Published in Proceedings of Signal Processing Workshop (SPW), 7-12, 2000
which should be used for any reference to this work

A LOW-COMPLEXITY VARIABLE BIT RATE SPEECH CODER FOR PORTABLE STORAGE APPLICATIONS

S. Grassi, M. Matthey, G. Biundo, A. Dufaux, M. Ansorge, and F. Pellandini

Institute of Microtechnology, University of Neuchâtel
Rue A.-L. Breguet 2, 2000 Neuchâtel, Switzerland
Phone: +41 32 7183432; Fax: +41 32 7183402; Email: Sara.Grassi@imt.unine.ch

Abstract*: This paper describes ongoing work in the development of a variable bit rate speech coder for storage applications in portable devices. The coder is based on the CELP FS1016, with different algorithmic modifications aimed at reducing the computational complexity. A silence compression scheme was adapted to this coder, to achieve reduction in both the bit rate and the power consumption. The physical implementation of the coder using an FPGA is being done.*

1 Introduction

The goal of speech coding is to reduce the bit rate needed to represent a speech signal, while keeping an acceptable quality of the decoded output. The term speech coding refers to narrow-band speech coding, the coding of telephone bandwidth (300-3400 Hz) speech sampled at 8 kHz. Speech coding for application in portable voice storage devices (e.g. personal digital assistant with voice input) has the following requirements:

- Moderate to low bit rate: to use efficiently the available memory and to decrease the amount of storage memory in the device, with the consequent reduction in power consumption and size.
- Near toll or good synthetic quality.
- Low computational complexity for low power consumption.

Additionally, the use of silence compression scheme (see Section 3) is highly desirable, for both bit rate reduction (resulting in optimal use of the available memory) and power saving (by switching off some computational expensive coder blocks during silence periods).

These requirements also apply to devices in which the speech information is compressed and stored before its transmission (e.g. vocal pager).

On the other hand, other requirements, which are important for communications applications, are less stringent in storage applications. A longer delay is acceptable, allowing longer frames, with the consequent decrease in bit rate and complexity. There is also no requirement on robustness toward channel error, as the storage memory can be seen as an error-free channel.

This paper presents the development of a low complexity, variable bit rate (VBR) speech coder suitable for voice storage in portable devices. The coder is based on the CELP FS1016 coder, explained in Section 2. The adaptation of a silence compression scheme to this coder is given in Section 3. Optimization of the main functional blocks is given in Sections 4 and 5. The physical implementation on FPGA is described in Section 6. Conclusions and further work are drawn in Section 7.

2 The FS1016 coder

The U.S. Federal Standard 1016, is a Code-Excited Linear Predictive (CELP) speech coder operating at 4.8 kbps [1]. This coder uses an 8 kHz sampling rate and a 30 ms frame size (240 speech samples), with four sub-frames of 7.5 ms each. Its main functional blocks are:

- Spectral analysis and quantization;
- Adaptive codebook search and gain calculation;
- Stochastic codebook search and gain calculation.

The FS1016 was originally intended for secure voice transmission, but it has some very desirable characteristics for low-power storage applications. As this coder is relatively aged we have added some features and optimizations, inspired from more recent research and newer telecommunications speech coding standards.

2.1 Preliminary work

A public domain C-code of the FS1016 is available [1]. This implementation was used as reference to write two C programs (encode and decode) interfaced as Matlab functions (mex functions). These programs are used as reference system to try out different algorithmic options and for the fixed-point analysis (see Section 5). Unnecessary options and functions, such as bit permutation and forward error correction (FEC) were removed. The bits that are unnecessary for a storage application (synchronization, FEC and future expansion) were removed. The bit rate is thus reduced to 4.6 kbps (138 bits / 30 ms frame). The variables needed for the VAD and the CNG (see Section 3) were identified and returned from the FS1016 mex encode function.

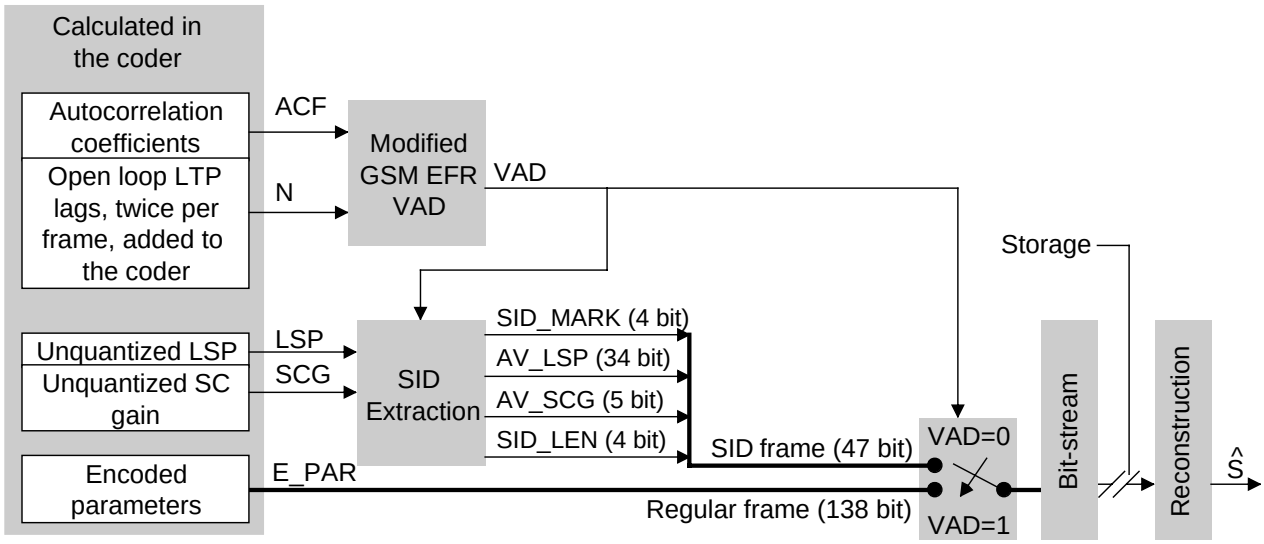

Figure 1: The silence compression scheme adapted to the CELP FS1016 (ACF = Autocorrelation Coefficients, N = Pitch lags, LSP = Line Spectrum Pairs, SCG = Stochastic Codebook Gain, E_PAR = Encoded Parameters, SID = Silence Description. SID_MARK, AV_LSP, AV_SCG, SID_LEN are defined in Subsection 3.3).

3 Silence Compression Scheme

To reduce the bit rate, a silence compression scheme was adapted to the FS1016, as shown in Figure 1. The voice activity detector (VAD) uses the autocorrelation coefficients (ACF) and pitch lags (N), calculated in the coder, to decide upon presence of active speech or background noise. The VAD output is a (1-bit) flag that indicates if active speech was detected (VAD=1) or not (VAD=0). Active speech is coded with the FS1016, producing a regular frame of 138 bits, whereas up to 16 contiguous frames of background noise can be coded using one 47-bit silence description (SID) frame (see Subsection 3.3). The regular and SID frames are written to memory as a continuous bit stream. The decoder reads this bit stream from memory, retrieving the regular and SID frames, to reconstruct either the speech or the background noise.

Silence compression is also used to decrease power consumption, by switching off some computationally expensive parts of the coder during silence.

3.1 Speech and noise databases

For the adaptation and test of the silence compression scheme we have used a subset of the BDSONS French speech database [2], consisting of 30 speakers reading the same story. The original files at 16 kHz were downsampled to 8 kHz. The average duration of the stories is 42 sec. Thus the total length of the speech material is about 21 min. Hereafter this subset will be referred to as LABIS database. Different levels of white noise coming from the NOISEX database [3] were added, generating four sets of 30 speech files for the following conditions: clean speech, 30 dB SNR, 20 dB SNR and 10 dB SNR.

3.2 VAD

We have adapted the VAD scheme used in the GSM speech coders [4] to the FS1016. There exist three different VADs, one for each of the three GSM speech coders: the full-rate (FR), the half-rate (HR), and the enhanced full rate (EFR). We did a comparative study of the three existing GSM VADs [5], and decided to use the EFR VAD because the EFR coder is the most similar to the FS1016 in the speech pre-processing. The EFR VAD was implemented and tested, and then adapted to the FS1016, as explained next. The tone detection function was removed, as it is not needed for storage applications.

The variables needed for the VAD (ACF and LTP lags) were identified and returned from the FS1016 encode mex file, and used instead of those calculated in the GSM EFR. To account for differences in pre-processing between the GSM EFR coder and the FS1016, the ACF calculated in the FS1016 are multiplied by a correcting factor, found by using linear regression and scatter plots, between ACF calculated in the EFR and ACF calculated in the FS1016 [5].

In the FS1016, four LTP lags are calculated (one per subframe) from the coded residual using a close-loop (adaptive codebook) approach. In the EFR coder, the LTP lags are first calculated in an open loop fashion, twice per frame (every two subframes), from non-coded, perceptually weighted speech. These open loop LTP lags are used in the EFR coder for both, reducing complexity of the subsequent closed-loop LTP lag calculation, and as input to the VAD. We have tried different variants of the LTP lags [5] and decided to add an open loop pitch calculation, twice per frame, to the FS1016, for good VAD performance. The open loop pitch calculation will also be used in the FS1016 for reducing complexity of the

adaptive codebook search (see Subsection 4.3). It also allows an earlier VAD decision in order to switch off the computationally expensive codebook searches in the case of VAD=0.

In the EFR VAD there are several parameters and variables that represent time, and are a function of the frame rate period (20 ms). These variable were scaled to take into account the 30 ms frame rate of the FS1016.

3.3 CNG and "Channel" Bit Stream

If we apply a clean speech signal at the input of the VAD, and then we set the signal to zero when VAD=0, then the resulting file is intelligible and of good quality. But if the input speech signal is noisy, zeroing it when VAD=0 would cause an annoying modulation of the background noise, known as "noise contrast effect". To reduce this effect, Comfort Noise Generation (CNG) is used.

When a frame is marked VAD=1 (speech detected) the regular encoder parameters (E_PAR) coming from the FS1016 are passed to the speech reconstruction (through the channel bit stream). When no active speech is detected (VAD=0) the regular coder operation is switched off, and a simpler coder produces a 47-bit silence description frame which is used by the decoder to insert a coarse reconstruction of the background noise. The bit allocation of and SID frame is shown in Figure 1. The SID contains information about the noise average spectrum (AV_LSP) and gain (AV_SCG). As the noise is stationary over longer periods, we allow an SID frame to code silence periods that go from 30ms, to 16*30 ms. The length of the silence period is coded in a 4 bit field (SID_LEN).

The input signal is thus converted to a succession of regular and SID frames which are serialized as a "bit stream" and stored sequentially in the storage memory. Thus, a method to differentiate both frames is needed. We decided to use a marker (SID_MARK = 1110) at the beginning of an SID frame, corresponding to an impossible sequence of the coded LSP values in the regular frames. This approach is efficient from the point of view of bit-rate decrease.

3.4 Test and achieved compression factor

The whole LABIS database, for the four noise conditions was used to test the developed VBR coder. The quality of the decoded signals is considered sufficient for the application. The achieved bit rate for the 21 min. of speech material with the four noise conditions (see Subsection 3.1) is given in Table 1.

SNR	*Amount of storage*	*Bit rate*
10	684 kBytes	4.40 kbps
20	641 kBytes	4.12 kbps
30	618 kBytes	3.98 kbps
clean	607 kBytes	3.91 kbps

Table 1: Achieved bit rate.

4 Functional Block Optimization

4.1 Spectral Analysis and Quantization

LPC analysis is performed by 10-th order autocorrelation analysis (Levinson-Durbin recursion) using high-pass filtering, a 30 ms Hamming window, no pre-emphasis, and 15 Hz bandwidth expansion [1]. The bandwidth expanded LPC coefficients are converted to a set of LSPs and quantized using a 34-bit, independent, non-uniform scalar quantization. Then, two adjacent quantized sets of LSPs are linearly interpolated, obtaining four sets of interpolated LSP, one set per subframe. Each of these LSP sets is converted back to LPC, to be used in the LPC synthesis filter for the codevector searches.

The optimization of the FS1016 spectral analysis and quantization and its implementation on a DSP56001 is presented in [6]. It is based on careful algorithmic optimization and study of the fixed-point quantization effects (see Section 5). Algorithmic optimization deals with the choice and modification of the algorithms, their optimal interrelation in the whole system, and a good match between algorithms and architecture. Finally, implementation of the spectral analysis and quantization block on a DSP56001 results in 0.2930 MIPS [6].

In particular, we have studied different algorithms for LSP calculation and proposed several techniques for complexity reduction [6], [7], retaining the method of [7] for utilization in the FS1016. This algorithm does simultaneous LSP calculation and quantization, using a fast and reliable binary-tree search on the quantization tables. Additionally, it uses the reflection coefficients instead of the LPC coefficients, which is beneficial for fixed-point implementation.

4.2 Stochastic Codebook search

Stochastic codebook search consists of searching the codevector (**c**) that maximizes the following term [8]:

$$\eta = \frac{\left[\mathbf{c}^{\mathrm{T}}\mathbf{H}^{\mathrm{T}}\mathbf{d}\right]^2}{\mathbf{c}^{\mathrm{T}}\mathbf{H}^{\mathrm{T}}\mathbf{H}\ \mathbf{c}} \qquad (1)$$

where **H** is the square matrix, containing the truncated impulse response of the combined synthesis and perceptual filters, and **d** is the target vector (perceptually filtered speech minus zero impulse response of the combined filter).

The numerator term is optimized through the use of backward filtering [8]. For optimization of the denominator calculation (energy of the filtered codevector) several approaches are found in scientific literature and in recent speech coder standards. We have adapted the method described in [9] to the CELP FS1016 structure, obtaining two configurations of the basis vectors. Using the first configuration, the computational complexity was decreased by a factor of 4, with a slight improvement in speech quality, but the bit rate was

increased to 4.73 kbps. With the second configuration, the computational complexity was decreased by a factor of 8, the bit rate was kept to 4.6 kbps, but the speech quality was slightly degraded, with a measured decrease of about 1 dB in SNR. Currently we are working in the use of Algebraic codebook (ACELP) schemes, which is promising for decreasing complexity, but produce degradation at low bit rates due to the decreased number of pulses and pulse position candidates. To overcome this problem, the use of phase dispersion [10] is under consideration.

4.3 Adaptive codebook search

The original FS1016 coder uses the closed-loop search for all (integer and fractional) pitch delays in the odd subframes. This pitch search requires a large computation time. We used a two-stage pitch analysis for complexity reduction, as it is done in recent speech coder standards, such as the ITU G.729 [11]. Open-loop pitch analysis is done first, in order to simplify the pitch analysis by confining the closed-loop pitch search to a small number of lags around the open-loop estimated lags. The open-loop estimate is also processed to favor lag values in the lower range, avoiding pitch multiples.

This work is still ongoing, to evaluate the effects on the performance of the coder (speech quality degradations). Nevertheless, no significant degradation is expected, as this kind of search is currently used in several toll quality speech coders.

5 Quantization Effects and scheduling

Fixed-point quantization analysis is mandatory for a low-power implementation, because the needed word-length determines the size and power consumption of the final implementation.

The study of the fixed-point quantization effects was done for the spectral analysis and quantization block, using the method described in [6], determining the optimum scaling and minimum wordlength needed at every node of the algorithm. The algorithms were then modified to include the scaling, and to fit their dynamic range needs into the dynamic range available in the DSP56001 registers. The same work is still to be done for the codebook searches. The sequence of operations of the spectral analysis and quantization was carefully optimized for implementation on a DSP56001.

Each functional block was implemented on a DSP56001 in assembly language and on a workstation in C language, including the DSP56001 arithmetic effects [6]. This bit-true C program is thus a “model” of the corresponding DSP56001 implementation. Each of these C “models” is interfaced as a Matlab function. It was verified that the DSP56001 implementation and the bit-true C program have exactly the same output under the same input, using the whole TIMIT database (6300 speech files) [12]. After that, the C model is used to measure the performance of the DSP56001 implementation. This bit-true C program was also adapted to the precision used in the FPGA implementation (see Section 6) and used for testing this implementation.

6 FPGA implementation

An FPGA implementation of the spectral analysis and quantization block of the coder is done using a FLEX10k100 from Altera. We have chosen to implement a micro-programmed DSP, using an architecture similar to the one of the DSP56001 [13], with a bit-parallel MAC and separate X and Y data memories, busses and address generation units. This in order to use directly the optimal sequencing of the DSP56001 program already implemented.

The different units were written using synthesizable VHDL, and interconnected, compiled and tested (before synthesis) using the program Renoir and ModelSim from Mentor Graphics. Logic Synthesis was done using Design Analyzer from Synopsys, and placing and routing was done using MaxPlusII from Altera, which is also used for post-synthesis simulation.

MaxIIplus also produces the programming file for the FPGA. An FPGA PC-based development card is used for testing. The loading of the FPGA programming file is done from a C program, as well as passing the data to be processed by the DSP and retrieving the results. These results are then compared, within Matlab, with the results produced by the bit-true C-program implementation of the algorithms running on the PC. Systematic test is done using the TIMIT database.

The architecture of the implemented DSP was defined based on the study of the FS1016 algorithm, its existing DSP56001 implementation, and the architecture of the DSP56001. To decrease size and power consumption, we decided to use a 16-bit architecture, instead of 24-bit as in the DSP56001. The developed DSP had to be able to execute the existing DSP56001 program without too many modifications (ideally with automatic translation). The DSP contains three execution units operating in parallel: the Arithmetic Logic Unit (ALU), the Address Generation Unit (AGU) and the Controller Unit (CU). These units contain several registers, and interface over the system busses with the (X and Y) memories and the Host Interface.

The parallelism of the DSP is fully exploited with instructions that keep all the units busy. In the best case we can do one arithmetic operation and two data moves in parallel, plus calculation of two memory pointers, e.g.:

```
mac x0,y1,a   x:(r0)+,x0   y:(r4)-,y1
```

This operation executes a multiplication of the values contained in the register X0 and Y1 and add the result to the value contained in the accumulator A. At the same time, X0 is loaded with the value in X memory pointed by the register R0 and this pointer is post-incremented by one. Simultaneously, the register Y1 is loaded with the

value in Y memory pointed by the register R4 and this pointer is post-decremented by one.

6.1 Busses

There are three bidirectional data busses. One for the X memory, one for the Y memory, and a global bus, which allows transfers between the Controller Unit and the registers / memory. It also allows transfers from / to the Host Interface. A bus switch allows data transfers between the three busses.

6.2 Memory

The DSP has two independent X and Y memory spaces, Each memory space contains 512 words of 16-bit-wide, asynchronous, static memory (256-words of RAM and 256 words of ROM). The maximum addressable amount is 1024 words per memory space (see Subsection 6.4).

The two ROMs contain the fixed values of the algorithm (filter coefficients, Hamming window, LSP quantization table, bandwidth expansion factors). The position of these value is chosen in order to use modulo arithmetic on the addresses (see Subsection 6.4).

6.3 Arithmetic Logic Unit (ALU)

ALU operations use fractional two's complement arithmetic. The ALU has a 16-bit input / 32-bit output multiplier and a 40-bit adder, with two 40-bit accumulators (32-bit + 8-bit extension). The multiplier output is connected to one of the adder inputs. The A and B accumulators are connected to the other input, and to the adder output to perform accumulations.

Four 16-bit input registers (X0, X1, Y0, Y1) can be independently selected as input of the multiplier. Typically two of these registers can be loaded in parallel. There is also the possibility to put the same register at both inputs of the multiplier, to carry out operations such as X0*X0. Any of these registers can be loaded from the X or Y data bus. The accumulator outputs are independently connected to the X and Y data busses.

There is also a unit, between the adder output and the accumulators, which performs shifting, limiting, rounding, and normalization. The adder is capable of doing one (non-restoring) division iteration.

No logical operations were implemented, as they are not used in the algorithm. The ALU allows the following operations:

- Multiplication with and without rounding;
- Multiply accumulate with and without rounding;
- Rounding;
- Division (one non-restoring division iteration);
- Normalization (one normalization step);
- Addition, subtraction, negation and absolute value.

6.4 Address Generation Unit (AGU)

This unit performs all the storage and calculations needed for addressing data operands in memory. The AGU contains eight address registers (R0-R7), eight offset registers (N0-N7) and eight modifier registers (M0-M7). All these are 10-bit registers, allowing a maximum of 1024 words per memory space. The Rx registers contain addresses, while Nx an Mx registers are used to update or modify the Rx registers. All AGU registers can also contain generic data. The AGU has two independent arithmetic units and can thus generate two addresses per cycle. One unit is reserved for the X memory space, together with the registers R0-R3, N0-N3, and M0-M3, while the other is reserved for the Y memory space together with the registers R4-R7, N4-N7, and M4-M7. Each unit can implement two types of arithmetic: linear and modulo (for circular addressing). Thus the following addressing operations are possible:

- (Rx) : indirect addressing;
- (Rx)+ : indirect addressing with post-increment of the address, with and without modulo;
- (Rx)- : indirect addressing with post-decrement of the address, with and without modulo;
- (Rx +/- Nx) : indirect addressing with a positive or negative offset Nx, with and without modulo.

6.5 Host Interface

This unit allows data transfer between the PC and the DSP. The data width of the development card is 8 bit, while the DSP data is 16-bit wide. Two 8-bit registers are written by the PC, and are read as a concatenated 16-bit register by the DSP. Similarly, a concatenated 16-bit register is written by the DSP, and read as two 8-bit registers by the PC. An 8-bit third register (Status DSP) can be read by the PC, containing flags issued by the DSP. Other hardware flags are used to avoid either the DSP or the PC to overwrite the registers or to read invalid data.

6.6 Controller Unit (CU)

The Controller Unit fetches and decodes the instructions contained in its internal program ROM, and generates the different signals controlling the other blocks, necessary for the instruction execution. The CU also performs hardware loop control and conditional branching. Every instruction is executed in one clock cycle. The fetch and decode of the next instruction are done during the execution of the current instruction.

The assignment of the binary code to the instructions was optimally organized [14] to minimize the amount of decoding logic, and thus the decoding time. All the instructions are 24-bit wide. The program size is limited to 1024 instructions, but it is easily extendable with minor modifications of the hardware.

The program counter contains the address for the program ROM. A program controller manages this program counter, incrementing it, or loading it in case of a branch or a loop. When a conditional jump is decoded, if the tested condition is true, the destination address is loaded in the program counter. For hardware loop (do) instruction the program controller keep track of the number of times the loop has been executed, and it exits the loop when it was executed the desired number of times, or return to the start address. Three registers are used to store the loop counter, the start address and the end address. These registers are stored in a hardware stack when a do instruction is encountered, allowing nesting of do instructions. The execution of branching and hardware loops implies no overhead other than fetching and decoding the instruction itself.

6.7 Programming the DSP

The implementation of the spectral analysis and quantization algorithms is the final step in the FPGA implementation. Ideally, the program already available in DSP56001 assembly language (see Section 5) should be automatically translated into binary code of the developed DSP. In practice, some instructions of the DSP56001 were not implemented because they would imply too much hardware, with respect to the hassle / overhead of implementing them in two instructions.

A rudimentary assembler program was written using C language, in order to translate the DSP56001 assembly language into DSP binary code. This program eases the programming of the DSP, allowing a higher level than using the DSP machine language. The assembler input is an ASCII file containing DSP mnemonics, and it outputs a file in MIF format which is used by the MAXPlusII for initialization of the program ROM.

Before using the assembler, the DSP56001 program is modified manually, to take into account the 16-bit architecture (instead of 24-bit), the reduced memory size (2*512 words instead of 2*64k words), and the substitution of instructions not available in the developed DSP.

6.8 Status of the work

The DSP is basically finished and operational. The host interface was extensively tested using the whole TIMIT database. The filtering, windowing, auto-correlation, and part of the LPC calculation were implemented and tested. The results calculated in the DSP and calculated in the bit-true C program are exactly the same. This implementation and testing allowed the detection and correction of different faults of the DSP under development.
The FLEX10k100 contains about 100′000 gates and 4 kBytes of memory. Currently, the developed device uses 91 % of the available memory and 60 % of the logic.

To have an idea of the size of the circuit, a very first VLSI implementation was done: the DSP was synthesized using a CMOS 0.25 µm technology. The resulting die size is about $1mm^2$.

7 Conclusions and further work

This paper describes the elaboration of a variable bit rate (VBR) speech coder suitable for voice storage in portable devices. This coder is based on the CELP FS1016 speech coder modified to add a silence compression scheme and to decrease computational complexity. The different optimization strategies and the FPGA implementation were described. Part of this work is currently being done. Ongoing and future work involves finishing the optimization, scheduling and studying of the quantization effects for the dictionary searches, as well as implementing them using the developed DSP, following the same methodology used for the implementation of the spectral analysis block. The implementation of specific hardware units to ease / optimize these dictionary searches is also under consideration.

References

[1] http://www.plh.af.mil/ddvpc/celp.htm (Version 3.2a).

[2] http://www.icp.grenet.fr/Relator/french/bdson.html

[3] http://www.speech.cs.cmu.edu/comp.speech/Section1/Data/noisex.html

[4] http://www.etsi.org/

[5] S. Grassi, “A Silence Compression Scheme for the CELP FS1016”, IMT internal report, IMT - University of Neuchâtel, Switzerland, June 2000.

[6] S. Grassi, "Optimized Implementation of Speech Processing Algorithms", Ph.D. Thesis, University of Neuchâtel, Switzerland, November 1997.

[7] S. Grassi, M. Ansorge, F. Pellandini, "An Algorithm for Fast Direct Calculation of Quantized LSP Parameters", Fifth Bayona Workshop on Emerging Technologies in Telecommunications (COST 254), Bayona, Spain, pp. 158-162, Sept. 1999.

[8] I. M. Trancoso and B. S. Atal, "Efficient Search Procedures for Selecting the Optimum Innovation in Stochastic Coders", IEEE Trans. ASSP, vol. 38, No. 3, pp. 385-396, March 1990.

[9] Cheung-Fat Chan, "Fast stochastic codebook search through the use of odd-symmetric crosscorrelation basis vectors", Proc. ICASSP'95, Vol.1, pp. 21 -24.

[10] R. Hagen et al., “Removal of sparse-excitation artifacts in CELP”, Proc. ICASSP’98, Vol. 1, pp. 25-28.

[11] http://www.itu.int/

[12] J. Garofolo et al., "Darpa TIMIT, Acoustic-phonetic Continuous Speech Corpus CD-ROM", National Institute of Standards and Technology, NISTIR 493, Oct. 1990.

[13] *DSP56000/DSP56001 Digital Signal Processor User's Manual*, DSP56000UM/AD Rev.2, Motorola Inc., 1990.

[14] M. Matthey, “VHDL Implementation of a CELP Speech Coder” (in French), IMT internal report, IMT- University of Neuchâtel, Switzerland, July 2000.