Application Report
SPRA564B - February 2000

G.729/A Speech Coder: Multichannel TMS320C62x Implementation

Chiouguey Chen
Xiangdong Fu

C6000 Applications

ABSTRACT

This document provides a detailed description of the implementation of the G.729 speech encoder and decoder (codec) on the Texas Instruments (TI™) TMS320C6201 digital signal processor (DSP). Topics include program structure, code writing rules, data/program memory requirements, and performance evaluation. Issues regarding multichannel implementation and interrupts are also addressed.

Contents

List of Figures

List of Tables

1 Introduction

This document provides a detailed description of the implementation of the G.729/A speech encoder and decoder (codec) the Texas Instruments (TI™) TMS320C6201 digital signal processor (DSP). Issues on multichannel implementation and interrupts are also addressed.

The G.729/A speech codec is defined by the Telecommunication Standardization Sector of the International Telecommunication Union (ITU-T).

The G.729 speech codec uses the conjugate-structure algebraic code excited linear-prediction (CS-ACELP) algorithm, which is an analysis-by-synthesis algorithm and belongs to the class of speech coding algorithms known as code excited linear prediction (CELP). Therefore, the speech decoder is primarily a subset of the speech encoder. For every 10-ms speech frame, the speech signal is analyzed to extract the parameters of the CELP model (linear-prediction filter coefficients, adaptive and fixed-codebook indices and gains). The G.729A is the annex A for G.729 to reduce complexity of the CS-ACELP speech codec.

Each speech frame is equally divided into two subframes. Except for LP parameters, which are extracted once per frame (and therefore are frame parameters,) the rest of the parameters are subframe parameters. These parameters are quantized into 80 bits, resulting in a transmission rate of 8 kbits/s. In the G.729 decoder, the quantized parameters are decoded and a synthetic excitation is generated by adding the adaptive and fixed codevectors scaled by their respective gains. The synthetic excitation is then filtered through the LP synthesis filter to generate the synthetic speech waveform, which is processed by a so-called post-filter to further improve speech quality.

2 Multichannel System

The multichannel system can run more than one algorithm at the same time. Any algorithm compliant with the eXpress DSP Algorithm Standard (xDAIS) [7][8][9][10][11] is capable of multichannel processing. An xDAIS-compliant algorithm requires three functional modules: initialization, freeing and kernel. The kernel module performs the algorithm processing while the initialization and freeing module initializes/frees the algorithm context data[4].

The XDAIS application programming interface (API) for the initialization is defined as

```
Int     (*algInit)(IALG_Handle, const IALG_MemRec *,
IALG_Handle, const IALG_Params *);
```

The XDAIS application programming interface (API) for freeing is defined as

```
Int     (*algFree)(IALG_Handle, const IALG_MemRec *);
```

where the first parameter of type IALG_Handle is a pointer that contains the current context data memory location. During framework initialization, the framework calls the algorithm initialization functions as shown in Figure 1. All the initialization functions perform the same tasks: initialize context data and store to the desired memory location. After the initialization function is completed, the system repeatedly calls an algorithm/algorithms until all the frames have been processed.

The system may need to call the algFree() function to free the algorithm instance after all processing is done.

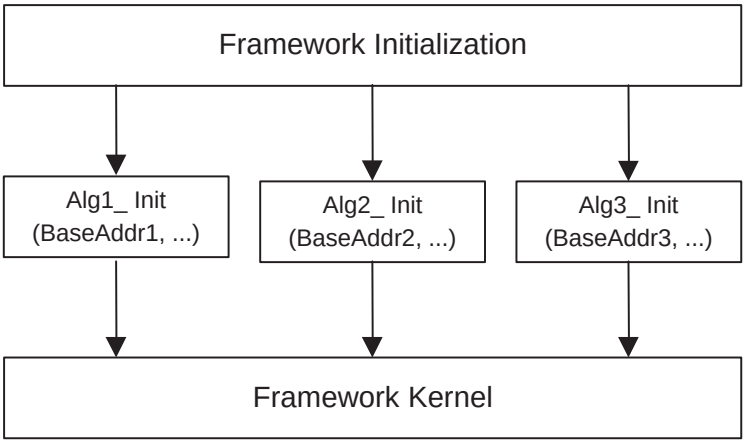

Figure 1. Framework Initialization

After the initialization function is completed, the framework repeatedly calls an algorithm/ algorithms until all the frames have been processed as shown in Figure 2.

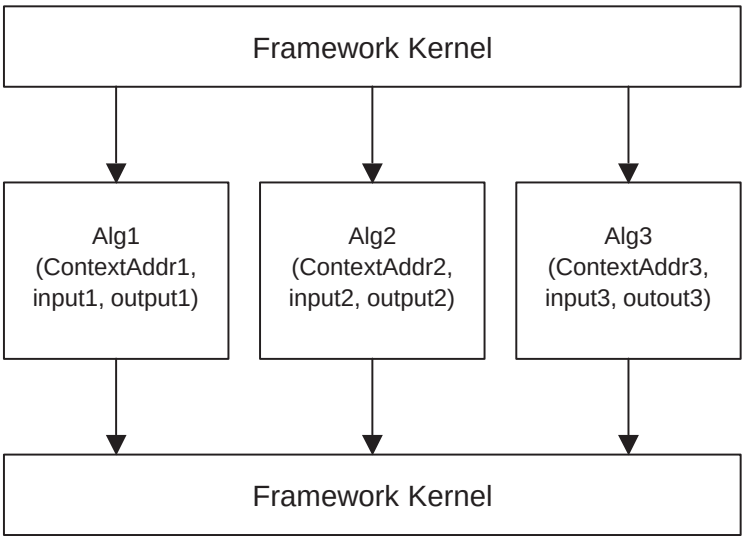

Figure 2. Framework Kernel

The APIs for the called algorithm, encoder and decoder, is defined as

```
extern DAIS_Int8 G729E_TI_encode(IG729ENC_Handle handle,
DAIS_Int16* in, DAIS_Uint16* out);
extern DAIS_Int8 G729E_TI_encode(IG729ENC_Handle handle,
DAIS_Int16* in, DAIS_Uint16* out);
```

where handle points to the start address of the context data which contains the start address of the constant tables of the algorithm. The pointers in and out point to the input and output data. Since XDAIS does not specify standard APIs for algorithm processing, these two APIs are algorithm specific.

3 Algorithm Description

The complete speech codec is implemented in C using the TMS320C6201 optimizer. Each basic math operation defined in basic_op2.c either is replaced by its corresponding TMS320C62x intrinsic or is static inlined. All of the 32-bit functions in oper_32b.c are expressed in intrinsics and are static inlined.

The codec is multichannel enabled. After processing one frame of one channel, the codec is capable of processing one frame of another channel. Additional effort is needed to make the codec interruptible within frame processing to allow the decoder higher priority than the encoder and/or to deliver the parameters timely to the next stage of the whole process.

Both encoder input and decoder output are 16-bit PCM values.

The interface is an 82-bit stream to the preliminary channel coding stage.

3.1 Encoder Structure

The G.729 speech encoder is divided into five submodules:

- Submodule 1 contains pre-processing, the LP analysis and LPC to LSP conversion, including Pre_Process(), Autocorr(), Lag_window(), Levinson() and Az_lsp().
- Submodule 2 calls Qua_lsp() to conduct LSP quantization.
- Submodule 3 generates the interpolated LPC parameters, computes weighted speech, and finds the open-loop pitch, including Int_qlpc(), Int_lpc(), Weight_Az(), Residu(), Syn_filt() and Pitch_ol().
- Submodule 4 performs the closed-loop fractional pitch search and the adaptive codebook search, calling Pitch_fr3(), Enc_lag3(), Pred_lt_3(), Convolve() and G_pitch().
- Submodule 5 performs the innovative codebook search and filter memory update, calling ACELP_Codebook(), Corr_xy2(), Qua_gain() and Syn_filt().

Submodules 1 through 3 are for frame processing and should be done once per frame. Submodules 4 and 5 are for subframe processing and should be repeated twice per frame.

3.2 Decoder Structure

The G.729 speech decoder is divided into two submodules:

- Submodule 1 deals with frame parameter recovering, which should be done once per frame.
- Submodule 2 contains subframe parameter recovering, speech synthesizing, and post-filtering. Submodule 2 should be repeated twice per frame.

4 Coding Guidelines

This program is an optimized C implementation.

4.1 Variable, Array, and Pointer Names

We use the same name conventions for variables, arrays, and pointers as in the reference C code.

4.2 Math Operations

All math operations defined in basicop2.c and oper_32b.c in the reference C program are replaced by TMS320C62x intrinsics, if possible. In addition to the use of intrinsics, basic operations div_s and negate and all the math functions defined in oper_32b.c are inlined. Their expressions in intrinsics or inlined versions reside in basic_op.h and oper_32b.h, respectively. For a description and usage of the TMS320C62x intrinsics, see the *TMS320C62x/C67x Programmer's Guide* (SPRU198) [4].

4.3 Tables and Functions

Tables are defined in the G729_TABLES structure (tab_ld8k.h). The G729_TABLES is used in both encoder and decoder of G.729 and G.729A speech coders.

The functions are stored based on their functionality for either the G.729 or G.729A speech coder.

- A function name ending with the letter "A" or "a" belongs to G.729A code; for example, Acelp_ca.c is used in the G.729A speech coder. A function name similar to the previous-stated function name belongs to G.729 code; for example, Acelp_co.c is used in the G.729 speech coder.
- The function name embedded with the prefix "dec" or "de" belongs to the decoder of either the G.729 or G.729A code; for example, Dec_ld8k.c is used in the G.729 decoder. A function name similar to the previous-stated function name belongs to the encoder; for example, Cod_ld8k.c is used in the G.729 encoder.
- All functions except the previously mentioned functions are used by both the encoder and the decoder of G.729 and G.729A.
  - If it is in natural C, this function is named *.c in this implementation.
  - If it is in optimized C, this function is named *_opt.c in this implementation.
  - If it is in assembly, this function is named *_ho.asm in this implementation.

4.4 Interrupt Issues

The codec is multichannel enabled in the sense that, after the process of one frame of one channel, the code can process one frame of another channel. That is, the codec is interruptible at the frame process boundary. Additional study is required to make the codec interruptible within a frame process.

4.4.1 Interrupts at the Frame Boundary

To ensure access to the correct memory of each channel, static variables and arrays with values that must be kept from one frame to the next are sorted into one of the two structures:

- ENCODER_G729_MEM_BLK for the encoder
- DECODER_G729_MEM_BLK for the decoder

Both of these two structures are defined in Coder.c and Decoder.c for encoder and decoder, respectively. The static variables and arrays of the codec of each channel are initialized with standard IALG initialization functions *g729eInit()* and g*729dInit()*, respectively.

To start the encoding process for a channel, the API is

```
extern DAIS_Int8 G729E_TI_encode(IG729ENC_Handle handle,
DAIS_Int16* in, DAIS_Uint16* out);
```

where the pointer handle denotes the start address of the channel context data of that particular channel, the pointer in points to the start address of the input speech buffer of the frame to be processed, and the pointer out points to the start address of the encoded bit stream buffer.

Similarly, to start a decoding process of a channel, the API is

```
extern DAIS_Int8 G729D_TI_decode(IG729DEC_Handle handle,
DAIS_Uint6* in, DAIS_Int16 out[]);
```

where the pointer *handle* denotes the start address of the channel context data of that particular channel, the pointer *in* points to the start address of the input bit stream buffer, and the pointer *out* points to the start address of the synthesized speech buffer.

4.4.2 Interrupts at the Submodule Boundary

You can interrupt the codec anywhere other than software pipelined loops. (Of course, you can even choose not to interrupt the codec until the end of a frame process as long as the overall system design requirement is met.) If interrupts are necessary, we suggest that, in general, interrupts occur in certain places so that the increases in data memory and cycle counts are kept minimal. Here we describe one particular interrupt scheme for this codec in which the interrupt occurs at the end of each submodule.

5 Data Memory Requirements

The data memory is divided into three groups:

- Context data
- Tables
- Local variables and arrays

5.1 Memory for Context Data

The context data are the static variables and arrays with values that must be kept from one frame to the next. Table 1 and Table 2 summarize the encoder/decoder context data names and their sizes. The context data for the G.729/A codec are sorted into one of the two structures, ENCODER_G729_MEM_BLK and DECODER_G729_MEM_BLK.

Table 1. Encoder Context Data Structure (ENCODER_G729_MEM_BLK)

Array			Size (Bytes)
prm			22
*new_speech			4
*preProcMemBlk (Pre_Proc.c)	y2		4
	y1		4
	x0		2
	x1		2
*coderMemBlk (cod_ld8k.c)	old_speech		480
	*speech		4
	*p_window		4
	*new_speech		4
	old_wsp		446
	*wsp		4
	old_exc		468
	*exc		4
	ai_zero		102
	*zero		4
	lsp_old		20
	lsp_old_q		20
	mem_syn		20
	mem_w0		20
	mem_w		20
	mem_err		100
	*error		4
	sharp		2
	L_exe_err		16
	*Levinson_mblk (Lpc.c)	old_A	22
		old_rc	4
	*Lsp_encw_mblk (Qua_lsp.c)	freq_prev	80
	smooth		2
	LarOld		4
	extra		2
	past_qua_en		8
Total			**1902**

Table 2. Decoder Context Data Structure (DECODER_G729_MEM_BLK)

Array			Size (Bytes)
synth_buf			190
voicing			2
*decoderMblk (Dec_ld8k.c)	old_exc		468
	*exc		4
	lsp_old		20
	mem_syn		20
	sharp		2
	old_T0		2
	gain_code		2
	gain_pitch		2
	*Lsp_decw_mblk (lspdec.c)	freq_prev	80
		prev_ma	2
		prev_lsp	20
	seed		2
	past_qua_en		8
*postfilterMblk (Pst.c)	apond2		40
	mem_stp		20
	mem_zero		20
	res2		384
	*res2_ptr		4
	*ptr_mem_stp		4
	gain_prec		2
*postprocessMblk (Post_pro.c)	y2		4
	y1		4
	x0		2
	x1		2
Total			**1310**

5.2 Memory for Tables

The constant tables are sorted into the G729_TABLES defined in tab_ld8k.h. This structure is common to both the encoder and decoder of G.729/A codec and takes 6024 bytes of data memory structure. Table 3 summarizes the tables in the source code. Notice that the memory for tables is independent of the number of channels running on the chip.

Table 3. List of G.729_TABLES

Table	Size (Bytes)
Hamwindow	480
lag	40
table	130
slope	128
table2	128
slope_cos	128
slope_acos	128
lspcb1	2560
lspcb2	640
fg	160
fg_sum	40
fg_sum_inv	40
inter_3l	62
pred	8
gbk1	32
gbk2	64
map1	16
map2	32
coef	8
L_coef	16
thr1	8
thr2	16
imap1	16
imap2	32
b100	6
a100	6
b140	6
a140	6
bitsno	22
tabpow	66
tablog	66
tabsqr	98
tab_zone	306
gridG729A	102
inter_3	26
grid	122
tab_hup_s	56
tab_hup_l	224
Total	**6024**

5.3 Memory for Local Variables and Arrays

The local variables and arrays are stored in the stacks and consume about 2780 bytes of data memory space.

6 Performance/Code Size Results

Table 4 summarizes the measured performance and code size using the TMS320C6201 tools version 2.1.

Table 4. G.729/A Performance/Code Size Results

	Performance (MHz)		Code Size (K bytes)
G.729	30.7	G.729/A	92.1
G.729A	18.2		

7 References

1. *ITU-T Recommendation G.729—CS-ACELPD*, March 1996.
2. *ITU-T Recommendation G.729 Annex A—Reduced Complexity CS-ACELPD*, March 1996.
3. M. Wang*, GSM Enhanced Full Rate Speech Coder: Multichannel TMS320C62x Implementation*, SPRA565.
4. Xiangdong Fu and Zhaohong Zhang*, A Multichannel/Algorithm Implementation on the TMS320C6000,* SPRA556.
5. Xiangdong Fu and Zhaohong Zhang*, TMS320C6000 Multichannel Vocoder Technology Demonstration Kit Host Side Design,* SPRA558.
6. Xiangdong Fu, *TMS320C6000 Multichannel Vocoder Technology Demonstration Kit Target Side Design,* SPRA560.
7. *eXpressDSP Algorithm Standard (Rules & Guidelines)*, SPRU352.
8. *eXpressDSP Algorithm Standard (API Reference)*, SPRU360.
9. Stig Torud, *Making DSP Algorithms Compliant to the eXpressDSP Algorithm Standard*, SPRA579.
10. Carl Bergman, *Using the eXpressDSP Algorithm Standard in a Static DSP System,* SPRA577.
11. Carl Bergman, *Using the eXpressDSP Algorithm Standard in a Dynamic DSP System*, SPRA580.

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

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

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.