Application Report
SPRA566B - February 2000

IS-127 Enhanced Variable Rate Speech Coder: Multichannel TMS320C62x Implementation

Min Wang
Xiangdong Fu
C6000 Applications

ABSTRACT

This application report provides a detailed description of the implementation of the IS-127 enhanced variable rate speech codec (EVRC) on the Texas Instruments (TI™) TMS320C62x digital signal processor (DSP). Topics include program structure, code writing rules, data/program memory requirements, and performance evaluation. Issues on multichannel implementation and interrupts are also addressed.

Contents

List of Figures

List of Tables

1 Introduction

This application report provides a detailed description of the implementation of the IS-127 enhanced variable rate speech codec (EVRC) on the TMS320C6000 DSP. Issues on multichannel implementation and interrupts are also addressed.

The EVRC is a standard for the “Speech Service Option 3 for Wideband Spread Spectrum Digital System,” which has been employed in both IS-95 cellular systems and ANSI J-STC-008 PCS (personal communications systems). The EVRC algorithm is specified in IS-127. A bit-exact reference C program is also provided in IS-127, along with a set of testing sequences. The code implemented on the TMS320C62x is fully bit compatible with the bit-exact reference code on all testing sequences.

The EVRC uses the relaxed code excited linear prediction (RCELP) algorithm, an analysis-by-synthesis algorithm that belongs to the class of speech coding algorithms knows as code excited linear prediction (CELP). Unlike most existing CELP-type codecs, RCELP extracts pitch information once for every 20-ms speech frame, which results in larger algebraic codebook for prediction residuals and robustness under impairment channel conditions. The EVRC processes every 20-ms speech frame using three of the four primary traffic packet types permitted by IS-95 Multiplex Option 1:

- Rate 1 (171 bits/packet)
- Rate 1/2 (80 bits/packet)
- Rate 1/8 (16 bits/packet)

The rate is determined by the contents of the current speech frame and the history of the characteristics of the previous frames, as well as the command from the high level of the traffic controller. Table 1 summarizes the bit allocations of the traffic packets.

Table 1. Bit Allocation of the Traffic Packet Types of EVRC

Field	Rate1	Rate 1/2	Rate 1/8
Spectral transition indicator	1		
LSP	28	22	8
Pitch delay	7	7	
Delta delay	5		
ACB gain	9	9	
FCB shape	105	30	
FCB gain	15	12	
Frame energy			8
(unused)	1		
Total	**171**	**80**	**16**

Every 20-ms speech frame is divided into three subframes of lengths 53, 53, and 54 speech samples. Because rates 1 and 1/2 use similar speech coding algorithms, their bit allocations are very similar, except for the spectral transition indicator and delay difference parameters. However, rate 1/8 usually applies to background noise rather than speech, only allocating bits for LSP and frame energy estimation.

Refer to IS-127 [1] for a detailed description of the EVRC vocoder.

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[5][6][7][8][9] is capable of multichannel processing. An XDAIS compliant algorithm requires three functional modules; initialization, freeing and kernel. The kernel module performs the algorithm processing and the initialization and freeing module initializes/frees the algorithm context data[2].

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. 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(s) 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.

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

```
extern DAIS_Int8 EVRC_TI_encode(IEVRCENC_Handle handle, DAIS_Int16* in,
DAIS_Uint16* out);
extern DAIS_Int8 EVRCD_TI_decode(IEVRCDEC_Handle handle, DAIS_Uint6*
in, DAIS_Int16 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 mixed C and TMS320C62x assembler, with the majority of the code in C (TMS320C62x optimized C). Each basic math operation defined in mathevrc.c either is replaced by its corresponding TMS320C62x intrinsic or is static inlined. The two 32-bit multiply functions in mathdp31.c are expressed in intrinsics and are static inlined.

The codec is multichannel enabled. After processing one frame of one channel, the codec can process one frame of another channel. Additional effort is needed to make the codec interruptible within a frame processing to allow the decoder higher priority than the encoder.

Features are implemented, including input signal preprocessing, determining the data rate, and adaptive post-filter.

TI's implementation of this algorithm is fully bit compatible with the bit-exact Reference C code, version 1.21 (program date April 11, 1997) on all testing sequences.

3.1 Encoder Structure

The EVRC speech encoder is divided into submodules common for all of the rates and submodules unique to each rate. The common submodules include submodule 1 to submodule 3 and are executed once per frame:

- Submodule 1 contains the input signal preprocessing and all the subroutines associate with noise suppression.
- Submodule 2 conducts pre-encoding which includes LPC analysis, conversion between LPC coefficients and LSP, computing weighted residual signal and determining the frame pitch period, and rate selection.
- Submodule 3 performs LSP quantization for all rates, quantizes pitch period for rate 1 and rate 1/2, and computes delta delay for rate 1.

The submodules unique to each rate include submodule 4_8 for rate 1/8 and submodule 4_12 for rate 1 and rate 1/2:

- Submodule 4_8 generates the random excitations of all three subframes and estimates the energy of the excitation.

- Submodule 4_12 performs the innovative codebook search and filter memory update and should be repeated three times per frame.

3.2 Decoder Structure

The EVRC 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 three times per frame.

3.3 Main Program

Figure 1 shows the file tree of the EVRC speech codec software package.

```
main  ---------  code --------- common ---------------- functions
  |                |                         |
           |                 |----- encoder              |------  intrinsics, inlined math
           |                 |                                      tables & constants
           |                 |----- decoder
           |
           |----- test  --------- enc
                             |---- dec
                             |---- run_evm
```

Figure 1. File Tree of the EVRC Speech Coder Software Package

The code directory contains the codec.c file that is the top file of the whole codec; the linker command file, codeclnk.cmd; the make file, makefile; and the simulator initialization file, siminit.cmd.

The common directory contains the include files *.h; the table files, *.tab; and the source files (in *.c or *.asm) common to both the encoder and the decoder.

The encoder directory contains the encoder top file, encoder.c, and the include files, *.h, used only in the encoder. The program follows the structure of the encoder described above. The files specific to one submodule are stored in the corresponding subdirectory.

The decoder directory contains all of the files used in the decoder only.

The test directory contains test files to run the encoder and decoder with a simulator or an EVM (evaluation module).

4 Coding Guidelines

This program is a mixed C and assembly 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 mathevrc.c and mathdp31.c in the reference C program are expressed with TMS320C62x intrinsics, if possible. In addition to using intrinsics, basic operations divide_s and negate and all of the two multiply functions defined in mathdp31.c are inlined. Their expressions in intrinsics or inlined versions reside in mathevrc.h and mathdp31.h, respectively. For a description and usage of the TMS320C62x intrinsics, see the *TMS320C62x/67x Programmer's Guide* (SPRU198) [10].

4.3 Tables and Functions

The names of tables are the same as in the reference C program, except for the tables used in the routines Interpol and Interpol_delay. Each of these two routines contains two look-up tables with the same names, frac and shft_frac, but different elements. To eliminate such ambiguity and to help the memory management and accessibility of these tables, we reserve such names for the tables in the Interpol routine and append delay to each of the tables for those in Interpol_delay. Refer to Table 4 for a list of all tables in this codec.

The functions are sorted in three categories based on their functionality in the vocoder: encoder, decoder, and common functions.

- All functions belonging to the encoder only are located in the encoder directory.
- All functions belonging to the decoder only are located in the decoder directory.
- All functions used by both the encoder and the decoder are placed in the common directory.
- Each function is either in C or assembly. The name of a function closely matches its name in the reference C program. For a function named * in the reference C program,
  - If it is in C, this function will be named *.c in this implementation.
  - If it is in hand-optimized assembly, this function will be named *_ho.asm in this implementation.

4.4 Interrupt Issues

The codec is multichannel enabled in the sense that after the processing 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 the access to the right memory of each channel, the static variables and arrays whose values have to be kept from one frame to the next are sorted into one of the two structures, EVRC_Sta_Encoder for the encoder and EVRC_Sta_Decoder for the decoder. Both of these two structures are defined in EVRC_StaMem.h. The static variables and arrays of the codec of each channel are initialized with standard IALG initialization functions evrceInit() and evrcdInit(), respectively.

To start an encoding process of a channel, the API is

```
      DAIS_Int8 EVRCE_TI_encode(IEVRCENC_Handle handle,DAIS_Int16* in,
      DAIS_UInt8* out)
```

where the pointer handle denotes the start address of the context memory block that containes the structure defined as EVRC_Sta_Encoder 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

```
      DAIS_Int8 EVRCD_TI_decode(IEVRCDEC_Handle handle,DAIS_UInt8* in,
      DAIS_Int* out)
```

where the pointer handle denotes the start address of the context memory blcok that contains the structure defined as EVRC_Sta_Decoder 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 at any place 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, like the end of each submodule, so that the increases in data memory and cycle counts are kept minimal.

5 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 2 and Table 3 summarize the encoder/decoder context data names and their sizes. The context data for the EVRC codec are sorted into one of the two structures, EVRC_Sta_Encoder and EVRC_Sta_Decoder.

Table 2. EVRC Encoder Context Data Structure (EVRC_Sta_Encoder)

Array/Variable	Size (Bytes)	Submodule			
		1	2	3	4
bq_xsave	36	x			
bq_ysave	72	x			
first	2		x		
pre_emp_mem	2		x		
de_emp_mem	2		x		
update_count	2		x		
hster_count	2		x		
last_update_cnt	2		x		
frame_cnt	2		x		
last_normb_shift	2		x		
ch_enrg	64		x		
ch_noise	64		x		
ch_enrg_long_db	32		x		
overlap	96		x		
window_overlap	48		x		
ConstHPspeech	320		x		
Oldlsp_nq	20		x		x
SynMemoryM	20				x
WFmemFIR	20				x
WFmemIIR	20				x
residual	660		x		x
OldlspE	20				x
Excitation	384				x
residualm	128				x
memA	20				x
memA1	20				x
memA2	20				x
LPCgain	2		x		
accshift	2			x	
pdelay	2			x	
dpm	2				x
shiftSTATE	2				x
DECbuf	80			x	
lastgoodpitch	2			x	
lastbeta	2			x	
memory	6			x	
iset	2				x
gset	4				x

Table 2. EVRC Encoder Context Data Structure (EVRC_Sta_Encoder) (Continued)

Array/Variable	Size (Bytes)	Submodule			
		1	2	3	4
Sum	6				x
Seed	2				x
buffer	384				x
a1	16				x
a2	16				x
a3	16				x
band_noise_sm	8	x			
last_rate	2	x			
last_rate_1st_stage	2	x			
last_rate_2nd_stage	2	x			
num_full_frame	2	x			
hangover	2	x			
hangover_in_progress	2	x			
signal_energy	4	x			
adaptcount	2	x			
pitchrun	2	x			
band_power_last	4	x			
snr_map	4	x			
snr_stat_once	2	x			
max_rate	2		x		
min_rate	2		x		
noise_suppression	2	x			
Total	**2670**				

Table 3. EVRC Decoder Context Data Structure (EVRC_Sta_Decoder)

Array/Variable	Size (Bytes)	Submodule 1	Submodule 2
iset	2		x
gset	4		x
FIRmem	20		x
IIRmem	20		x
OldlspD	20	x	x
SynMemory	20		x
PitchMemoryD	384		x
PtichMemoryD_back	256		x
Residual	364		x
last	2		x
Seed	2		x
Sum	6		x
PrevBest	2		x
erasureFlag	2	x	
errorFlag	2	x	
pdelayD	2	x	x
fer_counter	2	x	x
FadeScale	2		x
ave_acb_gain	2		x
ave_fcb_gain	2		x
last_valid_rate	2	x	
last_fer_flag	2	x	
post_filter	2		x
Total	**1124**		

5.2 Memory for Tables

Table 4 summarizes the tables in the EVRC codec. Tables common to both the encoder and decoder require 10920 bytes of data memory. Running encoder only requires 13924 bytes of data memory for tables. Running decoder only requires 10980 bytes of data memory for tables. Running both encoder and decoder on the same C62x chip requires 13974 bytes of data memory for tables.

Notice that the memory for tables is independent of the number of channels running on the chip.

Table 4. List of EVRC Tables

Table	Size (Bytes)	Encoder	Decoder
ppvq	16	x	x
ppvq_mid	14	x	
gnvq_4	32	x	x
gnvq_8	64	x	x
nsize8	4	x	x
lognsize8	4	x	x
nsub8	4	x	x
rnd_delay	10		x
nsize22	6	x	x
nsub22	6	x	x
lognsize22	6	x	x
nsize28	8	x	x
lognsize28	8	x	x
nsub28	8	x	x
lsptab8	320	x	x
lsptab22	3580	x	x
lsptab28	4352	x	x
Logqtb	1536	x	
Powqtbl	1536	x	x
wfac_GAMMA1	20	x	
wfac_GAMMA2	20	x	x
wfac_Gamma_4	20	x	
wfac_ALPHA	20		x
wfac_BETA	20		x
bq_w	32	x	
ch_tbl	64	x	
ch_tbl_sh	64	x	
vm_tbl	180	x	
window	208	x	

Table 4. List of EVRC Tables (Continued)

Table	Size (Bytes)	Encoder	Decoder
hamm_table	320	x	
w_long	68	x	
Table	112	x	x
Table1	272	x	x
LSTEPS	16	x	
cos129_table	258	x	x
sin129_table	258	x	x
frac	8	x	x
shft_fctr	8	x	x
frac_delay	10	x	x
shft_fctr_delay	10	x	x
A	6	x	
t_inv_sqrt	98	x	
r_filt_x2	68	x	
Ipos	22	x	
PswPCoef	6	x	
phs_tbl	256	x	
Index	16	x	
Total	**13974**	**13924**	**10980**

5.3 Memory for Local Variables and Arrays

The local variables and arrays are stored in the stacks and consume about 13924 bytes and 10984 bytes of data memory space for the encoder and decoder, respectively.

5.4 Program Memory

The codec is mixed C and assembly. The overall hand coded assembly routines, total of 15 routines, require 7744 bytes of program memory. All of these 15 routines are called in the encoder and only 5 of them, requiring 1824 bytes, are called in the decoder. With 2.10 compiler, the overall code sizes of the rest C routines of the encoder and the decoder takes 60970 bytes and 18076 bytes of program memory respectively. And the overall code size of the EVRC codec with the compiler version 2.10 release, is 81690 bytes.

Table 5 summarizes the estimated number of bytes required for the GSM EFR vocoder program memory and data memory. All of the numbers are in bytes. The *N* in the table represents the number of channels running on the same device.

Table 5. Program and Data Memory Requirements of EVRC

Function	Static RAM (Bytes)	Dynamic RAM (Bytes)	ROM Table (Bytes)	Program Memory (Bytes)
Encoder	2670 x N	4780	13924	68714
Decoder	1124 x N	1558	10984	19900
Total	**3794 x N**	**4780**	**13924**	**81690**

6 Performance Results

Table 6 gives the worst-case cycle count of one EVRC channel for all operating rates with the compiler version 2.10 release . The testing sequence is vec_01 with both noise suppression and post-filter enabled. The cycle count presented in is obtained by adding the worst-case cycle count with the version 2.00 fast simulator and 10% overhead because of possible memory bank hits. (Because the EVRC is intended to run on the C6202 with 256 Kbytes of on-chip program memory, and the EVRC code size is ~80Kbytes, the EVRC program can fit into the C6202 on-chip program memory. Therefore, there is no need to consider the extra cycle count from external program memory access.)

Table 6. Performance Estimation of EVRC With C62x DSP Compiler Version 2.1

Function	Rate 1 (MHz)	Rate 1/2 (MHz)	Rate 1/8 (MHz)
Encoder	22.30	17.04	10.65
Decoder	2.31	2.23	2.32
Total	**24.61**	**19.27**	**12.97**

The worst case for full duplex operation is encoder worst case + decoder worst case. For the current implemented codec, the worst-case cycle count is 22.30 + 2.32 = 24.64 MHz. If the codec is completely in hand-coded assembly, each EVRC channel will take no more than 12.5 MHz.

7 References

1. *Enhanced Variable Rate Codec, Speech Service Option 3 for Wideband Spread Spectrum Digital Systems*, January 1997, IA/EIA/IS-127.
2. Xiangdong Fu and Zhaohong Zhang*, A Multichannel/Algorithm Implementation on the TMS320C6000,* SPRA556.
3. Xiangdong Fu and Zhaohong Zhang*, TMS320C6000 Multichannel Vocoder Technology Demonstration Kit Host Side Design,* SPRA558.
4. Xiangdong Fu, *TMS320C6000 Multichannel Vocoder Technology Demonstration Kit Target Side Design,* SPRA560.
5. *eXpressDSP Algorithm Standard (Rules & Guidelines)*, SPRU352.
6. *eXpressDSP Algorithm Standard (API Reference)*, SPRU360.
7. Stig Torud, *Making DSP Algorithms Compliant to the eXpressDSP Algorithm Standard*, SPRA579.

8. Carl Bergman, *Using the eXpressDSP Algorithm Standard in a Static DSP System,* SPRA577.
9. Carl Bergman, *Using the eXpressDSP Algorithm Standard in a Dynamic DSP System*, SPRA580.
10. *TMS320C62x/67x Programmer's Guide*, SPRU198.