Application Report
SPRA656 - March 2000

Implementation of G.729 on the TMS320C54x

Lim Hong Swee

Texas Instruments Singapore (Pte) Ltd
Marketing & Sales

ABSTRACT

The main objective of this project is to implement the ITU-U G.729 8-kbit/s Vocoder (Voice Coder). The ANSI C code is available from ITU but it is not suitable for implementation in real time using a digital signal processor (DSP) due to the large computational time required for the encoder and decoder. The main task of this project is to study and understand the ITU-U G.729 standard and convert the ANSI C codes into TMS320C54x assembly language for real-time implementation.

The implemented vocoder should be tested using real-time speech signals captured from an analog-to-digital converter (ADC). This project uses the TMS320C54x evaluation module (EVM) board as the development platform.

THis application report is structured as follows: section 1 gives a background of speech coding, section 2 gives an overview of the ITU-U G.729 standard, section 3 discusses software design and conversion to assembly language, and section 4 gives the result of the real-time testing and problems encountered.

Contents

List of Figures

List of Tables

1 Properties of Speech Signal

Traditionally, speech has always been transmitted over a twisted pair with a bandwidth of about 3.4 kHz. In order to digitize speech, the analog waveform is first sampled at the Nyquist rate of 8 kHz, quantized and companded to either 8- or 16-bit pulse code modulation (PCM). Hence, a raw uncoded speech can be represented using a bit rate of 128 Kbit/s (16-bit PCM multiplied by 8000 samples/s), which is an extremely high bit rate. Specifically, for a signal with a bandwidth of 3.4 kHz and a signal-to-noise ratio (SNR) of 30 dB (which corresponds to very good speech quality, i.e., toll quality), and assuming an additive white Gaussian noise (AWGN) channel, the bit rate *C*, based on Shannon's Channel Capacity Theorem is given by:

$$C = W \log_2\left[1 + \frac{P}{G}\right]$$

Where W is the bandwidth and *P/G* is the SNR. The equation above shows that the bit rate required to represent speech with a small error is about 34 kbit/s. Although this a reduction of a factor of 3, it is still too high for modern telecommunication systems. The Shannon limit of 34 kbit/s is in fact an upper bound since it does not take into account the redundancies in speech signals. By using techniques such as linear prediction, the long- and short-term redundancies in speech may be recovered to yield an even low bit rate.

It is important to have a good understanding of speech signal properties in order to understand the operation of the speech coder. Speech properties have to do with the time and frequency domain characteristics of speech as well as the factor of human perception.

The time domain characteristics of speech can be broadly characterized into unvoiced and voiced segments. Unvoiced segments are aperiodic and have a noise-like appearance. A good example of unvoiced speech is the enunciation of the silent ‘f’. The fact that unvoiced segments are noise-like in their appearance suggests that we can replace it with say a Gaussian noise source and the human hear will not be able to perceive the difference. This fact is made use of in the decoder in the event of frame erasure. Voiced segments, on the other hand, are found to be periodic and have short- and long-term correlations. An example of voiced speech is the enunciation of vowel sounds. Short-term correlation implies redundancies between adjacent samples, while long-term correlation suggests periodicity, i.e., similarity between sequential cycles of samples. Both forms of redundancies may be removed using a short- and long-term predictor respectively. Figure 1 shows a speech production model for vocoders.

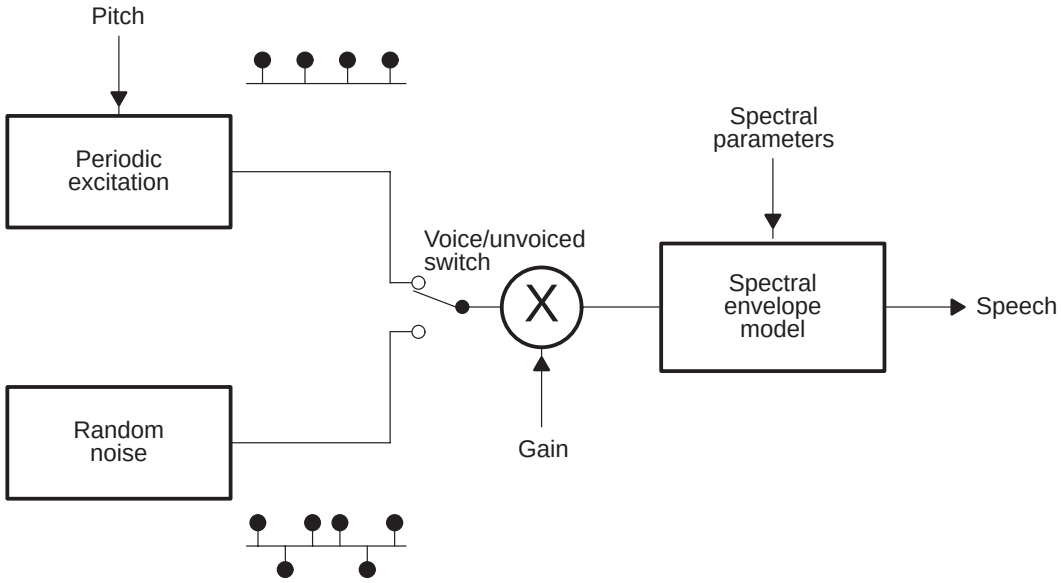


Figure 1. Speech Production Model for Vocoders

The frequency domain characteristics of speech lie in regions called formants or formant frequencies. Formant corresponds to the resonant frequencies of the vocal tract and is usually below 4 kHz. It is the region where most of the speech energy is concentrated. The Formant Vocoder is formed by encoding the locations of the formants. This information can also be used by the decoder (synthesizer) for postprocessing to enhance the formant of the synthesized speech.

The final important characteristic of speech has to do with human perception. The human ear, as it turns out, is susceptible to what is known as spectral masking. This means that a signal can be made inaudible to the human ear if it is masked out by another signal of higher energy and is in the same frequency range.

1.1 Attributes of a Speech Coder

Speech coders have several attributes, some of which are conflicting. Speech coders are usually optimized along some of these attributes according to the application needs. The main attributes of a speech coder are as follows:

1.1.1 Bit Rate

Bit-rate reduction is the primary motivation for speech coding. Most existing standards, including the G.729 (8 kbit/s), specify fixed-rate coders. The required bit rates are often dependent on the communication channel and the intended application. For example, for satellite and cellular phones, bit rates ranges from 3.3 to 13 kbit/s, and for general telephone network, the bit rates in use are 16 kbit/s and above.

1.1.2 Speech Quality

Speech quality is a vital attribute of speech coders. The main difficulty is in finding an objective criterion that correlates well for a variety of speech coders and input signals. Subjective tests such as the Mean Opinion Score (MOS), where different groups of trained and untrained listeners are asked to characterize each set of utterances on a scale of 1 (unacceptable quality) to 5 (excellent quality). An MOS of 4 or higher defines “toll” quality which means that the reconstructed speech is indistinguishable from the original speech.

1.1.3 Delay

Delay is important especially for real-time full-duplex communications. The delay threshold is dependent on the nature of the application. For example, for highly interactive conversations, delay above 150 ms may be perceived as an impairment. On the other hand, for normal conversation, a delay of 400 to 500 ms may be tolerated without significant reduction in overall performance. However, it must be noted that in a system without echo cancellers, the delay threshold can be as low as 100 ms.

Coder delay is often divided into four components. The first is the algorithm delay, which is delay due to the accumulation of one frame of speech plus any look ahead (G.729 has an algorithm delay of 15 ms). The second is the transmission delay, which is incurred in transmitting the encoded parameters of the particular frame. The third delay is due to multiplexing, which is incurred in cases of multiple-access channel. The final delay is the computational delay due to the actual processing of the speech frame. This delay is incurred by the encoder and decoder.

1.1.4 Channel Error Sensitivity (Robustness)

Channel errors are divided into two main types. The first is random errors, which are usually due to channel noise. This is normally specified as bit error rate (BER) and is limited to about 1%. To counter random errors, a sufficient signal-to-noise ratio must be achieved. In addition, channel coding, which is different from speech coding (source coding), is performed. Channel coding is often implemented by adding redundancies to the transmitted information to make it more robust against channel error. The drawback, however, is the additional overhead incurred due to the redundancies added. The second type of error is burst error, which is more common in mobile channels and arises due to mechanism such as fading. To guard against such error, error detection schemes are implemented. In G.729, a parity bit is inserted in the encoded parameter (80 bits) for error detection.

2 ITU-U G.729 Recommendation

This section attempts to explain how the G.729 vocoder works as stated in the ITU standard. The ITU-U G.729 recommendation contains the description of an algorithm for the coding of speech signals at 8 kbit/s using Conjugate-Structure Algebraic-Code Excited Linear Prediction (CS-ACELP). The CS-ACELP coder is designed to operate with a digital signal by first performing telephone bandwidth filtering (Recommendation G.712) of the analog signal, then sampling it at 8000 Hz, followed by conversion to 16-bit linear PCM for input to the encoder. The output of the decoder is converted back into analog signal by similar means.

2.1 General Description of the Coder

The G.729 vocoder is based on the Code-Excited Linear-Prediction (CELP) model. The coder operates on a speech frame of 10 ms which corresponds to 80 samples at a sampling rate of 8000 samples per second. For every 10-ms frame, the speech signal is analyzed to extract the parameters of the CELP parameters. They are the Linear-Prediction Filter coefficients, adaptive and fixed-codebook indices and gains. These parameters are encoded and transmitted. The bit allocation of the coder parameters is shown in Table 1. At the decoder, these parameters are used to retrieve the excitation and synthesis filter parameters. Speech is reconstructed by filtering this excitation through the short-term synthesis filter shown in Figure 2. The short-term synthesis filter is based on a tenth-order Linear Prediction filter. The long-term synthesis filter is implemented using the so-called adaptive-codebook approach. After computing the reconstructed speech, it is further enhanced by a post-filter.

Table 1. Bit Allocation of the G.729 CS-ACELP 8-kbit/s Vocoder (10-ms frame)

Parameter	Codeword	Subframe1	Subframe 2	Total per frame
Line Spectrum pairs	*L0,L1,L2,L3*	–	–	18
Adaptive-Codebook Delay	*P1,P2*	8	5	13
Pitch-Delay Parity	*P0*	1	–	1
Fixed-Codebook Index	*C1,C2*	13	13	26
Fixed-Codebook Sign	*S1.S2*	4	4	8
Codebook gains (stage 1)	*GA1, GA2*	3	3	6
Codebook gains (stage 2)	*GB1, GB2*	4	4	8
Total				**80**

Note that the 10-ms speech frame is subdivided into two 5-ms subframes. All the parameters except line spectrum pairs (LSPs) are encoded for each subframe. The reason for splitting into two subframes is to prevent the spectral transition from one frame to another from being abrupt and hence improving the speech quality.

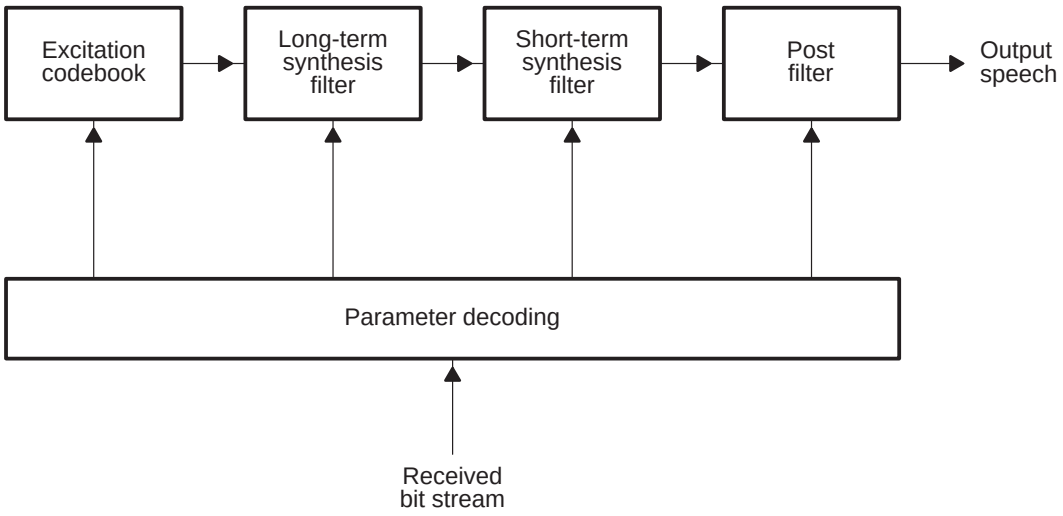


Figure 2. Block Diagram of Conceptual CELP Synthesis Model

2.2 G.729 Encoder

The encoding principle is shown in Figure 3. The input signal is high-pass filtered and scaled in the preprocessing block. The preprocessed signal serves as the input for the subsequent analysis. Linear Prediction is done once in a 10-ms frame to compute the linear prediction (LP) coefficients. These coefficients are converted to LSP and quantized (18 bits) using predictive two-stage Vector Quantization (VQ) . The excitation signal is chosen by using an analysis-by-synthesis search procedure in which the error the original speech and reconstructed speech is minimized according to a perceptually weighted distortion measure. This is done by filtering the error signal with a perceptual weighting filter, whose coefficients are derived from the unquantized LP filter coefficients. The amount of perceptual weighting is made adaptive to improve the performance for input signals with a flat frequency-response.

The excitation parameters (fixed- and adaptive-codebook parameters) are determined per subframe of 5 ms (40 samples) each. The quantized and unquantized LP filter coefficients are used for the second subframe (see Figure 3), while in the first subframe, interpolated LP filter coefficients (both quantized and unquantized) are used. An open-loop pitch delay is estimated once per 10-ms frame based on the perceptually weighted speech signal *sw(n)*. Then, the following operations are repeated for each subframe. The target signal *x(n)* is computed filtering the LP residual through the weighted synthesis filter *W(z)/A(z)*. Next, the impulse response *h(n)* of the weighted synthesis filter is computed. Closed-loop pitch search is then done to find the adaptive codebook delay and gain, using target signal *x(n)* and impulse signal *h(n),* by searching around the value of the open-loop pitch delay. A fractional pitch with a resolution of 1/3 is used. The pitch is encoded with 8 bits for the first subframe and differentially with 5 bit for the second subframe. The target signal *x(n*) is then updated before being used in the fixed-codebook search to find the optimum excitation. An algebraic codebook with 17 bits is used for the fix-codebook excitation. With this, the gains of the adaptive codebook and fixed codebook are vector quantized using 7 bits, (with moving average prediction applied to the fixed-codebook gain). Finally, the filter memories are updated using the determined excitation signal. Figure 4 shows the overall encoding steps in flowchart form.

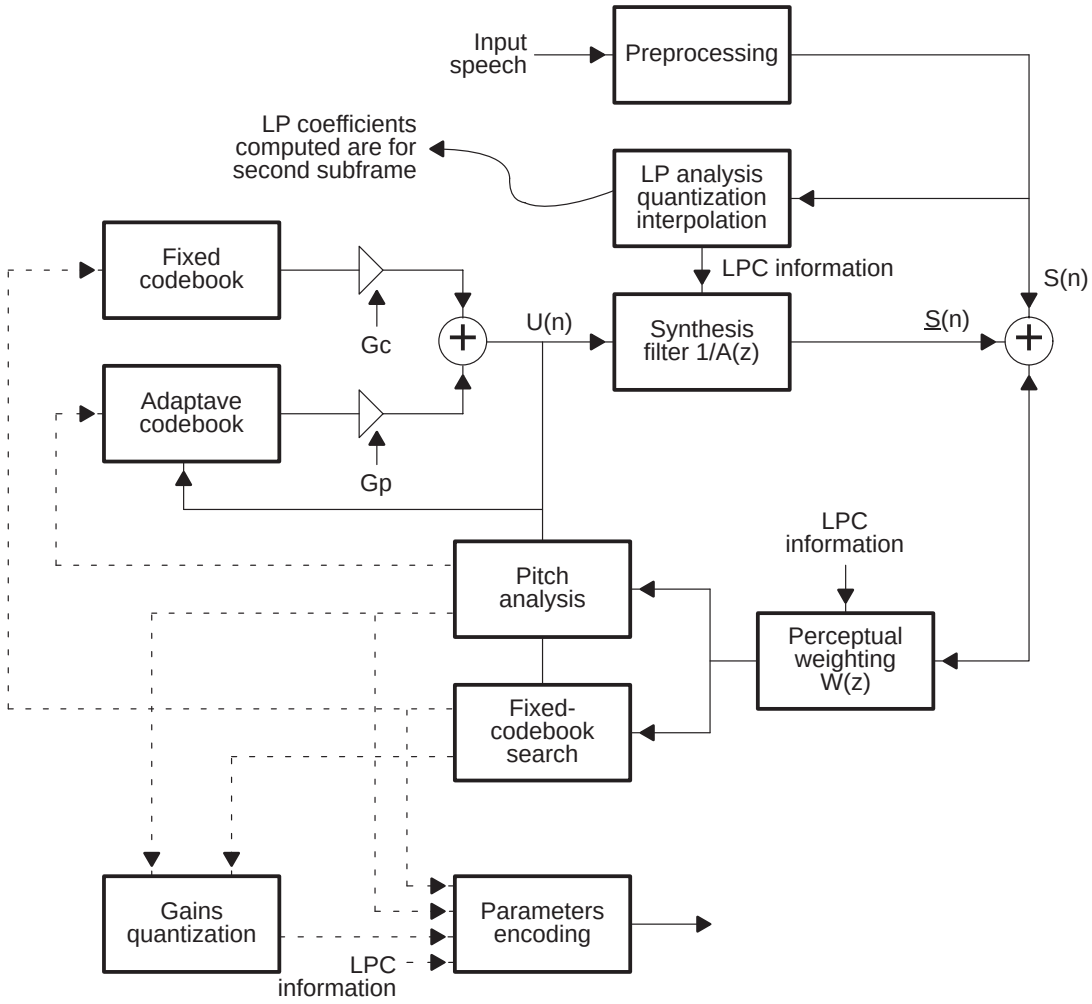


Figure 3. Encoding Principle of the CS-ACELP Encoder

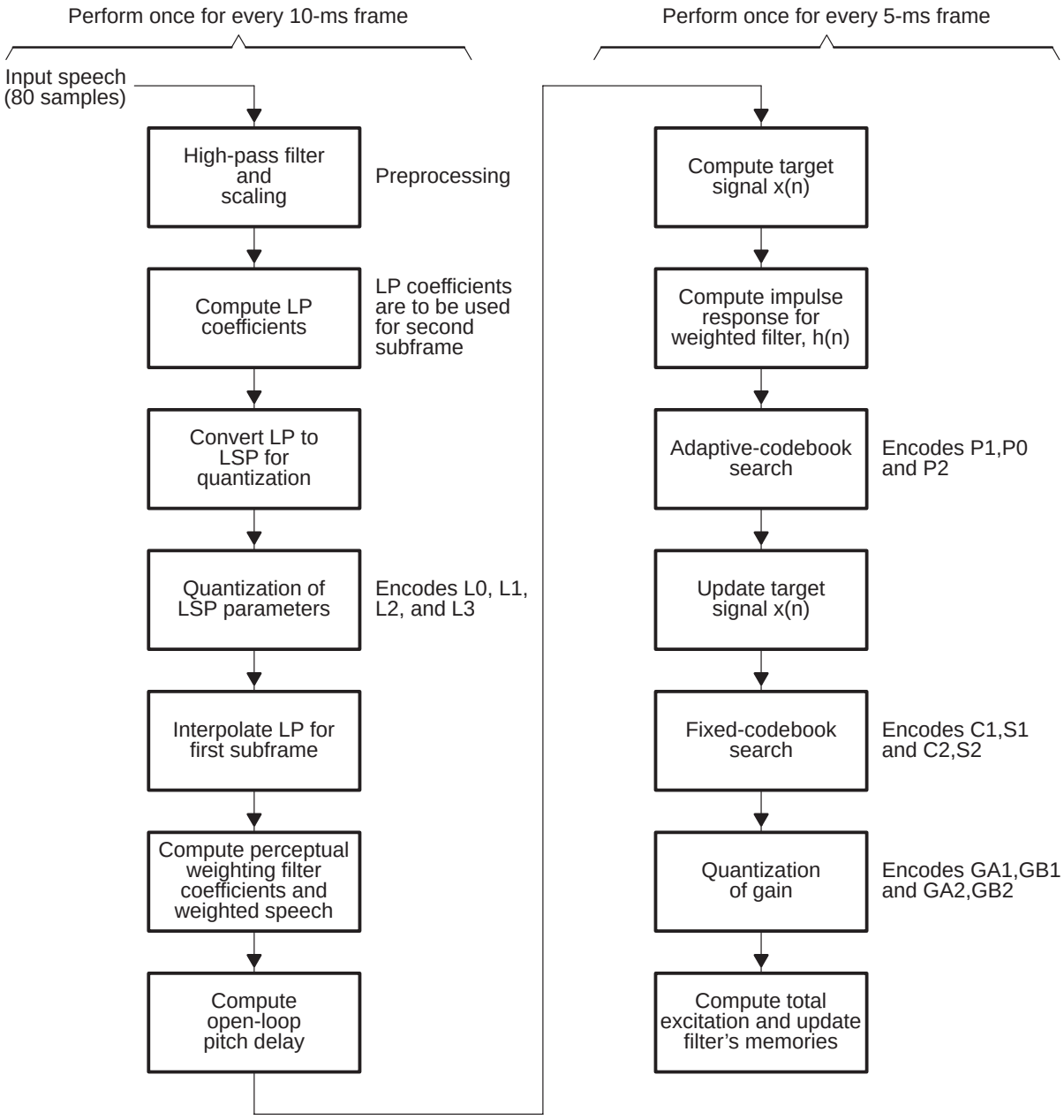


Figure 4. Flowchart of G.729 Encoder Routine

2.3 G.729 Decoder

The decoder principle is shown in Figure 5. First the parameter's indices are extracted from the received bitstream. These indices are decoded to obtain the coder parameters corresponding to a 10-ms frame. These parameters are the LSP coefficients, the two fractional pitch delays, the two fixed-codebook vectors, and the two sets of adaptive-codebook and fixed-codebook gains. The LSPs are interpolated and converted to LP coefficients for each subframe. For every 5-ms subframe, the following operations are repeated. First, the excitation is constructed by adding the adaptive and fixed-codebook vectors and scaling by their respective gains. Next, the speech is reconstructed by filtering the constructed excitation signal through the LP synthesis filter. Finally, the reconstructed speech is passed through a postprocessing stage, which includes an adaptive postfilter based on long- and short-term synthesis filters, followed by a high-pass filter and scaling operation.

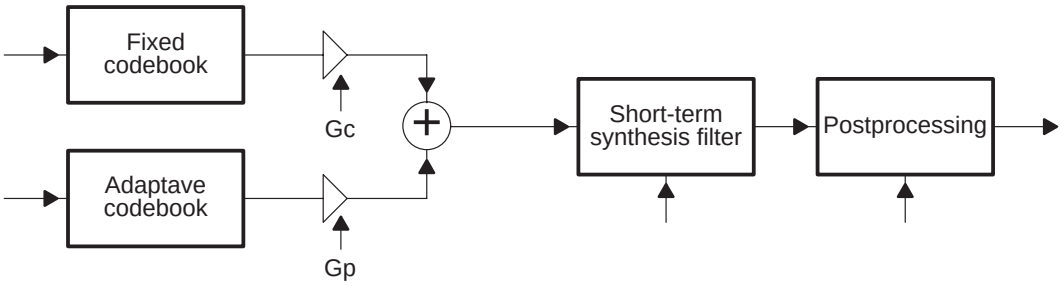


Figure 5. Decoding Principle of the CS-ACELP Encoder

3 Implementation of G.729

The software is implemented on the TMS320C54x 16-bit fixed-point DSP. The fixed-point DSP represents a fractional number by using a fixed decimal point, is one of the limitations of the fixed-point DSP. In classifying the different ranges of the decimal number, a Q-format is used. Different Q-formats represent different locations of the decimal point, and thus the integer range. Figure 6 shows a format of a Q15 number. Note that every bit after the decimal point has a resolution of ½ and the most-significant bit is used as the sign bit.

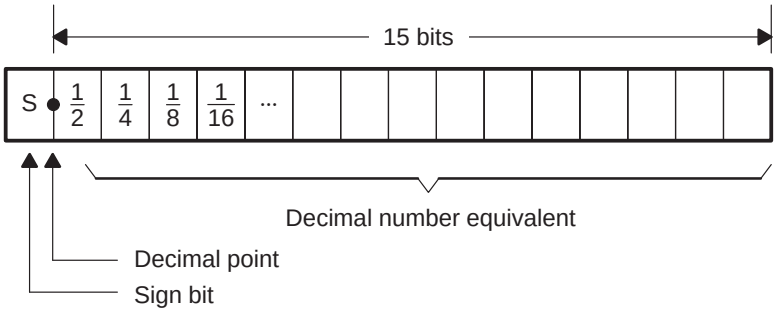


Figure 6. Q15 Format Representation

From Figure 6, we can see that the maximum positive number is obtained when the sign bit is 0 and the rest are 1s (7FFFH). The negative limit is obtained when the sign bit is 1 and the rest of the bits are 0s (8000H). This gives a range for the Q15 format to be –1.0 to 0.9999694 ≅ 1.0.

Therefore, we can increase the decimal range by shifting the decimal point to the right as shown in Figure 7.

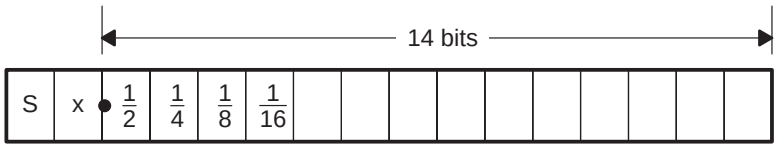


Figure 7. Q14 Format Representation

In Q14, the decimal range has been increased to –2.0 to 1.9999694 ≅ 2. However, the increase in range means a decrease in precision. Table 2 shows some of the decimal values for different Q formats. In the case of 32-bit representations, the decimal range is still the same except for a better precision.

Table 2. Decimal Value Representation of Hex Values for Different Q Formats

	Decimal values					
Hex Value	**Q15**	**Q14**	**Q13**	**Q12**	**Q11**	**Q10**
7FFFH	1.00000000	2.000000	4.000000	8.000000	16.000000	32.000000
3FFFH	0.50000000	1.00000000	2.00000000	4.00000000	8.00000000	16.00000000
1FFFH	0.25000000	0.50000000	1.00000000	2.00000000	4.00000000	8.00000000
0FFFH	0.12500000	0.25000000	0.50000000	1.00000000	2.00000000	4.00000000
07FFH	0.06250000	0.12500000	0.25000000	0.50000000	1.00000000	2.00000000
03FFH	0.03125000	0.06250000	0.12500000	0.25000000	0.50000000	1.00000000
01FFH	0.01562500	0.03125000	0.06250000	0.12500000	0.25000000	0.50000000
00FFH	0.00781250	0.01562500	0.03125000	0.06250000	0.12500000	0.25000000
007FH	0.00390625	0.00781250	0.01562500	0.03125000	0.06250000	0.12500000

3.1 Multiplication of numbers

When two numbers are multiplied, an extra sign bit is generated. For example, multiplication of two Q4 numbers is shown in Figure 8.

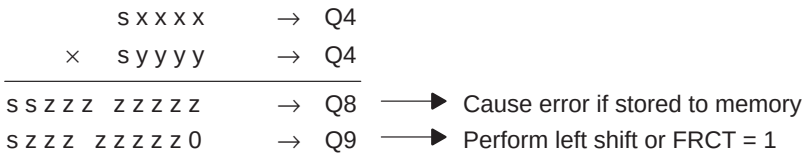


Figure 8. Multiplying Two Q4 Numbers

Therefore, an additional left shift is required to remove the redundant sign bit. This results in a Q9-format number. The removal of the additional sign bit can be done automatically by setting the FRCT bit in the DSP.

For a 16-bit multiplication, a 32-bit result is obtained. However, only the high word is stored in memory, the low word of the 32-bit result is truncated.

For better accuracy, a 32-bit result is maintained during a series of multiplications (convolution), i.e., multiply and accumulate. The truncation is performed only on the final result. To further improve the accuracy or the resolution of multiplication, a special number format called the double precision format (DPF) is used in this implementation. This format is used when single precision is not enough but full 32-bit precision is not needed. The mathematical operations pertaining to the DPF is not standard 32-bit operation. For example, a 32-bit by 32-bit multiplication would only produce a 32-bit result instead of a 64-bit result. However, note that a 16-bit DSP is used. As such, no matter how many bit is used in the multiplication process, only 16 bits will be stored. In DPF, the 32-bit integer is extracted into hi_word and lo_word. Both high and low words contain the sign, this allows fast multiplication. The format is as follows:

```
L_32 = hi_word<<16  +  lo_word<<1
hi_word = L_32>>16
lo_word = (L_32 – hi_word)>>1
```

The next subsection shows some operations involving DPF.

3.1.1 32-Bit by 32-Bit Multiplication

This operation multiplies two 32-bit integers and produces a 32-bit integer product. All operands in this operation are in DPF. The operation is given as:

```
L_32 =  (hi1*hi2)<<1 + ( (hi1*lo2)>>15 +(lo1*hi2)>>15) )>>1
```

where hi and lo are the high and low parts of the 32-bit number, respectively. This operation can also be viewed as the multiplication of two Q31 numbers which produces a Q31 result.

3.1.2 32-Bit by 16-Bit Multiplication

This operation multiplies a 32-bit number (DPF) with a 16-bit number, and produces a 32-bit product. The operation is given as:

```
L_32 =  (hi1*lo2)<<1 + ( (lo1+lo2)>>15)<1
```

where lo2 is the 16-bit number.

3.1.3 Overflow Handling

An overflow occurs when the value of the number stored in an accumulator exceeds a certain range. In the G.729 implementation, the value in the accumulator is always limited to the range 80000000H to 7FFFFFFFH; these values are the most negative and the most positive limit, respectively. Overflow is handled automatically by setting the OVM bit in the PMST register.

3.2 Conversion of C-Code to Assembly

The purpose of converting from the C language format to the assembly language format is to reduce the execution time of the vocoder. When the entire C file was run on the EVM board, it takes about 500 ms to encode a frame. This is very impractical for real-time applications because the delay for processing a frame is too large. Routines that consume the most computational resources are rewritten in assembly language. Before subroutines can be rewritten, the procedure of passing parameters in the C routine must be determined.

3.2.1 Parameter Passing in C Functions

Parameter passing in a C routine is accomplished via the stack. The C compiler imposes a strict set of rules on function calls. Except for special-support functions, any function that calls or is called by a C function must follow these rules. Instead of explaining the rules, Example 1 illustrates the function-calling convention. Note from the example, the first parameter always passes through accumulator A, and the rest of the parameters pass through the stack. The called function must reserve whatever stack space it requires (through the “Frame” command) to prevent corrupting data of the calling function. The required space is calculated based on the local variables (in C) and the maximum number of variables passed to called function (nested call) . When calling a C function, the stack pointer must always point to an even address. This can be achieved by ensuring that the number of PSHM instructions plus the amount of space reserved adds up to an odd number. Whenever the DSP encounters a CALL instruction, the return address is pushed onto the stack. Thus, prior to the entrance of the called function, the stack pointer is already pointing to odd address. Making sure that the stack pointer always points to an even address is to facilitate a 32-bit instruction (i.e., DLD, DADD, etc.). Example 1 explains parameter passing.

As shown in Example 1, interfacing an assembly subroutine to a main C routine is very easy if the function-calling convention is followed correctly.

In C Environment

```
void func1(int x, int y, int Array[ ] ); /*prototype declaration */
void func2(int x, int y );               /*prototype declaration */
int  k1,k2;                              /* global variables */
void main(void)
 {
   int  Data[2];                         /*  main routine local variables */
    …
   func1(k1,k2,Data);                    /* k1 passed to accumulator A */
                                         /* k2 passed to accumulator SP(0)+ */
                                         /* Address of Data[ ] passed to SP(1) */
}
```

Subroutine

```
void func1(int k1, int k2, int Data[ ] )
{
    int var1,var2,
    var1 = k1+k2;
    …
    func2( var1,var2);                   /* var1 passed to accumulator A */
                                         /* var2 passes to SP(0)+   */
}
+ SP(#offset from current stack pointer)
```

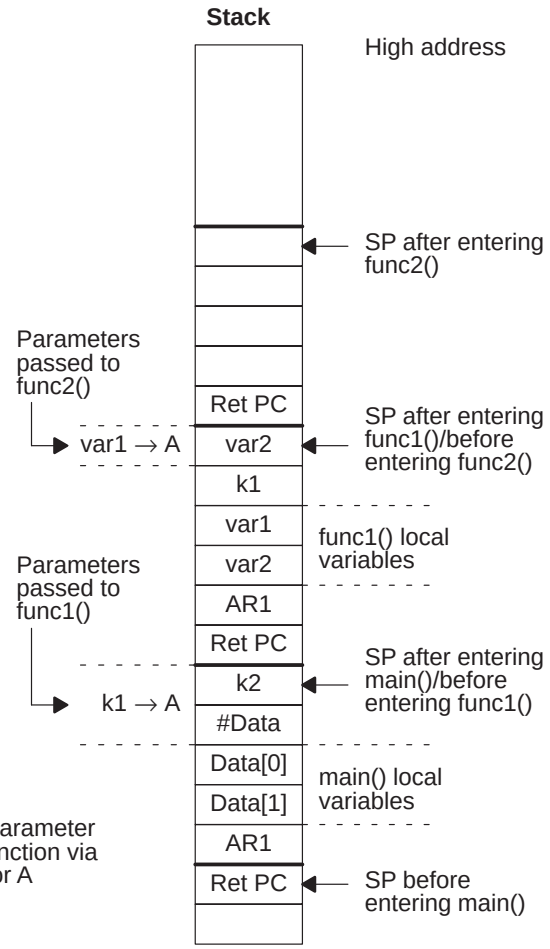
Assembly Language Environment

```
  ;main routine
PSHM          AR1  ; PSHM + FRAME #
FRAME         #-4  ; → ODD NUMBER
…
…
;func1(k1,k2,Data)

LD            *(_k2),A ;
STL           A,*SP(0)
LDM            SP,A ;
ADD           #2,A ;#Data
STL           A,*SP(1)
CALL          #_func1
…
…
FRAME          #4       ;
POPM          AR1       ;
RET           ;
…
```

func1( ) routine

```
PSHM          AR1 ;
FRAME        # -4  ;Reserve space
…
..
STL           A,*SP(1)          ; k1
….                            ; k2 stored in SP(6)
;func2( var1,var2)
LD            *SP(3),A          ; var2
STL           A,*SP(0)          ;
LD           *SP(2),A           ; var1
CALL         #_func2            ;
…
…
FRAME        #4
POPM         AR1
RET
```



Example 1. Example of Parameter Passing to Subroutine

3.3 Pipeline Issue

The C54xx DSP has a six-stage instruction pipeline. The six stages are independent of each other, which allows overlapping execution of instructions in a given clock cycle. During any one given cycle, one to six instructions can be active, each at a different stage of execution. Although this speeds up execution, it also poses the problem of pipeline conflict. That is, two stages of the pipeline might demand the same resources (i.e., memory, address unit etc.). Programmers writing C54 assembly code must take note of this constraint or unexpected errors will occur. The logic of the code written could be correct but pipeline conflict might force the DSP to fetch the wrong data or address. However, if the code is written entirely in ANSI C, the C compiler will take care of the pipeline issue. The different stages of the pipeline are shown in Figure 9.

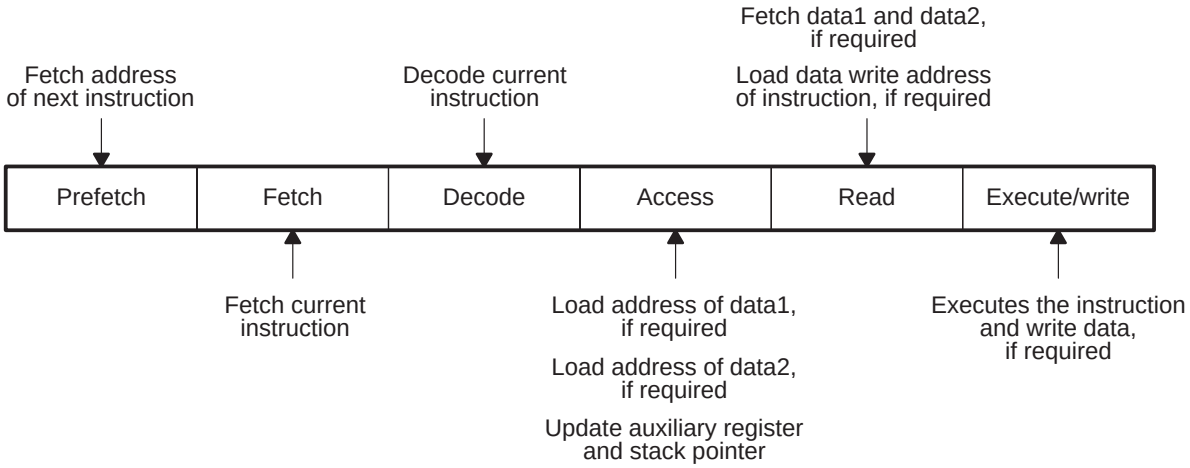


Figure 9. TMS320C54x Pipeline Stages

The C54x also has a set of protected instructions that is executed at the read stage — that is, one stage before the execute/write stage. These protected instructions are aimed to reduce pipeline conflicts but some might crash the pipeline if not used properly. Figure 10 shows how a pipeline conflict for a standard instruction is prevented by adding one latency. To overcome the problem of pipeline crash, a NOP instruction is added to force a delay of 1 cycle. The C54xx also has a mechanism to avoid pipeline conflict but it only applies to protected instructions. It simply delays the protected instruction from progressing to the execution phase, which is the same as a standard instruction. This is illustrated in Figure 11.

The converted assembly routines were checked thoroughly for pipeline conflicts. In most cases, the errors caused the DSP to fetch the wrong data and the auxiliary registers were not updated correctly.

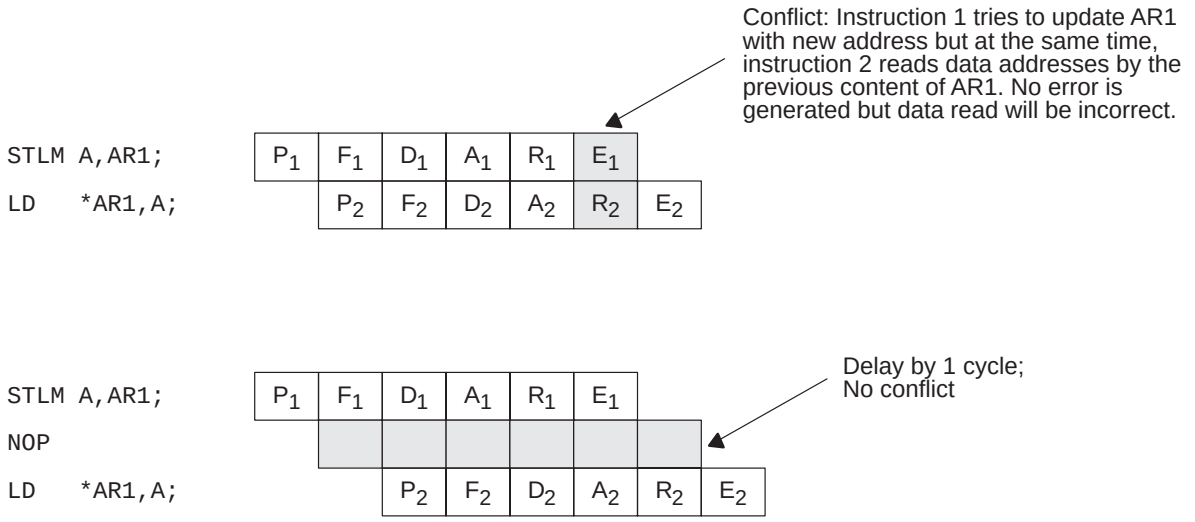


Figure 10. Example of Pipeline Conflict of Standard Instruction

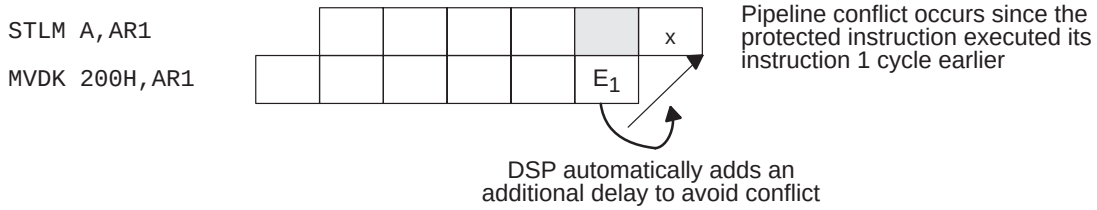


Figure 11. Pipeline Conflict Handling by the DSP

4 Real-Time Testing

The software was tested in real time using an audio analog-to-digital converter to capture human speech. One of the ways to meet the real-time requirement for the G.729 vocoder is to ensure that it does not take more than 10 ms to encode and decode a frame for full-duplex transmission. As already mentioned, the speech frame is 10 ms long (80 samples). The vocoder should complete any encoding and decoding of the previous frame before the current frame is captured. Of course, other measures can also be implemented to meet the real-time requirement if the 10-ms timing cannot be measured. The real-time requirement of the G.729 vocoder is shown in Figure 12.

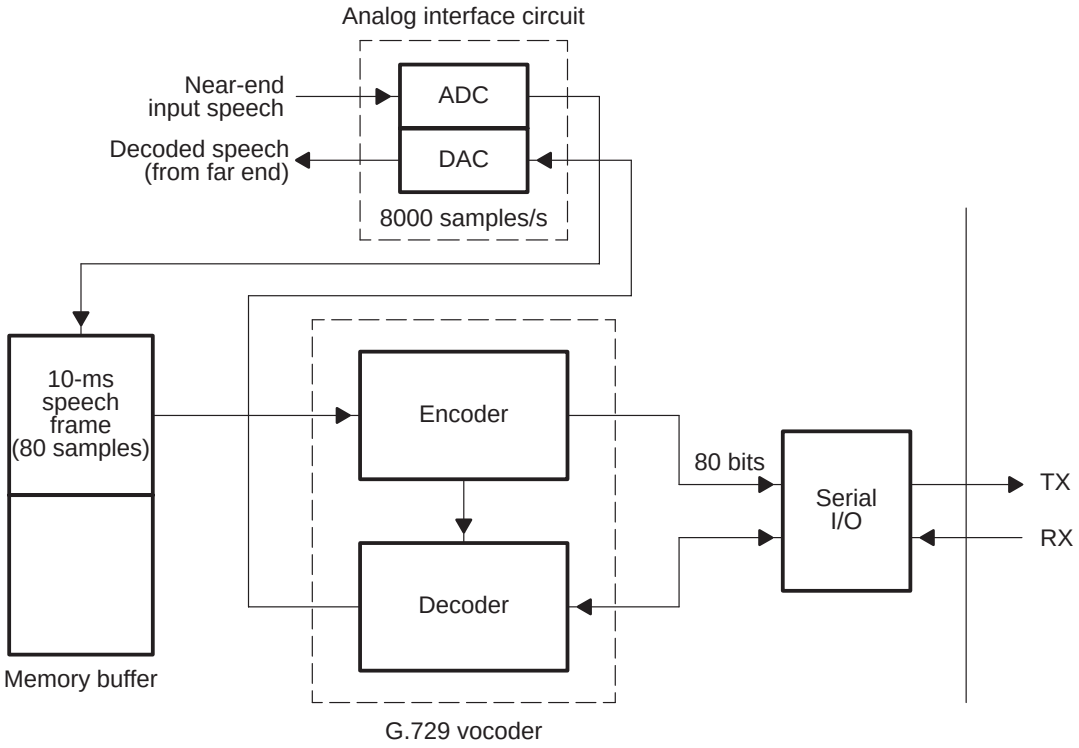


Figure 12. Practical Implementation of the G.729 Vocoder

From the timings shown, it can be seen that even after the whole vocoder's routines were converted to assembly language, the vocoder could meet the real-time requirement. For the purpose of evaluating the quality of the decoded speech, a non-real-time test was set up (see Figure 13). First, 500 frames corresponding to 40000 samples were captured. Next, each frame was encoded with the parameters sent to the decoding routine. From there, the simulated received bitstream was decoded and the reconstructed speech was stored back to the original speech buffer.

The decoded speech was sent to the Analog-Interface Circuit only when all the frames had been processed. The decoded speech was amplified (using on-board power amplifier) and sent to a pair of speakers. From the result obtained, the reconstructed speech was near toll-quality except for some interferences. The interferences might be an attribute of the noisy environment of the PC's motherboard.

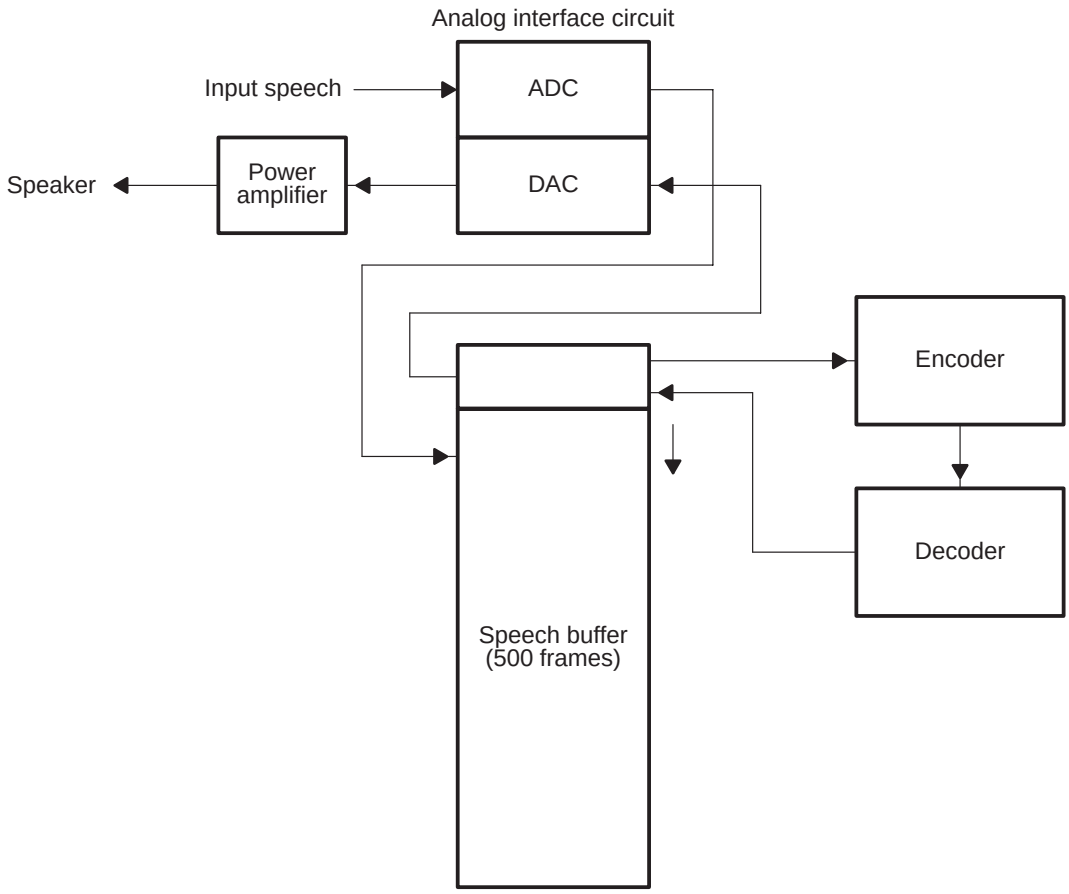


Figure 13. Non-Real-Time Implementation of the G.729 Vocoder

5 Conclusion

A non-real-time ITU G.729 8kbit/s vocoder was successfully implemented on a TMS320C54x DSP. This vocoder works on an 80-sample (80x16 bits = 1280 bits) speech frame and encodes it with 80 bits, thus, it has a compression ratio of 16:1. Vocoder plays a very important role in digital communication, especially in a cellular system. With less bits to send, bandwidth requirement per user will decrease and hence, user capacity will increase.

The vocoder can be made real-time if the codes are optimized so that the computational time is reduced to 10 ms or less. The vocoder was tested using voice samples captured through an analog-to-digital converter and the decoded speech was found to be near toll quality.

6 References

1. Panos E. Papamichalis, *Practical Approaches to Speech Coding*, Prentice-Hall, Englewood Cliffs, NJ, 1987.
2. “An Introduction to Speech Coding,” in W. B. Kleijn and K. K. Paliwal (eds.), *Speech Coding and Synthesis*, Elsevier, 1995.
3. ITU-T Recommendation G.729, March 1996, *C Source Code and Test Vectors for Implementation Verification of the G.729 8 Kbit/s CS-ACELP Speech Coder*.

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 acknowledgment, 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.

Customers are responsible for their applications using TI components.

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.