Application Report
SWRA491A–September 2015–Revised October 2016

Implementing Eddystone™ Bluetooth® Smart Beacons Using the TI BLE-Stack™

Chester Kim

ABSTRACT

This application note describes the basic concept of Eddystone™ Beacon, how connectable beacon applications can be implemented on top of the existing sample application SimpleBLEPeripheral, and how the compatibility to Eddystone specification cab be tested. This note also introduces events, callbacks, and usages of hardware drivers for obtaining information to populate the TLM frame.

Contents

List of Figures

List of Tables

Trademarks

1 Introduction

As described in the *Bluetooth® low energy Beacons* Application Note (SWRA475), a Bluetooth® low energy (BLE) beacon is a device broadcasting a non-connectable advertising packet carrying small pieces of information to nearby devices. Depending on the timing and the information conveyed within the packet, beacons can enable a variety of use cases, including but not limited to proximity awareness, synchronization, identification, or just informing.

Some vendors have defined a protocol or format for how the beacon is implemented in a larger ecosystem. For example, Apple has defined the iBeacon protocol, available under an MFi license, for use with iOS devices.

Eddystone™ is an open beacon protocol specification from Google aimed at improving proximity-based experiences, with support for both Android and iOS smart device platforms. These experiences are implemented by specifying various beacon payload formats, defined as frames-types, as well as a corresponding set of APIs used to access these payloads from the smart device (such as an Android smart phone).

This application note describes how to implement an example Eddystone beacon device using the TI BLE-Stack™ V2.2 SDK on the SimpleLink™ CC2640 Bluetooth Smart wireless MCU. The lower power consumption and wide operating voltage capability make the CC2640 an ideal platform for implementing battery-powered Eddystone beacons.

Although beacons are, by definition, broadcast-only devices, the concept of a connectable beacon will be described. Having the beacon enter a connectable state (technically, switching from the GAP broadcaster to the peripheral role) is desirable for the purposes of provisioning and updating the metadata of the beacon. The Eddystone protocol defines a mode that allows the beacon data to be updated by an authorized client.

The project files and the source code files created or modified for Eddystone Beacon device implementation are available as a GitHub repository, which can be found on the TI BLE Wiki.

2 Abbreviations

ADV	Advertising Packet
API	Application Program Interface
BLE	Bluetooth low energy
CCS	Code Composer Studio
ID	Identifier
GAP	General Access Profile
MCU	Microcontroller
OAD	Over the Air Download
SDK	Software Development Kit
TI-RTOS	Texas Instruments Real Time Operating System
TLM	Telemetry
TX	Transmission
UID	Unique Identity
URL	Uniform Resource Locator

3 Objective

This application shows what must be implemented on top of the existing SimpleBLEPeripheral sample application to make a complete working Eddystone beacon example project named SimpleEddystoneBeacon.

4 Project Overview and Prerequisites

Prior to following the examples described in this application note, the designer should have a detailed understanding of the TI BLE-Stack SDK as described in the SW Developer's Guide (SWRU393), the Bluetooth low energy Beacons application note (SWRA475), and the Google Eddystone protocol specification, which can be found at https://github.com/google/eddystone/blob/master/protocol-specification.md.

The sample application SimpleEddystoneBeacon requires TI BLE-Stack V2.2. Either IAR Workbench for ARM 7.50.3 or CCS 6.1 IDEs can be used to build the project. The SimpleEddystoneBeacon project runs on the CC2650 LaunchPad reference platform. An optional smartphone and test application can be used to validate the Eddystone beacon implementation.

The TI GitHub repository ble_examples includes the minimum number of files necessary to build the project when they are put on top of the existing BLE-Stack V2.2 installation. The repository consists of project and workspace files for IAR/CCS, as well as *simpleEddystoneBeacon.c* and *simpleEddystoneBeacon.h*, which are for the application, and *eddystoneURLCfg.c* and *eddystoneURLCfg.h*, which provide URF Configuration service. To install the SimpleEddystoneBeacon project, copy the following folders into the appropriate directories in the SDK:

- ble_examples/examples/cc2650lp/simple_eddystone
- ble_examples/src/examples/simple_eddystone/cc26xx
- ble_examples/src/profiles/EddystoneURLCfg

5 Design and Implementation

5.1 Operation Modes Overview

The Eddystone protocol defines three frame-type formats (referred to as "frames"), which are transmitted by the beacon device as described in Table 1.

Table 1. Eddystone Frame Formats

Frame-type	Description
UID	Unique identifier that can identify a particular beacon. Useful for location aware applications.
URL	Compressed web URL, such as https://goo.gl/Aq18zF, that can be launched by the smart device application.
TLM	Telemetry information, such as battery voltage and temperature of the beacon.

These frames are described in more detail in the Eddystone protocol specification. Each frame represents an advertising packet, up to the maximum 31-byte payload defined by the Bluetooth 4.2 core specification.

The Eddystone Beacon defines two modes of operation: regular advertising mode and an optional URL configuration mode. These operating modes are related by five individual operational states. In regular advertising mode, the beacon operates as a traditional broadcaster sending non-connectable advertisements consisting of UID, URL, or TLM frames. In URL configuration mode, the beacon transmits connectable advertisements to allow a client to connect, and if authorized, update the URL content and the system parameters through the URL configuration service.

The operating state transitions for these modes of operation are depicted in Figure 1, along with the corresponding actions used to transition states in the SimpleEddystoneBeacon example project. States shown with a grey circle represent URL configuration mode, while the other states represent regular advertising mode.

The Eddystone protocol does not specify the type, when, or how often a particular frame is transmitted, or the actions that trigger a particular state transition. These details are implementation-specific. An Eddystone Beacon may, for example, transmit only URL frames, or a combination of all three frames at different intervals.

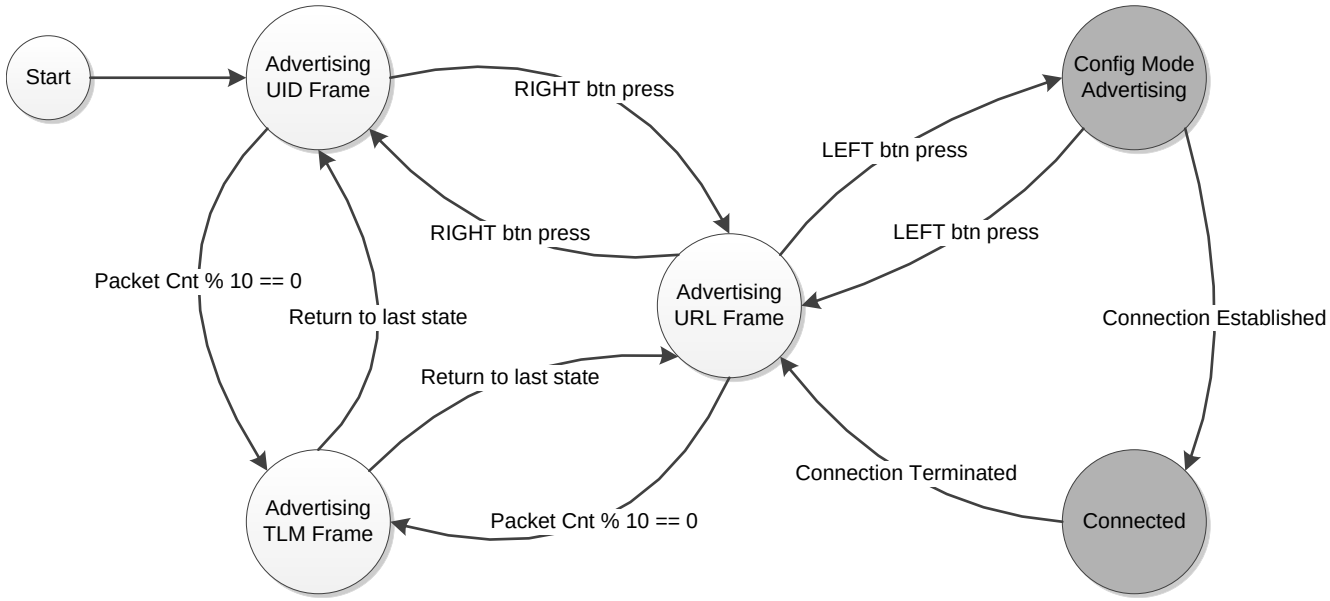

Figure 1. SimpleEddystoneBeacon Operation Mode Transitions

In this design, the beacon device sends either UID frames or URL frames regularly, and TLM frames intermittently, while in regular advertising mode. The state transitions between UID and URL are toggled by a RIGHT button press. For practical reasons, TLM frames are expected to be observed less frequently than UID or URL frames, so in this design, they are sent only every 10th advertise of UID or URL.

URL configuration mode can be entered by entering the config mode advertising state in this design. It is triggered by a LEFT button press. Exiting configuration mode can be caused by a LEFT button press, or any type of termination of the connection. Generally, the central terminates the connection when the beacon is finished updating, to let the beacon resume regular advertising mode.

To use a button press to trigger some of the transitions, the key event handler must be implemented. Button press handling is implemented slightly differently by hardware platform. In this implementation, functions in *board_key.c* are used for the CC2650 LaunchPad. In any case, *SimpleEddystoneBeacon_keyChangeHandler()* is registered as the callback and eventually the key event SEB_KEY_CHANGE_EVT is handled by *SimpleEddystoneBeacon_handleKeys()*.

For the CC2650 LaunchPad, all RIGHT btn presses correspond to BTN-2, and all LEFT button presses correspond to BTN-1.

5.2 Regular Advertising Mode

Advertising is used for four different purposes, regardless of the operation mode. Properties of each type of the ADVs used in this implementation are summarized in Table 2.

Table 2. Properties of Advertising Packets

	UID/URL/TLM Frame	URL Configuration Mode ADV
ADV Event Type	ADV_NONCONN_IND	ADV_IND
ADV Type Flags	BREDR_NOT_SUPPORTED	BREDR_NOT_SUPPORTED | GENERAL_DISCOVERABLE
ADV Interval	From URL Cfg for UID/URL, 10 times longer for TLM	Default Interval
TX Power	From URL Cfg	Medium Power Mode
Connectable	No	Yes

As switching from one advertising packet to another at run time is required, how to set the parameters to change the properties for regular advertising mode is described in this section. How to get the information to populate the payload of each packet is also covered.

5.2.1 Setting Advertising Parameters

As shown in Table 2, there are five properties to be changed when toggling between regular advertising mode and URL configuration mode.

ADV Event Type can be configured using *GAPRole_SetParameter()*, with GAPROLE_ADV_EVENT_TYPE as the parameter ID and GAP_ADTYPE_ADV_NONCONN_IND as the value, because regular advertising mode is non-connectable.

ADV Type Flags is an item included in an advertising packet. The GAP_ADTYPE_FLAGS field of the struct variable *eddystoneAdv* for regular advertising mode is initialized with GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED, because regular advertising mode is non-discoverable.

ADV Interval is set by specifying minimum and maximum intervals for limited and general discoverable ADVs. Typically, all four parameters are set to the same value to get the desired interval as follows:

```
GAP_SetParamValue(TGAP_LIM_DISC_ADV_INT_MIN, advInt);
GAP_SetParamValue(TGAP_LIM_DISC_ADV_INT_MAX, advInt);
GAP_SetParamValue(TGAP_GEN_DISC_ADV_INT_MIN, advInt);
GAP_SetParamValue(TGAP_GEN_DISC_ADV_INT_MAX, advInt);
```

That particular design should select advertising parameters that balance power usage with beacon power use. Using faster advertising intervals allows for quicker discovery by the smart device, while longer intervals allow for longer beacon battery life.

The value from the parameter corresponding with URLCFGSVC_BEACON_PERIOD of the URL configuration service is reflected on *advInt*. Unit conversion is necessary in this case because the value from the URL configuration service is in millisec, while *GAP_SetParamValue()* expects a number of 625-microsec ticks. See Section 5.3.2 for more details about URL configuration service. Basically the beacon period from the URL configuration service is supposed to be applied to URL frame advertising. However, it is used for UID as well in this implementation, because no external means to setup the interval for UID is defined in the Eddystone specification. The interval for TLM depends on the interval for UID/URL, because it is based on the number of UID/URL frames that have sent since the last TLM frame in this implementation.

Use the value from the parameters corresponding with URLCFGSVC_ADV_TX_PWR_LVLS and URLCFGSVC_TX_POWER_MODE of the URL configuration service to get the designated TX Power in dBm.

Once all of the above properties are set and the ADV data is updated, advertising is ready to start. Section 5.2.2 describes more about updating ADV data.

Lastly, non-connectable advertising mode is enabled to start sending out the packets at the given rate and power. *GAPRole_SetParameter()*, with the parameter ID GAPROLE_ADV_NONCONN_ENABLED, is used to start or stop advertising in regular advertising mode.

All the procedures described in this section are done in *SimpleEddystoneBeacon_applyConfiguration()* and *SimpleEddystoneBeacon_startRegularAdv()*.

5.2.2 Populating the Eddystone Frame Payload

Data in UID and URL frames generally don't need to be changed unless the beacon device has been updated during the last configuration mode operation. The Ranging Data subfield of UID frame and all subfields of URL frame except for Frame Type must be updated upon exiting configuration mode. Updates are implemented in *SimpleEddystoneBeacon_applyConfiguration()*.

TI recommends a few algorithms to generate a 10-byte ID Namespace subfield and a 6-byte ID Instance subfield of UID by the Bluetooth core specification. This implementation example uses some meaningless numbers, but for an actual product, the number assignments in function *SimpleEddystoneBeacon_initUID()* must be replaced with proper algorithms.

For URL frame, *SimpleEddystoneBeacon_encodeURL()* facilitates conversion of regular null-terminated string URL into encoded numbers.

TLM frames require special handling compared to UID and URL frames, because the frame data must be updated with the most recent information each time a TLM frame is broadcasted. Updating the TLM payload is done in *SimpleEddystoneBeacon_updateTLM()*.

The battery voltage VBATT subfield can be obtained from *AONBatMonBatteryVoltageGet()*. To use the battery monitor function, include *aon_batmon.h* of driverlib.

A constant value is currently used for the temperature TEMP subfield, because the CC2650 LaunchPad does not have a temperature sensor.

The ADV_CNT subfield needs an event from the stack, notifying the application upon completion of each advertising operation. *HCI_EXT_AdvEventNoticeCmd()* is the API to register an event that is sent upon completion of an ADV send out. Register SEB_ADV_COMPLETE_EVT and check it in the processing loop of stack events that come with the signature of 0xFFFF. The event SEB_ADV_COMPLETE_EVT is eventually processed in *SimpleEddystoneBeacon_processAdvCompleteEvt()*. This event processing function is not only for counting the number of ADVs, but also which type of Eddystone frame should be sent.

A 1-sec resolution clock tick counter is used, as an example, for SEC_CNT subfield, though the Eddystone spec requires 100-ms resolution. UTC_clock.c, originally used by the TimeApp example project, was added for this. Typically, a TLM frame is supposed to be sent intermittently, so in this example it is sent every 10 seconds by default, allowing for a 1-sec resolution.

When a local Eddystone frame is populated with all necessary information, it must be handed over to the BLE stack. *GAPRole_SetParameter()*, with the parameter ID GAPROLE_ADVERT_DATA, is used for that purpose.

All the procedures described in this section are handled by *SimpleEddystoneBeacon_selectFrame()*.

5.3 URL Configuration Mode

The URL configuration mode is an optional method to update the URL content of the beacon. A designer may wish to omit this mode, or modify the accompanying service based on implementation-specific requirements.

5.3.1 Setting Advertising Parameters

To create the set up for configuration mode advertising, use the same method described in Section 5.2.1 but with different properties than shown in Table 2. In this section, only the differences from Section 5.2.1 are noted.

For ADV Event Type, GAP_ADTYPE_ADV_IND as the value is used because configuration mode is connectable.

For ADV Type Flags, the GAP_ADTYPE_FLAGS field of the struct variable *eddystoneCfgAdv* for configuration mode is initialized with GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED, together with GAP_ADTYPE_FLAGS_GENERAL, because configuration mode is discoverable.

advInt for ADV Interval is set to GAP_ADTYPE_ADV_IND. Typically the interval for configuration mode is shorter than that for regular advertising mode.

TX Power should be set to the value corresponding to TX_POWER_MODE_MEDIUM.

When all of the above properties are set and the ADV data is updated using *GAPRole_SetParameter()* with the parameter ID GAPROLE_ADVERT_DATA, advertising is ready to start.

Lastly, connectable advertising mode is enabled by calling *GAPRole_SetParameter()* with the parameter ID GAPROLE_ADVERT_ENABLED.

All the procedures described in this section are handled in *SimpleEddystoneBeacon_startConfigAdv()*.

5.3.2 URL Configuration Service

URL Configuration service is implemented in *eddystoneURLCfg.c*. The application can access the nine characteristics of the service using *URLCfgSvc_GetParameter()* and *URLCfgSvc_SetParameter()*. The application registers a callback called when changes happen in the characteristics by the central. In this implementation, only a change in the Reset characteristic notifies the application.

Once the beacon device is connected with a central (such as a smart phone) in URL configuration mode, some of the system properties on the beacon device can be modified by the central through writing to characteristics, if the Lock State is FALSE. Only Unlock characteristic can be written to if Lock State is TRUE.

Updated characteristics are reflected on the corresponding properties in *SimpleEddystoneBeacon_applyConfiguration()* when the beacon device exits URL configuration mode to enter regular advertising mode by being disconnected for any reason, such as termination or time out. However, actions should be immediately taken when the Reset characteristic changes.

6 Test

To verify all the features and functions described in the application note, devices implementing the observer or central GAP roles are necessary. A GitHub Eddystone repository comes with validation tools for that purpose. Using Eddystone Validator and Eddystone-URL Config Validator, we can verify URI/URL/TLM frames and URL configuration mode, respectively. Figure 2 shows the screenshots of Eddystone Validator and Eddystone-URL Config Validator that have been used to verify SimpleEddystoneBeacon sample application. The validators used to validate SimpleEddystoneBeacon project are based on the commit SHA b5f23b3f895007c761e2303db4c10df7af2f997f of the GitHub Eddystone master branch.

Besides the validators, there are several Android/iOS applications that can be used to test Regular Advertising Mode functionalities of Eddystone-compatible beacons. For example, using Estimote Android App, we can see how Eddystone-compatible advertising packets are parsed and used by applications as shown in Figure 3. These smart phone applications are not developed or supported by Texas Instruments.

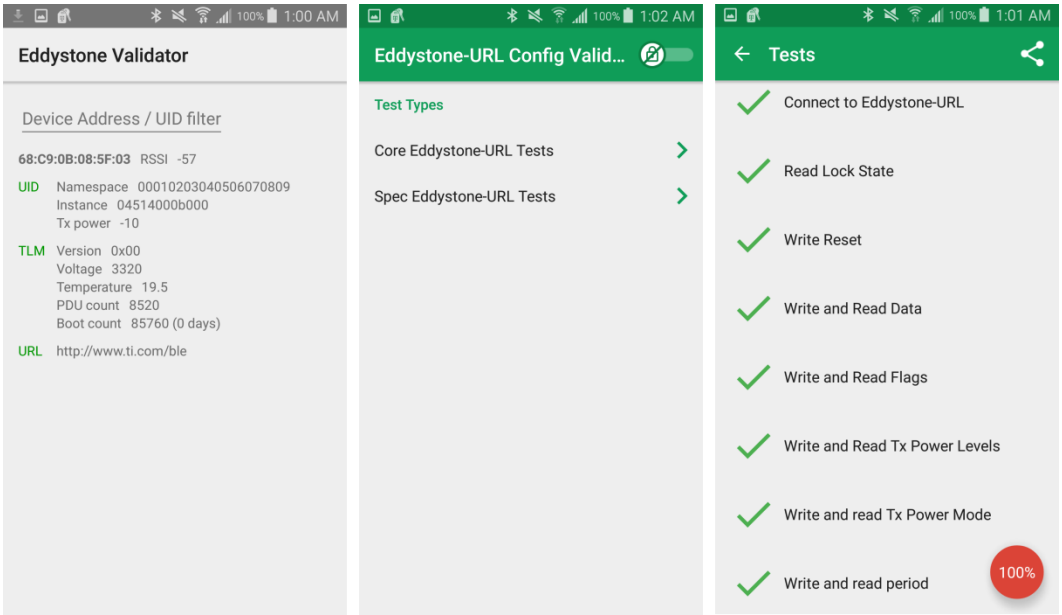

Figure 2. Eddystone Validator and Eddystone-URL Config Validator

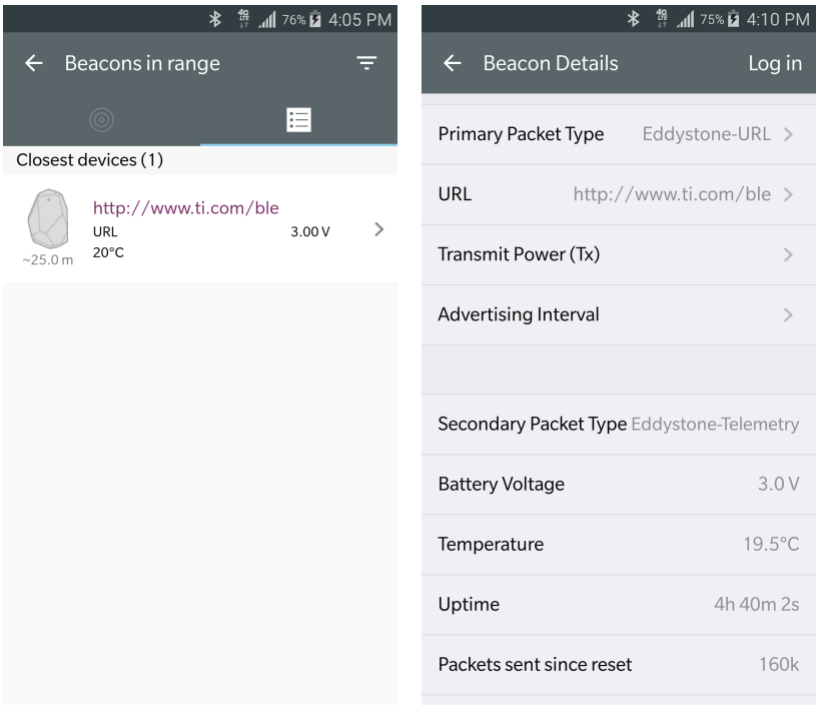

Figure 3. Estimote Android App

7 Summary

This application note describes the basic concept of Eddystone Beacon, how connectable beacon applications can be implemented on top of the existing sample application SimpleBLEPeripheral, and how the compatibility to Eddystone specification can be tested. Also, events, callbacks, and usages of hardware drivers are introduced for obtaining information to populate the TLM frame.

With a simple feature addition and modification on the sample application, connectable beacon devices such as a Eddystone-compatible beacon device can be easily developed.

8 References

1. Bluetooth low energy Software Developer's Guide (SWRU393)
2. CC2640 OAD User's Guide
3. Bluetooth low energy Beacons (SWRA475)
4. Google Eddystone Github

Revision History

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

Changes from Original (September 2015) to A Revision	Page
• Updated TI BLE-Stack v2.1 SDK to TI BLE-Stack v2.2 SDK.	2
• Updated Project Overview and Prerequisites section.	3
• Updated ARM 7.40.2 to ARM 7.50.3	3
• Updated Populating the Eddystone Frame Payload section.	5

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