ALL ASSEMBLY IMPLEMENTATION OF G.729 ANNEX B SPEECH CODEC ON A FIXED POINT DSP

Manish Arora[1], *Nitin Lahane and Amit Prakash*

Sasken Communication Technologies Limited
Survey #139/25, Kasaba Hobli, Domlur Layout, Bangalore, India-560038
Email: nitin@sasken.com, aprakash@sasken.com

ABSTRACT

A lot of effort has been spent over the last few years in the development of digital speech coding methods and their subsequent standardization. Algorithms have evolved which provide good quality speech at sub 8 kbps bit rates although at a much higher computational expense. DSP processors have also improved with time providing specific signal processing functionalities aiding in easier codec implementations along with lower power consumption at higher clock speeds. Software development tools and compilers have also improved although they still do not work well in high volume, low cost systems. The cost of development tools may also be prohibitive for non-vendors and at times high level code conversion tools may not be present at all. This paper describes techniques and approaches commonly used to realize such systems where the codec implementation in all assembly is necessary. The specific codec implemented was International Telecommunication Union (ITU-T) G.729 Annex B. The techniques described in this paper are applicable to any other speech codec.

1. INTRODUCTION

The need for interoperability between different telecommunication networks has prompted a lot of standardization activity in the recent past. Organizations such as the ITU-T, the European Telecommunication Standards Institute (ETSI) and the Telecommunication Industry Association (TIA) act as standard bodies, defining and publishing telecommunication standards. Since 1990, Study Group 15 of the ITU-T has been involved in a standardization process for a speech coding algorithm at 8 kbps [1]. The main applications for this coder are personal communication systems, digital satellite systems and packetized speech. The proposed algorithm G.729, based upon the CS-ACELP technique produces speech quality equivalent to that of 32 kbps ADPCM under operating conditions. Since the coder is intended for wireless applications, error resilience to random and bursty errors is incorporated.

Over the recent past there has been a tremendous growth in programmable DSP processing hardware. Typical of the growth are the wireless DSP processors where the hardware has been well molded by applications. These enhanced conventional DSP processors are characterized with irregular data paths, small register files, specialized and non-orthogonal instruction sets. Consequently they make very poor compiler targets [2] and providing high-level language support is a difficult task.

In this paper we describe techniques to obtain optimal performance while reducing development time and effort when working with such systems. The organization of the paper is as follows. In section 2, we describe the G.729 coding algorithm in detail. The detailed assembly implementation methodology is explained in section 3. Section 4 describes the testing and memory optimization techniques applied in our work. The results of the implementation are given in section 5.

2. G.729 SPEECH CODEC

2.1. G.729

The G.729 coder [1] is designed to operate with a digital signal obtained by performing telephone bandwidth filtering of the analogue input signal (G.712), then sampling it at 8000Hz, followed by conversion to 16 bit linear PCM for the input to the encoder. The output of the decoder should be converted back to an analogue signal by similar means. The G.729 coder operates on 10-ms speech frames, each corresponding to 80 samples. In addition, there is a look-ahead of 5ms, resulting in a total algorithmic delay of 15ms. The overall encoding configuration is shown in Figure 1.

Linear prediction analysis is done once per 10 ms frame to compute the LPC filter coefficients. These coefficients are converted into LSP and quantized using predictive two-stage vector quantization (VQ) with 18 bits. The short-term synthesis filter is based on 10^{th} order LPC filter. The excitation signal is chosen by using an analysis-by-synthesis search procedure in which the error between the original and reconstructed speech is minimized according to a perceptually weighted distortion measure. The speech frame is divided into two 5-ms sub-frames (40 samples), with excitation parameters determined for each sub-frame.

The pitch excitation vector is found from the previous adaptive codebook. The best adaptive codebook delay is determined in two stages by using open-loop analysis and closed-loop analysis. The open loop analysis is done one per

[1] The author is presently working with Emuzed India Pvt. Limited, 839, 2nd Cross, 7th Main, HAL II stage, Bangalore-560008, India. Email: arora@emuzed.com.

frame (10ms) and uses the weighted speech signal. The closed-loop pitch search minimizes the mean-squared weighted error between the original and synthesized speech. The search is limited around the candidate delay obtained from the open-loop pitch analysis. A fractional pitch delay with 1/3 resolution is used. The pitch delay is encoded with 8 bits in the first sub-frame and differentially encoded with 5 bits in the second sub-frame.

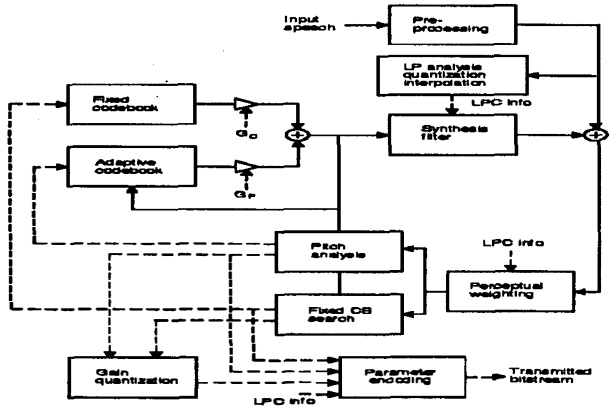

Figure 1. Block diagram of the G.729 Encoder.

A 17-bit algebraic codebook is used for the fixed innovation codebook. The innovation vector contains four non-zero pulses. The codebook vector is constructed by taking a zero vector, and putting the four unit pulses at the found locations, multiplied by their corresponding sign. The pulse positions of the first three pulses are each encoded with 3 bits, while the fourth pulse position is encoded with 4 bits. This gives a total of 17 bits for the four pulses. In this structure, 4 bits are determined through straightforward quantization, whereas 13 position bits are determined in an analysis-by-synthesis manner. The 13-bit pulse positions are found by minimizing the mean square error between the weighted input speech and the weighted reconstructed speech. The gains of the adaptive and fixed codebook contributions are vector quantized with 7 bits using a conjugate structure codebook with 4th order MA prediction applied to the fixed-codebook gain.

In decoding, the parameter indices are extracted from the received bit-stream. These indices are decoded to obtain the coder parameters corresponding to a 10-ms speech frame. These parameters are the LSP coefficients, two fractional pitch delays, two fixed codebook vectors, and two sets of adaptive and fixed codebook gains. The LSP coefficients are interpolated and converted to LP filter coefficients for each subframe. Then, for each 5-ms subframe, the excitation is constructed by adding the adaptive and fixed codebook vectors scaled by their respective gains, the speech is reconstructed by filtering the excitation through the LPC synthesis filter, and the reconstructed speech signal is passed through an adaptive postfilter to enhance the speech quality.

2.2. G.729 Annex B

The Annex B [3] provides a high level description of the Voice Activity Detection (VAD), Discontinuous Transmission (DTX), and Comfort Noise Generator (CNG) algorithms. These algorithms are used to reduce the transmission rate during silence periods of speech. The VAD algorithm makes a voice activity decision every 10 ms and a set of difference parameters is extracted and used for an initial decision. The parameters are the full band energy, the low band energy, the zero-crossing rate and a spectral measure. The long-term averages of the parameters during non-active voice segments follow the changing nature of the background noise. A set of differential parameters is obtained at each frame, which are the difference measure between each parameter and its respective long-term average.

The initial voice activity decision is obtained using a piecewise linear decision boundary between each pair of differential parameters. A final voice activity decision is obtained by smoothing the initial decision. The output of the VAD module is either 1 or 0, indicating the presence or absence of voice activity respectively. If the VAD output is 1, the G.729 speech codec is invoked to code/decode the active voice frames. However, if the VAD output is 0, the DTX/CNG algorithms are used to code/decode the non-active voice frames. If the background noise is not stationary, a mere comfort noise insertion does not provide the naturalness of the original background noise. Therefore it is desirable to intermittently send some information about the background noise in order to obtain a better quality when non-active voice frames are detected. The coding efficiency of the non-active voice frames can be achieved by coding the energy of the frame and its spectrum with as few as fifteen bits. The bits are transmitted only when an appreciable change has been detected with respect to the last transmitted non-active voice frame. At the decoder side, the received bit stream is decoded. If the VAD output is 1, the G.729 decoder is invoked to synthesize the reconstructed active voice frames. If the VAD output is 0, the CNG module is called to reproduce the non-active voiced frames.

3. ASSEMBLY IMPLEMNTATION

3.1. ITU-T C code

Along with the documentation describing the algorithms and references, the standard bodies provides an ANSI C code corresponding to a fixed-point fractional arithmetic implementation of the codec. A set of test vectors comprising input and output are given to verify the bit exact development of the codec on the target platform. The ANSI C code is written in a manner to aid porting to different platforms. In order to achieve this the basic mathematical operations of addition, subtraction, multiplication etc. are implemented as separate functions. This is done so that these operations can be rewritten in the most optimized manner for the specific DSP architecture. Also since the C language does not support fixed-point fractional arithmetic all the mathematical operations are simulated and functionalized.

Before the start of the coding in assembly, it is significant to have an idea of the routines that are the best candidates of optimization. Since the coding and debugging time in an all assembly implementation is very high, the prior knowledge of the relative importance of functions would help in their optimal implementation. C code profiler was used to extensively profile

both the encoder and decoder standard reference code. The profile results proved the importance of the multiply and multiply -accumulate operations and their optimal implementation. Table 1 shows the time spent in some basic operations for ITU-T test vector "tstseq4" with DTX disabled for the encoder.

Function	Time (%)	Function Desciption
L_mac	42.1	Multiply two 16 bit nos. and accumulate 32 bit result with saturation
L_mult	18.1	Multiply two 16 bit numbers with saturation
L_add	13.8	Addition of 32 bit numbers with saturation
Sature	9.3	Saturate to 16 bit

Table 1. Time spent in some basic operation in ITU test vector "tstseq4" for the encoder with DTX disabled.

The percentage of time spent in doing the L_mac operation is a very significant 42.1%. Of this a major portion is the calling overhead of the L_mult and L_add functions. This should be reduced in the final implementation. Saturate to 16 bit also takes about 9.4%. The profile information should be utilized now to use the hardware of the target platform for the intensive routines. Table 2 shows the time spent in major functions in the encoder for the test vector "tstseq4" with DTX disabled.

Function	Time (%)	Function Desciption
ACELP_Codebook	31.2	Algebraic codebook search procedure
Pitch_ol	14.7	Open loop pitch lag search
Pitch_fr3	11.1	Closed loop pitch search
Qua_lsp	8.6	Quantize lsp parameters
Syn_filt	8.1	Synthesis filtering
Az_lsp	4.9	Lsp from lpc coefficients
Convolve	4.8	Convolution between 2 vectors
AutoCorr	4.7	Autocorrelations with windowing
Residue	4.0	Compute the LPC residual

Table 2. Time spent in major functions of the encoder in ITU test vector "tstseq4" with DTX disabled.

3.2. Porting Methodology

In systems where a code conversion tool is present, the C code is first built for the target platform, then profiled and the optimization starts on selected functions. The ITU-T C code may be modified to a working reference code [4]. The lack of a code conversion tool posed a serious limitation in our development process. Lower level functions had to be first coded and tested before the testing of the higher-level functions. A code analysis tool [5] was extensively used to study the C code first and the reference code was divided into 4 portions. Coding was done incrementally in this order:

- Basic Operations
- Functions common to the Encoder and Decoder
- Functions specific to the Encoder
- Functions specific to the Decoder

With reference to Figure 2 the assembly coding was done with a bottom-up approach, starting from the functions on the right to the left.

While coding, a strong naming convention of variables and labels was used. This is needed to prevent illegal accesses and jumps because of the probable use of the same name in different

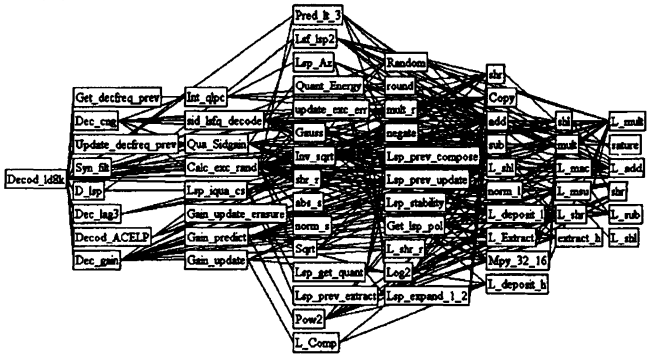

Figure 2. Decod_ld8k function invocation tree.

modules. We appended the function name to all the names of local variables and labels used in the function in our naming convention. A parameter passing conventions also needs to be used consistently. We used registers for parameters passing and return values from functions.

3.3. Code Optimizations

The optimization process needs to exploit the target DSP hardware, instruction set and memory configuration optimally. Our DSP hardware was a typical wireless enhanced conventional DSP chip. Initially we concentrated on arithmetic operations. Most of the basic math functions were directly inlined into the assembly code. This was done to save the overhead with subroutines as most of the basic operations can be performed in a single cycle on typical DSP hardware. With time internal DSP data path widths have increased to aid higher precision arithmetic [2]. Saturation and rounding arithmetic hardware available is with 32 bit precision. Speech codecs contain a lot of 16 bit arithmetic and saturation operations. In our case we used 32-bit saturation hardware to carry out 16-bit saturation arithmetic by explicit shifting.

The second optimization stage concentrated on the loop optimizations. The main cycle count advantage was gained by optimizing the inner loops. Reducing the memory access by using registers instead of local variables was done. Instruction shuffling to reduce pipeline latencies was used. Some loops were unrolled and other instructions were inserted in place of 'nop' operations. Delayed call and branch instructions were extensively used to reduce the pipeline latencies associated with flushing of the pipeline at a jump. Since the entire memory map was in the programmers control parallel load/store and temporary variable alignment techniques were well combined with post increment addressing to optimize nested loops.

Since in our development process testing whole of the codec is only possible after all the functions are assembled and integrated its very difficult to track a target encoder/decoder

cycle count from the initial phase of the work. We used an innovative method to track and predict the overall cycle counts after deciding upon a target initially. The method used the profile results from the C code obtained earlier. Suppose a function took a worst case of X% time amongst all test vectors in the C code. We optimized the corresponding assembly code until it took less than X% time of our assembly cycle target. Once this is achieved for all functions the overall codec target cycle count is easily met.

4. TESTING AND MEMORY MAP

As discussed above the whole code was divided into 4 portions. At each portion once the coding is over the functions need to be extensively unit tested. Unit test cases are generated from all possible hits of the function from all the test vectors. Verification was done for all possible hits of the function at each level. This would ensure that during the integration testing the defects found in the code are only because of function interface problems rather than the function assembly code. Debugging at each level after this was very easy and this method saved a lot of our testing time. The integration testing for whole of the codec was carried out on all the non-exhaustive ITU-T test vectors as well as a few other samples of recorded speech.

The exact method for organizing the memory and runtime stack will depend on the development tools being used, which will in turn depend on the DSP being used. We partitioned the memory area into ROM, temporary and static RAM areas. Each of these areas was relocatable in the form of an incremental define file as shown below. The target system is 16 bit integer addressed. The alignment of integer and long integer variables to 16 bit and 32 bit boundaries was also done explicitly.

The temporary memory was extensively optimized by overlapping memory areas of functions, which were at the same level in the invocation tree. As an example the Open Loop Pitch lag and Closed Loop Pitch lag temporary memory areas can be overlapped since these functions are called concurrently from the encoder main routine. Table 3 shows the temporary memory alignment procedure used for the encoder. The areas B, C and D are overlapped with each other to give the temporary RAM size as 1422 words.

Name	ITU C Code Files	Size (Words)
A	Coder_ld8k	520
B	Util, qsidgain, lspgetq, lpcfunc, gain_pred, lpc, pre_proc, qua_lsp, pitch, pwf	848
C	Taming, calcexc, filter, dtx, vad, qua_gain, qsidlsf	888
D	Acelp	902
Total Temporary Memory (A+D)		1422

Table 3. Encoder temporary RAM overlap.

5. RESULTS AND CONCLUSIONS

In this paper, we presented the real-time G.729 Annex B speech codec implementation on a fixed point DSP. Due to the unavailability of a code conversion tool whole of the codec was written in assembly. Table 4 shows the performance achieved in our work. The cycle ratings presented require the listed amount of on chip single access RAM, or external zero wait state program Rom and data RAM. The codec could have been optimized more at the cost of development time. The overall development time was very high (>1 man year effort). The key points of our work can be summarized as:

- Profiling of the ITU-T reference code;
- Analysis of the reference code and assembly coding in a bottom-up approach;
- Code optimization with Basic operation in-lining, loop unrolling and delayed branch/ function calls;
- Codec cycle target tracking with function performance comparison in C and Assembly;
- Unit test case generation and subsequent unit and integration testing;
- Re-locatable memory map using incremental defines;

The implementation was tested with the set of test sequences provided by ITU-T for bit exact implementation. We have presented techniques essential for all assembly speech codec implementations. These methods are well applicable to any speech codec and processor platform.

Measure	Codec Performance
Program Size	14.5 K Words
Rom Size	3.28 K words
Ram Size	4.52 K Words
Encoder Complexity	18.4 Mega Cycles (Worst Case)
Decoder Complexity	4.2 Mega Cycles (Worst Case)

Table 4. Results of real time implementation.

6. REFERENCES

[1] ITU-T Recommendation G.729, "Coding of Speech at 8kbit/s Using Conjugate Structure Algebraic-Code-Excited Linear-Prediction (CS-ACELP)," June 1995.

[2] Phil Lapsley, Jeff Bier, Amit Shoham and Edward A. Lee, *DSP Processor Fundamentals: Architectures and Features*, IEEE Press, Berkeley, California: Berkley Design Technology Inc., 1996.

[3] ITU-T Recommendation G.729 Annex B, "A Silence Compression Scheme for G.729 Optimized for Terminals Conforming to Recommendation V.70," June 1995.

[4] David H. Crawford and Emmanuel Roy, "Techniques for Real-Time DSP Implementation of Speech Coding Algorithms", *DSPWorld ICSPAT*, pp. 2-7, November 1999.

[5] ANSI C, C++ and K&R C reverse engineering, code navigation, and metrics tool, *Understand for C++*, Scientific Tools Inc. (www.scitools.com).