AN OPTIMIZED TTS SYSTEM IMPLEMENTATION USING A MOTOROLA STARCORE SC140-BASED PROCESSOR

Dragos Burileanu[*], *Andrei Fecioru*[*], *Dragos Ion*[*], *Madalin Stoica*[**], a*nd Costel Ilas*[**]

[*] Faculty of Electronics and Telecommunications, "Politehnica" University of Bucharest, Romania
[**] Motorola Software Center Romania

ABSTRACT

One of the key technologies for spoken language processing is the automatic synthesis of speech. For an important number of current or future applications (including various telecommunication services and voice interfaces for mobile devices), the synthesis of good quality speech starting from unrestricted text as well as the efficient implementation of the corresponding synthesis systems still represent very difficult tasks. This paper presents an optimized implementation of a text-to-speech synthesis system for the Romanian language using a Motorola development platform built around a StarCore SC140-based processor. The paper emphasizes the key requirements for such an embedded implementation (especially the intelligibility/footprint combination), the problems that were encountered and the solutions found to these problems.[1]

1. INTRODUCTION

The last two decades brought significant advances in spoken language processing technologies and lots of speech-enabled applications became available. The idea that low-cost friendly and natural speech interfaces will become soon a necessity is today largely accepted [5].

One of the key technologies for spoken language interfaces development is the automatic synthesis of speech. In response to the increasing demand for good quality speech output, one may notice that many highly intelligible text-to-speech (TTS) systems are now available. But most of the commercial TTS systems represent completely software solutions and use large speech unit databases. This is perfectly acceptable for PC/multimedia applications, but not for embedded speech applications such as cellular phones or PDAs. The new generation of small-scale computing devices has severe resource constraints, since it is not always possible to gain access to a central computer or mainframe that could manage a "large" TTS system; low CPU resources and small memory footprints are mandatory. This is way the efficient implementation of a TTS system for this kind of application is a difficult task.

Several companies currently provide embedded TTS solutions for their speech interfaces [4]. Some of these systems are formant-based, which results in rather poor quality speech and are mainly used for cost effective devices.

On the other hand, concatenative systems can produce high quality speech, but they generally need large speech segment databases; most of these solutions are therefore based on client/server (telephony) configurations [6]. Reduced size databases (to meet the memory constraints) and also simplifications in TTS systems' architecture are usually adopted [3].

The aim of the paper is to present the main implementation issues in developing an embedded version of a TTS system for the Romanian language, starting from a software ("reference") one. The reference TTS system and the simplifications performed in order to achieve a viable hardware implementation are briefly described in Section 2. Section 3 presents the embedded implementation on a Motorola DSP platform, with emphasis on the main attributes of the chosen processor and the restrictions imposed by the particular platform, the problems that were encountered and the solutions found to these problems. Section 4 concludes the paper with conclusions and final remarks.

2. A SIMPLIFIED TTS SYSTEM ARCHITECTURE

Our collective started a few years ago the project of developing a complete TTS synthesis system for the Romanian language. A concatenative approach was followed, using diphones as the basic acoustic segments and a dynamic unit selection procedure. The system presently includes a two-level parser for preprocessing and syntactic/prosodic analysis, a neural network-based letter-to-phone converter, a dynamic unit selection procedure (a spectral-distance measure is used to find an

[1] This research was supported by Motorola Software Center Romania

optimal sequence that minimizes acoustic discontinuities at unit boundaries), and a modified PSOLA algorithm for diphone concatenation and speech signal generation [1], [2], [3]. This software version runs in real-time on a medium-PC.

But hardware implementing all the above-mentioned modules leads to a very complex system with prohibitive memory and computational requirements. For the embedded version our main goal was to create a TTS system capable of generating highly intelligible speech while keeping the computational and memory requirements as low as possible.

The simplifications performed on the reference system finally led to the version presented in figure 1. The main features of this simplified structure are next briefly described.

- The preprocessor performs several basic operations on the input text: the replacement of some non-native Romanian letters with graphemes corresponding to their basic phonetic values, input text segmentation, substitution of upper cases into lower cases, and the removal of hyphens and other common punctuation marks. We mention that the embedded version presented in this paper does not allow yet special constructions like abbreviations, acronyms and numerals.
- The letter-to-phone converter is based on a parallel neural network architecture, and is practically the only module unmodified with respect to the complete software version [1]. Starting from a basic set of 33 phones and an articulatory description of them, we use a number of 30 fully connected feed-forward neural networks. Each of them is associated to one articulatory characteristic and is capable of determining whether the phone associated with the current input grapheme has the corresponding articulatory features or not, based on a specific binary codification table. After training, the converter is able to provide as his output the complete phone string in the testing phase.
- We carefully designed a minimal acoustic database consisting of 634 diphones. The segmentation from the speech corpus (speech files recorded at 8 kHz with 16 bit precision) was manually performed. Finally, diphones were labeled and stored in digital format.
- Using the phonetic transcription and the diphone database, the speech generation module (see figure 1) concatenates the corresponding segmental units in order to obtain words and sentences. To ensure a smooth transition between adjacent phones, a Hamming weightening function is applied on the last 100 samples of the current phone and the first 100 samples of the next one.

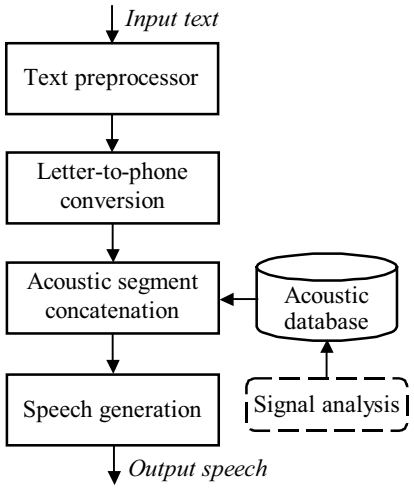

Figure 1. The simplified TTS system architecture

3. THE EMBEDDED IMPLEMENTATION OF THE TTS SYSTEM

3.1. Memory requirements

The main concern when creating an embedded implementation of a TTS system is related to memory requirements figures. In our case, the program's data (stored as plain constant tables within the program) comprises two main components:

- the acoustic database;
- the coefficients used by the neural networks that compose the phonetic converter.

For the acoustic database (see the previous section), an A-law voice-coding scheme reduced the memory requirement figure to 692 kB.

The coefficients for the neural networks were initially stored as simple look-up tables of floating point values (32 bits for each coefficient), further requiring additional 194 kB of available memory space.

However, the chosen processor employs a fixed-point architecture, making the floating-point computations extremely time consuming. For this reason the original algorithm was transformed from floating-point to fixed-point, meaning that all the coefficients were translated into their fixed-point equivalents. Tests showed that a precision of only 16 bits is sufficient not to alter the output of the phonetic converter. This operation lowers the memory requirement figure for the coefficient tables to half of its original size (from 194 kB to 97 kB). Considering the fact that the code-size reaches 107 kB, the total memory required for the program to run is around 900 kB of memory (including the memory space for the temporary variables). With a final memory footprint under the 1MB threshold the application can easily face scalability issues.

3.2. A brief hardware description

The hardware platform chosen for this version of our TTS system is a Motorola MSC8101 ADS development board built around the MSC8101 processor (based on a StarCore SC140 core). This platform incorporates the following features [8]:

- 512 kB on-chip memory space;
- an external SDRAM memory module with a capacity of 16 MB;
- two RS-232C compatible communication ports (used in our application to establish the DSP–PC communication link);
- a 16-bit audio-codec with a sampling frequency of 8 kHz (used for real-time play-back of the output synthesized speech).

Our newly developed TTS system also benefits from the high computational performances offered by the SC140 DSP core (a highly parallel architecture as depicted in figure 2).

The core's main architectural features consist in a data arithmetic logic unit (DALU – which includes four ALUs) and an address generation unit (AGU – containing two ALUs and a bit mask unit – BMU). Other features like separate buses for the data and program memory spaces, hardware support for both fractional and integer data types and a rich 16-bit wide orthogonal instruction set, allow the SC140 to perform up to 4 MMACS (million multiply-accumulate operations per second) for each megahertz of clock [7].

This considerable computational power helped us coping with the most time consuming parts of our application as outlined in the next section.

3.3. Application architecture

The application implements a three-level hierarchical architecture as shown in figure 3. This type of architecture ensures maximum portability since only the functions on the lowest level are platform dependent.

The bottom level implements the functions that handle all hardware setup procedures: initialization of the interrupt mechanism, initialization of the DSP's internal timers, setting up the DSP – audio-codec communication link as well as the communication between the DSP and the PC host (via serial interface).

The middle level handles the communication protocol routines, ensuring a reliable connection between the host PC, the DSP and the audio-codec for real-time playback of the output speech. The input text is transferred from the host PC to the DSP using a RS-232C compatible serial communication interface. The user interacts with the system through a standard windows application developed in Microsoft Visual C++ 6.0.

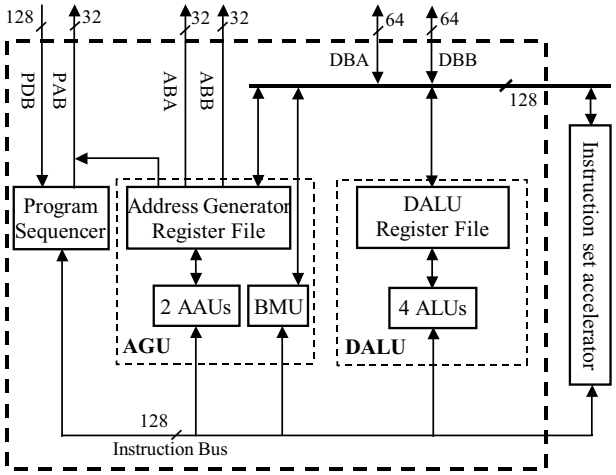

Figure 2. Simplified block diagram of the SC140 DSP core (adapted from [7])

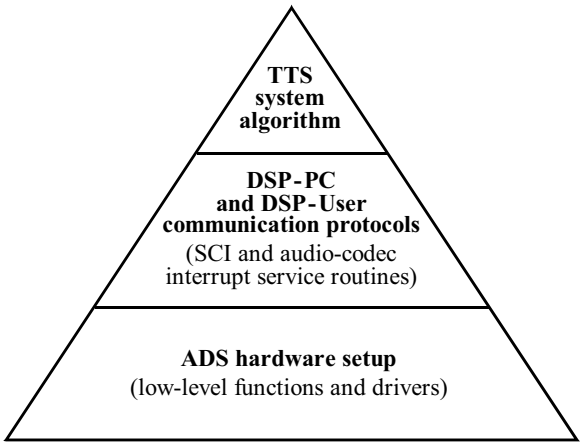

Figure 3. The embedded TTS system architecture

At the top of this structure stands the actual TTS algorithm. As it was described in Section 2, this algorithm basically comprises the text preprocessor, the phonetic converter and the speech generation module. The tasks performed at this level include:

- text preprocessing and transformation of the input stream into its corresponding phonetic transcription;
- decompressing the speech samples extracted from the acoustic database using the corresponding A-law decompression scheme;
- applying the Hamming weightening function;
- sending the processed samples to the audio-codec for playback.

Note that all routines are implemented in high-level C language in order to ensure easy portability on other similar platforms.

3.4. Computational load issues

Besides the memory requirement problem, another critical issue is the one related to the execution speed of our application. As mentioned, the main goal was to obtain an embedded TTS system capable of delivering in real-time highly intelligible output speech.

The profiling tools clearly indicated that the most computational intensive routine is the one that handles the phonetic conversion of the input text. This routine fills more than 80% out of the entire execution time. All optimization attempts aimed to speed up this section of the program.

First, all the neural network coefficients were cached into the internal memory, thus reducing the stalling time related to data access through external memory busses.

Second, the phonetic converter routine was adapted to the specific architecture of the SC140 core, by transforming the initial floating-point algorithm into a fixed-point equivalent. This transformation implied the following steps:

- transforming the neural network coefficients from the 32 bit floating-point format into their 16 bit integer equivalents. This was done by simply multiplying the floating-point value with 1024 and then converting it to an integer;
- replacing the floating-point multiplication operation with its 16-bit fixed-point equivalent (consisting of an integer multiplication and a 10-bit right shift);
- the non-linear neuron activation function was implemented using a simple look-up table of 97 integer values.

The effects of these optimizations were dramatic. The coefficient tables now take half of their original size (97 kB). The classic floating-point multiplication took 336 cycles to compute, while the modified fixed-point version of the same operation requires only 4 cycles. The speed gained at this point is significant considering that the algorithm performs this operation 96,116 times (for an input text of 4-letters long). The initial computation of the activation function required no less than 15,241 cycles. This function was replaced with a plain indexing operation within a vector that takes no more than 16 cycles.

The overall figures show obvious improvement: the cycle count for the floating-point version reaches 2.5e+08 cycles (or 833 ms) for an input text of 4 letters. In other words, for an input text of about 100 characters the processing time evolves around 17 seconds. On the other hand, the fixed-point implementation only requires 13,504,736 cycles (about 22.5 ms) for the same input text of 4 letters. Thus, the cycle count is reduced 18 times and the processing time corresponding to an input stream of 100 characters long is well under one second.

4. CONCLUSIONS

This paper presented the design philosophy and the main implementation issues in developing an embedded version of a TTS system for the Romanian language, starting from a complete software version. Due to the particular requirements implied by the use of embedded systems, the structure of the reference synthesis system has been first reduced to a minimal configuration. Then, this simplified system was ported on a Motorola DSP platform. Special attention was paid to the amount of available resources and the way these resources are managed.

This embedded version provides practically an unlimited vocabulary and is capable of synthesizing speech with a very good intelligibility. Therefore, the system is perfectly suited for a wide variety of applications, ranging from simple prompt generation to warning system controls and consumer applications.

We would like to mention that our collective works now on adding other modules from the software version to the present embedded implementation of the TTS system, taking into account the remaining computational and memory resources.

5. REFERENCES

[1] D. Burileanu, M. Sima, and A. Neagu, "A Phonetic Converter for Speech Synthesis in Romanian", *Proc. of the XIVth International Congress on Phonetic Sciences ICPhS'99*, San Francisco, USA, vol. 1, pp. 503-506, 1999.

[2] D. Burileanu, "Basic Research and Implementation Decisions for a Text-to-Speech Synthesis System in Romanian", *International Journal of Speech Technology*, vol. 5, no. 3, Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 211-225, 2002.

[3] D. Burileanu, A. Fecioru, and D. Ion, "On Automatic Speech Synthesis for Spoken Language Interfaces", *Speech Technology and Human-Computer Dialogue*, Publishing House of the Romanian Academy, Bucharest, Romania, pp. 127-138, 2003.

[4] L. Comerford et al., "The IBM Personal Speech Assistant", *Proc. of the IEEE ICASSP2001*, vol. 1, 2001.

[5] L. Deng et al., "Distributed Speech Processing in MiPad's Multimodal User Interface", *IEEE Trans. on Speech and Audio Processing*, vol. 19, no. 8, pp. 605-619, 2002.

[6] A. Monaghan et al., "Multilingual TTS for Computer Telephony: The Aculab Approach", *Proc. of Eurospeech'2001*, Aalborg, Denmark, vol. 1, pp. 513-516, 2001.

[7] Motorola SC140 DSP: Reference Manual, Rev. 3, 11/2001.

[8] Motorola MSC8101: Reference Manual, Rev. 2, 05/2002.