C2000 AES SHA Library

USER'S GUIDE

Copyright

Texas Instruments
13905 University Boulevard
Sugar Land, TX 77479
http://www.ti.com/c2000

Revision Information

This is version v2.00.00.00 of this document, last updated on September 14, 2021.

Table of Contents

1 Introduction

The C2000 Crypto Library is a collection of publicly available software AES and SHA algorithms. AES APIs offer encryption and decryption options of common AES modes such as ECB, CMAC, CBC and CTR. The SHA APIs contains SHA256 software algorithms. APIs are offered for implementations with the inputs gotten byte-wise and word-wise.

Note: Include the predefined symbol MEM_PACKED_SPEED_OPT to use the version of the speed optimized implementation of AES which accepts 16-bit packed input plaintext, key and gives out a 16-bit packet ciphertext. Please note that when the predefined symbol MEM_PACKED_SPEED_OPT is included, only the speed optimized implementation of AES is available for usage i.e., the space optimized implementation of AES is disabled and not available for usage.

This user guide provides details on the AES, SHA API. Performance and Memory details of implemented API is also described in the userguide.

Minimum Requirements:
Code Composer Studio v10, C2000 Compiler v21.6.0.LTS, ARM Compiler v21.6.0.LTS

Chapter Overview:

Chapter 2 - **AES APIs**
Describes the AES APIs
Chapter 3 - **SHA APIs**
Describes the SHA APIs
Chapter 4 - **Information on examples**
Details use of the different crypto APIs
Chapter 5 - **Performance and Memory Details**
Describes the Performance and Memory details of the supported crypto APIs
Chapter 6 - **Revision History**
The revision history of the software

Term	Definition
CCS	Code Composer Studio
AES	Advanced Encryption Standard
ECB	Electronic Codebook
CMAC	Cipher-based Authentication Code
CBC	Cipher Block Chaining
CTR	Counter
API	Application Programming Interface
SHA	Secure Hashing Algorithm

Table 1.1: Terms and Abbreviations

2 AES APIs

This chapter details the AES API.

2.1 AES ECB API

2.2 AES CMAC API

Functions

uint16_t * AES_performCMAC (uint16_t *text, uint16_t length, uint16_t *key, AES_KeySize keysize)

uint16_t * AES_performCMACFast (uint16_t *text, uint16_t length, uint16_t *key, AES_KeySize keysize)

2.2.1 Detailed Description

2.2.2 Function Documentation

2.2.2.1

uint16_t * text,

uint16_t length,

uint16_t * key,

AES_KeySize keysize) Perform AES CMAC to generate Message Authentication Code (MAC)

Parameters *text* is the input plaintext or ciphertext and once operation is complete, this parameter contains the output plaintext or ciphertext

---

length is the length of the input data in blocks (block size is 128-bits)

---

key is the AES key

---

keysize is the size of the AES key

This function performs AES Cipher-based Message Authentication (CMAC) on the requested number of 128-bit input plain text blocks. The calculated Message Authentication Code(MAC) is returned.

Note: Size of the plain text must be a multiple of 128-bits

Prototype:

```
AES_performCMAC uint16_t *AES_performCMAC(
```

Returns MAC Starting address of a 128 element array containing the MAC(128-bits)

2.2.2.2 uint16_t* AES_performCMACFast (

uint16_t * text,

uint16_t length,

uint16_t * key,

AES_KeySize keysize)

Perform AES fast CMAC to generate Message Authentication Code (MAC)

Parameters *text* is the input plaintext or ciphertext and once operation is complete, this parameter contains the output plaintext or ciphertext

---

length is the length of the input data in blocks (block size is 128-bits)

---

key is the AES key

---

keysize is the size of the AES key

This function performs AES fast Cipher-based Message Authentication (CMAC) on the requested number of 128-bit input plain text blocks. The calculated Message Authentication Code(MAC) is returned.

Note: Size of the plain text must be a multiple of 128-bits

Returns MAC Starting address of a 128 element array containing the MAC(128-bits)

2.3 AES CBC API

Functions

void AES_performCBC (uint16_t *text, uint16_t length, uint16_t *key, AES_KeySize keysize, AES_OperationMode mode, uint16_t *iv)

void AES_performCBCFast (uint16_t *text, uint16_t length, uint16_t *key, AES_KeySize keysize, AES_OperationMode mode, uint16_t *iv)

2.3.1 Detailed Description

2.3.2 Function Documentation

2.3.2.1

uint16_t * text,

uint16_t length,

uint16_t * key,

AES_KeySize keysize,

AES_OperationMode mode,

uint16_t * iv) Perform AES CBC Encryption/Decryption

Parameters *text* is the input plaintext or ciphertext and once operation is complete, this parameter contains the output plaintext or ciphertext

---

length is the length of the input data in blocks (block size is 128-bits)

---

key is the AES key

---

keysize is the size of the AES key

---

mode is the AES operation mode of encryption or decryption

---

iv is the initialization vector of size 128 bits

This function performs AES Cipher Block Chaining (CBC) mode encryption or decryption on the requested number of 128-bit input blocks.

Prototype:

```
AES_performCBC void AES_performCBC(
```

Returns None.

2.3.2.2 void AES_performCBCFast (

uint16_t * text,

uint16_t length,

uint16_t * key,

AES_KeySize keysize,

AES_OperationMode mode,

uint16_t * iv)

Perform AES CBC fast Encryption/Decryption

Parameters *text* is the input plaintext or ciphertext and once operation is complete, this parameter contains the output plaintext or ciphertext

---

length is the length of the input data in blocks (block size is 128-bits)

---

key is the AES key

keysize is the size of the AES key

mode is the AES operation mode of encryption or decryption

iv is the initialization vector of size 128 bits

This function performs AES Cipher Block Chaining (CBC) mode fast encryption or decryption on the requested number of 128-bit input blocks.

Returns None.

2.4 AES CTR API

Functions

void AES_performCTR (uint16_t *text, uint16_t length, uint16_t *key, AES_KeySize keysize, AES_OperationMode mode, uint16_t *nonceCounter)

void AES_performCTRFast (uint16_t *text, uint16_t length, uint16_t *key, AES_KeySize keysize, AES_OperationMode mode, uint16_t *nonceCounter)

2.4.1 Detailed Description

2.4.2 Function Documentation

2.4.2.1

uint16_t * text,

uint16_t length,

uint16_t * key,

AES_KeySize keysize,

AES_OperationMode mode,

uint16_t * nonceCounter) Perform AES CTR Encryption/Decryption

Parameters *text* is the input plaintext or ciphertext and once operation is complete, this parameter contains the output plaintext or ciphertext

length is the length of the input data in blocks (block size is 128-bits)

key is the AES key

keysize is the size of the AES key

mode is the AES operation mode of encryption or decryption

nonceCounter is the required number-used-once 128-bit counter value

This function performs AES Counter (CTR) mode encryption or decryption on the requested number of 128-bit input blocks.

Prototype:

```
AES_performCTR void AES_performCTR(
```

Returns None.

2.4.2.2 void AES_performCTRFast (

uint16_t * text,

uint16_t length,

uint16_t * key,

AES_KeySize keysize,

AES_OperationMode mode,

uint16_t * nonceCounter)

Perform AES CTR fast Encryption/Decryption

Parameters *text* is the input plaintext or ciphertext and once operation is complete, this parameter contains the output plaintext or ciphertext

length is the length of the input data in blocks (block size is 128-bits)

key is the AES key

keysize is the size of the AES key

mode is the AES operation mode of encryption or decryption

nonceCounter is the required number-used-once 128-bit counter value

This function performs AES Counter (CTR) mode fast encryption or decryption on the requested number of 128-bit input blocks.

Returns None.

3 SHA256 APIs

This chapter details the SHA256 API.

3.1 SHA256 API

Macros

#define SHA256_STATUS_LENGTH_TOO_LARGE

#define SHA256_STATUS_NOT_ALIGNED

#define SHA256_STATUS_NULL_INPUT

#define SHA256_STATUS_SUCCESS

Functions

uint16_t SHA256_addDataByteWise (SHA256_HandleByteWise handle, uint16_t *data, size_t length)

uint16_t SHA256_addDataWordWise (SHA256_HandleWordWise handle, uint32_t *data, size_t length)

uint16_t SHA256_addDataWordWiseSwapped (SHA256_HandleWordWise handle, uint32_t *data, size_t length)

void SHA256_cancelByteWise (SHA256_HandleByteWise handle)

void SHA256_cancelWordWise (SHA256_HandleWordWise handle)

uint16_t SHA256_finalizeByteWise (SHA256_HandleByteWise handle, uint32_t digest[8])

uint16_t SHA256_finalizeWordWise (SHA256_HandleWordWise handle, uint32_t digest[8])

uint16_t SHA256_hashByteWise (SHA256_HandleByteWise handle, uint16_t *data, size_t length, uint32_t digest[8])

uint16_t SHA256_hashWordWise (SHA256_HandleWordWise handle, uint32_t *data, size_t length, uint32_t digest[8])

uint16_t SHA256_hashWordWiseSwapped (SHA256_HandleWordWise handle, uint32_t *data, size_t length, uint32_t digest[8])

void SHA256_processBlockByteWise (uint32_t digest[8], uint16_t Ws[64])

void SHA256_processBlockWordWise (uint32_t digest[8], uint32_t Ws[16])

uint16_t SHA256_startByteWise (SHA256_HandleByteWise handle)

uint16_t SHA256_startWordWise (SHA256_HandleWordWise handle)

3.1.1 Detailed Description

3.1.2 Macro Definition Documentation

3.1.2.1

SHA256- length of input too large

3.1.2.2

SHA256- input not aligned

3.1.2.3

SHA256- NULL input error

3.1.2.4

SHA256 operation successfull

3.1.3 Function Documentation

3.1.3.1

SHA256_HandleByteWise handle,

uint16_t * data,

size_t length) Add data to SHA256 operation when inputs are supplied byte-wise

Adds data to a hash operation. The `handle` must have been initialized by a call to SHA256_startByteWise first.

The total length of data that can be hashed by this implementation is 512MiB (0x20000000 bytes.).

After passing in all data to be hashed, call SHA256_finalizeWordWise() to obtain the final digest.

Parameters *handle* A 'SHA256_HandleByteWise' handle.

---

data 16-bit pointer pointing to the memory location of the starting byte the data to be hashed.

---

length Length of the data (in number of bytes) to be hashed

Returns SHA256_STATUS_SUCCESS hash operation succeeded. SHA256_STATUS_LENGTH_TOO_LARGE - requested length of data to hash is more than supported length. SHA256_STATUS_NULL_INPUT- One or more of the pointer inputs is NULL

3.1.3.2 uint16_t SHA256_addDataWordWise (

SHA256_HandleWordWise handle,

uint32_t * data,

size_t length)

Add data to SHA256 operation when inputs are supplied as 32-bit words

Adds data to a hash operation. The `handle` must have been initialized by a call to SHA256_startWordWise first.

The total length of data that can be hashed by this implementation is 512MiB (0x20000000 bytes.).

After passing in all data to be hashed, call SHA256_finalizeWordWise() to obtain the final digest.

Prototype:

```
SHA256_addDataByteWise uint16_t SHA256_addDataByteWise(
```

Parameters *handle* A 'SHA256_HandleWordWise' handle.

---

data 32-bit pointer pointing to the starting memory location of the data to be hashed.

---

length Length of the data (in number of bytes) to be hashed Note that for word-wise input, length of the data should be a multiple of 4.

Returns SHA256_STATUS_SUCCESS The hash operation succeeded.

SHA256_STATUS_LENGTH_TOO_LARGE The requested length of data to hash is more than the implementation supports.

SHA256_STATUS_NULL_INPUT One or more of the pointer inputs is NULL.

SHA256_STATUS_NOT_ALIGNED Length is not a multiple of 4

3.1.3.3 uint16_t SHA256_addDataWordWiseSwapped (

SHA256_HandleWordWise handle,

uint32_t * data,

size_t length)

Add data to SHA256 operation when inputs are supplied as 32-bit words

Input words are Swapped High-Low

Adds data to a hash operation. The `handle` must have been initialized by a call to SHA256_startWordWise first.

The total length of data that can be hashed by this implementation is 512MiB (0x20000000 bytes.).

After passing in all data to be hashed, call SHA256_finalizeWordWise() to obtain the final digest.

Parameters *handle* A 'SHA256_HandleWordWise' handle.

---

data 32-bit pointer pointing to the starting memory location of the data to be hashed.

---

length Length of the data (in number of bytes) to be hashed Note that for word-wise input, length of the data should be a multiple of 4.

Returns SHA256_STATUS_SUCCESS The hash operation succeeded.

SHA256_STATUS_LENGTH_TOO_LARGE The requested length of data to hash is more than the implementation supports.

SHA256_STATUS_NULL_INPUT One or more of the pointer inputs is NULL.

SHA256_STATUS_NOT_ALIGNED Length is not a multiple of 4

3.1.3.4 void SHA256_cancelByteWise (

SHA256_HandleByteWise handle)

Cancels SHA256 operation by clearing intermediate data stored in the 'SHA256_HandleByteWise' handle.

The handle will not be ready for a new operation until after SHA256_startByteWise() is called on the handle.

Parameters *handle* A 'SHA256_HandleByteWise' handle

3.1.3.5 void SHA256_cancelWordWise (

SHA256_HandleWordWise handle)

Cancels SHA256 operation by clearing intermediate data stored in the 'SHA256_HandleWordWise' handle.

The handle will not be ready for a new operation until after SHA256_startWordWise() is called on the handle.

Parameters *handle* A 'SHA256_HandleWordWise' handle

3.1.3.6 uint16_t SHA256_finalizeByteWise (

SHA256_HandleByteWise handle,

uint32_t digest[8])

Finalize the SHA256 operation when input is supplied byte-wise, creating the final digest.

After calling this function, `handle` should not be used again until it has been reinitialized via a call to SHA256_startByteWise().

Parameters *handle* A 'SHA256_HandleByteWise' handle

digest Output location for the final digest

Returns SHA256_STATUS_SUCCESS The hash operation succeeded.

SHA256_STATUS_NULL_INPUT One or more of the pointer inputs is NULL.

3.1.3.7 uint16_t SHA256_finalizeWordWise (

SHA256_HandleWordWise handle,

uint32_t digest[8])

Finalize the SHA256 operation when input is supplied as 32-bit words, creating the final digest.

After calling this function, `handle` should not be used again until it has been reinitialized via a call to SHA256_startWordWise().

Parameters *handle* A 'SHA256_HandleWordWise' handle

digest Output location for the final digest

digest Output location for the final digest

Returns SHA256_STATUS_SUCCESS The hash operation succeeded.

SHA256_STATUS_NULL_INPUT One or more of the pointer inputs is NULL.

3.1.3.8 uint16_t SHA256_hashByteWise (

SHA256_HandleByteWise handle,

uint16_t * data,

size_t length,

uint32_t digest[8])

Perform a complete hash operation when input is supplied byte-wise, producing a final digest for the data.

This function wraps SHA256_startByteWise(), SHA256_addDataByteWise(), and SHA256_finalizeByteWise().

There is no need to call SHA256_startByteWise() prior to calling this function.

The total length of data that can be hashed by this implementation is 512MiB (0x20000000 bytes.)

Parameters *handle* A 'SHA256_HandleByteWise' handle

data 16-bit pointer pointing to the memory location of the starting byte the data to be hashed.

length Length of the data (in number of bytes) to be hashed.

digest Output location for the final digest

Returns SHA256_STATUS_SUCCESS The hash operation succeeded.

SHA256_STATUS_LENGTH_TOO_LARGE The requested length of data to hash is more than the implementation supports.

SHA256_STATUS_NULL_INPUT One or more of the pointer inputs is NULL.

3.1.3.9 uint16_t SHA256_hashWordWise (

SHA256_HandleWordWise handle,

uint32_t * data,

size_t length,

uint32_t digest[8])

Perform a complete hash operation when input is supplied as 32-bit words, producing a final digest for the data.

This function wraps SHA256_startWordWise(), SHA256_addDataWordWise(), and SHA256_finalizeWordWise().

There is no need to call SHA256_startWordWise() prior to calling this function.

The total length of data that can be hashed by this implementation is 512MiB (0x20000000 bytes.)

Parameters *handle* A 'SHA256_HandleWordWise' handle

data 32-bit pointer pointing to the starting memory location of the data to be hashed

length Length of the data (in number of bytes) to be hashed. Note that for word-wise input, length of the data should be a multiple of 4.

digest Output location for the final digest

Returns SHA256_STATUS_SUCCESS The hash operation succeeded. SHA256_STATUS_LENGTH_TOO_LARGE - The requested length of data to hash is more than the implementation supports. SHA256_STATUS_NULL_INPUT - One or more of the pointer inputs is NULL. SHA256_STATUS_NOT_ALIGNED - Length. is not a multiple of 4

3.1.3.10 uint16_t SHA256_hashWordWiseSwapped (

SHA256_HandleWordWise handle,

uint32_t * data,

size_t length,

722.7

Parameters *digest* pre-loaded with previous block's digest (or initial digest if call is to process first block) On function return digest will hold the updated digest.

Ws loaded with data block, including padding as specified by the SHA256 standard.

3.1.3.13 uint16_t SHA256_startByteWise (SHA256_HandleByteWise handle)

Initialize a 'SHA256SW_HandleByteWise' handle, preparing for hashing data.

Parameters *handle* A 'SHA256_HandleByteWise' handle

Returns SHA256_STATUS_SUCCESS The hash operation succeeded.

SHA256_STATUS_NULL_INPUT One or more of the pointer inputs is NULL.

3.1.3.14 uint16_t SHA256_startWordWise (SHA256_HandleWordWise handle)

Initialize a 'SHA256SW_HandleWordWise' handle, preparing for hashing data.

Parameters *handle* A 'SHA256_HandleWordWise' handle

Returns SHA256_STATUS_SUCCESS The hash operation succeeded.

SHA256_STATUS_NULL_INPUT One or more of the pointer inputs is NULL.

4 examples

The crypto library provides folder named examples under path **libraries/security/crypto/c28**. This folder has examples on usage of the all the crypto API supported.

5 Performance and Memory Details

This chapter details the performance and memory details for the different crypto APIs supported.

The table below details how many CPU cycles the various modes require for AES encryption and decryption (space optimized implementation).

	Execution from RAM				Execution from Flash			
	AES-128		AES-256		AES-128		AES-256	
	Encrypt	Decrypt	Encrypt	Decrypt	Encrypt	Decrypt	Encrypt	Decryp
AES ECB	9068	12576	12674	17708	11465	14158	16125	20094
AES CBC	9311	12873	12915	18004	11701	14431	16353	20374
AES CTR	9494	9494	13098	13098	11868	11868	16461	16461
AES CMAC	18453		25530		22844		32039	
Expand Key	2755		4075		1857		3168	

Table 5.1: AES Space Optimized CPU Cycle Usage –Optimization Level 2 (Global Optimizations) used

The table below details how many CPU cycles the various modes require for AES encryption and decryption (speed optimized implementation).

	Execution from RAM				Execution from Flash			
	AES-128		AES-256		AES-128		AES-256	
	Encrypt	Decrypt	Encrypt	Decrypt	Encrypt	Decrypt	Encrypt	Decryp
AES ECB	2683	2978	3519	3854	3331	3636	4418	4768
AES CBC	2930	3279	3764	4184	3490	3846	4588	4971
AES CTR	3111	3111	3945	3945	3924	3924	4738	4738
AES CMAC	5733		7292		6568		8627	
Expand Key	1309	5281	1956	7558	1523	6265	2198	8936

Table 5.2: AES Speed Optimized CPU Cycle Usage –Optimization Level 2 (Global Optimizations) used

The table below details how many CPU cycles the various modes require for AES encryption and decryption when the predefined symbol MEM_PACKED_SPEED_OPT is included (speed optimized implementation) i.e., when the AES inputs are given as 16-bit packed words.

	Execution from RAM				Execution from Flash			
	AES-128		AES-256		AES-128		AES-256	
	Encrypt	Decrypt	Encrypt	Decrypt	Encrypt	Decrypt	Encrypt	Decryp
AES ECB	2444	2681	3299	3587	3357	3661	4546	4896
AES CBC	2544	2769	3398	3722	3468	3805	4656	5040
AES CTR	2544	2544	2979	2979	3505	3505	4694	4694
AES CMAC	2722		2979		3737		5095	
Expand Key	1285	4840	1908	6945	1635	6860	2278	9688

Table 5.3: AES Speed Optimized 16-bit packed input mode CPU Cycle Usage –Optimization Level 2 (Global Optimizations) used

The table below details how many CPU cycles the various implementations of SHA256(hashing) API.

	Execution from FLASH in MB/s	**Execution from RAM in MB/s**
sha256(Byte-wise implementation)	0.53	0.57
sha256(Word-wise implementation)	0.76	0.79

Table 5.4: SHA256 CPU Cycle Usage –Optimization Level 2 (Global Optimizations) used

This table details memory size of different AES modes supported(space optimized implementation).

File	**16 bit words in memory**
aes_common	1793
aes_ecb	79
aes_cbc	223
aes_ctr	273
aes_cmac	285

Table 5.5: Space optimized AES Memory Usage

This table details memory size of each AES API(speed optimized implementation).

File	**16 bit words in memory**
aes_common	7122
aes_ecb	79
aes_cbc	224
aes_ctr	274
aes_cmac	287

Table 5.6: Speed optimized AES API Memory Usage

This table details memory size of supported SHA256 implementation.

File	**16 bit words in memory**
sha256(Byte-wise implementation)	1793
sha256(Word-wise implementation)	751

Table 5.7: SHA256 Memory Usage

6 Revision History

v2.01.00.00: Third Release

- 16-bit packed plaintext and key input mode introduced using MEM_PACKED_SPEED_OPT predefined symbol
- Example provided for the newly introduced 16-bit packed input mode
- Performance and Memory Details updated for the newly introduced 16-bit packed input mode

v2.00.00.00: Second Release

- API names for AES functions changed (the API now take key size as input), fast implementations of AES API added
- SHA256 API support added
- examples provided for all supported crypto algorithms
- Performance and Memory Details updated for supported algorithms

v1.00.00.00: First Release

- AES API Functions ECB, CMAC, CBC, CTR
- Using the APIs
- Performance and Memory Details

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**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Broadband	www.ti.com/broadband
DSP	dsp.ti.com	Digital Control	www.ti.com/digitalcontrol
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Military	www.ti.com/military
Logic	logic.ti.com	Optical Networking	www.ti.com/opticalnetwork
Power Mgmt	power.ti.com	Security	www.ti.com/security
Microcontrollers	microcontroller.ti.com	Telephony	www.ti.com/telephony
RFID	www.ti-rfid.com	Video & Imaging	www.ti.com/video
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Wireless	www.ti.com/wireless

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