide easy implementation on the MSP430FR6047 of the wM-Bus serial protocol to communicate with a wM-Bus device. The functions provided send out the corresponding commands over UART with the selected parameters. Each command is identified by its CMD/type. These are divided into three types:

- Generic commands: these commands can be used for both meter devices and data collectors, and they are generally used for generic tasks such as pinging the device and requesting information on the wM-Bus device's configuration settings.
- Application layer commands: these are more specific commands that are used for tasks such as sending data, reading data, and getting the status of the wM-Bus device
- Events: these are received by the host processor and indicate events such as a telegram being ready to be read. The appropriate actions can then be taken based on the information in the event data frame.

The implementation of all of the command functions and the definitions for the commands and events can be found in WMBusSerialCommands.c, and the descriptions and prototypes of these functions can be found in WMBusSerialCommands.h. The following subsections describe which functions respond to each command and what the possible responses from the wM-Bus device are for each command.

NOTE: The STACKFORCE serial interface defines commands that can be used for meter devices, data collectors, or both. The MSP430FR6047 Wireless M-Bus Serial Library and this document omits the commands that only apply to data collectors, as the MSP430FR6047 would only be used as a host processor for a meter device. Additionally, a few commands that can apply to meter devices or data collectors have parameters that only apply to data collectors, so those parameters have been removed from the serial library functions.

Copyright © 2018, Texas Instruments Incorporated

3.1 Generic Commands

Table 1 lists all of the generic commands and the corresponding MSP430FR6047 Wireless M-Bus Serial Library functions, along with the possible responses that can be expected from sending that command.

NOTE: The following responses can be received as a result of any command that expects a response, so they were not included in the table:

- Confirmation: SERIAL_CONFIRM_FAILED
- Confirmation: SERIAL_CONFIRM_TOO_FEW_BYTES (if required parameters are missing)

Table 1. Generic Commands and Responses

CMD	Serial Library Function Name	Possible Responses
SERIAL_CMD_TYPE_CONFIRM (0x00)	WMBusSerialCommands_confirmation	N/A (this is a response command)
SERIAL_CMD_TYPE_RF_DATA (0x01)	WMBusSerialCommands_RFData	Confirmation: SERIAL_CONFIRM_OK
SERIAL_CMD_TYPE_LOCAL_DATA (0x02)	WMBusSerialCommands_localData	N/A (this is a response command)
SERIAL_CMD_TYPE_SET_CONFIG (0x05)	WMBusSerialCommands_setConfig	• Confirmation: SERIAL_CONFIRM_OK • Confirmation : SERIAL_CONFIRM_CFG_NOT_SUPPORTED • Confirmation: SERIAL_CONFIRM_MODUS_NOT_SUPPORTED • Confirmation: SERIAL_CONFIRM_CHANNEL_NOT_SUPPORTED • Confirmation: SERIAL_CONFIRM_DEVICE_NOT_SUPPORTED
SERIAL_CMD_TYPE_GET_CONFIG (0x06)	WMBusSerialCommands_getConfig	• Local data with requested configuration data • Confirmation: SERIAL_CONFIRM_CFG_NOT_SUPPORTED
SERIAL_CMD_TYPE_PING (0x0A)	WMBusSerialCommands_ping	Confirmation: SERIAL_CONFIRM_OK
SERIAL_CMD_TYPE_STATUS (0x20)	WMBusSerialCommands_status	Status command with status byte
SERIAL_CMD_TYPE_SYNCHRONIZE (0x21)	WMBusSerialCommands_synchronize	• Confirmation: SERIAL_CONFIRM_OK • Confirmation: SERIAL_CONFIRM_NVM_NOT_READY
SERIAL_CMD_TYPE_NVM_ERASE (0x22)	WMBusSerialCommands_NVMErase	Confirmation: SERIAL_CONFIRM_OK
SERIAL_CMD_TYPE_BUFFER_CLEAN_UP (0xFA)	WMBusSerialCommands_bufferCleanUp	Confirmation: SERIAL_CONFIRM_OK
SERIAL_CMD_TYPE_MANUFR (0xFB)	WMBusSerialCommands_manufr	Depends on implementation
SERIAL_CMD_TYPE_INTERNAL_EVENT (0xFC)	WMBusSerialCommands_internalEvent	Confirmation: SERIAL_CONFIRM_OK
SERIAL_CMD_TYPE_UPDATE (0xFD)	WMBusSerialCommands_update	None
SERIAL_CMD_TYPE_RESET	WMBusSerialCommands_reset	None

3.2 Application Layer Commands

Table 2 lists the generic commands and the corresponding MSP430FR6047 Wireless M-Bus Serial Library functions, along with the possible responses that can be expected from sending that command.

NOTE: The following responses can be received as a result of any command that expects a response, so they were not included in the table:

- Confirmation: SERIAL_CONFIRM_FAILED
- Confirmation: SERIAL_CONFIRM_TOO_FEW_BYTES (if required parameters are missing)

Copyright © 2018, Texas Instruments Incorporated

Table 2. Application Layer Commands and Responses

CMD	Serial Library Function Name	Possible Responses
SERIAL_CMD_TYPE_APL_SETALARM (0x31)	WMBusSerialCommands_setAlarm	Confirmation: SERIAL_CONFIRM_OK
SERIAL_CMD_TYPE_APL_SET_PROPERTY (0x32)	WMBusSerialCommands_setProperty	Confirmation: SERIAL_CONFIRM_OK
SERIAL_CMD_TYPE_APL_SETUD_BYTE (0x37)	WMBusSerialCommands_setUDByte	Confirmation: SERIAL_CONFIRM_OK
SERIAL_CMD_TYPE_APL_SET_FAC (0x3B)	WMBusSerialCommands_setFACMode	Confirmation: SERIAL_CONFIRM_OK
SERIAL_CMD_TYPE_APL_GET_PROPERTY (0x3D)	WMBusSerialCommands_getProperty	Local data with status of the requested property
SERIAL_CMD_TYPE_APL_DESTROY_TLG (0x40)	WMBusSerialCommands_destroyTLG	Confirmation: SERIAL_CONFIRM_OK
SERIAL_CMD_TYPE_APL_TX_IR (0x41)	WMBusSerialCommands_transmitInstallationRequest	Local data with the ID of the installation request telegram
SERIAL_CMD_TYPE_APL_SET_ACCESSIBILITY (0x44)	WMBusSerialCommands_setAccessibility	Confirmation: SERIAL_CONFIRM_OK
SERIAL_CMD_TYPE_APL_GET_ACCESSIBILITY (0x45)	WMBusSerialCommands_getAccessibility	Local data with current accessibility configuration
SERIAL_CMD_TYPE_APL_SET_STATUS_BYTE_FLAG (0x4C)	WMBusSerialCommands_setStatusByteFlag	Confirmation: SERIAL_CONFIRM_OK
SERIAL_CMD_TYPE_APL_CLR_STATUS_BYTE_FLAG (0x4D)	WMBusSerialCommands_clrStatusByteFlag	Confirmation: SERIAL_CONFIRM_OK
SERIAL_CMD_TYPE_APL_CREATE_SPONTANEOUS (0x54)	WMBusSerialCommands_createSpontaneousTelegram	Local data with ID of the telegram
SERIAL_CMD_TYPE_APL_TX_SPONTANEOUS (0x55)	WMBusSerialCommands_transmitSpontaneousTelegram	Confirmation: SERIAL_CONFIRM_OK
SERIAL_CMD_TYPE_APL_RX_DATA (0x56)	WMBusSerialCommands_readData	Local data with the read data bytes
SERIAL_CMD_TYPE_APL_SET_LONG_HEADER (0x5C)	WMBusSerialCommands_setLongHeader	Confirmation: SERIAL_CONFIRM_OK
SERIAL_CMD_TYPE_APL_RX_WHOLE_TLG (0x5E)	WMBusSerialCommands_readWholeTelegram	Local data with the payload of the telegram

3.3 Events

Events are received by the host processor as a way of the wM-Bus device communicating information based on the wireless communication. When one is received, the host processor can take the necessary action. It is up to this application level code to handle these events and determine what actions need to be taken, if any. Table 3 lists the events that are available for meter devices and descriptions of each. These event defines can be found in WMBusSerialCommands.c.

Table 3. Events for Meter Devices

Event	Description
SERIAL_CMD_TYPE_APL_EVT_TLG_AVAILABLE (0x33)	Informs the host processor that a telegram is available to be read
SERIAL_CMD_TYPE_APL_EVT_UD_REQ (0x36)	Informs the host processor that user data has been requested
SERIAL_CMD_TYPE_APL_EVT_TX (0x38)	Sent to the host processor when a telegram has been transmitted
SERIAL_CMD_TYPE_APL_EVT_ALARM_TX (0x39)	Sent to the host processor when an alarm telegram has been transmitted
SERIAL_CMD_TYPE_MTR_EVT_STATUS_IDLE (0x3C)	Informs host processor that transmission process is finished

3.4 Other MSP430FR6047 Wireless M-Bus Serial Library Functions

In addition to the functions for the wM-Bus commands, the serial library also contains functions for setting up and configuring the UART communication and for calculating the CRC. The following subsections briefly explain these functions and describe where and how to use them.

3.4.1 CRC Calculation Function

The following function calculates the CRC for the wM-Bus serial commands: WMBusSerialCommands_crcCalc. This function calculates a 16-bit CRC according to the wM-Bus block checksums described in EN-13757-4. It is called on a single byte at a time, and it is used on the wM-Bus commands by being called sequentially on each byte, starting with the MSB. The CRC polynomial that is used is CRC_POLYNOM, which can be found in WMBusSerialCommands.c. By default, it has a value of 0x3D65, which corresponds to the following polynomial:

Copyright © 2018, Texas Instruments Incorporated

$x^{16} + x^{13} + x^{12} + x^{11} + x^{10} + x^8 + x^6 + x^5 + x^2 + 1$

The initial value is 0, and the final CRC is complemented.

The CRC is calculated using all of the bytes in a wM-Bus serial command except for the SFD byte and the two length bytes.

3.4.2 UART Initialization Function

The following function initializes UART for the MSP430FR6047 evaluation board for serial communication with the wM-Bus device:

`WMBusSerialCommands_UARTInit`

This function performs the following tasks:

- Configures the UART RX and TX pins
- Configures the clocks, selects the eUSCI clock source input, and sets the clock prescaler and modulation values to produce a baud rate of 115200 bps (given a source clock of 8 MHz)
- Sets the UART parameters (no parity, LSB first, 1 stop bit)
- Enables UART RX interrupts

4 Wireless M-Bus Water Meter Demo

This section describes how to use the provided water meter demo to show how the wM-Bus serial commands can be used on the MSP430FR6047 to communicate with a CC1350 running the wM-Bus stack. This demo shows the use of a few serial commands and provides a guide on how to use the commands in general to extend to other applications.

This demo takes ADC captures with the MSP430FR6047 and then uses the CC1350 running the wM-Bus stack to send the data to a collector device running on another CC1350. Multiple commands are shown, including ones to configure wM-Bus parameters on the meter device and to send data over the wM-Bus protocol.

4.1 Required Hardware and Software

To run the demo, the following hardware is necessary:

- One MSP430FR6047 EVM (EVM430-FR6047)
- Two CC1350 LaunchPad development kits (LAUNCHXL-CC1350) or two CC1310 LaunchPad development kits (LAUNCHXL-CC1310)

NOTE: Other hardware configurations can be used instead of the CC1350 LaunchPad development kit, as long as the correct wM-Bus software stack is used. More information on this can be found on the Wireless M-Bus Protocol Software page.

The following software is required:

- Wireless M-Bus Protocol Software
- Ultrasonic Sensing Software Library for water metering with wireless M-Bus support
- Ultrasonic Sensing Design Center GUI
- Uniflash Standalone Flash Tool for TI Microcontrollers (MCU), Sitara Processors, and SimpleLink devices
- Code Composer Studio™ IDE or IAR Embedded Workbench® IDE
- Wireless M-Bus Suite by STACKFORCE GmbH (formerly Steinbeis or STZEDN)

Copyright © 2018, Texas Instruments Incorporated

4.2 Setting Up the Demo

To set up the demo, first flash one of the CC1350 (or CC1310) LaunchPad development kits with a meter image, and the other with the compatible collector image. For this demo, the T2 mode images is used (using other modes requires a similar process). For the default install location, the necessary images can be found at the following location:

```
C:\ti\wmbus_cc13x0_rtos_1_2_0\wmbus-cc13xx-rtos-1.2.0\hex\CC13xx_RTOS
```

To flash the devices, the Uniflash Standalone Flash Tool can be used. For each LaunchPad development kit, plug the device into the computer using the micro USB cable. In the Uniflash GUI window, any supported devices connected through USB are automatically detected. These devices can be seen at the top of the GUI window. Press the *start* button to bring the GUI to a new window that is used to choose the image to flash the device with (see Figure 3).

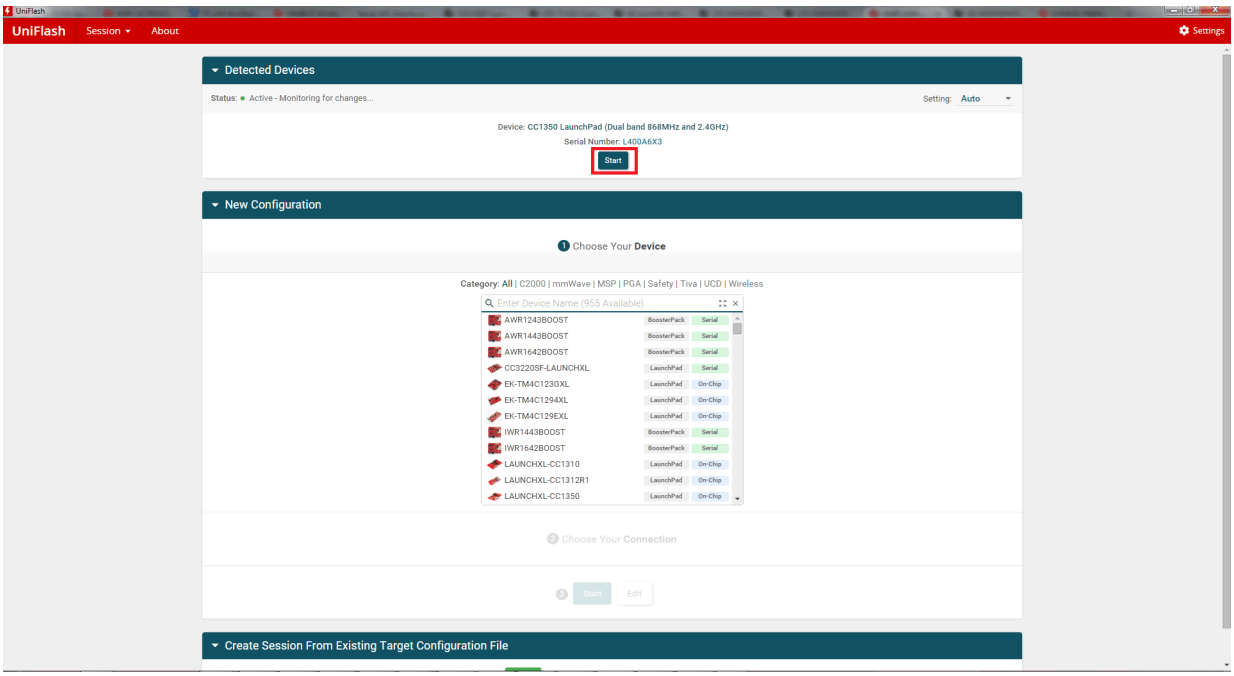

Figure 3. Uniflash Standalone Flash Tool Setup

In the new GUI window, chose the desired image to flash the device. After this, press the *Load Image* button to flash the device with the selected image. The console window at the bottom of the GUI shows when the program is loaded successfully (see Figure 4).

Copyright © 2018, Texas Instruments Incorporated

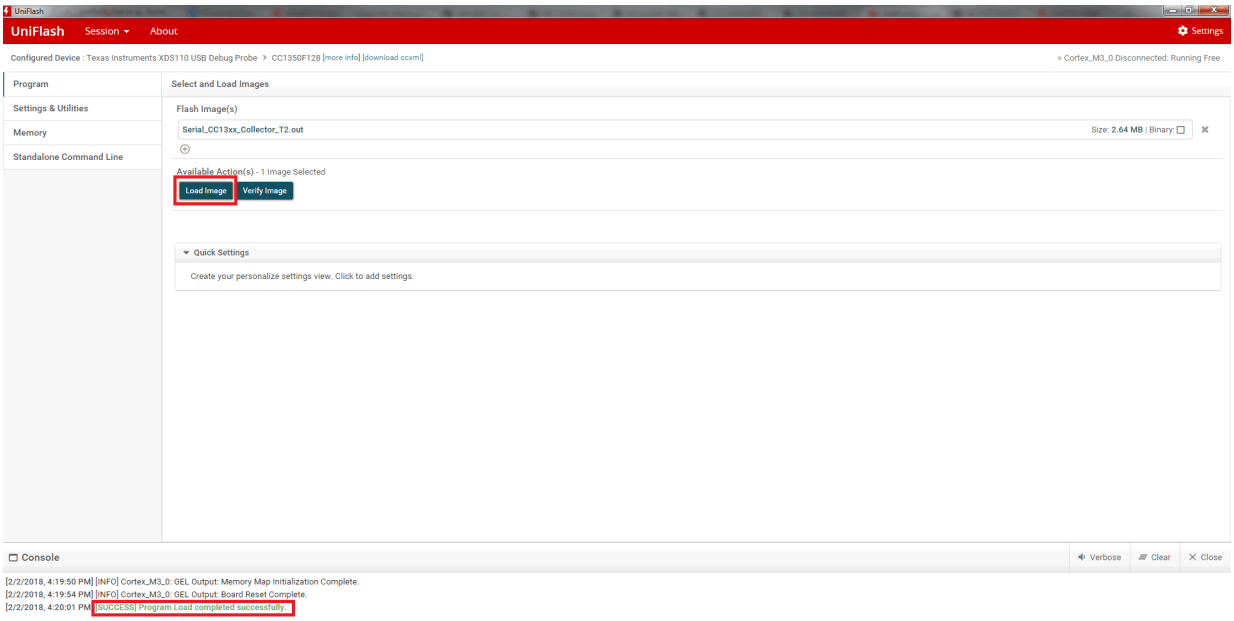

Figure 4. SmartRF Flash Programmer 2 Flash Success

After flashing both LaunchPad development kits, the wM-Bus devices are ready to use. Next, connect the LaunchPad development kit with the meter image to the MSP430FR6047 EVM using the BoosterPack™ plug-in module connectors. Make sure J1 on the LaunchPad development kit is connected to J5 on the MSP430FR6047 EVM, and J2 on the LaunchPad development kit is connected to J6 on the MSP430FR6047 EVM (see Figure 5). Before attaching the LaunchPad development kit, make sure that the two RF_POW jumpers (next to the LCD) on the MSP430FR6047 EVM are populated.

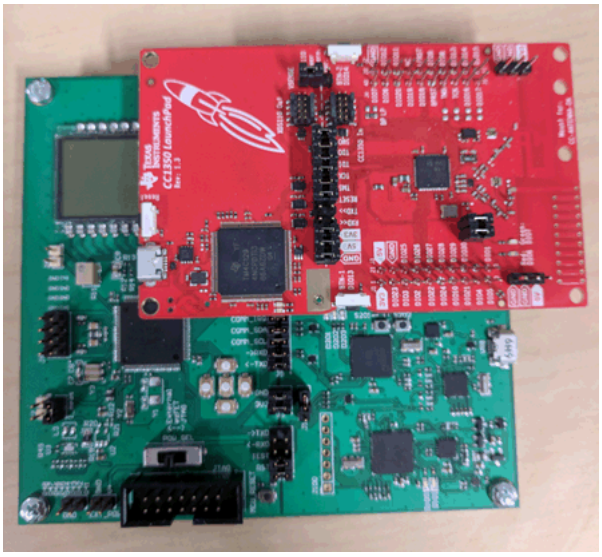

Figure 5. Meter Hardware Setup

To program the MSP430FR6047, use a micro USB cable to connect it to a computer (make sure the attached LaunchPad development kit is no longer connected to the computer), and make sure that the POW_SEL switch is set to the ezFET position (middle position). Using Code Composer Studio IDE (CCS) or IAR Embedded Workbench IDE, program the device using the Ultrasonic Sensing Design Center GUI Application project.

Copyright © 2018, Texas Instruments Incorporated

4.3 Running the Demo

After the devices have all been programmed, make sure that the MSP430FR6047 EVM (with the attached LaunchPad development kit) and the LaunchPad development kit with the collector image are both connected to the computer through USB. Next, open the Wireless M-Bus Suite. This Java-based GUI tool is used to configure and serially communicate with the collector device (the LaunchPad development kit with the collector image). An introduction to using this tool is in the Design Guide for the Low-Power wM-Bus Communications Module Reference Design. For this demo, the tool is used a little differently, as the serial console is used for the most part, because the rest of the GUI is intended mostly for using the Wireless M-Bus Suite with both the collector and meter, while in this case the meter is the MSP430FR6047 EVM and CC1350 LaunchPad development kit combination.

NOTE: STACKFORCE also provides and supports a command line tool for configuring devices running the wireless M-Bus stack, similar to the Wireless M-Bus Suite.

In the *Navigator* pane on the left of the GUI, right click *Data Collectors* to add a new collector. In the window that is opened to create a new collector device, enter a name for the collector, and enter the encryption key (which must match that of the collector). In this case, the key should be 00112233445566778899AABBCCDDEEFF (see Figure 6). This is all that is required to set up the collector for this demo, so press *Finish* to complete the creation of the collector device.

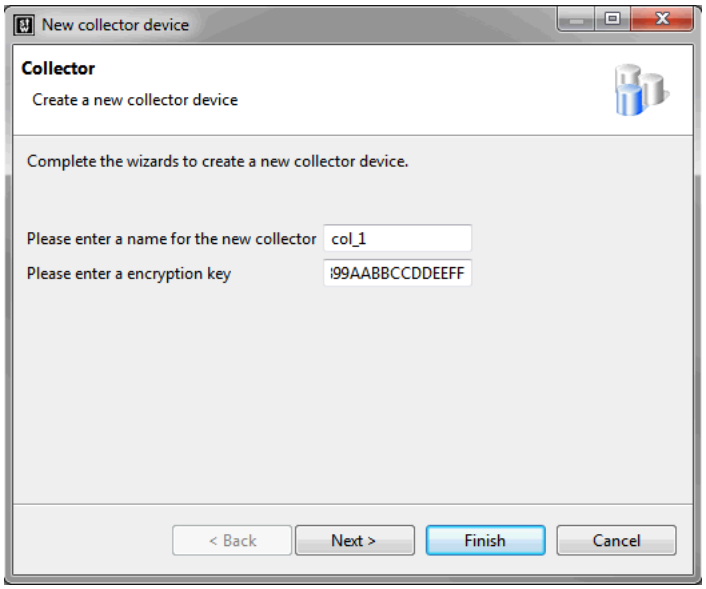

Figure 6. New Collector Device Creation

The console at the bottom of the window now shows text describing how to get started with the serial communication. First, type "open" into the console and press enter. When prompted, select the port that to communicate on. When you have determined what port your collector LaunchPad development kit UART connection corresponds to, type this into the console window and press enter. You are then prompted you to select the baud rate. Because the wM-Bus serial interface that is being used supports only a baud rate of 115200, type this value in and press enter. The port is opened for UART communication. Figure 7 shows these steps.

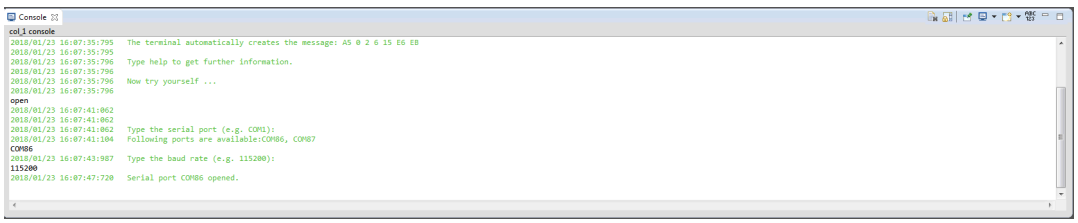

Figure 7. Opening the Port for UART Communication

Copyright © 2018, Texas Instruments Incorporated

At this point, the collector is ready to start being configured for wM-Bus communication. The MSP430FR6047 EVM starts configuring the meter LaunchPad development kit over UART upon startup. The following steps are taken upon startup:

1. Ping the device to make sure that it is receiving UART commands
2. Clean the buffer so that the wM-Bus device can receive the maximum amount of UART commands.
3. Set the meter address.
4. Set the encryption key.
5. Set the device as connected (because we are about to tell the collector to add this device as a connected meter).
6. Set the periodic interval at which data is transferred.

There are a few notes to keep in mind about these steps:

- The wM-Bus device can only handle commands so quickly. If a time-out is occurring, that means that the device was receiving bytes faster than it could handle them. This means that it may be necessary to put a delay in between commands to allow the wM-Bus device to process them.
- The meter address that is used for this demo is 0x5133800000012307. Information on how the address is constructed can be found in the Design Guide for the Low-Power wM-Bus Communications Module Reference Design.
- The encryption key used is 0x00112233445566778899AABBCCDDEEFF. This can be changed to any 16-byte key that has at least 8 unique bytes, as long as it is the same as that used by the collector.
- The periodic interval is set to 100 seconds. This is to make it easier to send commands to the collector in the demo without being interrupted by telegrams that are received from the meter. This value can be changed depending on the application.

After the MSP430FR6047 has sent the initial commands to the meter LaunchPad development kit on startup, the Wireless M-Bus Suite can be used to send initial commands to the collector LaunchPad development kit and to add the desired meter to the list of meters. The following steps can be taken to do this:

1. Ping the device to make sure that it is receiving UART commands.
2. Ask the wM-Bus collector device for any parameters (version string was chosen as an example).
3. Clean the buffer so that the wM-Bus device can receive the maximum amount of UART commands.
4. Clear the meter list.
5. Add the desired meter to the meter list using both the meter address and the meter's encryption key.

Figure 8 shows these steps. The text in black is what was entered by the user, and the text in blue shows what was sent to the collector device (arrow pointing right) and what was received from the collector device (arrow pointing left). Note that these commands are all in the raw bytes. For a description of what each command does and what parameters it includes, see the documentation provided with the wM-Bus stack. The SFD byte, length bytes, and CRC bytes do not need to be input into the console window.

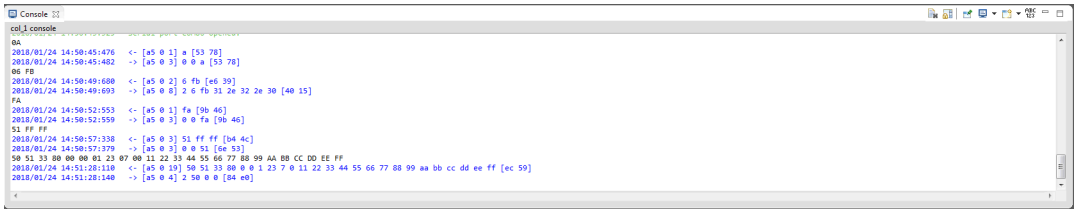

Figure 8. Adding the Meter to the Meter List

At this point, the meter is sending out periodic telegrams at the rate that was set on startup, and the collector is looking for telegrams from a meter with that address. To complete the demo, the data in the telegram is filled with meter data obtained by the Ultrasonic Sensing Solution module on the MSP430FR6047 EVM. To do this, first open the Ultrasonic Sensing Design Center GUI. First, connect to the MSP430FR6047 EVM by choosing *Connect* under the *Communications* tab at the top of the GUI. If

Copyright © 2018, Texas Instruments Incorporated

this is successful, the GUI reports "HID connected to MSP430FR6047 on Evaluation Module" at the bottom of the GUI. Next, clicking on the *ADC Capture* tab at the top of the GUI shows the ADC capture screen, which contains a button labeled *Capture* see Figure 9) to capture data from the ADC in the Ultrasonic Sensing Solution module on the MSP430FR6047 EVM and display it in the GUI. The software used for this demo also populates the next wM-Bus periodic telegram with this data.

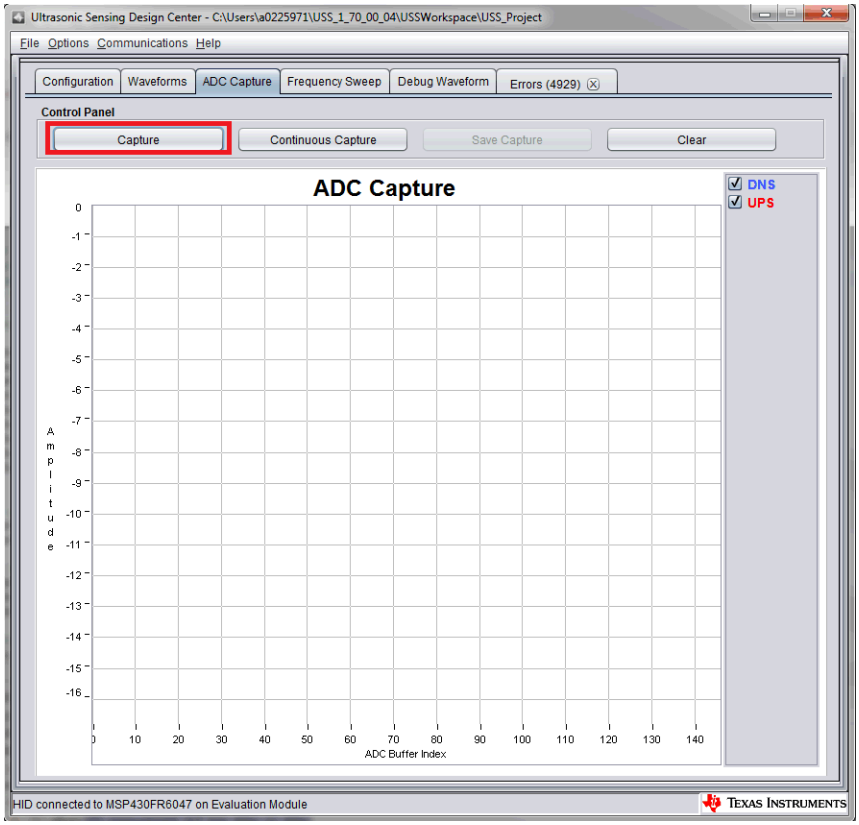

Figure 9. ADC Capture

After clicking the *Capture* button, the next telegram sent to the Wireless M-Bus Suite contains this data (see Figure 10). The first user command (the text in black) again shows that the meter has been added to the collector's meter list. The three lines in blue starting at time 14:41:51:958 show received telegrams. The fourth byte in this line (after the first set of brackets) gives the telegram ID, which can be used to read the data in the telegram. For the first telegram, the telegram ID is 0. Entering 5E 00 reads the unencrypted data of this telegram. After the first set of brackets in the last line in Figure 10, the first 2 bytes indicate that this is a response to the data read request (2 5e). The rest of the bytes before the second set of brackets, are the data received from the MSP430FR6047 of the wM-Bus protocol.

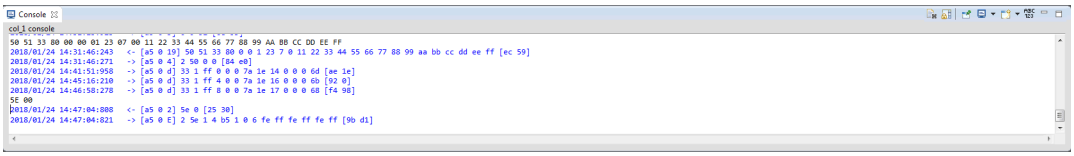

Figure 10. Reading a Received Telegram

Copyright © 2018, Texas Instruments Incorporated

5 Using Wireless M-Bus Serial Library for Customer Applications

The functions and data structures provided in the wireless M-Bus serial library can be used to customize meter applications running on the MSP430FR6047. Functions are provided to configure many different settings on the wireless M-Bus device in addition to the few shown in the demo. Additionally, the MSP430FR6047 can also receive serial commands. For bidirectional communication modes, these commands can come from telegrams and requests that are sent from the collector device. Most commonly, though, this is useful because almost all serial commands sent to the device running the wM-Bus stack cause a response to be sent back. Handling these responses can be done using the following steps:

- Uncomment the following line the WMBusSerialCommands_UARTInit function in WMBusSerialCommands.c:

  ```
  //EUSCI_A_UART_enableInterrupt(EUSCI_A3_BASE,EUSCI_A_UART_RECEIVE_INTERRUPT);
  ```

- Include an interrupt service routine in the application code that handles UART receive interrupts. An example ISR can be seen below. It makes use of a variable of type rxDataFrame, which stores the different data fields of a wM-Bus command and is defined in WMBusSerialCommands.h. Code can be added to the application to process the data accordingly after each command (or command response) is received.

```
/* SFD byte        = 0
 * 1st length byte = 1
 * 2nd length byte = 2
 * Code byte       = 3
 * Data bytes      = 4
 * 1st CRC byte    = 5
 * 2nd CRC byte    = 6 */
uint8_t currentRxByte = 0;
rxDataFrame currentDataFrame;
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector = EUSCI_A3_VECTOR             // eUSCI ISR
__interrupt void USCI_A3_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(EUSCI_A3_VECTOR))) USCI_A3_ISR (void)
#else
#error Compiler not supported!
#endif
{
    switch(__even_in_range(UCA3IV, USCI_UART_UCTXCPTIFG))
    {
        case USCI_NONE: break;
        case USCI_UART_UCRXIFG:
            uartTestCount++;
            RXData = UCA3RXBUF;
            switch(currentRxByte)
            {
                case 0:
                    if(RXData == SFD)
                    {
                        currentRxByte++;
                    }
                    __no_operation();
                    break;
                case 1:
                    tempLength1 = RXData;
                    currentRxByte++;
                    break;
                case 2:
                    currentDataFrame.length = RXData + (tempLength1 << 8);
                    currentRxByte++;
                    __no_operation();
```


Copyright © 2018, Texas Instruments Incorporated

```
                    break;
                case 3:
                    currentDataFrame.CMD = RXData;
                    currentRxByte++;
                    __no_operation();
                    break;
                case 4:
                    if(currentDataFrame.length)
                    {
                        currentDataFrame.data[tempDataByte] = RXData;
                        tempDataByte++;
                        if(tempDataByte >= currentDataFrame.length-1)
                        {
                            tempDataByte = 0;
                            currentRxByte++;
                        }
                    }
                    break;
                case 5:
                    tempCRC1 = RXData;
                    currentRxByte++;
                    break;
                case 6:
                    currentDataFrame.CRC = RXData + (tempCRC1 << 8);
                    currentRxByte = 0;
                    __no_operation();
                default: break;
            }
            break;
        case USCI_UART_UCTXIFG: break;
        case USCI_UART_UCSTTIFG: break;
        case USCI_UART_UCTXCPTIFG: break;
        default: break;
    }
}
```

6 Summary

The provided MSP430FR6047 Wireless M-Bus Serial Library can be used to easily implement a wireless water meter using the MSP430FR6047 and a wireless device running the wireless M-Bus software stack.

7 References

1. CC13xx wM-Bus S-Mode
2. CC13xx Combined wM-Bus C-Mode and T-Mode

Copyright © 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