Telephone Bandwidth Extension Using DIOPSIS 740

By SIPL Team[*]

Signal & Image Processing Laboratory, Dept. of Electrical Engineering, Technion – IIT, Israel

Abstract

The degradation of speech quality using analog telephone systems is caused by band-limiting filters along telephone networks. These filters have a passband from approximately 300 Hz up to 3400 Hz. The goal of the presented system is to demonstrate a novel application of speech bandwidth extension that increases the quality of telephone speech signals while maintaining backward compatibility to currently available telephone networks and telephones. This novel application is implemented using the DIOPSIS 740 platform. The details of the application design and the implementation, performance benchmarks and the verification report, installation and setup instructions, are all presented in this document. The DIOPSIS 740 platform proves to be a very successful choice for speech bandwidth extension implementation due to its powerful DSP processor and rich peripherals set.

[*] Oleg Kuybeda, Yair Moshe, Idan Kepten, Yevgeni Litvin, Jonathan Shimonovich are the team leader, team mentor, and team members, respectively

Table of Contents

Introduction

Public telephone systems reduce the bandwidth of the transmitted speech signal from an effective bandwidth of 50-7000Hz to a bandwidth of 300-3400Hz. The reduced bandwidth leads to a characteristic thin and muffled sound of the so called telephone speech. A unique speech bandwidth extension application, in which the transmission from and to the talker's handset is analog, intended for the public telephony system, is demonstrated in this work. This system is based on a research thesis carried out at SIPL [1] and published in [2].

The input speech signal is analyzed and its high frequencies are represented by a relatively small amount of data (750 bits per second). The low frequencies are transferred to the receiver and at the receiver end the wideband speech is reconstructed using a concise representation of the high frequencies and the low band signal. The concise representation of the high frequencies is based on a frequency selective spectral linear prediction technique [4].

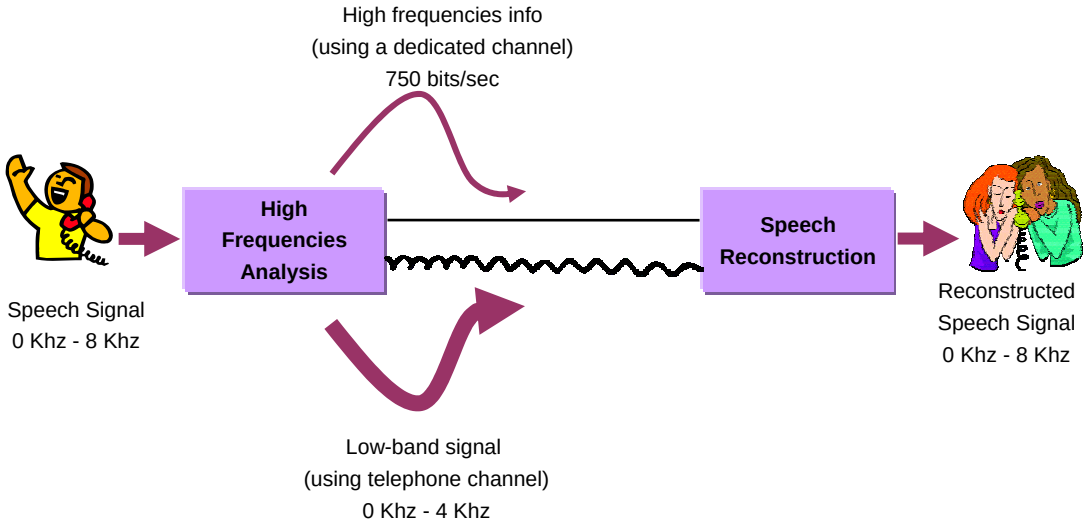
This document presents the system based on the DIOPSIS 740 platform implementing the speech bandwidth extension (SBE) algorithm. The document contains the following sections:

- *System Layout* – Introduces the application that embodies the functionality of the bandwidth extension system on a single DIOPSIS JTST board.
- *Using the DIOPSIS 740 Development Board* – Presents how DIOPSIS processing power and peripherals are used in the application.
- *Memory Management* – Presents challenges imposed by the large memory demand of the application as well as memory management strategies used in the SBE application.
- *ARM-to-mAgic communication* – Describes the mechanism for moving input/output data to and from mAgic.
- *Design and Development Strategies* – Describes the guidelines we followed during the design and development of the system.
- *Encountered Difficulties* – Presents pitfalls encountered during the development.
- *Performance* – Presents DSP processor computational power utilization together with memory requirements.
- *System Verification*
- *Installing the Application* – Shows how to install the application on the DIOPSIS 740 board and how to test it in real-time.
- *Troubleshooting* – Summary of encountered problems and their workarounds.

System Layout

Figure 1 depicts a conceptual layout of the system in the basic configuration. In this configuration, the concise representation of high frequencies is transferred to the receiver using a dedicated data channel.

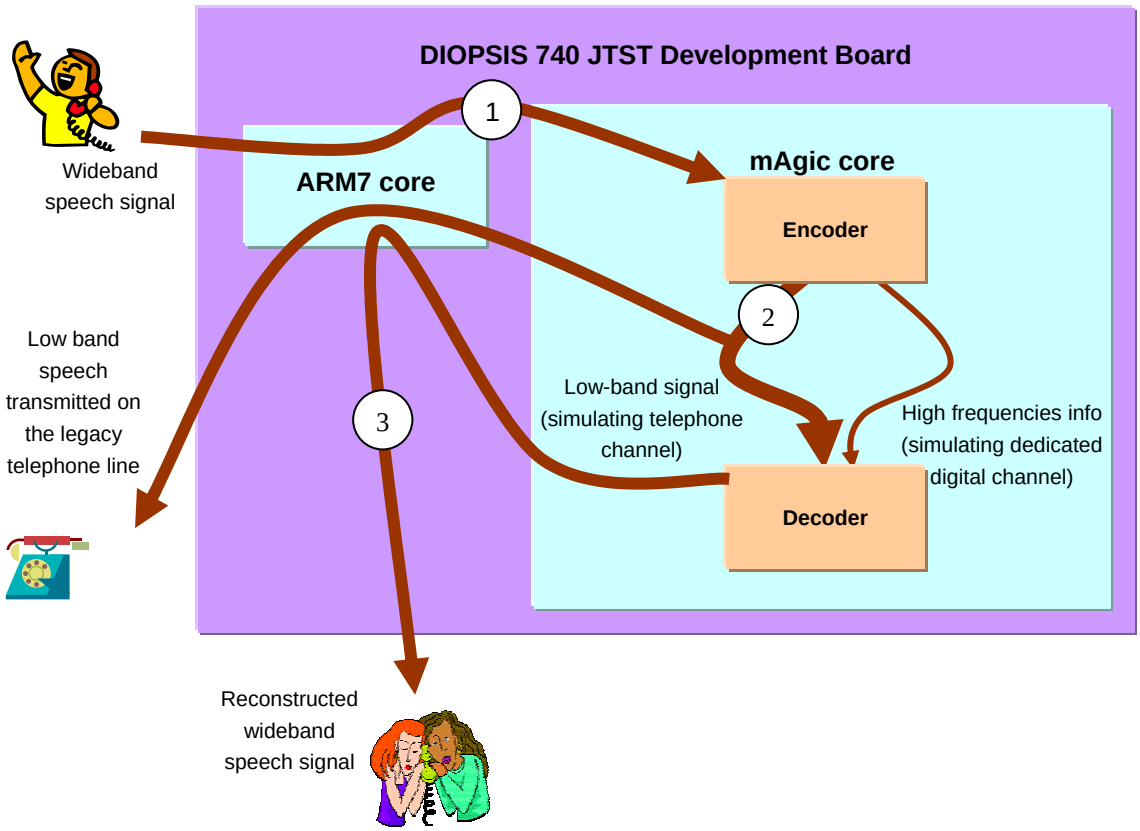
Figure 1. System layout



We used a single DIOPSIS 740 JTST development board. The presented implementation demonstrates both the encoder and the decoder modules of the system running simultaneously in real-time. The telephone line audio signal and high frequency ancillary info are short circuited internally within the mAgic core. Figure 2 depicts system layout as implemented on the board.

Embedding of digital high frequencies information into the lowband signal transmitted on the telephone line, thus canceling the need for dedicated digital line is possible. Such technique is called data embedding and is based on watermarking concepts. It can be used in the current system. More information on this technique and ways of possible integration into the presented system is available in [1], [2] and [3].

Figure 2. Implementation layout



The ARM core acquires a wideband input speech signal from the analog audio input line. It passes the input speech signal to the mAgic core. mAgic feeds the input signal into the encoder that analyses the highband of the input speech signal and produces a lowband audio signal suitable for the telephone line transmission together with concise digital high frequencies information. The lowband audio signal together with the highband frequencies information are fed directly into the decoder The decoder reconstructs the highband of the speech and recombines it with the lowband signal received from the simulated telephone line. The output of the decoder is the reconstructed wideband speech signal which is transferred back to the ARM core that outputs the reconstructed speech to the analog output audio line.

The presented solution is based on research done in the SIPL laboratory [1] [2] accompanied by MATLAB code. C version of the SBE system was created on the DIOPSIS 740 platform. Special effort was made to implement efficient resource management in order to fit within strict constraints of memory and processing bandwidth of the real-time platform. While starting with a plain C implementation of common DSP algorithms, we gradually moved to Atmel's DSP library [18] counterparts. The improvements achieved both due to the improved memory management and due to the usage of DSP library functions, are presented in the *Performance* section.

Encoder/Decoder Structure

Figure 3 depicts the conceptual block diagram of the encoder. Wideband speech signal is fed into the encoder. In the lower branch high frequencies of the input speech are analyzed using selective linear prediction (LP) method. LP coefficients are transformed to the line spectral frequencies (LSF) representation. In the upper branch of the encoder diagram the input signal is downsampled to the regular telephone line bandwidth. The wideband excitation signal is generated from the downsampled speech. Based on the LSF coefficient and the excitation signal the gain is estimated. The quantized gain and the LSF codebook index are transmitted to the decoder. These two parameters, together with the narrowband speech transmitted over the legacy telephone line are sufficient for the reconstruction of the speech high frequencies.

Figure 3. SBE encoder structure overview

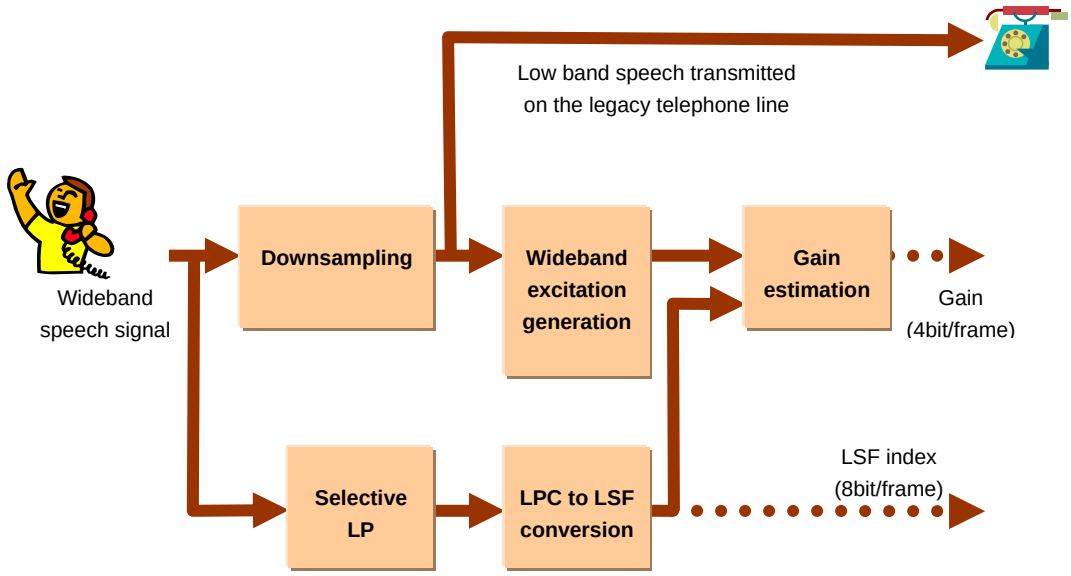
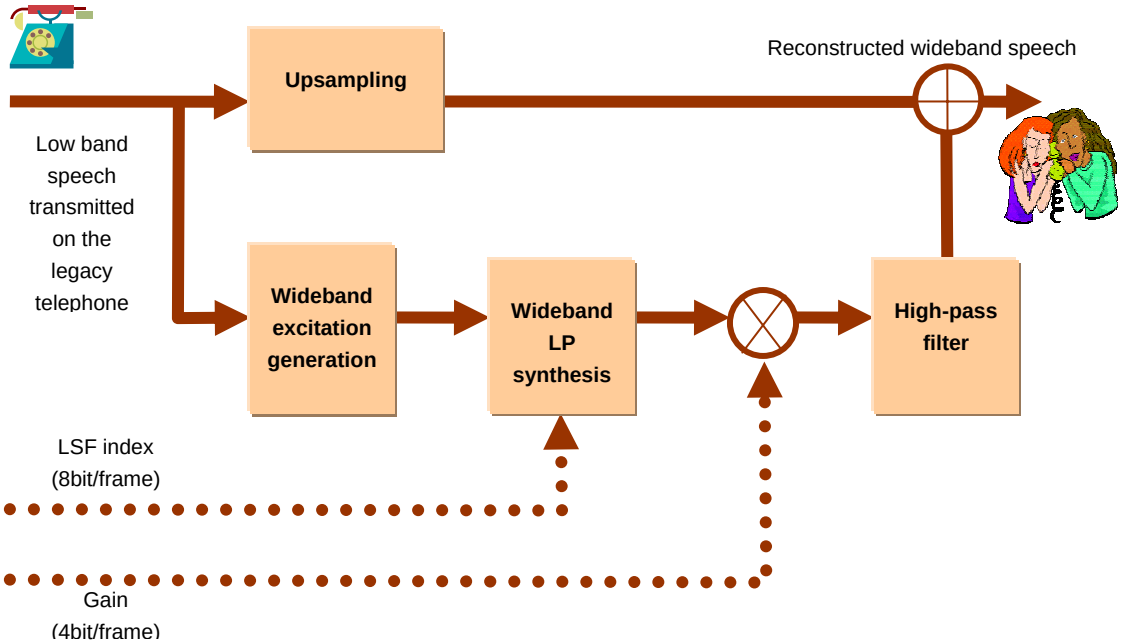


Figure 4 depicts the conceptual block diagram of the decoder. The narrowband speech signal is fed into the decoder together with gain and the LSF index. In the lower branch the high frequencies of the speech are reconstructed. First the wideband excitation signal is generated from the narrowband speech. Then, the LP synthesis is performed (the LP coefficients are reacquired from the LSF coefficients) followed by gain adjustment. Finally, the reconstructed high frequency components are recombined with the narrowband speech signal.

Figure 4. SBE decoder structure overview



Detailed explanation on the internals of the encoder and the decoder can be found in Appendix A and in [1], [2].

Using the DIOPSIS 740 Development Board

The DIOPSIS 740 development board [16], [17] provided a friendly platform for the development of our application, both from the processing power point of view and the peripherals provided on the board.

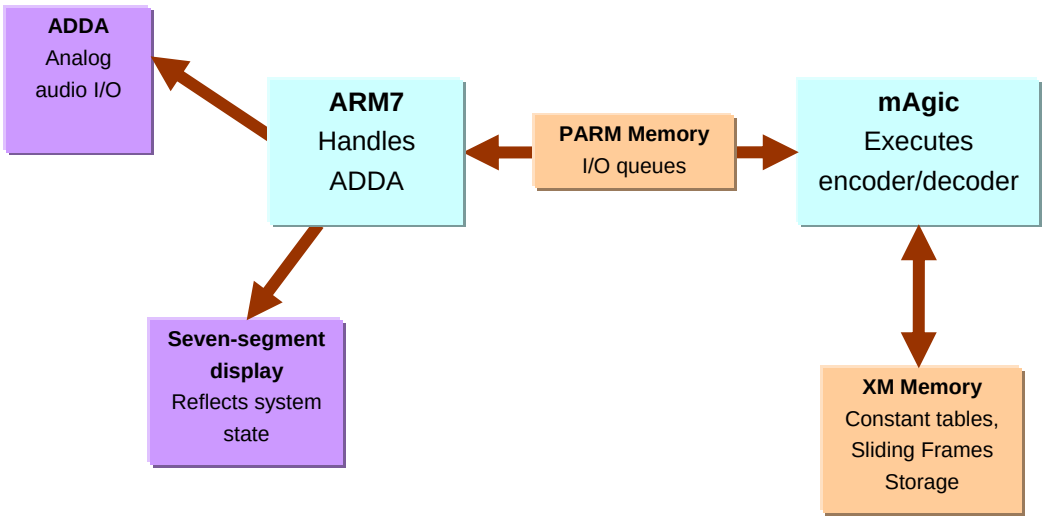
The powerful DSP processor in combination with a highly optimized and rich DSP library [18], allowed us to achieve, and even surpass, the performance required for this real-time application. Only half of the computational power of the mAgic processor is actually used.

We found the seven-segment display useful for displaying various run-time conditions.

However, working with the simulator, mostly due to slow execution times and inexact DMA simulation, was not always convenient. Using the command-line version of the simulator in unsynchronized mode partially shortened the simulation time.

Figure 5 depicts various blocks mapped to corresponding hardware components.

Figure 5. Mapping of application blocks to hardware components

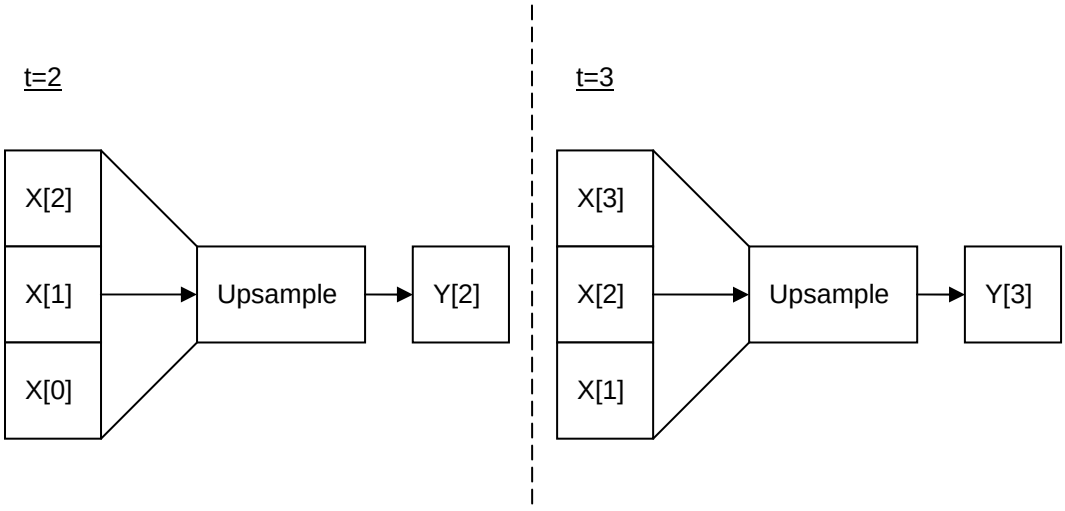


Memory Management

Almost every block in the encoder/decoder introduces an inherent delay of one frame (please notice the headings in most of the blocks in Figure 16 and Figure 17 in the *Appendix A*). These delays are required by the algorithm structure. As a consequence, a significant amount of memory is required to store the delayed signals.

To demonstrate this, let's examine the following example: prior to downsampling a single frame it should be filtered with a lowpass filter to prevent aliasing. To avoid artifacts adjacent to the beginning and the end of the frame, the signal is delayed by a single frame and concatenated with the previous and the next frames. Only then, the filter is applied to the sequence of three frames and the signal is downsampled. This pattern repeats itself almost in every block both in the encoder and the decoder.

Figure 6. Upsampling using a sliding window of frames



Naturally, some memory has to be allocated in order to keep the adjacent frames. We'll refer to this memory as "sliding frames storage". The amount of memory required for the sliding frames storage is fairly large and can not be sustained only in mAgic's internal data memory.

Extensive memory demand for temporary data, constant tables (mostly codebooks) and sliding frame storage made efficient memory management important for the success of this application. Several memory usage strategies were formulated:

- Call stack is used for temporary data storage to allow different blocks of the algorithm to reuse temporary memory space.
- Small constant tables are placed into page 3 memory region in order to free pages 0-2 as much as possible, thus preventing the call stack from growing into page 3. Keeping the call stack out of page 3 memory region allows us to use DSP library routines safely since most of them can not read and write to the single ported memory in the same invocation.
- Large constant tables (codebooks) are placed into mAgic's external memory. ARM sets up the constant tables during the system start up using the mAgicArmDataExchange library.
- External memory is used to keep the sliding frames storage for all algorithm blocks.

For more information on the memory system of the DIOPSIS 740 board please refer to the *Memory Organization* chapter in [17].

Global Variables

We found no synax for defining C external variables so that the compiler would assume other than p0.* prefix (i.e. globals located in page 3 would be treated as located in page 0 thus causing linkage error). In order to address global variables located in page 3 or page 4 memory space from C modules, other then the module containing the definition itself, a special effort had to be made. Special inline assembler macro (`SYM2PTR`) was created that explicitly overrides SLAMP pointer page field. We use this macro to get a correct pointer to the required symbol.

External Memory

Managing data in the external memory turned out to be not as straight forward as it seemed at the beginning due to the following pitfalls:

- All external memory variables had to be defined in a single C file. Otherwise the linker would locate these variables in overlapping locations. It appears that the linker resets allocated memory pointer for each C module.
- Macros found in magic.h for managing DMA assume only direct access to symbols. They do not provide any way for indirect address calculations as required for accessing data in complex data structures, such as accessing individual array within two dimensional array. This functionality is required for managing the sliding frames storage.

In order to manage the sliding frames storage in the external memory, a set of inline assembler macros was created that allowed using integers as external memory pointers. First, integer representation (linear address) of the symbol located in the external memory is acquired. This address can then be easily modified using integer arithmetic. Then, DMA transaction is initiated using the newly calculated memory address.

In our initial trial we used SLAMP pointers to address external memory data structures. Although this approach gives the required flexibility in manipulating addresses and is well supported by the C compiler, it had to be neglected since only 8K of the external memory can be addressed in this fashion.

Large constant tables (codebooks) are located in the external memory as well. Only certain part of the codebook table is transferred to the internal mAgic memory at a time. To set up the DMA transaction of a sub-block within the codebook, the base+offset must be specified. We use the same techniques (i.e. integer as an external memory pointer) to address a certain sub-block within the codebook table.

Since image loader can not initialize constant tables in the mAgic external memory, the constant table values are copied from ARM's memory to the mAgic external memory using mAgicARMDataExchange library at the application start up.

Sliding Frames Storage

Let's examine a common case (depicted in Figure 6) when an algorithmic block requires a sliding frames storage of three frames long. The sliding frames storage is managed as a queue of three frames. Incoming frames are added to the head of the queue (X[3] is added at t=3) and the frames at the tail of the queue are discarded (X[0] is discarded at t=3). The block accesses all three frames centered around X[2] (at t=3) (in this common case every block introduces an algorithmic delay of 1 frame).

Each sliding frames storage structure is maintained using two integer values: base pointer and current head element index. At each time slice the current head index is advanced using modulo arithmetics.

ARM-to-mAgic Communication

Input sampling and audio output are configured according to the ADDA example provided with the SDK.

Figure 7 depicts audio signal flow between two cores. Input signal, sampled at 16 Khz, passed to the mAgic core. After the signal is being encoded, the narrow band version of the signal is passed back to the ARM core to be transmitted to the telephone line.

Figure 7. Data flow between ARM and mAgic cores

Since it is impossible to configure different output channels of the ADDA to operate at different frequencies, we operate all channels at 16 KHz frequency. The lowband signal, intended for the telephone channel, is upsampled prior to sending it to ARM (still, it lacks the 4 KHz to 8 KHz frequency components).

Both the narrowband signal and highband information are fed immediately into the decoder. The reconstructed wideband signal is being transferred back to the ARM core which sends it to the analog output line.

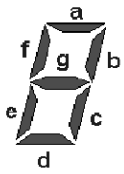
Three 256 sample queues located in PARM memory are used to support this data flow (one for each signal flow illustrated in Figure 7). These queues keep sample data in signed 32 bit integer format. ARM uses this queuing mechanism in a 'sample wise' manner, i.e. each sample acquired from the A/D is placed on the ARM-to-mAgic queue at every sampling point, and at every sampling point a data sample is moved from mAgic-to-ARM queue to the D/A. From mAgic's perspective, the queuing mechanism works in a frame-wise fashion, i.e., the samples are acquired or written back in groups of 256 samples.

In case there are not enough samples available on the queue for mAgic to start processing, mAgic will busy wait until all 256 samples become available. Once the entire frame is ready, it is immediately evicted into mAgic data memory and the ARM-to-mAgic queue is cleared so that new samples can be written to it.

In case that the ARM-to-mAgic is full or, on the other hand, there are no samples available for the output (should never happen because it would mean that mAgic is not keeping up the pace) then the seven segments LED display will indicate which queue is not functioning.

Figure 8 shows the seven segments LED layout. 'f' segment will turn on if the ARM-to-mAgic queue is full. 'g' segment will turn on if the mAgic-to-ARM reconstructed signal queue is empty and 'e' segment will turn on if the mAgic-to-ARM telephone line signal queue is empty. It should be stressed that the seven segments display is used for the debugging and malfunction detection purposes only.

Figure 8. seven segments LED layout



Design and Development Strategies

The following guidelines were formulated and used during the design and the implementation of the system:

- A Major part of the system is implemented in C. Some parts of code are written in inline assembler. (Inline assembler was used mostly for pointer manipulations and custom DMA invocations as described in the *Memory Management* chapter).
- DSP library routines are used extensively due to highly efficient implementation that is incomparably faster and results in much smaller code size then regular C implementation.
- eCos OS is used, mostly due to its stdio library that allowed easy message output of textual data throughout development/debugging process. We would've used MARMOS, had it supported stdio like output to the MADE simulator output window and gdb output window, when working with the development board.
- We used command line (armsim) simulator in the non-synchronized mode to overcome poor simulator performance which is inappropriate for debugging large systems under MADE environment (speedup of approx. x30 compared to the MADE environment). Auxiliary debug library was developed to allow 'debug prints' from mAgic that allowed easier debugging with the command line simulator.
- Each algorithmic block is implemented in a separate C file.

Additional memory usage strategies can be found in the *Memory Management* chapter.

Encountered Difficulties

Below is a summary of the pitfalls encountered throughout the development and debugging process:

Variable Allocation Problems

- We didn’t find a way to reference global variables defined in page 3 or 4 from C modules, other than the module in which the global is declared. It seems that when using `extern` keyword, the compiler always assume p0.* variable. In

order to address global variables located in page 3 or page 4 memory space from C modules, other then the module containing the definition itself, a special effort had to be made. A special inline assembler macro (`SYM2PTR`) was created that explicitly overrides SLAMP pointer page field. We use this macro to get a correct pointer to the required symbol.

- Complex data structures in the external memory can not be addressed using the existing macros in the `magic.h`. E.g., addressing a certain array in the two dimensional array located in the external memory is not possible. A set of inline assembler macros was created that made it possible to use plain integers as external memory pointers and allowed explicit memory address calculations using plain integer arithmetcs.

- Defining multiple external memory variables in several different C modules causes the linker to allocate these variables in the overlapping memory spaces. To workaround this problem, we defined all external memory variables in a single C module.

DSP Library

- IIR filtering is used to shape excitation signal into needed spectral envelope (see WB LP Synthesis block in Figure 17). The IIR coefficients change every frame to reflect a new spectral envelope that is updated every frame. Changing the filter coefficients on the fly caused a disturbing noise in the resulting signal.

  In order to produce a clean undisturbed signal with new filter coefficients, the filter was re-initialized but first half of the previous input frame signal was filtered with the new coefficients before filtering the current frame data. This way the filter coefficients were updated and the transient effect of re-initializing the filter is avoided. Using this approach increases IIR filtering time by the factor of x1.5.

  IIR filter function that allows continuous filtering while allowing coefficients modification could be useful. It would spare the overhead of bringing the filter to its steady state. IIR filtering consumes 19% of the total decoder execution time. According to Amdahl's law, the decoder performance would have improved by 7%.

- There is no vector abs function present in the DSP library. We had to implement the rectification of the signal using a loop written in C.

- Different DSP library functions misbehave if the processed data is located in single ported memory, i.e. page 3 and page 4 memory regions. It is safe, though, to locate input or output only parameters in page 3 and 4. We used page 3 for constant tables since they always serve as input parameters. The DSP library documentation [18] is not always explicit about which kind of problems might arise; which usage of DSP library functions is safe, and which is not.

Other

- While implementing sliding frames storage we've tried using an inline function that performed modulo division of two integers. We were surprised that suddenly the code size of the function grew enormously large. We then found out that integer division is poorly supported by mAgic core. We think that mentioning this point in the documentation or issuing a compiler warning might be useful.
- Macros found in magic.h, for managing DMA, assume only direct access to symbols and do not provide any way for indirect address calculations, as required for accessing data in complex data structures. Such is the case when accessing individual array within a two dimensional array. This functionality is required for managing the sliding frames storage.

Performance

Computational power utilization

Both the decoder and the encoder operate on 256 sample frames. The speech is sampled at the rate of 16 KHz meaning that the frames are processed at the rate of 62.5 frames per second, thus allowing 1,600,000 cycles for processing of a single frame. Currently the entire encoder/decoder loop uses approximately 800,000 cycles.

We began developing the application from pure C implementation of most of the DSP algorithms. Comparing to pure C version of the encoder/decoder, adapting the C code to be mAgic aware and using DSP library functions resulted in a speed up of approximately x50.

Figure 9 and Figure 10 depict encoder and decoder computational power utilization breakdown by functions. Total cycles required for encoding a single frame (1/62.5 sec) is $507x10^3$ cycles and for decoding a single frame is $283x10^3$ cycles. Both the encoder and the decoder together require approximately $790x10^3$ cycles per frame, which is approximately one half of the full mAgic capacity.

Figure 9. Functional breakdown of encoder processing power utilization

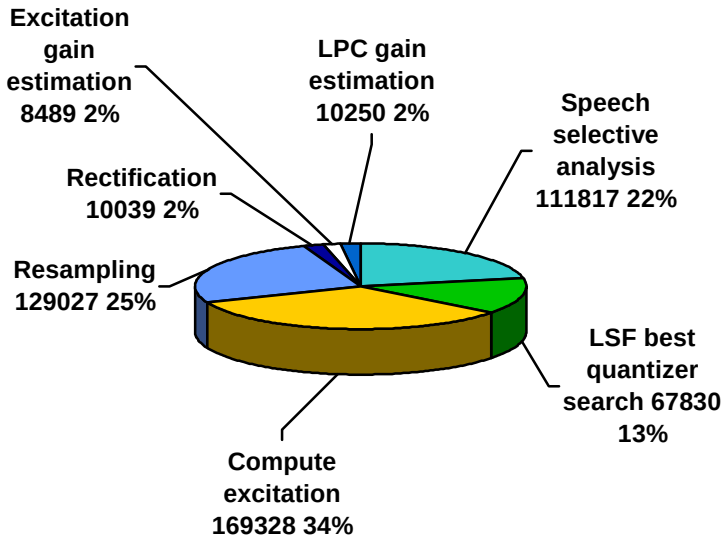
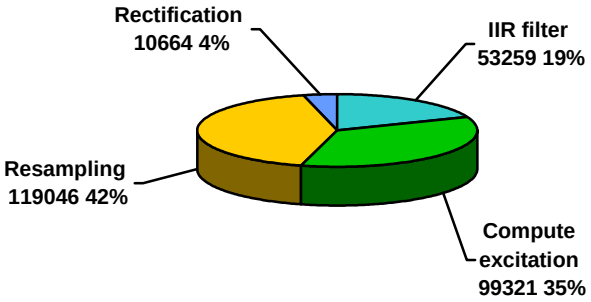


Figure 10. functional breakdown of decoder processing power utilization



Memory Requirements

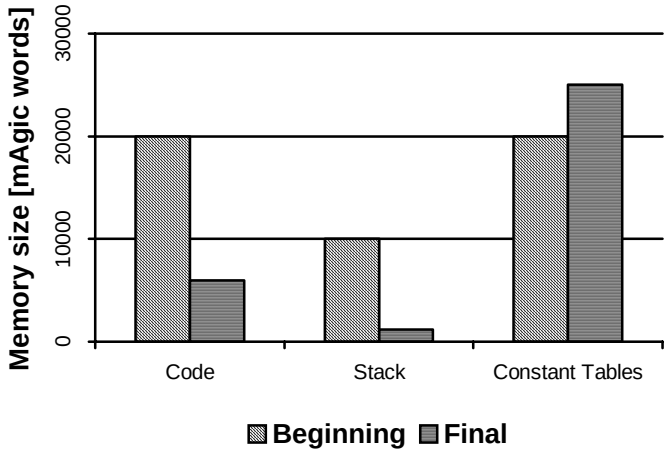
Table 1 Memory requirements

Memory type	Allocated (mAgic words)	Maximum available (mAgic words)
Page 0-2	2508	6144
Page 3	1826	2048
Page 4	390	512
External memory	22653	

Program memory	7521 (VLIWs)	8192

Figure 11 demonstrates the change in memory requirements corresponding to the beginning of the development process versus the final version of the application. It can be seen that the code size and the stack size improved dramatically. The total constant table sizes increases since some of the algorithms were optimized by using of additional constant tables. Large constant tables are affordable, since they are stored in the external memory. However, since we intended to avoid code overlaying, the code size had to be maintained fewer than 8K of instructions. Since the call stack resides in the internal data memory, its size is also limited.

Figure 11. Improvement in memory demand throughout the development process. Memory size is indicated in mAgic words (40 bit floating point)

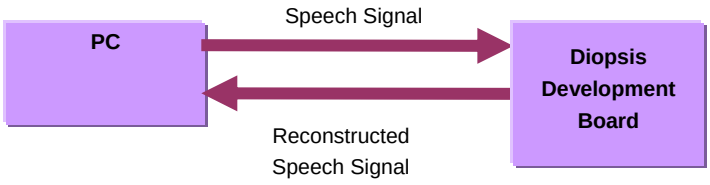


System Verification

Figure 12 depicts system setup used to verify the quality of the reconstructed speech. A PC sound card output was connected to the IN0 analog audio input line of the development board. Several wideband speech signals were played using standard playback software on the PC. The reconstructed speech was recorded using standard audio recording software and saved to wav files (plain PCM format). Then a group of people was asked to evaluate the quality of the reconstructed speech signal. All members of this group reported the original and reconstructed speech fragments to be of the same quality.

Both the original speech samples and the reconstructed speech samples can be evaluated using the *Setup and Demonstration Wizard* described in the *Installing and Running the Application* section.

Figure 12. Verification system setup



Installing and Running the Application

In this section we show how to install and run the SBE application.

All project files, including the source code, the *Setup and Demonstration Wizard* and documentation can be downloaded as a single package from the web site:

http://siglab.technion.ac.il/~atmel

Unzip the contents of the package to a local directory. In the rest of the document we'll refer to this local directory as <SBE_ROOT>.

Setup and Demonstration Wizard

The *Setup and Demonstration Wizard* that comes with the application provides fast and easy way of getting started. The wizard is accompanied by illustrations that guide you through the setup and installation process. Moreover, the wizard provides sample audio files and a real-time spectral analysis of the system's signals. It makes the exploration of system performance clear and easy.

The *Setup and Demonstration Wizard* is created using MATLAB 7.1 with data-acquisition toolbox. This wizard can be run either as stand-alone application, on machines that have no MATLAB 7.1 with data-acquisition toolbox installed, or as regular MATLAB source file. The following sub-sections describe these approaches. Please refer to the relevant sub-section.

Running the Wizard as Stand-alone Application

1. Install MATLAB Component Runtime by running:

```
<SBE_ROOT>\wizard\prerequisites\MCRInstaller.exe
```

2. Run the wizard:

```
<SBE_ROOT>\wizard\bin\SBE_wizard.exe
```

Running the Wizard form MATLAB 7.1

Simply run the following MATLAB script file:

```
<SBE_ROOT>\wizard\src\SBE_wizard.m
```

Note: MATLAB data-acquisition toolbox must be installed.

Running the SBE Application

Although, the wizard is the recommended way of setting up the application and exploring its performance, in this sub-section we provide the necessary instructions for installing and testing the application without the help of the wizard.

1. Connect the JTST board to the power supply.

2. Connect the RS232 port to the serial port on your PC (see Figure 15).

3. Configure AD/DA sampling rate to 16 KHz sampling frequency by setting the JP1-JP3 jumpers as indicated in Table 2

Table 2 Jumpers settings

Jumper label	Setting
JP1	HI
JP2	1
JP3	LO

4. Connect speech input/output. The board has 4 stereo audio I/O lines (see Figure 15), numbered IN0-IN3 and OUT0-OUT3. Table 3 shows input/output audio lines configuration used by the SBE application.

   Regular input/output configuration may include PC sound card output connected to the IN0 line. Standard PC software may be used to play speech samples. Speakers may be connected to OUT0-OUT2 so that speech signals at different stages of the processing can be evaluated.

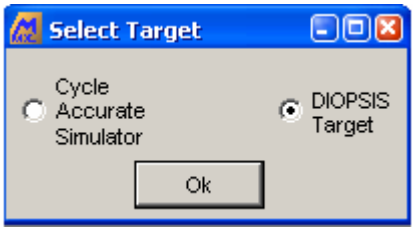
   Note: a standard dynamic microphone can not be connected directly to the IN0 line due to its output signal level. The microphone signal should be pre-amplified if you want to experiment with the real speech.

Table 3 Jumpers settings

Line	Usage
IN0	Narrowband input speech signal
OUT0	Reconstructed speech signal
OUT1	Narrowband speech (to be transmitted on the telephone line)
OUT2	Unmodified speech as received in IN0 (for reference purposes)
OUT3	Left: reconstructed speech (same as OUT0) Right: unmodified speech (same as OUT2) This output line is used by the *Setup and Demonstration Wizard*

5. Start MADE IDE and select 'DIOPSIS Target' operating mode as indicated in Figure 13.

Figure 13. Selecting MADE target mode



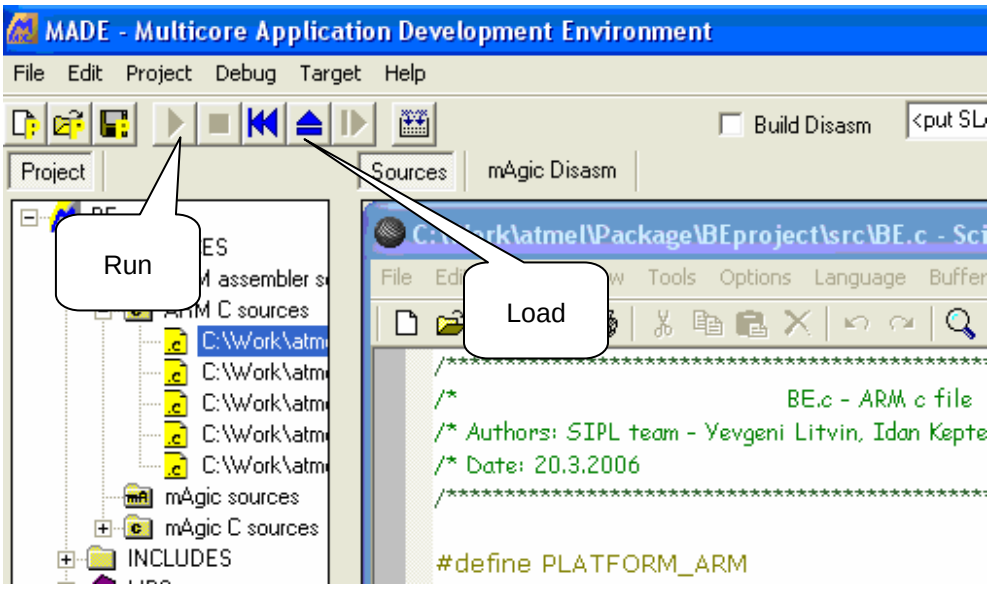
6. Open the project from the following location:

   ```
   <SBE_ROOT>\src\BE.jpf
   ```

7. The files provided with the package already include the pre-compiled version of the application so you do not have to build it.

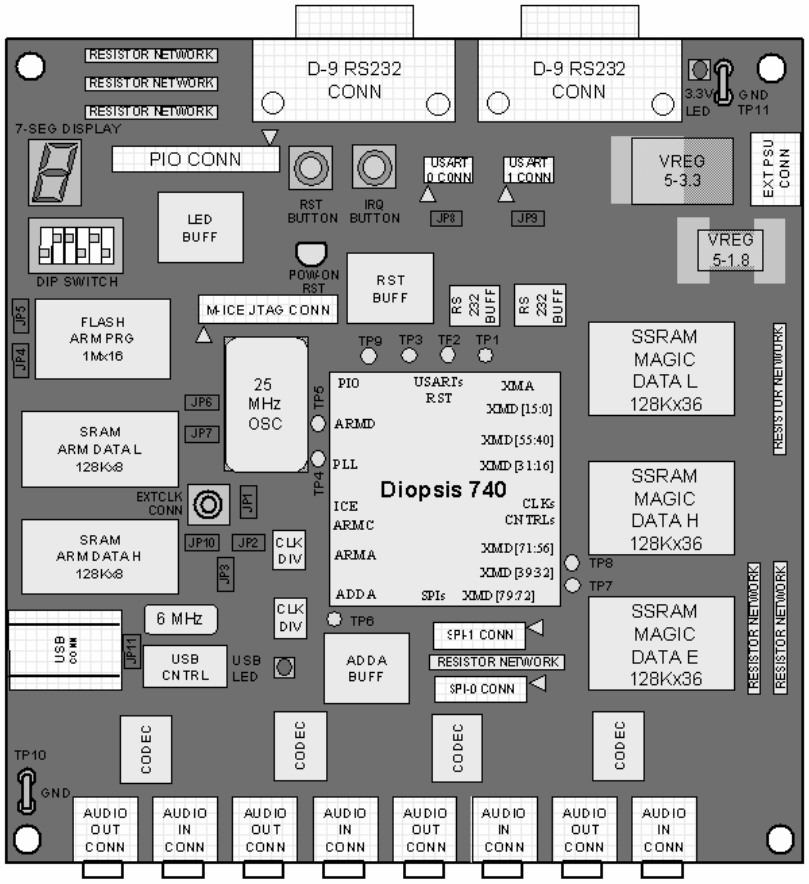
8. Load the application to the board and then run it (Figure 14 depicts these steps).

Figure 14. Loading and running the application



9. At this stage, the application is running. Experiment with different input speech samples (IN0 line). Evaluate the reconstructed speech signal (OUT0 line). Compare it to the original (OUT2 line) and telephone line (OUT1 line) counterparts.

Figure 15. JTST board



Troubleshooting

The JTST Board

Problem: The reconstructed speech signal (OUT0 line) is disrupted.

The unmodified speech (OUT2 line) sounds ok.

Some segments on the seven segment display may be flashing.

Solution: The sampling rate is not configured correctly on the board. It should be configured to 16 KHz sampling frequency. Refer to Table 2 for the correct jumper settings.

Problem: High frequencies of the reconstructed signal are disrupted.

Solution: Only speech signals are processed correctly by the application. High frequencies reconstruction will be inaccurate if other than speech signals are processed, such as music.

Problem: All output signals are very weak or can not be heard at all.

Solution: Make sure the input line signal has signal level suitable for the JTST input line. For example, regular dynamic microphones can not be connected directly to the IN0 line and have to be pre-amplified.

The Setup and Demonstration Wizard

Problem: On the spectral density view, the original and the reconstructed speech spectral graphs look exactly the same.

Solution: Make sure that the OUT3 line on the JTST board is connected to the stereo line-in line of the sound card on the PC. In most cases the regular microphone line in on the sound cards is mono line. The *Setup and Demonstration Wizard* contains the information on the configuration of the line-in sound card line as a default input audio source.

Problem: *"mclmcrrt73.dll is missing"* error message appears when SBE_wizard.exe is launched

Solution: MATLAB Components Runtime was not installed. Run <SBE_ROOT>\wizard\prerequisites\MCRInstaller.exe and try again.

If the problem persists, try restarting Windows.

Problem: Spectral density view shows only static noise.

Solution: Make sure that:

The JTST board receives input signal.

All OUT0-OUT3 lines produce output signals

Line-in sound card input line is configured to be the default audio for the PC. The *Setup and Demonstration Wizard* contains the information on the configuration of the line-in sound card line as a default input audio source.

Appendix A – Speech Bandwidth Extension Encoder/Decoder Internals

This section presents a brief overview of the encoder and the decoder internals. Please refer to [1], [2] and [3] for more detailed information.

Most of the works in speech bandwidth extension (SBE) use linear prediction (LP) techniques [5]. By these techniques, the wideband speech generation is divided into two separate tasks. The first task is the generation of a wideband excitation signal, and the second task is to determine the wideband spectral envelope, represented by linear prediction coefficients (LPCs). Once these two components are generated, wideband speech is regenerated by filtering the wideband excitation signal with the wideband linear prediction synthesis filter.

The generation of the wideband excitation signal and the wideband spectral envelope can be done by solely using the narrow-band (NB) speech signal [6][7]. The implicit assumption of such an approach is that there is a correlation between the low and high frequency bands of the speech signal. Another approach is to code and transmit side information about the highband (HB) portion of the speech signal. This approach is hybrid, because it artificially regenerates part of the high-frequency information from the NB speech signal, and completes the high-frequency information from the side information [8][9][10]. In our system the hybrid approach is used because it has the potential to provide a higher quality reconstructed wideband speech.

Spectral Linear Prediction

Spectral LP, suggested by Makhoul [4], is a spectral modeling technique in which the signal spectrum is modeled by an all-pole spectrum. In selective (spectral) LP, an all-pole model is applied to a selected portion of the spectrum.

In the case of SBE, the selective LP technique is applied to the HB of the original WB speech, and the spectral envelope of the HB is computed. If, alternatively, a time domain LP analysis is performed on the HB speech, it would require sharp filtering and down-sampling of the WB speech. These operations are costly and are completely avoided by working in the frequency domain, using the selective LP technique.

SBE Encoder Structure

The SBE encoder extracts the HB spectral parameters that will be embedded in the NB speech signal. The parameters include a gain parameter and spectral envelope parameters of the NB for each frame of the original WB speech signal. The explanation of the structure of the SBE encoder refers to Figure 16. The input to the SBE encoder is the original WB speech signal, denoted by s_{WB}. The WB speech signal is fed in parallel into three branches.

Upper Branch. In the upper branch, the WB speech is decimated (decimation includes filtering), and a NB speech signal, denoted by s_{NB}, is obtained. A time domain LP analysis is performed on the NB signal, and the NB excitation signal is obtained by inverse filtering the NB speech signal by the analysis filter. The NB excitation signal,

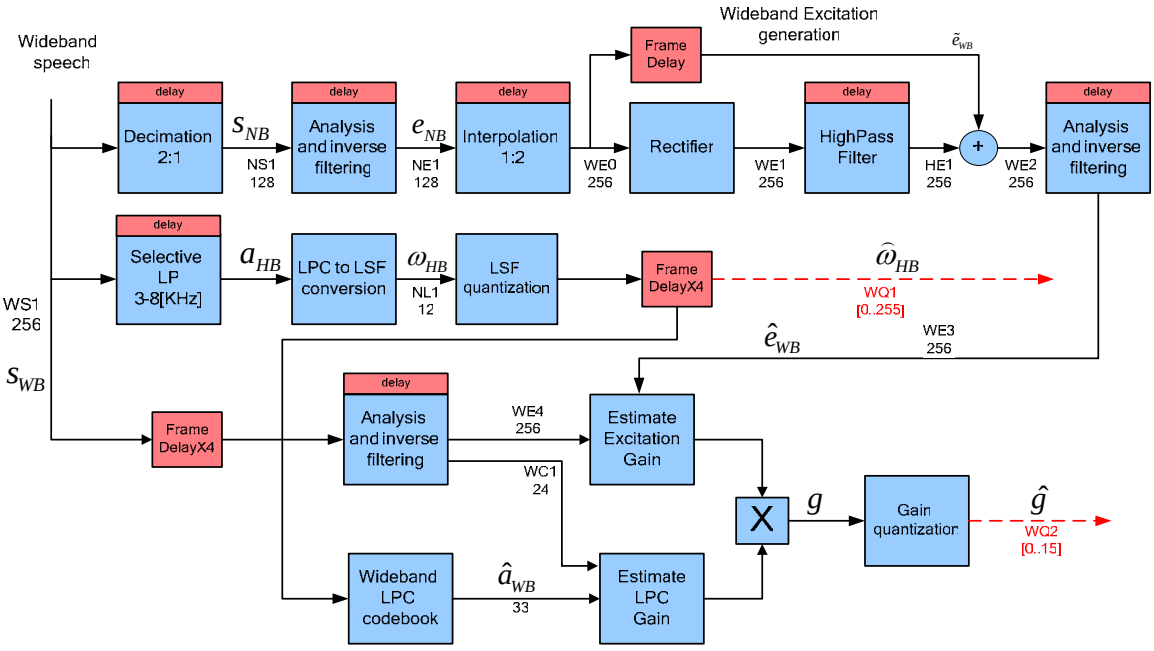
denoted by e_{NB} is then used for WB excitation regeneration. The reconstructed WB excitation signal is denoted by $\hat{e}_{WB}$.

Middle Branch. In this branch, the WB signal is analyzed by selective LP on the HB, in the range 3-8KHz. The selective LP coefficients, a_{HB}, are converted into an LSF (Line Spectral Frequencies) representation, ω_{HB}. The selective LSFs are quantized using a LSF vector quantizer. The LSFs codebook index is one of the transmitted parameters via embedding. The quantized selective LSFs are transformed into WB LPCs, as explained in the sequel, and are denoted by $\hat{a}_{WB}$, which correspond to the reconstructed WB spectral envelope. The WB LPCs are used to synthesize the WB reconstructed speech signal at the encoder, denoted by $\tilde{s}_{WB}$.

Lower Branch. In the lower branch, a gain parameter, denoted by g, is computed by minimizing the spectral distance between the original and synthesized WB speech signals, in the 3-8KHz frequency range. After gain computation, the gain is quantized, and the gain index is transmitted.

The transmitted parameters in each analysis frame (marked by dashed lines) include the LSF codebook index and the gain index (i.e., the indices of $\hat{\omega}_{HB}$ and $\hat{g}$).

Figure 16. SBE encoder structure



Wideband Excitation Generation

The WB excitation can be artificially generated from the NB excitation signal [11]. The NB excitation signal is the output of inverse filtering by the LP analysis filter, applied to

the NB speech signal. As shown in Figure 16, the NB excitation signal, e_{NB}, is first interpolated by a factor of 2 to the WB speech sampling rate. It is known that a non-linear operation expands the bandwidth, and in this case the interpolated NB excitation is passed through a full-wave rectifier, which performs sample by sample rectification. The interpolated NB excitation is combined with the HB portion of the rectified interpolated NB excitation, to produce an artificially extended WB excitation, denoted by $\tilde{e}_{WB}$. This artificially extended WB excitation has a tilt in the high-frequencies due to the rectifier operation. The tilt can be flattened by a whitening filter that performs inverse filtering using a LP analysis filter obtained by analyzing the artificially extended WB excitation, $\tilde{e}_{WB}$. The output of the whitening filter, the reconstructed WB excitation signal, is denoted by $\hat{e}_{WB}$.

LSF Parameters Computation and Quantization

To compute the HB spectral envelope, selective LP in the 3-8KHz frequency range is performed on each frame. The selective LPCs are subsequently converted to a LSF representation and are quantized using a LSF codebook. A LSF vector quantizer codebook was designed by the well known LBG vector quantization algorithm.

Wideband Spectral Envelope Computation

The transmitted side-information for each frame includes a LSF codebook index and a gain index. In order to perform WB speech synthesis, the frequency range of the spectral envelope should be 0-8KHz. The spectral envelope shape has no importance in the 0-3KHz range since the reconstructed WB speech, generated at the decoder, contains the NB speech in that frequency range.

The problem of WB spectral envelope computation is stated as follows: Given the selective LPCs (or equivalently LSFs) in the frequency range of 3-8KHz, the task is to find WB LPCs in the frequency range 0-8KHz such that the spectral distance between the selective and WB spectral envelopes will be small in the frequency range 3-8KHz.

The method suggested in [2] for WB spectral envelope computation is based on a symmetric duplication (mirroring) of the upper band into the lower band about the frequency 3KHz, followed by spectral LP.

Given a LSF codebook, the computation of the WB LPCs is done only once, in the design stage. After the computation of the WB LPCs codebook, the SBE decoder and encoder store the same codebook, and use it to generate the WB spectral envelope from a given index of quantized LSF vector. The WB LPCs codebook can be viewed as a conversion table between selective LPCs (or equivalently LSFs) that represents the spectral envelope in 3-8KHz frequency range and a WB LPCs that represents the spectral envelope in 0Khz-8KHz frequency range.

Highband Gain Computation and Quantization

The computation of the HB gain is done in order to minimize the spectral distance between the spectral envelopes of the original WB speech signal and the reconstructed WB speech signal, in the 3-8KHz frequency range. The spectral difference between these spectral envelopes originates from two main sources. First, the WB artificially

extended excitation is not identical to the original WB excitation. Second, the quantized WB LSFs introduce further spectral distortion between the two spectral envelopes.

The computed gain is quantized for transmission, by a scalar nonuniform quantizer of $\log(g)$.

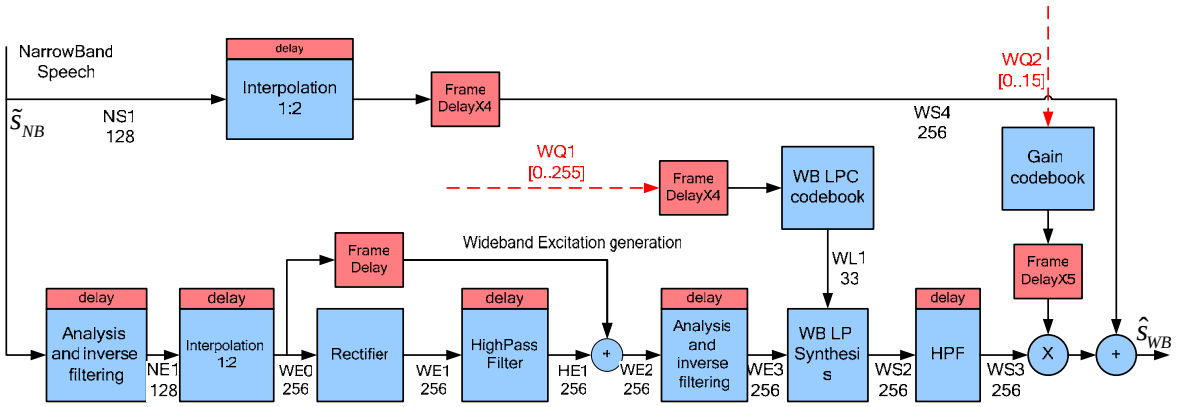
SBE Decoder Structure

The SBE decoder generates the reconstructed WB speech from the received NB speech signal and the side information. The description of the decoder structure refers to Figure 17. The side information for each frame includes the gain index and the LSF codebook index.

In the lower branch, the reconstructed WB excitation signal is generated from the NB speech signal, using the same technique as the technique used in the SBE encoder. In the middle branch, the WB LPCs are computed by using the selective LSF index and the mapped WB LPCs codebook. The WB artificial excitation together with the gain parameter and the WB LPCs are used to synthesize a WB speech signal. The HB part of the synthesized WB speech signal is filtered by a HPF, and combined with the interpolated NB speech signal, to produce the reconstructed WB speech signal, $\hat{s}_{WB}$.

It is desirable that the NB speech, which is input to the SBE decoder, will be close to the original NB speech signal generated at the input to the telephone channel. In the real world application the NB speech signal which is the output of a channel is close to the original NB speech. It is not identical to it because of two reasons. The first reason is the spectral distortion introduced by the non ideal channel. The second reason is the existence of embedded data in the NB speech in the fully backword compatible version of the system.

Figure 17. SBE decoder structure



A BPF is used to filter out the enhanced noise. The BPF is specified by a passband in the frequencies 100-3400Hz, and two transition bands in the frequencies 0-100Hz and 3.4-4KHz. The NB telephone speech signal which arrives from the channel, is filtered by the BPF. The filtered signal is denoted in Figure 17 as the NB speech, $\tilde{s}_{NB}$.

Appendix B - Directions for Further Improvements

- We are aware of several optimizations that could be done with the coder/encoder software that can reduce processing power demand. We chose not to proceed with these optimizations since only half of the processing power is actually used by the system. We estimate that additional x1.2 speed up can be achieved with reasonable amount of work.
- Letting ARM to use PARM memory for accumulating input audio samples and reading output audio samples from PARM for playback, appears to be sub-optimal, since most of the PARM memory is being constantly occupied by some data. PARM memory is a valuable resource that can be used for other needs as well, e.g., passing debug and trace information to and from mAgic. Probably, a better solution would use PARM memory regions exclusively in bursts, freeing it right after the burst and letting other parts of the program use larger regions of PARM, in a similar exclusive burst like fashion, as well.
- Implementing data-embedding functionality, if implemented, can allow speech bandwidth extension system to operate over legacy telephone lines without dedicated low data rate digital channel.
- The performance of the POLY2LSF algorithm can be significantly improved by optimizing parts of it, using assembler or DSP library functions, since it is written entirely in C.
- ARM image size can be reduced by porting the code to MARMOS operating system.
- Current implementation issues the DMA transactions in blocking mode since only half of mAgic computational power is utilized. It is possible to issue DMA transactions in advance in asynchronous mode thus eliminating stalls caused by waiting for the external memory.

Acknowledgments

We would like to express our gratitude to Professor David Malah for inspiring and reviewing the project, to all SIPL laboratory staff: Mr. Nimrod Peleg, the SIPL laboratory chief engineer, Mrs. Avni Ziva and Mr. Rosen Avi who supported our work, to Atmel Italy and Technion that sponsored our training in Rome and to Atmel's training and support staff that helped us a lot throughout the development by providing fixes to the DSP library, code snippets and answering our questions.

References

1. Sagi A., Data embedding in speech signals. M.Sc. thesis, Technion, Dept. of Electrical Engineering, May 2004, http://sipl.technion.ac.il/new/Research/Publications/Graduates/Ariel_Sagi/Ariel_Thesis_final_f.pdf .

2. A. Sagi and D. Malah. Bandwidth extension of telephone-speech aided by data embedding . Submitted for publication to Journal of Applied Signal Processing, February 2006.

3. A. Sagi and D. Malah, Data embedding in speech signals using perceptual masking. XII European Signal Processing Conference - Eusipco, pp. 1658-1660, September 2004.

4. J. Makhoul. Spectral analysis of speech by linear prediction. IEEE Trans. on Audio and Electroacoustic, vol. AU-21, no. 3, pp. 140-148, June 1973.

5. J. Makhoul. Linear prediction: A tutorial review. Proceedings of the IEEE, 63(4):561-580, April 1975.

6. J. A. Fuemmeler, R. C. Hardie, and W. R. Gardner. Techniques for the regeneration of wideband speech from narrowband speech. EURASIP Journal on Applied Signal Processing, 4:266-274, 2001.

7. P. Jax and P. Vary. Wideband extension of telephone speech using a hidden markov model. IEEE Workshop on Speech Coding, pages 133-135, September 2000.

8. A. McCree. A 14 kb/s wideband speech coder with a parametric highband model. In IEEE Proc. Of ICASSP, volume 4, pages 1153-1156, Istanbul, 2000.

9. A. McCree, T. Unno, A. Anandakumar, A. Bernard, and E. Paksoy. An embedded adaptive multi-rate wideband speech coder. In IEEE Proc. of ICASSP, volume 4, pages 2613-2616, Salt-Lake City, UT,May 2001.

10. J. M. Valin and R. Lefebvre. Bandwidth extension of narrowband speech for low bit-rate wideband coding. IEEE Workshop on Speech Coding, pages 130-132, September 2000.

11. J. Makhoul and M. Berouti. High-frequency regeneration in speech coding systems. In Proc. ICASSP, pages 428-431, Washington, DC, 1979.

12. ISO/IEC. Information technology - coding of moving pictures and associated audio for digital storage media at up to about 1.5 mbit/s-part 3: audio. Technical Report ISO/IEC 11172-3, ISO, 1992.

13. Q. Cheng and J. Sorensen. Spread spectrum signaling for speech watermarking. In Proc. ICASSP, pages 1337-1340, 2001.

14. R. N. Bracewell. Discrete hartley transform. J. Opt. Soc. Am., 73(12):1832-1835, 1983.

15. S. Haykin. Adaptive Filter Theory. Prentice Hall, 3'rd edition, 1996.

16. Atmel, JTST AT572D740-DK1 DIOPSIS 740 Development Kit, User Manual, d7003.pdf

17. Atmel, DIOPSIS 740, Dual-core DSP, AT572D740, d7001.pdf

18. Atmel, DSP Library, User Manual (draft), d7007.pdf