AN2261
Rev. 2, 3/2005

ITU-T G.729AB Implementation on the StarCore™ SC140/SC1400 Cores

By Corneliu Margina, Bogdan Costinescu, and Costel Ilas

This application note describes a methodology for porting and optimizing the ITU-T G.729 Recommendation with Annex A and Annex B (G.729AB) on the Freescale StarCore™-based DSPs.[1] This recommendation proposes three algorithms for reducing the transmission rate during periods of silence:

- Voice activity detection (VAD)
- Discontinuous transmission (DTX)
- Comfort noise generator (CNG)

The methodology integrates an optimized ITU-T G.729A implementation with the ITU-T G.729/Annex B Recommendation and considers an interface to the G.729AB library. The integration phase is a challenge because some optimizations from G.729A (`Levinson()`, for example) must be re-optimized according to the modifications introduced by Annex B. Of the new functions in Annex B, only six functions are optimized in C, and one is optimized using assembly language. The StarCore compiler also reduces the effort required in the optimization phase. The tests to verify the bit exactness use all available ITU test vectors, as well as an extended set of internal Freescale test vectors.

CONTENTS

1. The project was implemented according to the methodology for implementing ITU-T G.729A on the SC140 core [5]. The ITU-T G.729/Annex B Recommendation [3] was added to the optimized ITU-T G.729A implementation to maintain bit-exactness.

The ITU-T G.729AB implementation yields high performance on the SC140 core—61 channels on a 300 MHz SC140 core using only 202 KB of memory. These results demonstrate the power of the SC140 core for wireless infrastructure products.

1 G.729AB Background

G.729AB is an 8kbit/s Conjugate-Structure Algebraic-Code-Excited Linear-Prediction (CS-ACELP) speech codec employed in simultaneous voice and data applications. The Annex B of the ITU-T G.729 Recommendation proposes a silence compression scheme for terminals to be used with ITU-T G.729 or ITU-T G.729A. It contains three algorithms for reducing the transmission rate during periods of silence, as follows:

- *Voice Activity Detection*. Indicates the presence or absence of voice, thus activating the vocoder to code/decode active voice frames. This algorithm makes a decision on voice activity every 10 ms in correlation with the frame size of the G.729/G.729A vocoder. The output of the VAD module is either a 1 or 0, indicating the presence or absence of voice activity. The decision is based on a set of difference parameters: the full band energy, the low band energy, the zero-crossing rate and a spectral measure. If the output of a VAD module is 1, the speech codec is invoked to code/decode active voice frames (speech). If the output is 0, the DTX/CNG algorithms are used to code/decode non-active voice frames (silence).
- *Discontinuous Transmission*. Receives the active/non-active voice information from the VAD module and sends a set of non-active voice update parameters to the speech decoder by measuring the changes in the non-active voice signal. The update decision is based on the absolute and adaptive thresholds on the frame energy and the spectral distortion measure. If the update is required, the encoder sends information to generate a signal similar to the original non-active voice signal. This information comprises an energy level and a description of the spectral envelope. If no update is required, the decoder generates the signal on the basis of the last received energy level and the spectral shape information of the non-active voice frame. The decision output of the DTX module is as follows:
  - 0 untransmitted frame
  - 1 active speech frame
  - 2 Silence Insertion Descriptor (SID) frame
- *Comfort Noise Generator*. At the decoder, reproduces non-active voice frames based on the decisions of the VAD (output 0) and DTX modules at the encoder. The comfort noise is generated by introducing a pseudo-white excitation signal of a controlled level into interpolated LPC filters, in the same way that the decoder produces active speech by filtering the decoded excitation. The excitation level and LPC filters are obtained from the previous SID information.

2 Implementation Process

The process of porting and optimizing the ITU-T G.729 with integrated Annex A and Annex B Recommendations references the methodology used in the implementations of ITU-T G.729 [4] and ITU-T G.729A [5] Recommendations. **Table 1** summarizes the main phases of this process.

Table 1. Implementation Phases

Implementation Phase	Description
Porting ITU-T G.729AB to the SC140 core by integrating it into ITU-T G.729A optimized implementation	Adding new functions, channel data, and tables with respect to the changed functions and tables from G.729A. Data type definitions and introduction of pragmas and intrinsic functions (Section 2.1, “Differences Between ITU-T G.729A and ITU-T G.729AB” on page 4).
Project-level optimizations	Function inlining, data alignment, channel data transformations (Section 2.2, “Project-Level Optimizations” on page 8).
Algorithm changes	Changing algorithms based on better platform usage (Section 2.3, “Function-Level Optimizations” on page 9).
Function-level C optimizations	C optimization techniques (multisample, loop unroll, split summation, loop merging, code factorization) with a better use of intrinsic functions to increase performance (Section 2.3, “Function-Level Optimizations” on page 9).
Function-level assembly implementations and optimizations	Implementing selected function using assembly language in order to increase performance (Section 2.3, “Function-Level Optimizations” on page 9).

The following sections describe these implementation phases, indicating the performance results. **Table 2** summarizes the requirements of the ITU-T G.729AB implementation.

Table 2. ITU-T G.729AB Implementation Requirements

Requirement	Size (Encoder + Decoder)
Code Size	40 KB
Data ROM (Tables)	6 KB
Data RAM - Channel	$3 \times N^1$ KB
Data RAM - Stack	3 KB
Peak Performance (speed)	5.5 MCPS
Notes: 1. N = number of channels	

These requirements are based on the results obtained for the ITU-T G.729A optimized implementation on the SC140 core, presented in **Table 3**.

Table 3. G.729A Optimized Implementation Results

Type of Implementation	Speed (MCPS)	ROM		RAM	
		Program (KB)	Tables (Bytes)	Channel Data (Bytes)	Stack (Bytes)
C only	6.81	32.80	4736	2240 x N^1	2312
C and assembly	4.7	28.09	4736	2240 x N	2312
Notes: 1. N = number of channels					

After all implementation phases, a testing phase verified and ensured the bit exactness and measured the speed (MCPS) and code size (bytes) of the project. Section 2.1.4, “Verifying Bit Exactness Using Test Vectors” on page 6 describes the test vectors. The tests were performed on both the StarCore SC140 simulator and the MSC8101ADS board using the Metrowerks® CodeWarrior® for StarCore integrated development environment (IDE), Release 1.1.

2.1 Differences Between ITU-T G.729A and ITU-T G.729AB

The differences between Annex B of the ITU-T G.729 Recommendation and the ITU-T G.729A Recommendation are determined by comparing the reference C code from the ITU-T. The result consists of added channel data, functions, and tables for implementing Annex B. However, some data and functions must be changed, as summarized in the following sections. The comparison of the reference code is helpful in porting and integrating Annex B with the ITU-T G.729A optimized implementation.The compiler for implementing and porting ITU-T G.729AB is Metrowerks® CodeWarrior® for StarCore, Release 1.1.

Porting Annex B to the SC140 core required use of the following elements:

- Defined data types (for example, Word16, Word32) as defined in ITU-T G.729A to comply with the SC140 architecture.
- Intrinsic functions defined by the compiler instead of their emulated versions.
- Overflow handling intrinsics instead of the boolean overflow and carry flags at the base of the emulated functions.

Integrating Annex B with the ITU-T G.729A optimized implementation requires the following steps:

1. Integrate the new functions in the ITU-T G.729A optimized implementation by adding the calls and code of these functions (**Section 2.1.1**). Split these functions into files, one function per file, thus using the code structure of the ITU-T G.729A optimized implementation.
2. Add the modified code corresponding to the changed functions as they are implemented in the C reference code, optimizing them in the next project implementation phase (**Section 2.2**).
3. Add/modify the data structures of the channel data with the new global and static variables from each new/changed added function (**Section 2.1.2**);
4. Add the new Annex B-defined tables (**Section 2.1.3**).
5. Make the G.729AB library interface comply with the Application Binary Interface (ABI) as described in **Section 2.1.5**, and modify the initialization functions of the structures accordingly.

Table 4 summarizes the results of the SC140 porting phase of the ITU-T G.729AB. The results were obtained using only C versions of the functions:

- C optimized versions of the unchanged functions from ITU-T G.729A optimized implementation.
- C reference code of the functions implemented or modified by Annex B.

Table 4. Performance After Porting to the SC140 Core

Speed (MCPS)	ROM		RAM	
	Program (KB)	Tables (Bytes)	Channel Data (Bytes)	Stack (Bytes)
9.54	44.78	5640	2544 x N[1]	3104
Notes: 1. N = number of channels				

2.1.1 Integrating the New Functions

The C reference code of the ITU-T G.729A and ITU-T G.729AB recommendations was compared at the function-level. **Table** summarizes the results of the comparison. This analysis was used for integrating the new functions in the ITU-T G.729A optimized implementation. The functions were integrated in conjunction with the modification

of the channel data and data tables. The new functions implement three algorithms of the Annex B (VAD, DTX, and CNG) and the quantization algorithms for computing the SID. As **Table** shows, only a few functions differ, and most are identical. The differences consist of new added computations:

- `Autocorr()` computes an exponent of the autocorrelation vector used by the VAD algorithm.
- `Levinson()` computes a residual energy used by the DTX algorithm.
- `Post_Filter()` calls the function that performs the harmonic postfilter based on the input VAD decision.
- `Random()` returns a pointer to the generated seed.

Table 5. Differences Between ITU-T G.729A and ITU-T G.729AB Functions

Encoder Functions		Common Functions		Decoder Functions	
Function Name	**Difference**	**Function Name**	**Difference**	**Function Name**	**Difference**
Autocorr	few diff.	Random	few diff.	Post_Filter	few diff.
Levinson	few diff.	Calc_exc_rand	new	Dec_cng	new
Calc_pastfilt	new	Gauss	new	Init_Dec_cng	new
Calc_RCoeff	new	Init_lsfq_noise	new	sid_lsfq_decode	new
Calc_sum_acf	new	Qua_Sidgain	new	Get_decfreq_prev	new
Cmp_filt	new	Quant_Energy	new	Update_decfreq_prev	new
Cod_cng	new	Sqrt	new	D_lsp	same
Get_freq_prev	new	Copy	same	agc	same
Init_Cod_cng	new	Gain_predict	same	Dec_gain	same
lsfq_noise	new	Gain_update	same	Dec_lag3	same
MakeDec	new	Get_lsp_pol	same	Decod_ACELP	same
New_ML_search_1	new	Int_qlpc	same	Gain_update_erasure	same
New_ML_search_2	new	Inv_sqrt	same	Init_Post_Filter	same
Qnt_e	new	Log2	same	Lsp_decw_reset	same
Update_cng	new	Lsf_lsp2	same	Lsp_iqua_cs	same
Update_freq_prev	new	Lsp_Az	same	pit_pst_filt	same
Update_sumAcf	new	Lsp_expand_1_2	same	Post_Process	same
vad	new	Lsp_get_quant	same	preemphasis	same
vad_init	new	Lsp_prev_compose	same		
ACELP_Code_A	same	Lsp_prev_extract	same		
Az_lsp	same	Lsp_prev_update	same		
Chebps_10	same	Lsp_stability	same		
Chebps_11	same	Pow2	same		
Cor_h	same	Pred_lt_3	same		
Cor_h_X	same	Residu	same		
Corr_xy2	same	Set_zero	same		
D4i40_17_fast	same	Syn_filt	same		
Dot_Product	same	Weight_Az	same		
Enc_lag3	same				
G_pitch	same				
Gbk_presel	same				
Get_wegt	same				
Lag_window	same				
Lsp_encw_reset	same				
Lsp_expand_1	same				
Lsp_expand_2	same				
Lsp_get_tdist	same				
Lsp_last_select	same				
Lsp_lsf	same				
Lsp_lsf2	same				
Lsp_pre_select	same				

Table 5. Differences Between ITU-T G.729A and ITU-T G.729AB Functions (Continued)

Encoder Functions		Common Functions		Decoder Functions	
Function Name	**Difference**	**Function Name**	**Difference**	**Function Name**	**Difference**
Lsp_qua_cs	same				
Lsp_select_1	same				
Lsp_select_2	same				
Parity_Pitch	same				
Pitch_fr3_fast	same				
Pitch_ol_fast	same				
Pre_Process	same				
Qua_gain	same				
Qua_lsp	same				
Relspwed	same				
test_err	same				
update_exc_err	same				

2.1.2 Modifying the Channel Data

The modified channel data consists of global data used by both the encoder and decoder of the speech codec to perform the following tasks:

- Decode the SID information
- Make the DTX decision
- Make the VAD decision
- Generate the comfort noise (CNG algorithm)

The changed functions necessitated additions of data to the global data for Linear Prediction Coding (LPC), Linear Spectral Pair (LSP), and LSP quantization.

2.1.3 Adding the Annex B-Defined Tables

Table 6 presents the changes for some data tables from ITU-T G.729A required to integrate the ITU-T G.729 Annex B Recommendation.

Table 6. Data Table Modifications

Table Name	Comment
Word32 lag[]	Two values added (8 bytes added)
Word16 table2[]	Doubled size, thus removing the optimization from ITU-T G.729A optimized implementation (64 bytes added)
Word16 slope[]	Doubled size, thus removing the optimization from ITU-T G.729A optimized implementation (64 bytes added)
Word16 freq_prev_reset[]	New table (20 bytes added)

2.1.4 Verifying Bit Exactness Using Test Vectors

ITU provides a set of test vectors to verify the bit exactness of the encoder and decoder in the G.729AB vocoder (see **Table 7**). Other internal Freescale test vectors are also used. In addition, the ITU test vectors for the ITU-T G.729A optimized implementation maintain the bit exactness of the ITU-T G.729A Recommendation when the VAD is disabled.[2]

Table 7. ITU-T G.729AB Test Vectors

Encoder Input	Encoder Output Decoder Input	Decoder Output
Tstseq1.bin	Tstseq1.bit	Tstseq1.out
Tstseq2.bin	Tstseq2.bit	Tstseq2.out
Tstseq3.bin	Tstseq3.bit	Tstseq3.out
Tstseq4.bin	Tstseq4.bit	Tstseq4.out
	Tstseq5.bit	Tstseq5.out
	Tstseq6.bit	Tstseq6.out

2.1.5 Making the Library Interface ABI-Compliant

The G.729AB library must comply with the Application Binary Interface as specified in [4] and [5]. **Example 1** presents the ABI-compliant C functions of the vocoder external interface.

Example 1. G.79AB Library Interface

```
void MDCR_G729AB_encode_initialize(MDCR_G729AB_ENCODER_CHANNEL_INFO_T *enc_info);
Word16 MDCR_G729AB_enableVAD(MDCR_G729AB_ENCODER_CHANNEL_INFO_T *enc_info,
                             IN Word16 flag);
Word16 MDCR_G729AB_getVADStatus(MDCR_G729AB_ENCODER_CHANNEL_INFO_T *enc_info);

void MDCR_G729AB_encode(Word16 *signal,
                        Word16 *prm,
                        MDCR_G729AB_ENCODER_CHANNEL_INFO_T *enc_info);
void MDCR_G729AB_decode_initialize(MDCR_G729AB_DECODER_CHANNEL_INFO_T *dec_info);
void MDCR_G729AB_decode(Word16 *prm,
                        Word16 *synth,
                        MDCR_G729AB_DECODER_CHANNEL_INFO_T *dec_info);
```

The functions are as follows:

- `MDCR_G729AB_encode_initialize()`. Initializes the encoder channel data structures. By default, the encoder behavior is set to enable the VAD/DTX functionality described in G.729 Annex B.
- `MDCR_G729AB_enableVAD()`. Enables/disables the VAD part of the encoder. If the least significant bit of the second parameter is set to 1, the VAD is enabled. The function returns the old status of the VAD subsystem.
- `MDCR_G729AB_getVADStatus()`. Returns the current status of the VAD subsystem. Possible return values are `MDCR_G729AB_VAD_DISABLED` (VAD is disabled) and `MDCR_G729AB_VAD_ENABLED` (VAD is enabled).
- `MDCR_G729AB_encode()`. Using either G729 Annex A or G729 Annexes A plus B, this function encodes the input speech samples in the signal array. The encoder output is provided in the `prm[]` vector. The length of the data depends on the input speech type: active speech or silence. The channel history and data are provided in the enc_info data structure.
- `MDCR_G729AB_decode_initialize()`. Initializes the decoder channel data structures.

2. These test vectors are described in [5].

- `MDCR_G729AB_decode()`. To conform with the G729AB, this function decodes the input frame parameters in the prm array. The 160-byte decoder output (synthesized data) is provided in the `synth[]` vector. The channel history and data are in the dec_info data structure.

2.2 Project-Level Optimizations

The global project optimizations are built upon the ITU-T G.729A optimized implementation, considerably reducing the work required. The optimizations described in this section apply only to the new and changed functions integrated from Annex B, as follows:

- Data Precision Format (DPF) operations to speed up the 32-bit operations because the StarCore C compiler provides efficient support for the DPF operations based on intrinsic functions [6].
- Alignment of the new data structures added to the channel data.
- Alignment of some vectors and data tables of the new and changed functions to benefit from the compiler optimizations when `-O3` is used.
- Use of assembly versions of the functions from the ITU-T G.729A optimized implementation that were unaffected by the integration phase, thus gaining speed and minimizing code size.

The following assembly functions were integrated from the ITU-T G.729A optimized implementation, thus improving speed and program size:

- Get_lsp_pol()
- Pred_lt_3()
- Residu()
- Syn_filt()
- Post_Process()
- Autocorr()
- Az_lsp()
- Chebps()
- Cor_h()
- D4i40_17_fast()
- Dot_Product()
- Levinson()
- Pitch_fr3_fast()
- Pitch_ol_fast()
- Pre_Process()
- Qua_gain()

The function changes introduced by Annex B apply to the assembly versions of the `Autocorr()` and `Levinson()` functions. The assembly version of `Levinson()` is based on [6]. **Table 8** summarizes performance results after application of the global project optimizations. The processing load value of 7.73 MCPS is the worst case introduced by Annex B.

Table 8. Performance Results After Project-Level Optimizations

Speed (MCPS)	ROM		RAM	
	Program (KB)	Tables (Bytes)	Channel Data (Bytes)	Stack (Bytes)
7.73	40.52	5740	2544 x N[1]	2824
Notes: 1. N = number of channels				

2.3 Function-Level Optimizations

The algorithmic changes applied to the functions ported from Annex B use both C and assembly language. There are two types of C code transformations:

- *Platform-independent.* To avoid repeated computations or fetches of the same value and reduce the number of tests.
- *Platform-dependent.* To perform non-dependent computation in parallel and reorder vectors to use packed moves.

A few optimizations were made to a reduced number of functions from Annex B after a profiling session. Six functions were optimized using C language, and only one function required optimization using assembly language. Thus, the compiler proves to deliver highly-optimized generated code. The optimizations increased performance.

The C optimization techniques are as follows:

- Multisampling
- Loop merging
- Loop unrolling
- Loop splitting
- Split summation

These techniques were applied to the following functions:

- Calc_exc_rand()
- Gauss()
- Random()
- New_ml_search_1()
- New_ml_search_2()
- Lsp_lsf()

Several descendants of `Calc_exc_rand()` were optimized. The result of combining the `Gauss()` and `Random()` functions is the `Generate_Gauss_Exc()` function, which is used by `Calc_exc_rand()`. This result was optimized using assembly to increase speed. Also, the loops used for normalization were extracted into separate functions, `normalize_1()` and `normalize_2()`, that were optimized in C to increase speed. **Table 9** summarizes the results of optimizing the `Calc_exc_rand()` function.

Table 9. Performance Results for the Calc_exc_rand() Optimization Phases

C Version	Number of Cycles	Code Size (Bytes)
ITU-T Reference Code	22495	2168
Optimized Code + Generate_Gauss_Exc() (C optimized version)	7946	2442
Optimized Code + Generate_Gauss_Exc() (assembly version)	6822	2442
Optimized Code + Generate_Gauss_Exc() (assembly version) + normalize_1() and normalize_2() (C optimized code)	4684	2218

Special optimization techniques were applied to `New_ml_search_1()` and `New_ml_search_2()` to reduce the number of cycles. The goal was to optimize the loop that finds the index of the minimum value in a vector of 16-bit values. The optimization consisted of creating a 32-bit array that contains the 16-bit value on the high part and its index in the low part. Using the split-summation technique, four minimums and their indexes are computed at each loop iteration. Using this technique in conjunction with the C optimization techniques improved the results, as shown in **Table 10**.

Table 10. Performance Results for New_ml_search1() and New_ml_search2()

Function Name	C Version	Number of Cycles	Code size (Bytes)
New_ml_search_1()	ITU-T Reference Code	3063	666
	Optimized Code	980	748
New_ml_search_2()	ITU-T Reference Code	4040	1126
	Optimized Code	2800	1062

3 Results

Table 11 summarizes the performance results of the ITU-T G.729AB optimized implementation on the StarCore SC140 core. A profiling session after a small number of functions from Annex B were optimized demonstrated that the performance targets are more attainable under the following conditions:

- All C optimized functions are compiled for speed (`-O3` compiler flag) and the rest for size (`-Os -O3` compiler flag), thus obtaining smaller code with only a small decrease in the speed.
- All C functions are compiled for speed, thus obtaining a higher speed with only a small increase in code size.

As **Table 11** shows, the values for the C-only implementations differ significantly from the implementations using assembly versions of the functions. The assembly versions of the functions from the ITU-T G.729A optimized implementation provided a solid basis for performance increases. **Figure 1** shows the effort (in man-months) to meet the project targets.

Table 11. Performance Results of the G.729AB Optimized Implementation for the SC140 Core

Type of Optimizations	Speed (MCPS)	ROM		RAM (KB)	
		Program (KB)	Tables (KB)	Channel Data (Bytes)	Stack (Bytes)
(C and assembly project implementation) 85 functions compiled for speed 17 assembly functions	4.89	40.82	5.6	2544 x N[1]	2824
(C and assembly project implementation) 19 functions compiled for speed 66 functions compiled for size 17 assembly functions	5.12	37.24	5.6	2544 x N	2824
(C-only project implementation) 102 functions compiled for speed	6.62	45	5.6	2544 x N	2824
(C-only project implementation) 36 functions compiled for speed 66 functions compiled for size	6.85	41.44	5.6	2544 x N	2816
Notes: 1. N = number of channels					

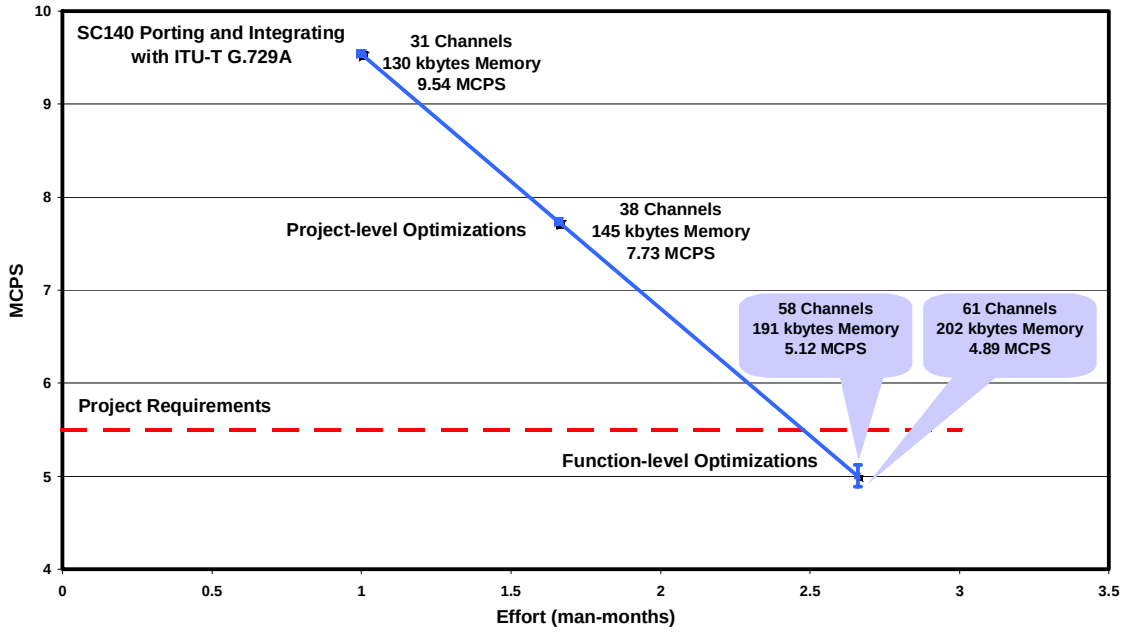

Figure 1. Processing Load Versus Implementation Effort

4 References

[1] ITU-T Recommendation G.729—*Coding of Speech at 8 kbit/s Using Conjugate-Structure Algebraic-Code-Excited Linear-Prediction (CS-ACELP)*, March 1996.

[2] ITU-T Recommendation G.729/Annex A—*Reduced Complexity 8 kbit/s CS-ACELP Speech Codec*, November 1996.

[3] ITU-T Recommendation G.729/Annex B— *A Silence Compression Scheme for G.729 Optimized for Terminals Conforming to Recommendation V.70*, November 1996.

[4] *ITU-T G.729 Implementation on StarCore SC140*, Freescale, AN2094/D.

[5] *ITU-T G.729A Implementation on StarCore SC140*, Freescale, AN2151/D.

[6] *Implementing the Levinson-Durbin Algorithm on the SC140*, Freescale, AN2197/D.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations not listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GMBH
Technical Information Center
Schatzbogen 7
81829 München, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
+800 2666 8080

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Order No.: AN2261
Rev. 2
3/2005