User's Guide
SLUU473–January 2011

bq78412 Application Programming Interface

This manual describes the operation of the Application Programming Interface (API) for the bq78412. The API is intended to allow users to automate their production flow. Essential tasks such as configuration and calibration of each battery pack can be accomplished using the API and the appropriate software and hardware interfaces.

Contents

1 API User Guide

1.1 How to Use This Manual

This manual is intended for the experienced software programmer. Code examples are written in C# unless otherwise noted.

1.2 Introduction

The following API is designed to support the production flow of bq78412-based hardware. This API is built on the .NET 2.0 Framework and contains a COM visible interface.

1.3 Preparation for Production Flow

The bq78412 Evaluation GUI software is used in the laboratory to develop a production prototype. It provides the necessary software tools to create a .dat file. The .dat file is what is used to clone the production exemplar created in the laboratory. When this file is created by the bq78412 Evaluation GUI, it is in a coded format that it is not easily read. A .dat file that is specific to the battery pack being produced must be created in order to complete the production process using the API.

2 API Overview

A listing of the methods, properties, and events included in the API follows:

2.1 Classes

Class: bq78412

Methods

- bq78412
- Connect
- DelegateForFuncPtr
- Disconnect
- GetPortNames
- LoadChemistry
- LoadPackConfiguration
- SaveChemistry
- SavePackConfiguration
- UartRead
- UartSendCommand
- UartWriteWord
- WritePendingChanges

Properties

- BaudRate
- FirmwareVersion
- Parameters
- Target

Events

- OnComm
- OnProgress
- OnProgressComplete
- OnProgressError
- OnProgressStart

Class: Parameter collection

Methods

- ClearAllPendingValues
- ClearAllValues
- FindByAddress
- FindByName
- Parameter collection

Properties

- FlashLength
- RamLength

Class: Parameter

Methods

- Parameter
- ToString

Properties

- AddressOffset
- BitField
- DataType
- Description
- Format
- IsBitField
- IsChem
- L1
- L2
- L3
- Length
- MaxValue
- MemoryType
- MinValue
- Name
- PendingValue
- ScaleFactor
- ScaleOffset
- Units
- Value

Events

- OnPendingValueError

Class: BitFieldInfo

Methods

- BitFieldInfo

Properties

- BitNames
- Value

2.2 Enumerations

Commands

- Control
- DeviceType
- Read
- SetSealedLevel0
- SetSealedLevel1From2
- Write

Status

- ComPortClosed
- ComPortError
- ComPortInUse
- FileParseError
- FileReadError
- InvalidAddress
- InvalidChecksum
- InvalidData
- InvalidFile
- InvalidFileChecksum
- InvalidFileType
- InvalidFileValue
- InvalidFileVersion
- InvalidLength
- Nack
- NoResponse
- NoTargetFound
- Ok
- ReadError
- ReadTimeout
- RespondMismatch
- WriteError
- WriteTimeout

SubCommand

- SetSealedLevel1From0
- SetSealedLevel2
- InitializeSoc
- ResetCumulativeData
- ResetDevice

Parameter.ParamDataType

- Byte
- Double
- Float
- Int
- Long
- Sbyte
- Short
- String
- Sword
- Uint
- Ulong
- Ushort

2.3 Minimum Requirements

- Microsoft™ Windows™ operating system – XP, Vista, 7
- .NET Framework 2.0

3 API Interface Definitions

3.1 Methods of TI.bq784XX.bq78412:

bq78412
bq78412()

Summary:
Instantiates the API object

Connect
Status Connect(string *portName*, byte *address*)

Summary:
Establishes a link between the API and the bq78412

Parameters:
portName: Name of COM port
address: Address of the device

Returns:
Status

DelegateForFuncPtr
System.Delegate DelegateForFuncPtr(long *ptr*, System.Type *t*)

Summary:
Converts an unmanaged function pointer to a delegate. This method is commonly used for COM interoperability. For more information see
http://msdn.microsoft.com/enus/library/system.runtime.interopservices.marshal.getdelegateforfunctionpointer.aspx

Parameters:
ptr: Function pointer
t: Type

Returns:
Delegate

Disconnect
void Disconnect()

Summary:
Disconnects the link between the API and bq78412 and releases all used resources

GetPortNames
string[] GetPortNames()

Summary:
Gets a list of all available COM ports

Returns:
Array of port names. Ex. COM1

LoadChemistry
Status LoadChemistry(byte *address*, string *filename*)

Summary:
Loads a chemistry file. Validation is automatically performed.

Parameters:
address: Address of the device
filename: Path to file

Returns:
Status

LoadPackConfiguration
Status LoadPackConfiguration(byte *address*, string *filename*)

Summary:
Loads a pack configuration file (also know as golden flash file) This file is used in production for cloning packs. Validation is automatically performed.

Parameters:
address: Address of the device
filename: Path to file

Returns:
Status

SaveChemistry
Status SaveChemistry(byte *address*, string *filename*)

Summary:
Saves a chemistry file from the bq78412

Parameters:
address: Address of the device
filename: Path to file

Returns:
Status

SavePackConfiguration
Status SavePackConfiguration(byte *address*, string *filename*)

Summary:
Saves a pack configuration file (also known as golden flash file) This file is used in production for cloning packs

Parameters:
address: Address of the device
filename: Path to file

Returns:
Status

UartRead
Status UartRead(byte *address*, int *register*, int *length*)
Status UartRead(byte *address*, int *register*, int *length*, out int[] *data*)

Summary:
Reads a Word of data from the bq78412; this method automatically updates the value of the corresponding parameter which can then be read by accessing the Parameters property.

Parameters:
address: Address of device
register: Register to read from
length: Number of integers to read
data: Array of integers

Returns:
Status

UartSendCommand
Status UartSendCommand(byte *address*, byte *command*, byte *param3*, byte *param2*, byte *param1*, byte *param0*)
Status UartSendCommand(byte *address*, byte *command*, byte *param3*, byte *param2*, byte *param1*, byte *param0*, int *returnLen*, out byte[] *data*)

Summary:
Sends a command to the bq78412

Parameters:
address: Address of device
command: Command to perform
param3: Sub parameter
param2 Sub parameter
param1: Sub parameter
param0: Sub parameter
returnLen: Amount of data expected in return
data: Array of bytes

Returns:
Status

UartWriteWord
Status UartWriteWord(byte *address*, int *register*, string *data*)
Status UartWriteWord(byte *address*, int *register*, byte *msb*, byte *lsb*)
Status UartWriteWord(byte *address*, int *register*, int *value*)
Status UartWriteWord(byte *address*, int *register*, byte[] *value*)
Status UartWriteWord(byte *address*, int *register*, int[] *value*)

Summary:
Writes a Word of data to the bq78412

Parameters:
address: Address of device
register: Register to write to
**msb:* MSB of data to write
**lsb:* LSB of data to write
**data:* Data to write
**value:* Array of bytes to write
* *Overloaded parameter*

Returns:
Status

WritePendingChanges
Status UartWriteWord(byte *address*)

Summary:
Automates the writing of all parameters that have been edited. This method is preferred over UartWriteWord when writing to the device parameters. This method performs all of the necessary data conversions and validates data ranges.

Parameters:

address: Address of the bq78412

Returns:
Status

3.2 Properties of TI.bq784XX.bq78412:

BaudRate
int BaudRate { set; get; }

Summary:
Gets or sets the baud rate for UART communications

FirmwareVersion
string FirmwareVersion { get; }

Summary:
Gets the firmware version of the connected device

Parameters
Parameter collection Parameters { get; }

Summary:
Gets a collection of all parameters of the connected device

Target
string Target { get; }

Summary:
Gets the name of the connected device

3.3 Events of TI.bq784XX.bq78412:

OnComm
CommunicationsEventHandler OnComm

Summary:
Fires when communications is detected on the UART

OnProgress
ProgressEventHandler OnProgress

Summary:
Fires each time a change in status occurs during a process

OnProgressComplete
ProgressCompleteEventHandler OnProgressComplete

Summary:
Fires when a process completes

OnProgressError
ProgressErrorEventHandler OnProgressError

Summary:
Fires when an error occurs during a process

OnProgressStart
bq78412ProgressStartEventHandler OnProgressStart

Summary:
Fires when a long running process starts

3.4 Methods of TI.bq784XX.ParameterCollection:

ClearAllPendingValues
void ClearAllPendingValues()

Summary:

Clears all of the pending value properties of each parameter

ClearAllValues
void ClearAllValues()

Summary:
Clears all of the value properties of each parameter

FindByAddress
Parameter FindByAddress(int *Address*)

Summary:
Finds a parameter in the collection by its memory address

Parameters:
Address: Address of register

Returns:
Parameter

FindByName
Parameter FindByName(string *name*)

Summary:
Finds a parameter in the collection by its name

Parameters:
Name: Name of parameter, case insensitive

Returns:
Parameter

Parameter collection
Parameter collection()

Summary:
Instantiates the Parameter collection object

3.5 Properties of TI.bq784XX.ParameterCollection:

FlashLength
int FlashLength { get; }

Summary:
Gets the number of flash parameters

RamLength
int RamLength { get; }

Summary:
Gets the number of ram parameters

3.6 Methods of TI.bq784XX.Parameter:

Parameter
Parameter()

Summary:
Instantiates the Parameter object

ToString
string ToString()

Summary:
Converts the ram parameter data to a scaled and formatted value for display

Returns:
Formatted data

3.7 Properties of TI.bq784XX.Parameter:

AddressOffset
int AddressOffset { set; get; }

Summary:
Gets the parameters register address

BitField
BitFieldInfo BitField { get; }

Summary:
Gets the parameters bit field information

DataType
Parameter.ParamDataType DataType { get; }

Summary:
Gets the parameters data type

Description
string Description { get; }

Summary:
Gets the parameters description

Format
string Format { get; }

Summary:
Gets the parameters data format

IsBitField
bool IsBitField { get; }

Summary:
Determines if the parameter is bit encoded

IsChem
bool IsChem { get; }

Summary:
Determines if a parameter is associated with the packs chemistry

L1
int L1 { get; }

Summary:
Gets the parameters availability at seal level 1

L2
int **L2** { get; }

Summary:
Gets the parameters availability at seal level 2

L3
int **L3** { get; }

Summary:
Not used

Length
int Length { get; }

Summary:
Gets the parameters length in bytes

MaxValue
double MaxValue { get; }

Summary:
Gets the parameters maximum value

MemoryType
string MemoryType { get; }

Summary:
Gets the parameters memory type

MinValue
double MinValue { get; }

Summary:
Gets the parameters minimum value

Name
string Name { get; }

Summary:
Gets the parameters name

PendingValue
string PendingValue { set; get; }

Summary:
Gets or sets the parameters pending value. Edit this value when a change in the parameter actual value is desired

ScaleFactor
double ScaleFactor { get; }

Summary:
Gets the parameters scale factor

ScaleOffset
double ScaleOffset { get; }

Summary:
Gets the parameters scale offset

Units
string Units { get; }

Summary:
Gets the parameters units of measurement

Value
string Value { set; get; }

Summary:
Gets or sets the parameters currently displayed value

3.8 Events of TI.bq784XX.Parameter:

OnPendingValueError
Parameter.PendingValueErrorEventHandler OnPendingValueError

Summary:
Pending value error EventHandler

3.9 Methods of TI.bq784XX.BitFieldInfo:

BitFieldInfo
BitFieldInfo ()

Summary:
Instantiates the BitFieldInfo object

3.10 Properties of TI.bq784XX.BitFieldInfo:

BitNames
String[] BitNames { get; }

Summary:
Array of bit names of a parameter

Value
int Value { get; }

Summary:
Not currently used

3.11 Command and SubCommand Enumerations

Using the API enumerations is optional. Its purpose is to help you create cleaner code by associating a constant command code to a user-friendly name that is consistent with the bq78412 Pb-Acid Battery State-of-Charge Indicator With Run-Time Display data sheet. See the data sheet for command definitions.

Commands

```
public enum Commands : byte
{
    DeviceType              = 0x12,
    Control                 = 0x13,
    SetSealedLevel1From2    = 0x14,
    SetSealedLevel0         = 0x15,
    Read                    = 0x16,
    Write                   = 0x17
}
```

SubCommands

```
public enum SubCommands : byte
{
    ResetDevice             = 0x01,
    ResetCumulativeData     = 0x02,
    SetSealedLevel1From0    = 0x03,
    SetSealedLevel2         = 0x04,
    InitializeSoc           = 0x05,
}
```

3.12 Status Enumerations

Most bq78412 API methods return Status.

Ok	Communications successful
InvalidAddress	Register address not valid
InvalidLength	Not used
InvalidChecksum	The checksum received was bad
InvalidData	The data received was bad
InvalidFile	The file is XML but not the correct file
InvalidFileType	The file is not the correct type
InvalidFileVersion	The file version is not compatible
InvalidFileChecksum	The file does not contain the original data
InvalidFileValue	A file contains an illegal parameter value
FileReadError	An error occurred while loading a file
FileWriteError	An error occurred while saving a file
FileParseError	Not used
Nack	The request was not acknowledged
RespondMismatch	The responding address does not match
ReadTimeout	The device failed to respond on time
ReadError	Lost communications to bq78412

WriteTimeout	The API failed to write on time
WriteError	Lost communications to bq78412
ComPortError	The COM port has a problem
ComPortClosed	The COM port is closed
ComPortInUse	The COM port is not available
NoTargetFound	Could not communicate to the bq78412
NoResponse	Not used
DataValidationError	The value written does not match the value read

4 Parameters

The several ways to read and write parameters in the bq78412 appear in the following table, which displays the properties needed to access the parameters for the various methods. Note that some of the data sheet parameter names are abbreviated. Use the parameter names shown in the table when using the API.

Name	Address	Length	Access	L1	L2	Scale x
Status	0x0000	2	R	x	x	1
Temperature	0x0002	2	R	x	x	1
Voltage	0x0004	2	R	x	x	0.001
Current	0x0006	2	R	x	x	0.1
RemainingCapacity	0x0008	2	R	x	x	0.1
FullChargeCapacity	0x000A	2	R	x	x	0.1
RunTimeToEmpty	0x000C	2	R	x	x	1
CycleCount	0x000E	2	R	x	x	1
AverageCurrent	0x0010	2	R	x	x	0.1
DeratedFCC	0x0012	2	R	x	x	0.1
AccumulatedMissedCharge	0x0014	2	R	x	x	1
RelativeStateOfCharge	0x0016	2	R	x	x	1
OTCount	0x0018	2	R	x	x	6
UTCount	0x001A	2	R	x	x	6
OVCount	0x001C	2	R	x	x	6
UVCount	0x001E	2	R	x	x	6
OccCount	0x0020	2	R	x	x	6
OcdCount	0x0022	2	R	x	x	6
DOD80Count	0x0024	2	R	x	x	1
DOD60Count	0x0026	2	R	x	x	1
DOD30Count	0x0028	2	R	x	x	1
DOD10Count	0x002A	2	R	x	x	1
DOD0Count	0x002C	2	R	x	x	1
DischargeAHCount	0x002E	2	R	x	x	16
ChargeAHCount	0x0030	2	R	x	x	16
DischargeTime	0x0032	2	R	x	x	6
MultiDropAddress	0x4000	2	R/W	3	1	1
InstallDate	0x4002	2	R/W	3	1	1
ActivationDate	0x4004	2	R/W	3	1	1
ActivationIndicator	0x4006	2	R/W	3	1	1
MFGCodeSN	0x4008	8	R/W	3	1	1
VoltageGain	0x4010	2	R/W	1	1	3.05E-05
TemperatureOffset	0x4012	2	R/W	1	1	1
MeasurementScale	0x4014	2	R/W	1	1	0.000244

Name	Address	Length	Access	L1	L2	Scale x
CurrentOffset	0x4016	2	R/W	1	1	0.1
MeasurementConfiguration	0x4018	2	R/W	1	1	1
OTThreshold	0x401A	1	R/W	1	1	1
UTThreshold	0x401B	1	R/W	1	1	1
OVThreshold	0x401C	2	R/W	1	1	0.001
UVThreshold	0x401E	2	R/W	1	1	0.001
OCCThreshold	0x4020	1	R/W	1	1	10
OCDThreshold	0x4021	1	R/W	1	1	10
DODThreshold	0x4022	1	R/W	1	1	0.1
DesignCapacity	0x4024	2	R/W	1	1	0.1
ChemistryID	0x4026	1	R/W	1	1	1
NumberOfCells	0x4027	1	R/W	1	1	1
ChargeTaperTime	0x4028	2	R/W	1	1	1
ChargeTime	0x402A	2	R/W	1	1	1
EndOfDischargeVoltage	0x402C	2	R/W	1	1	0.001
CapacityDerateL	0x402E	1	R/W	0	0	1
CapacityDerateH	0x402F	1	R/W	0	0	1
DerateChange	0x4030	2	R/W	0	0	1
EndOfLifeCapacity	0x4032	2	R/W	0	0	0.1
LifeCycles	0x4034	2	R/W	0	0	1
EndOfLifeCapacityWarning	0x4036	2	R/W	0	0	0.1
LifeCycleWarning	0x4038	2	R/W	0	0	1
DeviceConfiguration1	0x403A	2	R/W	0	0	1
DeviceConfiguration2	0x403C	2	R/W	0	0	1
DisplayConfiguration1	0x403E	2	R/W	0	0	1
DisplayConfiguration2	0x4040	2	R/W	0	0	1
DisplayConfiguration3	0x4042	2	R/W	0	0	1
DisplayConfiguration4	0x4044	2	R/W	0	0	1
DisplayConfiguration5	0x4046	2	R/W	0	0	1
MissedChargeLimit	0x4048	1	R/W	0	0	1
ChargeEfficiency	0x4049	1	R/W	0	0	1
P-Scale	0x404A	2	R/W	0	0	1
CurrentAverageTime	0x404C	2	R/W	0	0	1
IdleThreshold	0x404E	2	R/W	0	0	0.1
TransToActive	0x4050	2	R/W	0	0	0.1
SleepTime	0x4052	2	R/W	0	0	1
Level0Password	0x4054	4	R/W	0	0	1
Level1Password	0x4058	4	R/W	3	0	1

4.1 Parameter Read/Write Level Restriction

x = Read Only
L1 = Sealed at level 1
L2 = Sealed at level 2

Bit Encoded Value	Write	Read
3 = Read and Write	1	1
1 = Read Only	0	1
0 = No Access	0	0

5 Code Examples

The following code examples are snippets demonstrating some of the API's functionality. Snippets are not necessarily complete blocks of code but show the user how the methods can be used in their application in the simplest form.

5.1 Instantiate the API Object

```
bq78412 API = new bq78412();
```

5.2 Connecting to the bq78412

```
// Example: Searching all available COM ports
// for a bq78412
private bool Find_bq78412()
{
      // Get a list of all available COM ports
      string[] comports = API.GetPortNames();
      // Loop through each port searching for a bq78412
      for each (string s in comports)
      {
         try
         {
               // Call the connect method passing in the COM port
               // name and the address of device.
               if (API.Connect(s, batteryAddress) == Status.Ok)
               {
                   // Found target!
                   return true;
               }
          }
          catch
          {
               // An error has occurred, report error
               return false;
           }
      }

       // No target found!
       return false;
}
```

5.3 Disconnecting From the bq78412

```
API.Disconnect();
```

5.4 Reading Data

```
const int RAM_START        = 0:        // Start of Ram space
const int FLASH START      = 0x4000;   // Start of Flash space
// Read all Ram variables using method UartRead
// This method will read all Ram variables and populate all
// of the associated parameters which can then be read by name.
Status status = API.UartRead(BatteryAddress, RAM_START, API.Parameters.RamLength);

// Read all Flash variables using method UartRead
// This method will read all Flash variables and populate all of the
// associated parameters which can then be read by name.
// Note: This exact method call will only work if the device is unsealed
// When the device sealed, all Flash parameters are not readable. Please see
// datasheet for information covering parameters and the associated seal
// level.
Status status = API.UartRead(BatteryAddress, FLASH_START, API.Parameters.FlashLength);

// Read two bytes of data from register 0x0008 RemainingCapacity.
// The integer result is returned in the Value variable.
// You must apply the scale factor to achieve the correct result
// Result = Value * ScaleFactor
int[] Value;
Status status = API.UartRead(BatteryAddress, 0x0008, 2, out Value);

// Another way to perform the same task is to read the register
```

```
// and then read the Parameters collection to get the value.
Status status = API.UartRead(BatteryAddress, 0x0008, 2);
string Value = API.Parameters["RemainingCapacity"].Value;

// Or return the entire Parameter object
Parameter parameter = API.Parameters["RemainingCapacity"]
string Value = parameter.Value;
```

5.5 Writing Data

```
// Write a new value to DesignCapacity using UartWriteWord.
// When writing values using UartWriteWord, you must include
// the scale factor.
// Result = Value / ScaleFactor
// 1000 = 100Ah / 0.1
Status status = API.UartWriteWord(BatteryAddress, 0x4024, 2, 1000);

// An easier approach is to set the pending value of the Parameter object
// and then call the method WritePendingChanges. You do not have to apply
// scaling to the result, all of the calculations are done by the API
// You can set as many parameters as you wish and just call
// WritePendingChanges one time.
API.Parameters["DesignCapacity"].PendingValue = "100";
Status status = API.WritePendingChanges(BatteryAddress);
```

5.6 Sending Commands

Sending commands is explained in detail in the product data sheet under section **Command Set and Status Reporting**. Refer to this document for all available commands.

```
// Example: Setting the seal level to 0
// password in this case is a 32 bit uint.
byte b3 = Convert.ToByte((password & 0xFF000000) >> 24);
byte b2 = Convert.ToByte((password & 0x00FF0000) >> 16);
byte b1 = Convert.ToByte((password & 0x0000FF00) >> 8);
byte b0 = Convert.ToByte((password & 0x000000FF) >> 0);
Status status = API.UartSendCommand(batteryAddress,
(byte)Commands.SetSealedLevel0, b3, b2, b1, b0);

// Example: Resetting the device
Status status = API.UartSendCommand(batteryAddress, (byte)Commands.Control,
(byte)SubCommands.ResetDevice, 0, 0, 0);
```

5.7 Loading Files

Loading the .dat file is the main purpose for this API. The .dat file is created using the bq78412 Evaluation GUI and contains all of the necessary data to clone a pack for production. When loading files, data validation is automatically performed by the API.

```
// Loading a .dat file where FileName is the directory path to the file.
Status status = API.LoadPackConfiguration(batteryAddress, FileName);
```

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information 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.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Audio	www.ti.com/audio	Communications and Telecom	www.ti.com/communications
Amplifiers	amplifier.ti.com	Computers and Peripherals	www.ti.com/computers
Data Converters	dataconverter.ti.com	Consumer Electronics	www.ti.com/consumer-apps
DLP® Products	www.dlp.com	Energy and Lighting	www.ti.com/energy
DSP	dsp.ti.com	Industrial	www.ti.com/industrial
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Security	www.ti.com/security
Logic	logic.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Power Mgmt	power.ti.com	Transportation and Automotive	www.ti.com/automotive
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com	Wireless	www.ti.com/wireless-apps
RF/IF and ZigBee® Solutions	www.ti.com/lprf		
	TI E2E Community Home Page		e2e.ti.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265