Real Time Implementation of TETRA Speech Codec on TMS320C54x

B. Sheetal Kiran, Devendra Jalihal, R. Aravind
Department of Electrical Engineering,
Indian Institute of Technology Madras
Chennai 600 036
{sheetal, dj, aravind}@ee.iitm.ernet.in

ABSTRACT

This paper, partly tutorial in nature, describes an approach to the implementation of the 4.567 kbps TETRA speech codec on the TMS320C54x DSP family. The TETRA system is introduced and its speech codec is described in detail. The TETRA standard describes the speech codec in bit-exact, fixed-point C-code. Our paper compares the efficiency of implementing this code by first writing it in C and cross-compiling it and directly hand-coding it in TMS320C54x assembly code.

1. INTRODUCTION

Speech signals are by far the most prevalent of signals transported over telecommunication networks. The 'toll quality' A-/μ- law based digital representation of analog speech results in 64 kbps bit stream. The 64 kbps of data when transmitted on telecommunication networks occupy considerable bandwidth, which is not always available. A number of well-known compression techniques such as ADPCM, Sub-band Coding, etc., achieve lower bit rates in the range of 32 kbps to 16 kbps. Using radio and satellite links requires more efficient coding methods and near-toll quality speech below 16 kbps rate. LPC based techniques such as Code Excited Linear Prediction (CELP) coding, are used at lower rates and can provide good quality speech at bit rates of around 8 kbps and down to 4.8 kbps. These codecs perform very well in noisy environments. Speech communication is inherently real-time and to achieve this using a CELP coder in real-time on a DSP is a formidable task, as CELP coders are computationally intensive.

A DSP implementation of Speech Codec for a full-rate traffic channel defined in the document ETS 300 395-2[1] on TMS320C54x DSP chip is considered here. Section 2 introduces the TETRA system and describes the TETRA speech codec in detail. Section 3 highlights the importance of using fixed point DSP for speech codec implementation; introduces the TMS320C54x TI DSP and discusses our approach to implementation. Section 4 presents the conclusions.

2. The TETRA System

A number of organizations such as police, fire fighters, taxi operators, etc. operate mobile communications systems, which are independent of the PSTN network system. These vary in complexity from simple point to point communication radios to trunked networks requiring large infrastructures and supporting point to point and point to multipoint communications. Trunked radio networks are preferred by the radio regulators because of their high spectral efficiency (particularly where relatively short messages are exchanged) and by the users who benefit from the large number of supplementary services that can be supported. This proliferation of user requirements has resulted in the evolution of many standards, out of which TETRA (TErrestrial Trunked Radio) has been the most important one [2]. TETRA offers many features such as special safety services, cellular operating modes and low-bit rate data services.

TETRA Speech Codec

The TETRA speech codec is a CELP based codec, as shown in Figure1. Further, it uses the algebraic CELP (ACELP) where the innovation codebook have an algebraic structure and this has advantages in terms of storage, search complexity and robustness [3].

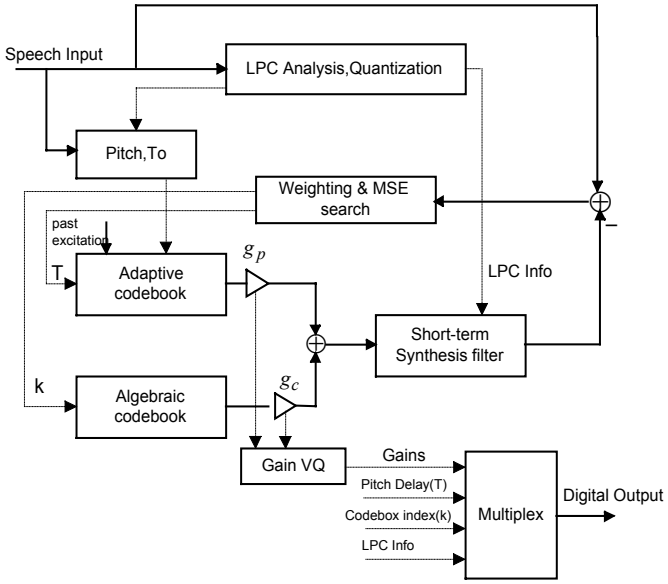

Figure1: *Block diagram of TETRA encoder*

The basic building blocks of the TETRA speech coder are (i) LPC analysis, (ii) LPC to LSP conversion and Quantization, (iii) Long-term prediction and (iv) Codebook search.

LPC analysis is performed every 30 ms. The different frames used by the codec are shown in Figure 2. The LPC coefficients are converted to the Line Spectral Pairs (LSP) for quantization and interpolation. In the TETRA codec, implementation, quantization and interpolation of the LSPs are performed in the cosine domain, and thus saving any trigonometric computations necessary to convert to the frequency domain. The computed LSPs are quantized with 26 bits using a split-VQ.

The long term prediction analysis or adaptive codebook search finds the delay and the gain values of the pitch filter for each 7.5 ms sub-frame. To simplify the pitch analysis procedure, a two-stage approach is used, comprising first an open loop pitch search followed by a closed loop search. The open loop pitch is computed once every speech frame (30 ms) using a weighted speech signal obtained from a shaping filter which is constructed using the unquantized LP parameters. The closed loop pitch analysis is performed around the open loop pitch delay on a sub-frame basis. The closed loop pitch search is done by minimizing the mean-square weighted error between the original and the synthesized speech.

The TETRA codec uses a specific dynamic algebraic excitation codebook whereby the fixed excitation vectors are shaped by a dynamic shaping matrix. The shaping matrix is a function of the LP model, and its main role is to shape the excitation vectors in the frequency domain so that their energies are concentrated in the important frequency bands. Here a 16-bit algebraic codebook is used in the innovative codebook search, which finds the best innovation and gain parameters. The innovation vector contains, at most, four non-zero pulses, with amplitudes of 1.4142, -1, +1 and –1, and result in fast codebook search [1]. The codebook is searched by minimizing the mean squared error between the weighted input speech and the weighted synthesis speech. A focussed search procedure is used so that a low percentage of the codebook is searched. Obviously this is where most of the complexity of the coder lies.

The bit allocation for various codec parameters is given in Table 1. Each 30 ms frame is represented by 137 bits, resulting in a bitrate of 4.567 kbps.

Table 1: *Bit Allocation for the TETRA codec (SF: Sub-frame)*

Parameter	SF1	SF2	SF3	SF4	Total
LP Parameter					26
Pitch Delay	8	5	5	5	23
Algebraic Code book	16	16	16	16	64
Gains(VQ)	6	6	6	6	24
Total Bits					137

3. IMPLEMENTATION DETAILS

3.1 Advantages of DSP

Digital Signal Processors can be divided into two categories, fixed-point and floating-point. Fixed-point DSPs usually represent each number with a minimum of 16 bits. Floating-point DSPs typically use a minimum of 32 bits to store each value. Floating point numbers are stored in ANSI/IEEE standard, dividing the 32 bits into 3 parts, one bit for sign, 8 bits for exponent and the rest for magnitude (mantissa). Fixed-point DSPs usually represent the data in two's complement with an assumed binary point in a fixed place [5]. Fixed-point DSPs are cheaper, consume less power but take longer code development time. Floating point DSPs are expensive, require more power but are easier to code.

CELP coding is very computationally intensive involving a large number of multiplications and divisions of real numbers and other operations. The most intensive part of the TETRA codec is the codebook search. For a complete implementation of the TETRA codec, channel encoding and decoding as per the ETSI document should also be included. All these complex algorithms must be done in real time i.e., the time lag between the speaker and the listener should not exceed frame-length/N s. Here frame-length is 30 ms, and N is the number of simultaneous channels the DSP is processing. Usually N=1 at the subscriber end. This calls for the use of a dedicated signal processor. We propose to use the TMS320C54x processor. This fixed-point DSP meets the specific needs of real-time embedded applications, such as telecommunications. This DSP has a number of advantages such as [4]:

- 2 accumulators of 40 bits.
- 8 auxiliary registers, simplifies indirect addressing.
- A SP (Stack Pointer) register for stack addressing.
- Circular Buffer.
- Instructions with 2 to 4 operands.
- Parallel instructions in one cycle.
- Arithmetic instructions with 32 bit operands (DADD, …)
- Instruction to normalize and shift numbers in one cycle.
- Conditional instructions as conditional store.

These and other advantages of TMS320C54x make this an ideal platform to implement real-time speech codec applications. The DSP used here is a TMS320VC5410, an evaluation module. This is a 100 MIPS processor with 16K words (16 bits) of on-chip Program ROM, 8K words of dual-access RAM and 56K words of single-access RAM.

3.2 Implementation Methodology

The TETRA codec is described in a fixed-point C code. To generate highly optimized codes on fixed-point DSP, it is still necessary to program in assembly [6]. The visual debugger called the Code Composer Studio tool from Texas Instruments is very useful in this regard. Using this tool it is possible to check the contents of any registers, program memory contents and data memory contents at any point of the code and also possible to set break points.

Our methodology to implement this is inspired from the fixed-point C code available from the standard. An implementation example of the C code on the ‘54x DSP is shown in the Figure 3. The C code shown is a fixed-point implementation to evaluate the LSP polynomial from the LP filter coefficients (evaluation of only one polynomial is shown for brevity). As can be seen, all intermediate arithmetic is done in double precision. Whenever there is a need to operate on two 16-bit operands each operand is scaled into a 32-bit temporary location and the corresponding arithmetic is performed. As the DSP used is a fixed-point one, the format of the numbers should always be taken care of depending on their possible dynamic range. For example, Q15 numbers have a range between -1 and +1, in Q14 the decimal range increases from –2 to +2 and so on. However, the increase in range means a decrease in precision. In the case of 32-bit representations, the decimal range is still the same except for a better precision. In the example, Q11 range has been chosen for LSP polynomial.

The ANSI C code uses a lot of low-level functions for basic arithmetic. The advantage of this type of coding is most of the function calls resemble low level general DSP instructions. For example, a call to a MAC function like ‘L_mac()’ can be replaced by a MAC instruction. Not all functions can be similarly replaced like the ones using double precision multiplication, 16-bit by 32-bit multiplications, single and double precision division, etc. These functions have to be hand coded in separate subroutines. The choice of a function call or a macro depends on the frequency of usage. If the subroutine is frequently called, say a double precision multiplication, the choice of a macro is economical in terms of MIPS.

From the comparison in Table 2, the advantage of hand coded assembly over cross-compiled code is clear. The C code was compiled using TMS320C5000 Code Composer Studio (version 1.1) and optimized at the highest level 3. The compile tools offer quick ways to generate assembly code but at the cost of MIPS. The table shows a 10-fold decrease in the instruction cycles with hand coded assembly. This may not be the case always, because the ANSI C code is written in DSP friendly format

as previously mentioned. If instead the C code were written with inline coding rather than using function calls for basic arithmetic, the cycles mentioned against cross-compiled coding may come down, but still large by any account.

Table 2: *Comparison of the two coding alternatives for LSP polynomial computation.*

Type of Coding	No. of cycles (approx.)
Cross compiled code	570
Hand coded Assembly	60

The use of macros simplifies the coding by avoiding the PUSH and POP stack operations of function calls, but the registers that are utilized in the macro should be properly documented and taken care of. Special macros were developed for various DSP routines that were called frequently. All the macros were archived using the **ar500.exe** application. Memory control was achieved by allocating data tables and various sections of the code to Program or Data memory locations on the chip accordingly. At each step of the assembly coding results were seen to be bit compatible to the corresponding results from the C code. We can do away with any pointer that has been declared, as we have a set of eight 16-bit Auxiliary Registers (AR0-AR7) which can be used as pointers to any array in data memory.

4. CONCLUSIONS

In this paper we have proposed an approach to the implementation of the TETRA speech coder in real-time on TMS320C54x DSP. The basics about TETRA system as a whole and the details of the speech codec involved have been discussed. We have highlighted the procedure followed to implement the codec. The two methods: cross compiling and hand coding have been compared for efficiency of the code generated, execution time, size of the code and ease of implementation. Hand coding of complex algorithms has been shown to generate much more efficient code than cross-compilation.

5. REFERENCES

[1] European Telecommunications Standards Institute (ETSI), ETS 300 395-2: Terrestrial Trunked Radio (TETRA); Speech codec for full-rate traffic channel, Feb 1998.

[2] John Dunlop, Demessie Girma, James Irvine, *Digital Moblie Communications and the TETRA System* – John Wiley & Sons, 1999.

[3] Salami R. et.al.: "Speech coding", Chapter 3 in *Mobile Radio Communications*, edited by R. Steele, Pentech Press, London, 1992.

[4] "TMS320C54x User's Guide", Texas Instruments, 1997.

[5] Junchen Du, George Warner, Erik Vallow and Tom Hollenbach, "High-Performance DSPs – Using DSP16000 for GSM EFR Speech Coding", *IEEE* Signal Processing Magazine, March 2000.

[6] Jean-Pierre Petit, "Real-time implementations of the recent ITU-T low bit rate speech coders on the TI TMS320C54x DSP: results, methodology and applications", France Telecom – CNET publications, March 1999.

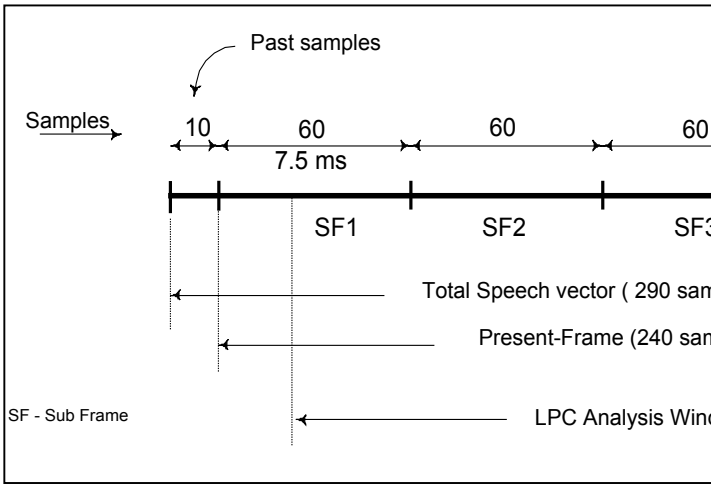

Figure 2: *Frame structure used in the T*

The fixed-point C code of an LSP Polynomial evaluation from ETSI document (code for only one polynomial is showed).

```
f1[0] = 2048;        /* f1[0] = 1.0 is in Q11 */

for (i = 0; i< nc; i++)
{
  t0 = Load_sh(a[i+1], (Word16)15);
                     /* a[i+1]  in Q27 */
  t0 = add_sh(t0, a[pp-i], (Word16)15);
                      /* +a[pp-i] in Q27 */
  t0 = sub_sh16(t0, f1[i]);
                     /* -f1[i]  in Q27 */
  f1[I+1] = extract_h(t0);
            /* f1[i+1] = a[i+1] + a[pp-i] - f1[i] */
            /* results in Q11  */
}
```

The corresponding TI DSP '54x implementation is:

```
            ST #2048,*(f1)        ; f1[0]=2048  {1.0 is in Q11 }
            STM #A_t+1,AR1        ; AR1<-a[i+1]
            STM #A_t+10,AR2       ; AR2<-a[pp]
            STM #1,AR0
            STM #f1,AR3           ; AR3<-f1

            ST #-32768,*SP(0)
            LD *SP(0),T           ; T <- -32768
            LD #5,B               ; B <- nc(=5)

loop:       SUB #1,B
            MPY *AR1,A            ; A <- a[i+1]*T
            NEG A
            MAS *AR2,A            ; A <- t0 - a[pp-i]*T
            DST A,*(t0)           ; t0 is 2 word temporary variable
            MPY *AR3+,A           ; A <- f1[i]*T
            SFTA A,1
            DADD *(t0),A          ; A<-t0+A
            STH A,*AR3            ; f1[i+1]
            BC loop,BGT
```

Figure 3: *An example implementation on the TMS320C54x DSP*