G.726/G.711 CST Algorithm User's Guide

SPIRIT CORP
DSP Software Source
www.spiritDSP.com/CST

Literature Number: SPRU637
March 2003

Texas Instruments

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 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.

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.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Preface

Read This First

About This Manual

The following abbreviations are used in this document:

ADPCM	adaptive differential pulse code modulation
PCM	pulse code modulation
XDAIS	TMS320 DSP Algorithm Standard

Related Documentation From Texas Instruments

Using the TMS320 DSP Algorithm Standard in a Static DSP System (SPRA577)

TMS320 DSP Algorithm Standard Rules and Guidelines (SPRU352)

TMS320 DSP Algorithm Standard API Reference (SPRU360)

Technical Overview of eXpressDSP-Compliant Algorithms for DSP Software Producers (SPRA579)

The TMS320 DSP Algorithm Standard (SPRA581)

Achieving Zero Overhead with the TMS320 DSP Algorithm Standard IALG Interface (SPRA716)

Related Documentation

ITU-T Recommendation G.726 (1990). 40, 32, 24, 16 kbit/s adaptive differential pulse code modulation (ADPCM).

Trademarks

TMS320™ is a trademark of Texas Instruments.

SPIRIT CORP™ is a tradmark of Spirit Corp.

All other trademarks are the property of their respective owners.

Software Copyright

If You Need Assistance . . .

❑ World-Wide Web Sites

TI Online	http://www.ti.com
Semiconductor Product Information Center (PIC)	http://www.ti.com/sc/docs/products/index.htm
DSP Solutions	http://www.ti.com/dsp
320 Hotline On-line™	http://www.ti.com/sc/docs/dsps/support.htm
Microcontroller Home Page	http://www.ti.com/sc/micro
Networking Home Page	http://www.ti.com/sc/docs/network/nbuhomex.htm
Military Memory Products Home Page	http://www.ti.com/sc/docs/military/product/memory/mem_1.htm

❑ North America, South America, Central America

Product Information Center (PIC)	(972) 644-5580		
TI Literature Response Center U.S.A.	(800) 477-8924		
Software Registration/Upgrades	(972) 293-5050	Fax: (972) 293-5967	
U.S.A. Factory Repair/Hardware Upgrades	(281) 274-2285		
U.S. Technical Training Organization	(972) 644-5580		
Microcontroller Hotline	(281) 274-2370	Fax: (281) 274-4203	Email: micro@ti.com
Microcontroller Modem BBS	(281) 274-3700 8-N-1		
DSP Hotline		Email: dsph@ti.com	
DSP Internet BBS via anonymous ftp to ftp://ftp.ti.com/pub/tms320bbs			
Networking Hotline		Fax: (281) 274-4027	
		Email: TLANHOT@micro.ti.com	

❑ Europe, Middle East, Africa

European Product Information Center (EPIC) Hotlines:		
Multi-Language Support	+33 1 30 70 11 69	Fax: +33 1 30 70 10 32
Email: epic@ti.com		
Deutsch	+49 8161 80 33 11 or +33 1 30 70 11 68	
English	+33 1 30 70 11 65	
Francais	+33 1 30 70 11 64	
Italiano	+33 1 30 70 11 67	
EPIC Modem BBS	+33 1 30 70 11 99	
European Factory Repair	+33 4 93 22 25 40	
Europe Customer Training Helpline		Fax: +49 81 61 80 40 10

❑ Asia-Pacific

Literature Response Center	+852 2 956 7288	Fax: +852 2 956 2200
Hong Kong DSP Hotline	+852 2 956 7268	Fax: +852 2 956 1002
Korea DSP Hotline	+82 2 551 2804	Fax: +82 2 551 2828
Korea DSP Modem BBS	+82 2 551 2914	
Singapore DSP Hotline		Fax: +65 390 7179
Taiwan DSP Hotline	+886 2 377 1450	Fax: +886 2 377 2718
Taiwan DSP Modem BBS	+886 2 376 2592	
Taiwan DSP Internet BBS via anonymous ftp to ftp://dsp.ee.tit.edu.tw/pub/TI/		

❑ Japan

Product Information Center	+0120-81-0026 (in Japan)	Fax: +0120-81-0036 (in Japan)
	+03-3457-0972 or (INTL) 813-3457-0972	Fax: +03-3457-1259 or (INTL) 813-3457-1259
DSP Hotline	+03-3769-8735 or (INTL) 813-3769-8735	Fax: +03-3457-7071 or (INTL) 813-3457-7071
DSP BBS via Nifty-Serve	Type “Go TIASP”	

❑ Documentation

When making suggestions or reporting errors in documentation, please include the following information that is on the title page: the full title of the book, the publication date, and the literature number.

Mail: Texas Instruments Incorporated
Technical Documentation Services, MS 702
P.O. Box 1443
Houston, Texas 77251-1443

Email: dsph@ti.com Email: micro@ti.com

Note: When calling a Literature Response Center to order documentation, please specify the literature number of the book.

For product price & availability questions, please contact your local Product Information Center, or see www.ti.com/sc/support http://www.ti.com/sc/support for details.

For additional CST technical support, see the TI CST Home Page (www.ti.com/telephonyclientside) or the TI Semiconductor KnowledgeBase Home Page (www.ti.com/sc/knowledgebase).

If you have any problems with the Client Side Telephony software, please, read first the list of Frequently Asked Questions at http://www.spiritDSP.com/CST.

You can also visit this web site to obtain the latest updates of CST software & documentation.

Contents

Figures

Tables

Notes, Cautions, and Warnings

Chapter 1

Introduction to G726 Encoder-Decoder (G726G711) Algorithms

This chapter provides the user with an introduction to the G.726 Encoder-Decoder algorithm.

For the benefit of users who are not familiar with the TMS320 DSP Algorithm Standard (XDAIS), brief descriptions of typical XDAIS terms are provided.

Topic **Page**

1.1 Introduction

This document describes the implementation of ITU-T Recommendation G.726 and G.711 encoder and decoder developed by SPIRIT Corp. for TMS320C54xx platform and is intended for integration into devices for encoding and decoding differential pulse coded signals. ITU-T G.726 implements adaptive differential pulse code modulation (ADPCM) encoding and decoding of voice frequencies.

The SPIRIT G726 software is a fully TMS320 DSP Algorithm Standard (XDAIS) compatible, reentrant code. The G726 interface complies with the TMS320 DSP Algorithm Standard and can be used in multitasking environments.

The TMS320 DSP Algorithm Standard (XDAIS) provides the user with object interface simulating object-oriented principles and asserts a set of programming rules intended to facilitate integration of objects into a framework.

The following documents provide further information regarding the TMS320 DSP Algorithm Standard (XDAIS):

- *Using the TMS320 DSP Algorithm Standard in a Static DSP System* (SPRA577)
- *TMS320 DSP Algorithm Standard Rules and Guidelines* (SPRU352)
- *TMS320 DSP Algorithm Standard API Reference* (SPRU360)
- *Technical Overview of eXpressDSP-Compliant Algorithms for DSP Software Producers* (SPRA579)
- *The TMS320 DSP Algorithm Standard* (SPRA581)
- *Achieving Zero Overhead with the TMS320 DSP Algorithm Standard IALG Interface* (SPRA716)

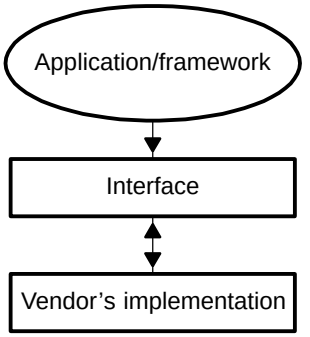
1.2 XDAIS Basics

This section instructs the user on how to develop applications/frameworks using the algorithms developed by vendors. It explains how to call modules through a fully eXpress DSP-compliant interface.

Figure 1-1 illustrates the three main layers required in an XDAIS system:

- Application/Framework layer
- Interface layer
- Vendor implementation. Refer to appendix A for a detailed illustration of the interface layer.

Figure 1-1. XDAIS System Layers



1.2.1 Application/Framework

Users should develop an application in accordance with their own design specifications. However, instance creation, deletion and memory management requires using a framework. It is recommended that the customer use the XDAIS framework provided by SPIRIT Corp. in ROM.

The framework in its most basic form is defined as a combination of a memory management service, input/output device drivers, and a scheduler. For a framework to support/handle XDAIS algorithms, it must provide the framework functions that XDAIS algorithm interfaces expect to be present. XDAIS framework functions, also known as the ALG Interface, are prefixed with "ALG_". Below is a list of framework functions that are required:

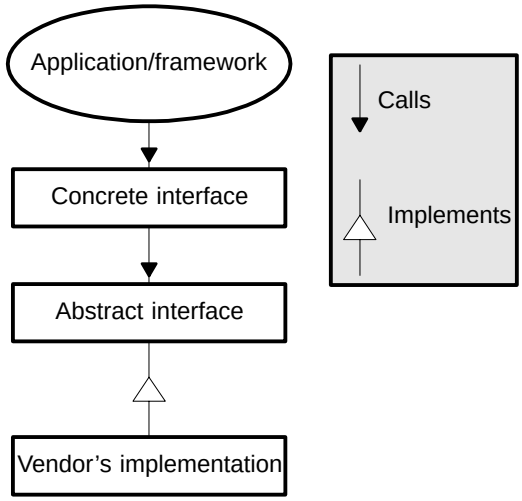
- `ALG_create` - for memory allocation/algorithm instance creation
- `ALG_delete` - for memory de-allocation/algorithm instance deletion
- `ALG_activate` - for algorithm instance activation

- ALG_deactivate - for algorithm instance de-activation
- ALG_init - for algorithm instance initialization
- ALG_exit - for algorithm instance exit operations
- ALG_control - for algorithm instance control operations

1.2.2 Interface

Figure 1-2 is a block diagram of the different XDAIS layers and how they interact with each other.

Figure 1-2. XDAIS Layers Interaction Diagram



1.2.2.1 Concrete Interface

A concrete interface is an interface between the algorithm module and the application/framework. This interface provides a generic (non-vendor specific) interface to the application. For example, the framework can call the function MODULE_apply() instead of MODULE_VENDOR_apply(). The following files make up this interface:

- Header file MODULE.h - Contains any required definitions/global variables for the interface.
- Source File MODULE.c - Contains the source code for the interface functions.

1.2.2.2 Abstract Interface

This interface, also known as the IALG Interface, defines the algorithm implementation. This interface is defined by the algorithm vendor but must comply with the XDAIS rules and guidelines. The following files make up this interface:

- Header file `iMODULE.h` - Contains table of implemented functions, also known as the IALG function table, and definition of the parameter structures and module objects.
- Source File `iMODULE.c` - Contains the default parameter structure for the algorithm.

1.2.2.3 Vendor Implementation

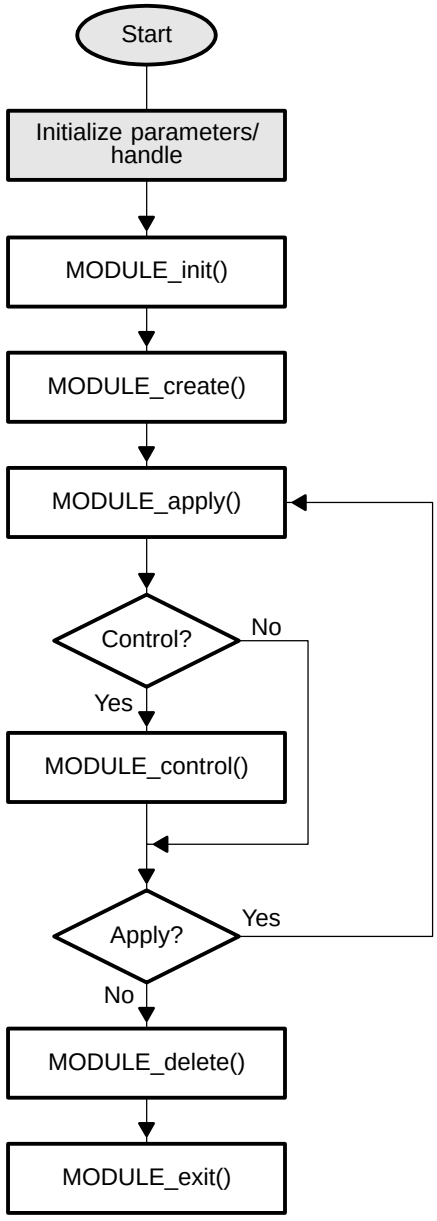
Vendor implementation refers to the set of functions implemented by the algorithm vendor to match the interface. These include the core processing functions required by the algorithm and some control-type functions required. A table is built with pointers to all of these functions, and this table is known as the function table. The function table allows the framework to invoke any of the algorithm functions through a single handle. The algorithm instance object definition is also done here. This instance object is a structure containing the function table (table of implemented functions) and pointers to instance buffers required by the algorithm.

1.2.3 Application Development

Figure 1-3 illustrates the steps used to develop an application. This flowchart illustrates the creation, use, and deletion of an algorithm. The handle to the instance object (and function table) is obtained through creation of an instance of the algorithm. It is a pointer to the instance object. Per XDAIS guidelines, software API allows direct access to the instance data buffers, but algorithms provided by SPIRIT prohibit access.

Detailed flow charts for each particular algorithm is provided by the vendor.

Figure 1-3. Module Instance Lifetime



The steps below describe the steps illustrated in Figure 1-3.

Step 1: Perform all non–XDAIS initializations and definitions. This may include creation of input and output data buffers by the framework, as well as device driver initialization.

Step 2: Define and initialize required parameters, status structures, and handle declarations.

Step 3: Invoke the `MODULE_init()` function to initialize the algorithm module. This function returns nothing. For most algorithms, this function does nothing.

Step 4: Invoke the `MODULE_create()` function, with the vendor's implementation ID for the algorithm, to create an instance of the algorithm. The `MODULE_create()` function returns a handle to the created instance. You may create as many instances as the framework can support.

Step 5: Invoke the `MODULE_apply()` function to process some data when the framework signals that processing is required. Using this function is not obligatory and vendor can supply the user with his own set of functions to obtain necessary processing.

Step 6: If required, the `MODULE_control()` function may be invoked to read or modify the algorithm status information. This function also is optional. Vendor can provide other methods for status reporting and control.

Step 7: When all processing is done, the `MODULE_delete()` function is invoked to delete the instance from the framework. All instance memory is freed up for the framework here.

Step 8: Invoke the `MODULE_exit()` function to remove the module from the framework. For most algorithms, this function does nothing.

The integration flow of specific algorithms can be quite different from the sequence described above due to several reasons:

- Specific algorithms can work with data frames of various lengths and formats. Applications can require more robust and effective methods for error handling and reporting.

- Instead of using the `MODULE_apply()` function, SPIRIT Corp. algorithms use extended interface for data processing, thereby encapsulating data buffering within XDAIS object. This provides the user with a more reliable method of data exchange.

1.3 Versions

SPIRIT G.726/G.711 product is supplied as:

Unified G.726 Encoder+Decoder + G.711.

1.4 Supported Standards

This implementation is fully compliant with ITU-T Recommendations G.726 and G.711.

Chapter 2

G726 Encoder-Decoder (G726G711) Integration

This chapter provides descriptions, illustrations, and examples of Encoders and Decoders for G.726 algorithm integration.

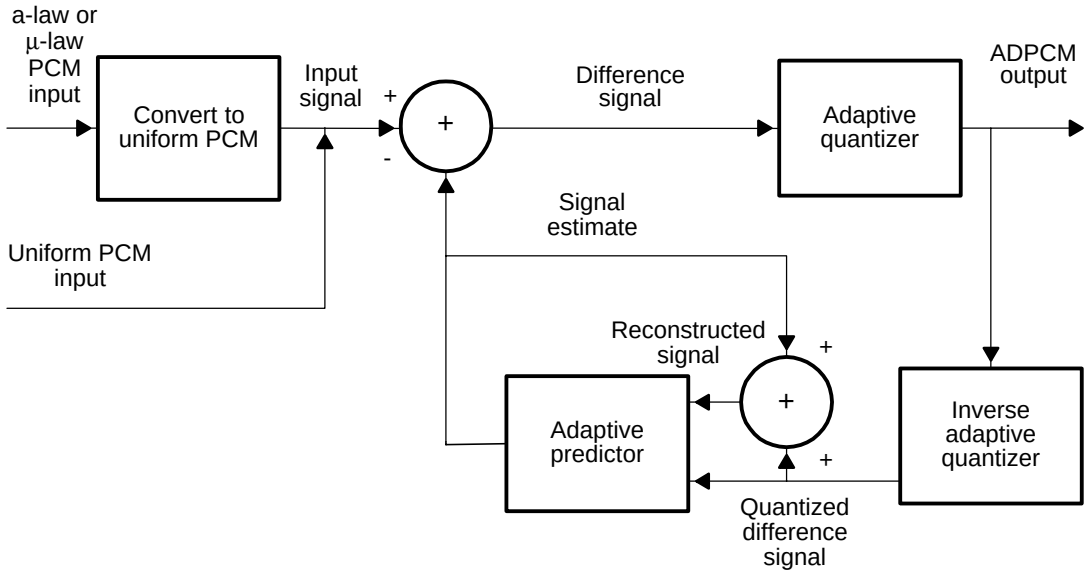
Topic **Page**

2.1 G.726 Encoder

2.1.1 Overview

Figure 2-1 illustrates a G.726 encoder.

Figure 2-1. G.726 Encoder Block Diagram



In brief, G726G711 module performs the following operations:

- Converts the A-law or μ-law PCM input signal to uniform (linear) PCM.
- Computes a difference signal by subtracting an estimate of the input signal from the input signal itself.
- Executes an adaptive 31-, 15-, 7-, or 4-level quantizing of a difference signal to assign five, four, three or two binary digits, respectively, to the value of the difference signal for transmission to the G726 decoder.
- Computes an estimate of the expected input signal.

An inverse quantizer produces a quantized difference signal from these same five, four, three or two binary digits, respectively. The signal estimation is added to this quantized difference signal to produce the reconstructed version of the input signal. Both the reconstructed signal and the quantized difference signal are operated upon by an adaptive predictor which produces the estimation of the input signal, thereby completing the feedback loop.

2.1.2 Integration Flow

In order to integrate G.726 encoder into a framework the user should follow these steps (as illustrated in Figure 2-2):

Step 1: Create a `G726G711_Params` structure, initialize the required rate (16, 24, 32 or 40 kb/s) and law (a-law, μ-law or linear law) of G.726 encoder.

Step 2: Call `G726G711_create` to create the instance of a G.726 encoder.

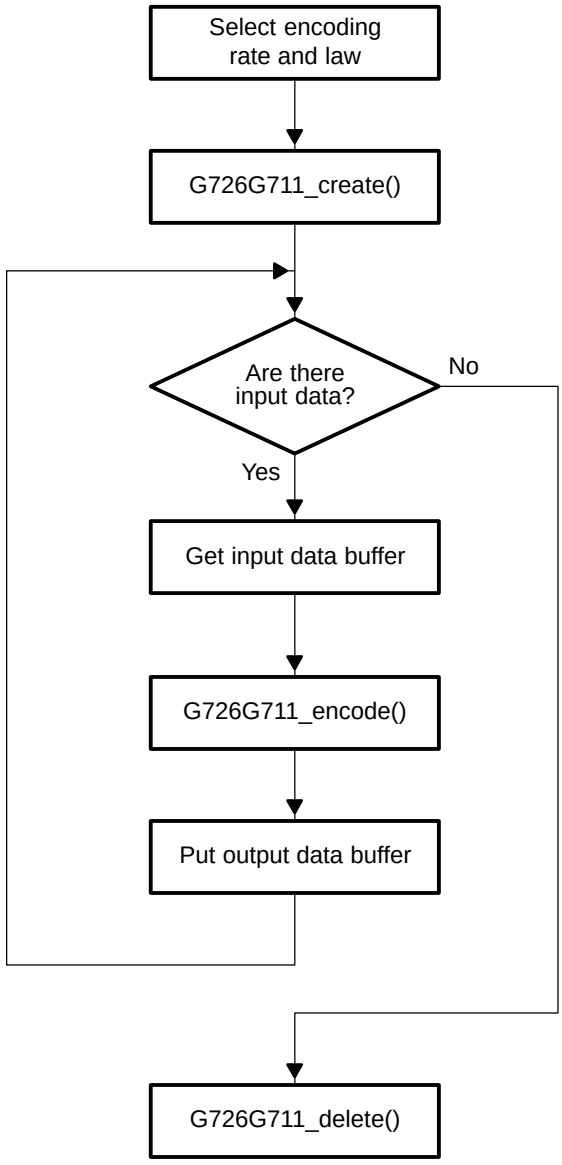
Step 3: Create a buffer of input samples (8 kHz, 14 bit for linear law or 8 bit for a-law or μ-law).

Step 4: Call `G726G711_encode` to get the output quantized difference signal buffer.

Step 5: Repeat a call to `G726G711_encode (Step 4)` for new input samples.

Step 6: Call `G726G711_delete` to delete the instance of a G.726 encoder.

Figure 2-2. G.726 Encoder Flowchart



2.1.3 Example of a Call Sequence for a G.726 Encoder

The example below demonstrates a typical call sequence for a G.726 encoder.

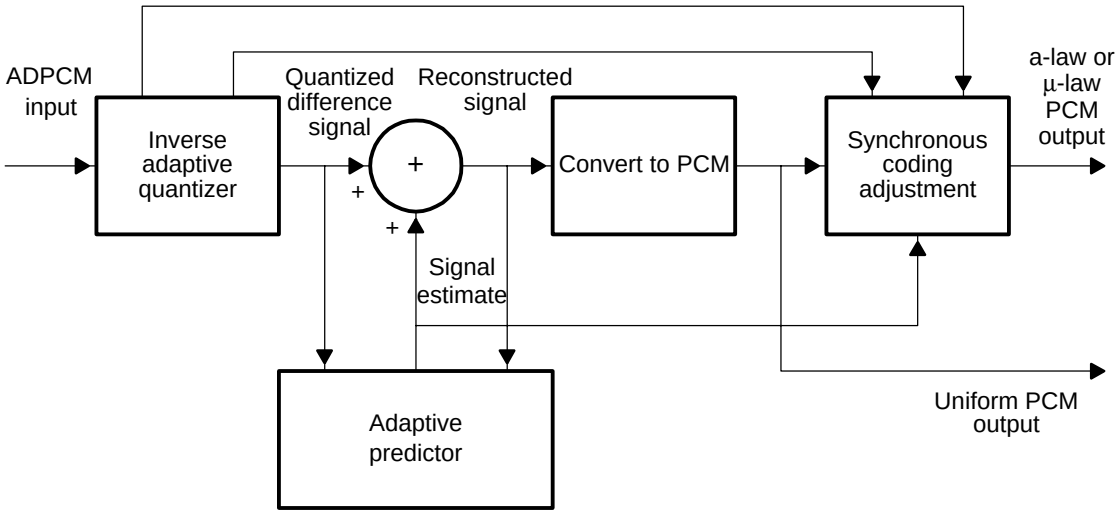
```
#define LENBUFF 80                  /* buffer for 10 msec */
XDAS_Int16      inputbuf[LENBUFF], outputbuf[LENBUFF];
G726G711_Params    encparams;
G726G711_Handle    enchandle;
encparams.rate= IG726G711_40KBPS; /* or another rate */
encparams.law = IG726G711_ALAW; /* or another law */
enchandle = G726G711_create (&G726G711_SPCORP_IG726G711, &encparams);
while(  /*there are input date*/ )
{
  /* get input data buffer */
  G726G711_encode(enchandle, inputbuf, ouputbuf, LENBUFF);
  /* put output data buffer */
}
G726G711_delete (enchandle);
```

2.2 G.726 Decoder

2.2.1 Overview

Figure 2-3 illustrates a G.726 decoder.

Figure 2-3. G.726 Decoder Block Diagram



In brief, G.726 decoder performs the following operations:

- An inverse quantizer produces a quantized difference signal from these same five, four, three or two binary digits, respectively.
- The signal estimation is added to this quantized difference signal to produce the reconstructed version of the input signal.
- Both the reconstructed signal and the quantized difference signal are processed by the adaptive predictor which computes an estimate of the expected output signal, reconstructs signal from difference form to uniform (linear) PCM.
- Compresses the linear PCM signal according to A-law or μ-law, thereby completing the feedback loop.

2.2.2 Integration Flow

In order to integrate G.726 decoder into a framework the user should follow these steps (as illustrated in Figure 2-4):

Step 1: Create a `G726G711_Params` structure and initialize it with the required rate (16, 24, 32 or 40 kb/s) and PCM type (A-law, μ-law or linear) of a G.726 decoder.

Step 2: Call `G726G711_create` to create the instance of G.726 decoder.

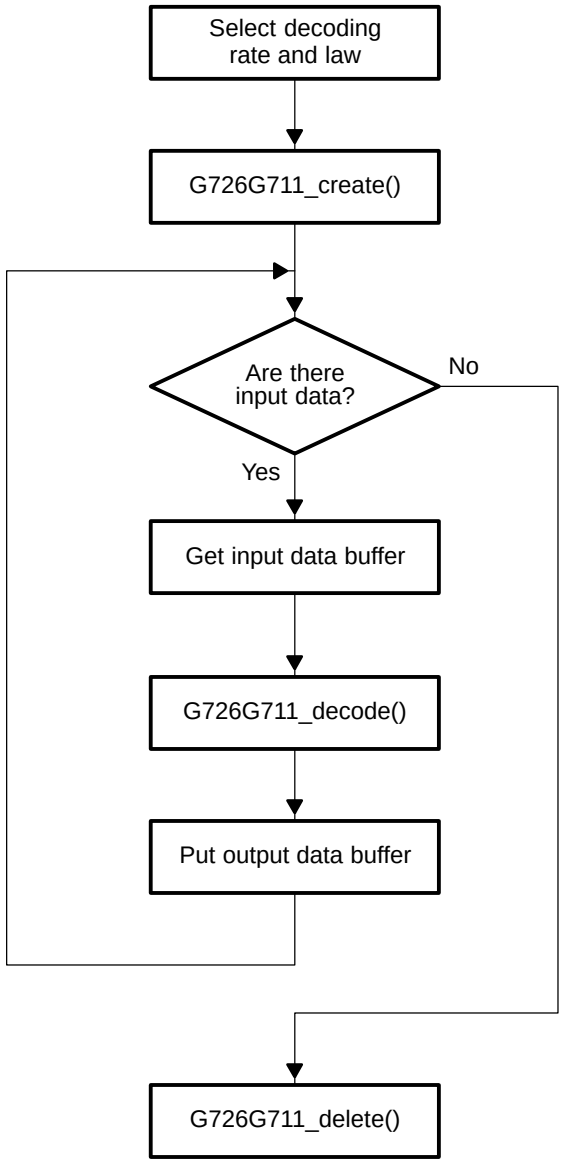
Step 3: Create a buffer of input samples (8 kHz, 2-5 bits).

Step 4: Call `G726G711_decode` to get the output signal buffer.

Step 5: Repeat call `G726G711_decode` for new input samples.

Step 6: Call `G726G711_delete` to delete the instance of G.726 decoder.

Figure 2-4. G.726 Decoder Flowchart



2.2.3 Example of a Call Sequence for a G.726 Encoder

The example below demonstrates a typical call sequence for a G.726 decoder.

```
#define LENBUFF 80                  /* buffer for 10 msec */
XDAS_Int16      inputbuf[LENBUFF], outputbuf[LENBUFF];
G726G711_Params    decparams;
G726G711_Handle    dechandle;
decparams.rate= IG726G711_40KBPS; /* or another rate */
decparams.law = IG726G711_ALAW; /* or another law */
dechandle = G726G711_create (&G726G711_SPCORP_IG726G711, &decparams);
while(  /*there are input data*/ )
{
  /* get input data buffer */
  G726G711_decode(dechandle, inputbuf, ouputbuf, LENBUFF);
  /* put output data buffer */
}
G726G711_delete (dechandle);
```

Chapter 3

G726 Encoder-Decoder (G726G711) API Descriptions

This chapter provides the user with a clear understanding of G726 Encoder-Decoder (G726G711) algorithms and their implementation with the TMS320 DSP Algorithm Standard interface (XDAIS).

Topic **Page**

3.1 Overview

The G.726 common object supports G.726 encoders, G.726 decoders, as well as G.711 encoders and decoders. Such unification allows to minimize required program memory at the cost of small MIPS increase. The object name is G726G711. To select G.726 or G.711 use the IG726G711_Rate (see Table 3-4).

3.2 Standard Interface Structures

The section describes parameter structures for G726G711 algorithm.

Table 3-1 lists the type and location of the Standard Interface structures.

Table 3-1. Standard Interface Structures Summary

Parameters	Located in Table...
Interface Creation	Table 3-2
PCM Type	Table 3-3
Data Rate	Table 3-4
Initial State	Table 3-5
Status Structure	Table 3-6

3.2.1 Instance Creation Parameters

Description This structure defines the creation parameters for the algorithm. A default parameter structure is defined in "`iG726G711.c`".

Structure Definition

Table 3-2. G726G711 Interface Creation Parameters

```
typedef struct IG726G711_Params {
```

Params Type	Params Name	Description
`IG726G711_Law`	`law`	PCM type (see Table 3-3)
`IG726G711_Rate`	`rate`	data rate (see Table 3-4)
`IG726G711_Reset`	`reset`	define the initial state (see Table 3-5)

```
} IG726G711_Params;
```

Types `IG726G711_Law`, `IG726G711_Rate`, `IG726G711_Reset` are defined as:

Table 3-3. PCM Type

```
typedef enum IG726G711_Law {
```

Enumerated Name	Value	Description
IG726G711_ULAW	0	μ-law PCM
IG726G711_ALAW	1	a-law PCM
IG726G711_UNIFORMLAW	2	uniform PCM

```
} IG726G711_Law;
```

Table 3-4. Data Rate

```
typedef enum IG726G711_Rate {
```

Enumerated Name	Value	Description
IG726G711_16KBPS	0	16k bits per second input data rate
IG726G711_24KBPS	1	24k bits per second input data rate
IG726G711_32KBPS	2	32k bits per second input data rate
IG726G711_40KBPS	3	40k bits per second input data rate
IG726G711_64KBPS	4	64k bits per second input data rate (equivalently G711)

```
} IG726G711_Rate;
```

Table 3-5. Initial State

```
typedef enum IG726G711_Reset {
```

Enumerated Name	Value	Description
IG726G711_RESETNO	0	reset is not made
IG726G711_RESETYES	1	reset is made

```
} IG726G711_Reset;
```

Type `IG726G711_Params`, `IG726G711_Law`, `IG726G711_Rate`, `IG726G711_Reset` are defined in "`iG726G711.h`".

3.2.2 Status Structure

Description This structure defines the status parameters for the algorithm. It is used by the control function to set or return algorithm parameters to the application.

Status structure is used for control purposes. Status can be received by function `G726G711_getStatus()`.

Structure Definition

Table 3-6. G.726G711 Status Parameters

```
typedef struct IG726G711_Status {
```

Status Type	Status Name	Description
`IG726G711_Reset`	`reset`	define the initial state of the G711oder

```
} IG726G711_Status;
```

Type `IG726G711_Status` defined in "`iG726G711.h`".

3.3 Standard Interface Functions

Table 3-7 summarizes the standard Interface functions of the G726G711 API. They are required when using G726G711.

`G726G711_apply()` and `G726G711_control()` are optional, but neither are supported by Spirit Corp.

Table 3-7. G726G711 Standard Interface Functions

Functions	Description	See Page...
`G726G711_init`	Algorithm initialization	3-5
`G726G711_exit`	Algorithm deletion	3-5
`G726G711_create`	Instance creation	3-6
`G726G711_delete`	Instance deletion	3-6

3.3.1 Algorithm Initialization

G726G711_init *Calls the framework initialization function to initialize G726G711*

Description This function calls the framework initialization function, `ALG_init()`, to initialize the algorithm `G726G711`. For G726G711, this function does nothing. It can be skipped and removed from the target code according to *Achieving Zero Overhead With the TMS320 DSP Algorithm Standard IALG Interface* (SPRA716).

Function Prototype `void G726G711_init()`

Arguments none

Return Value none

3.3.2 Algorithm Deletion

G726G711_exit *Calls the framework exit function to remove G726G711*

Description This function calls the framework exit function, `ALG_exit()`, to remove the algorithm `G726G711`.

For G726G711, this function does nothing. It can be skipped and removed from the target code according to *Achieving Zero Overhead With the TMS320 DSP Algorithm Standard IALG Interface* (SPRA716).

Function Prototype `void G726G711_exit()`

Arguments none

Return Value none

3.3.3 Instance Creation

G726G711_create *Calls the framework create function to create an instance object*

Description In order to create a new G726G711 object, `G726G711_create` function should be called. This function calls the framework create function, `ALG_create()`, to create the instance object and perform memory allocation tasks. Global structure `G726G711_SPCORP_IG726G711` contains `G726G711` virtual table supplied by SPIRIT Corp.

Function Prototype

```
G726G711_Handle G726G711_create (const IG726G711_Fxns
*fxns, const G726G711_Params *prms);
```

Arguments

`IG726G711_Fxns *`	Pointer to vendor's functions (Implementation ID). Use reference to `G726G711_SPCORP_IG726G711` virtual table supplied by SPIRIT Corp.
`G726G711_Params *`	Pointer to Parameter Structure. Use `NULL` pointer to load default parameters.

Return Value

`G726G711_Handle`	Defined in file "`G726G711.h`". This is a pointer to the created instance object.

3.3.4 Instance Deletion

G726G711_delete *Calls the framework delete function to delete an instance object*

Description This function calls the framework delete function, `ALG_delete()`, to delete the instance object and perform memory de-allocation tasks.

Function Prototype

```
void G726G711_delete (G726G711_Handle handle)
```

Arguments

`G726G711_Handle`	Instance's handle obtained from `G726G711_create()`

Return Value none

3.4 Vendor-Specific Interface Functions

This section describes the vendor-specific functions in the SPIRIT's algorithm implementation and interface (extended IALG methods). Table 3-8 summarizes SPIRIT's API functions of G726G711.

The whole interface is placed in header files `iG726G711.h, G726G711.h, G726G711_spcorp.h.`

Table 3-8. G.726G711-Specific Interface Functions

Functions	Description	See Page...
`G726G711_encode`	Encode data	3-7
`G726G711_decode`	Decode data	3-8

3.4.1 Encode data

G726G711_encode *Gets a selected buffer sample to encode*

Description Gets buffer of A-law, μ-law or uniform PCM samples and encode them to ADPCM bit stream.

Function Prototype

```
XDAS_Void G726G711_encode
(G726G711_Handle handle,
 const XDAS_Int16 *inputbuf,
 XDAS_Int16 *outputbuf,
 const XDAS_0UInt16 length)
```

Arguments

`handle`	Pointer to G726G711 object.
`inputbuf`	Pointer to input buffer; each sample resides in a separate word, even for 8-bit samples.
`outputbuf`	Pointer to output buffer; each set of bits (2,3,4 or 5 bits), corresponding to one input sample, resides in a separate byte (which is also a word on C54x platform).
`length`	Length of input and output buffers.

Return Value none

Restrictions Maximal length of input and output buffers is 65535, range of input value is from 0 to 255 for μ-law and A-law PCM and from -8192 to 8191 for uniform PCM.

3.4.2 Decode data

G726G711_decode *Gets a buffer of ADPCM value and decodes as specified by user*

Description

Gets buffer of ADPCM values and decode them to A-law, μ-law or uniform PCM.

Function Prototype

```
XDAS_Void G726G711_decode
(G726G711_Handle handle,
 const XDAS_Int16 *inputbuf,
 XDAS_Int16 *outputbuf,
 const XDAS_0UInt16 length)
```

Arguments

`handle`	Pointer to G726G711 object.
`inputbuf`	Pointer to input buffer; each set of bits (2,3,4 or 5 bits), corresponding to one input sample, resides in a separate byte (which is also a word on C54x platform).
`outputbuf`	Pointer to output buffer; each sample resides in a separate word, even for 8-bit samples.
`length`	Length of input and output buffers.

Return Value

none

Restrictions

Maximal length of input and output buffers is 65535, input bit stream consists of 2 bits 16 kbit/s, 3 bits for 24 kbit/s, 4 bits for 32 kbit/s, 5 bits for 40 kbit/s.

Appendix A

Test Environment

Note: Test Environment Location

This chapter describes test environment for the G726 object.

For TMS320C54CST device, test environment for standalone G726 object is located in the Software Development Kit (SDK) in `Src\FlexExamples\StandaloneXDAS\G726`.

Topic **Page**

A.1 Description of Directory Tree

The SDK package includes the test project “test.pjt” and corresponding reference test vectors. The user is free to modify this code as needed, without submissions to SPIRIT Corp.

Table A-1. Test Files for G726

File	Description
main.c	Test file
..\ROM\CSTRom.s54	ROM entry address
Test.cmd	Linker command file
Vectors*.pcm	Reference output test vectors

A.1.1 Test Vectors Format

All test vectors are raw PCM files with the following parameters:

- Bits per sample: 16, Mono
- Word format: Intel PCM (LSB goes first)
- Encoding:
  - nrm.pcm, out_l2t.pcm, out_l3t.pcm, out_l4t.pcm, out_l5t.pcm: Uniform
  - nrmu.pcm, out_u2t.pcm, out_u3t.pcm, out_u4t.pcm, out_u5t.pcm: μ-law
  - nrma.pcm, out_a2t.pcm, out_a3t.pcm, out_a4t.pcm, out_a5t.pcm: a-law

A.1.2 Test Project

To build and run a project, the following steps must be performed:

Step 1: Open the project: `Project\Open`

Step 2: Build all necessary files: `Project\Rebuild All`

Step 3: Initialize the DSP: `Debug\Reset CPU`

Step 4: Load the output-file: `File\Load program`

Step 5: Run the executable: `Debug\Run`

Once the program finishes testing, the file *Output.pcm* will be written in the current directory. Compare this file with the reference vector contained in the directory *Vectors*.

Note: Test Duration

Since the standard file I/O for EVM is very slow, testing may take several minutes. Test duration does not indicate the real algorithm's throughput.

Index

M

P

S

T

X