Hardware Reusable Design of Feature Extraction for Distributed Speech Recognition

V. RODELLAR-BIARGE, C. GONZALEZ-CONCEJERO, E: MARTINEZ DE ICAYA,
A. ALVAREZ-MARQUINA, and P. GÓMEZ-VILDA
Departamento de Arquitectura y Tecnología de Sistemas Informáticos
Facultad de Informática - Universidad Politécnica de Madrid
Campus de Montegancedo s/n – Boadilla del Monte. - 28660 (Madrid)
SPAIN

Abstract: - The design and implementation of a speech-feature extraction front-end module compliant to the Standard Aurora for Distributed Speech Recognition Systems (DSR) is presented in this paper. The design has been oriented towards its reusability in reconfigurable logic implementations. The number of samples of a frame, energy bands and mel-Cepstrum coefficients are parameterized. The design has been modelled in VHDL according to the restrictions and recommendations for high level synthesis, being portable among different EDA tools and technologically independent. The resulting design may be used as a core in the implementation of the client front-end part of Standard Aurora.

Key-Words: - Standard Aurora, client front-end, speech feature extraction, FPGA implementation.

1 Introduction

The popularity of remote and wireless devices as cellular phones, personal digital assistants (PDAs) and tablet computers have increased the interest on automatic speech recognition (ASR) in mobile communication systems [1]. They contain tiny keypads and small screens that make the interaction with these devices difficult. Speech communication can facilitate to the consumers the access to information application portals, voice navigation maps, finance transaction voice applications, form filling, etc. The use of a voice interface for mobile wireless devices is generally limited by computation capability, memory and battery energy. Currently many cellular phones have voice dialing capabilities but they are not able to carry out more sophisticated ASR tasks.

The Standard Aurora is an initiative from the European Telecommunication Standards Institute (ETSI) which promotes the standardization of the front-end and client-server protocols for Distributed Speech Recognition Systems (DSR), its structure being shown in Fig.1 [2][3]. The objective is to facilitate the access to computers and communication services by means of mobile terminals, avoiding the transmission channel affecting the recognition system performance. The DSR client part defines a standard front-end for the extraction of spectral features on the terminal. These features are compressed and transmitted to the server for speech reconstruction or conversion into text.

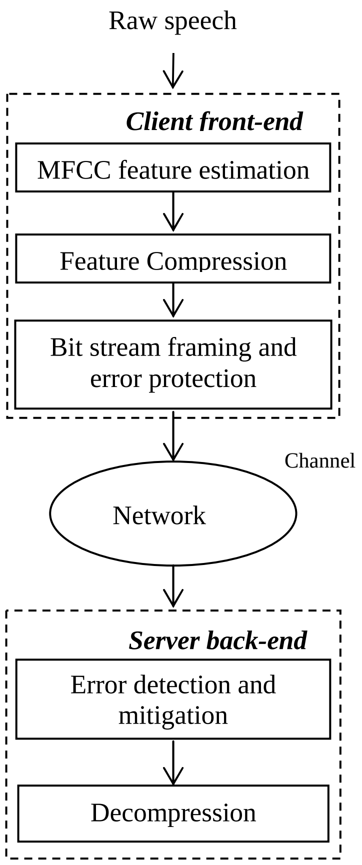

Fig. 1. Client-server structure.

The client front-end terminal processes data sampled at the rates of 8, 11 and 16 kHz and includes the following blocks: speech feature extraction, feature compression and framing, bit-stream and error protection. The server back-end terminal includes the blocks: bit-stream decoding and error mitigation, feature decompression and server feature processing. This last block is not defined in the standard, which is under evolution, new features being added in each new version in order to improve the intelligibility and quality of reconstructed speech. The principal challenges on the implementation of the front-end terminal for mobile applications are the restrictions of area, limited memory and power dissipation. On the other hand, semiconductor technology allows designing very complex devices that can be enclosed as part of a complete system in a single chip (SoC) [4]. If the system is conceived from scratch achieving the desired performance is costly and time consuming. Nowadays, to meet the tight time-to-market requirement, the electronic design re-uses pre-designed cores as a common practice. These cores may be parametrizable and customizable in order to be synthesized within a large or small application specification. They are available to the designer from heterogeneous sources, design teams, CAD tool libraries, CAD tool independent libraries, etc.

In this work we will concentrate on the speech feature extraction block to evaluate the Mel Frequency Cepstral Coefficients (MFCC) for the front-end terminal, and more specifically on its design as a soft-core. This block may be integrated with the rest of blocks of the standard in a SoC. The design has been modelled in VHDL according to the restrictions and recommendations for high level synthesis [5]. The resulting design is portable among different EDA tools and is technology independent.

The paper is organized as follows. First, we introduce the steps to be accomplished to extract the spectral features of speech according to the standard. Then, an overview of the major implementation issues of the basic sub-blocks is presented. And finally, the performance evaluation of the design for Altera FPGAs is discussed.

2 Feature Extraction overview

The process starts by de-noising raw speech. Then the spectral features for a 10 ms frame are quantified in 14 parameters, which are 13 mel-cepstral coefficients (C0 …C12) and the logarithm of the energy, E [6]. The measure of current frame energy is useful to detect if the signal under processing corresponds to silence or to speech and to detect the start-end limits of a word. The coefficient calculation involves the following steps:

1) *Pre-emphasis filtering,*
2) *Hamming windowing,*
3) *FFT,*
4) *Mel filtering*
5) *Discrete Cosine Transform.*

First a pre-emphasis filter is used in order to enhance the high frequency components with a factor of 0.97.

The window size is 256 samples with an overlapping each 128 samples, this characteristic being parameterized for it to be resized as needed. To smooth the signal and avoid the window side effect, a Hamming window is applied next.

The energy spectrum for the resulting Hamming-windowed frame is calculated by means of the FFT. The algorithm being used for such a transformation has been reused from other designs developed by the authors [7]. Later, the speech signal is passed through a Mel Filter Bank made of a set of overlapping triangular filters. Initially, the triangular filters are equally spaced for low-frequencies but as the frequency increases, their bands are logarithmically separated. The number of bands is also parameterized for reusability.

The energy of each band is calculated and finally, Mel Cepstrum parameters are estimated by the Cosine Transform. The number of MFCC coefficients is also parameterized.

3 Implementation issues

The design is organized in three main blocks, as shown in Fig. 2. The first block prepares the data to be processed by the FFT. It implements frame overlapping, pre-emphasis filtering, Hamming windowing and sample reordering. The second one carries out the FFT calculation. And the last one implements speech feature extraction by calculating the frame and band energies and MFCC parameters. The output from this last block is compressed, framed and error-protected before being sent to the network, according to the client front-end structure shown in Fig. 1. The complete system has been modelled in VHDL according to the restrictions and recommendations for high level behavioural synthesis [5]. The synthesis restriction that critically affects our particular design is the limitation related to floating-point number use, for this reason we have normalized the data multiplying by 1024 (2^{10}). The data format adopted is two's complement binary and 16 bits. As it was said in the previous paragraph, the design is parameterized in window size, number energy bands and number of MFCC coefficients, which provides enough flexibility to re-scale and reuse the design. A change in these parameters will determine RAM memory sizes used as temporal store, and the number of memory read/write accesses. The values of the Hamming window coefficients, energy bands, and cosines to calculate the MFCC coefficients are evaluated off-line and stored into ROM memories, therefore the reusability of the complete design will be also limited by this fact, because the values of these memories must be updated accordingly to re-scaled conditions. On the other hand, the FFT block implemented is based on a radix-2 decimation-in-time algorithm, and the number of window samples and sample coding number of bits are parameterized as well in order to control the quantization error when using fixed-point arithmetics [7].

In the next paragraphs we will describe some implementation details for the pre-processing and parameter extraction blocks.

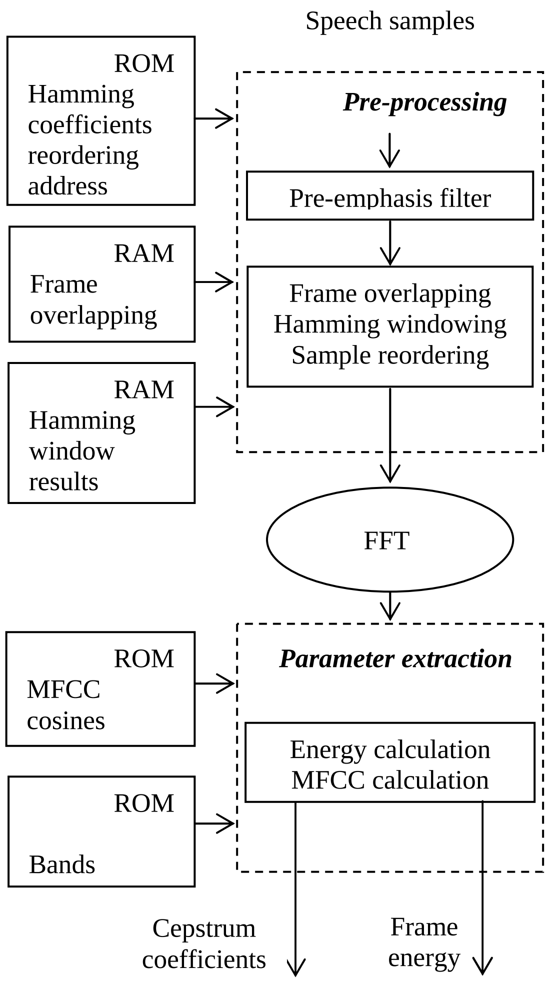
Speech samples
ROM Hamming coefficients reordering address
Pre-processing
Pre-emphasis filter
RAM Frame overlapping
Frame overlapping Hamming windowing Sample reordering
RAM Hamming window results
FFT
ROM MFCC cosines
Parameter extraction
Energy calculation MFCC calculation
ROM Bands
Cepstrum coefficients
Frame energy

Fig. 2. Design structure

3.1 Pre-processing block

In order to make the implementation of the pre-emphasis filter easier and to avoid an operation of multiplication by a constant, the parameter *k=0.97* has been approximated by 31/32. It allows writing the expression for the filtered sample as:

$$\bar{s}_n = s_n - s_{n-1} + \frac{s_{n-1}}{32} \tag{1}$$

Then, the operation can be carried out by means of two parallel registers storing current and previous samples *s(n)* and *s(n-1)*, a five-position shift-right register to evaluate s*(n-1)/32*, a subtracter and an adder. The filtered sample is obtained after three clock cycles.

3.1.1 Frame overlapping

This procedure has been implemented by means of an auxiliary RAM memory, which is managed under the control of a two-clock scheme. This strategy is adopted in order to avoid losing pre-filtered samples. The *clk2* controls the write operation and works twice slower than the clock controlling the read operation *(clk1)*. The memory is divided into two parts of the same size. One half is updated with the new data and the other half remains unchanged containing the overlapped part from the previous frame. An example for a memory of 8 positions is shown next.

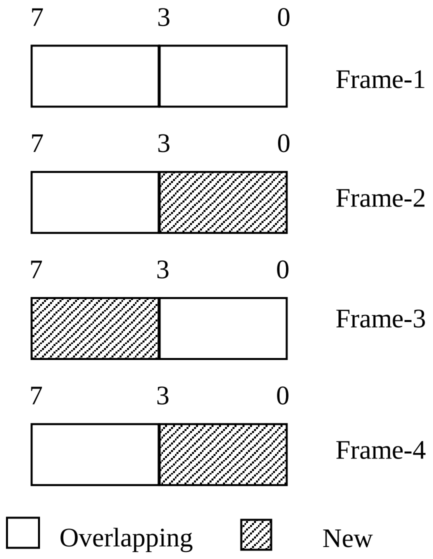

Fig. 3. Memory-update strategy

Initially, a complete frame contained in the memory positions from 0 to 7 is read. While these data are processed by other functional blocks, new data inputs from the next frame are updated and written in the memory positions 0 to 3. For the next frame positions 4 to 7 will be read first and positions 0 to 3 later on. So the actual frame and the previous one have their overlapping samples stored in positions 4 to 7. For the next frame, new data inputs are written in positions 4 to 7. In this case, the memory will be read in an opposite way than before, positions 0 to 3 first and positions 4 to 7 next. Now the current frame and the previous one have in common the samples contained in positions 0 to 3. And the process goes on.

3.1.2 Hammig windowing

The implementation of the Hamming windowing has been carried out by a ROM and RAM memories, and a multiplier. All of them are controlled by *clk2*. The ROM memory contains the values of the Hamming function and the address where the windowed results are to be stored in the auxiliary RAM memory. The reason to enclose this address together with the window value is to prepare the samples in the right order to be processed properly by the FFT algorithm. So, they do not need any later group sample shuffling into even and odd as is required by decimation-in-time FFT algorithms. The addresses are generated according to a bit reversal addressing mapping. An example for 8 bits is shown next.

Reading address in ROM	*Writing address in RAM*
000 (0)	000 (0)
001 (1)	100 (4)
010 (2)	010 (2)
011 (3)	110 (6)
100 (4)	001 (1)
101 (5)	101 (5)
110 (6)	011 (3)
111 (7)	111 (7)

Table 1. Sample addressing by bit reversal reordering

3.2 Parameter extraction block

This block has been structured as three main sub-blocks named: bands of energy, natural logarithm and mel-cepstrum. We will introduce the structures designed for them in the next sections.

3.2.1 Bands of energy

The calculation of the frame energy and band energy are done in parallel, as can be seen in Fig. 4. The design is done with the following basic sub-blocks: shifter_right, multiplier, counter, ROM and two accumulators. The shifter_right unit normalizes the data coming from the FFT and the multiplier squares the resulting data from the previous operation. The counter addresses the ROM memory that contains the accumulation limits of the bands, and controls the accumulation operation to obtain the energy of a frame.

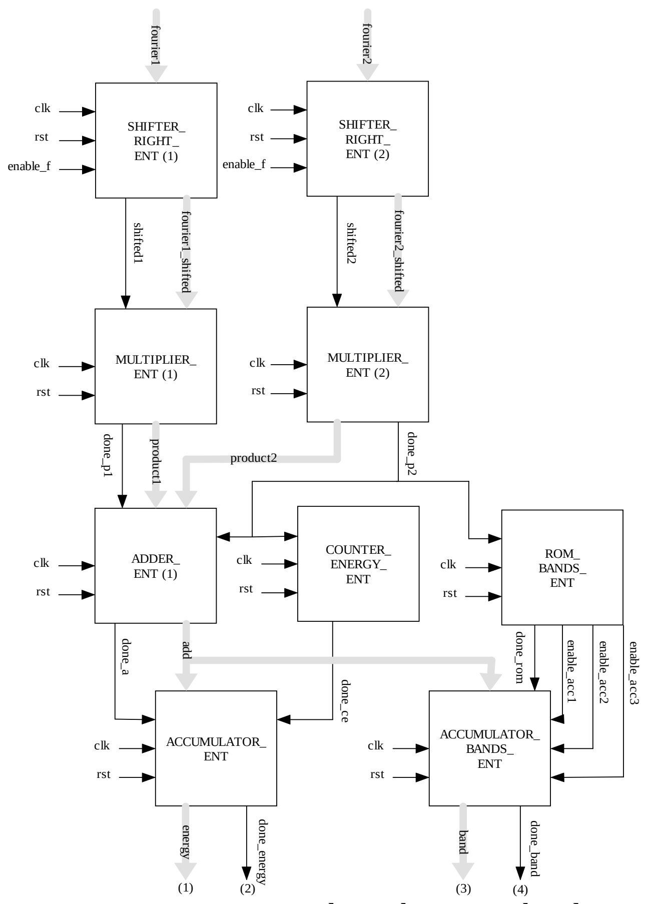

Fig. 4. Structure to evaluate the energy bands

3.2.2 Natural logarithm calculation

The final result of one band energy is obtained after calculating the natural logarithm. This logarithm is calculated taking starting from the logarithm in base two. A classical way to implement logarithmic functions is by pre-calculated look-up tables; this solution demands a lot of area and lacks reusability. In order to avoid these inconveniences, the base-two logarithm has been implemented by using the method developed by Mitchell [8]. The implementation is very simple and appropriate for supporting reusability. The method in many cases is not exact but gives a good approximation; nevertheless the resulting values may be corrected to improve accuracy if required. The basic idea is to split the calculation into two parts, obtaining the final result as the addition or concatenation of them as in:

$$(\log_2 Z)_{aprox} = c + m \qquad (2)$$

The characteristic part *c* is the binary value of the position of the first non-zero bit starting from the most significant bit to the least significant one. The mantissa *m* is determined by the remaining bits to the right, starting from the first non-zero bit detected. As an example, the base-2 logarithm of 21 (decimal) is

4.3922. And its binary representation is 00010101. The characteristic part is determined by the bit located in position number four (2^4), then *c*=100. The mantissa is defined by the remaining bits, *m*= 0101. The final result is 100.0101, which is 4.3125.

The structure designed to calculate the natural logarithm is shown next.

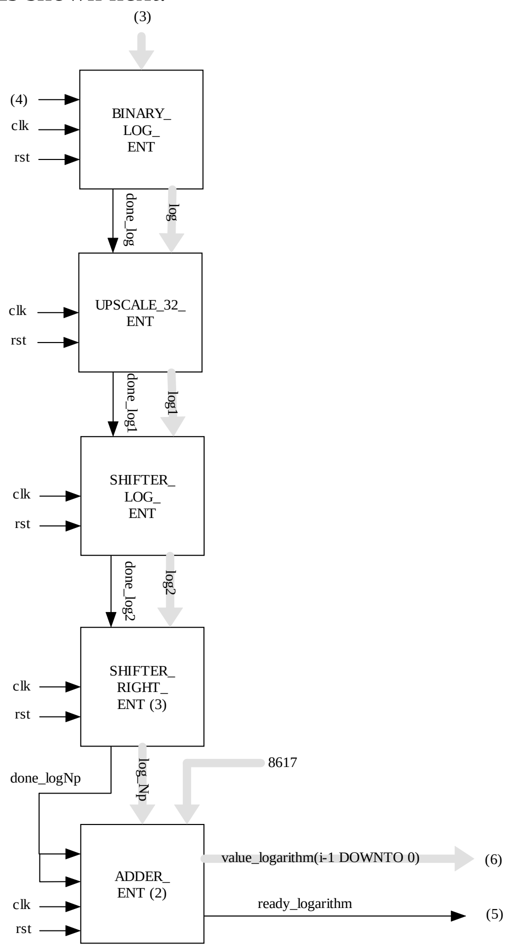

Fig. 5. Computation of the natural logarithm from base-two logarithms

Once the logarithm in base two is calculated, the 16 bit data format is expanded into 32 bits by means of the functional unit upscale_32. And then the transformation into the natural logarithm is done according to:

$$\ln = (\log_2 * 11357) + 8617 \tag{3}$$

Initially the result of the logarithm in base two is multiplied by the number 11357. The multiplication has been implemented by shift-left operations:

$$shifter\_\log \Rightarrow \sum \log_2 << x \tag{4}$$

Considering that 11,357 may be represented as:

$$11357 = 2^{13} + 2^{11} + 2^{10} + 2^6 + 2^4 + 2^3 + 2^2 + 2^0$$

the values of the shifts (x) will be then: 13, 11, 10, 6, 4, 3, 2, 0. Next, a normalization operation consisting in fourteen right shifts carried out. Finally, the partial result is corrected by adding the amount of 8,617, which is equivalent to calculate *ln* (64x2048).

After the operations mentioned above are finished, data are ready for Mel Cepstrum coefficient calculation.

3.2.3 Cepstrum coefficients

Finally, the basic structure implemented to obtain the mel-cepstrum coefficients is shown in Figure 6. It consists of a counter to address the ROM memory, which contains the cosine values to compute the Cosine Transform, a set of registers for synchronization purposes and a substructure consisting of a multiplier and accumulator to compute a single coefficient (encircled within the dash line) in the scheme. This structure may be replicated as many times as coefficients to be computed in parallel.

4 Performance evaluation

The system was synthesized with the EDA tool Quartus II from Altera without using any pre-designed component available in its libraries [9]. The device selection was automatically carried out by the tool, choosing in this case the EP1S20F484C5 from the Stratix family. Then, the results presented here may be optimized having into consideration the synthesis tool recommendations and the particularities of the device. Each block was first synthesized and the complete system after, following a bottom-up standard procedure.

The physical resource demand after synthesis is shown in Table 2. The pre-processing block requires more logic elements and memory bits than the feature extraction block. However, this last one demands more DSP units because it has to do more arithmetic computations. It may be seen that the complete system fits in the selected device. Extra resources are still available, except for logic elements, which are almost finished.

The performance results in terms of frequency and power dissipation are shown in Table 3. In this table the results for *clk1* and *clk2* used in the frame overlapping sub-block, and for *clk* that controls the feature extraction block are evaluated. The results in frequency obtained for the stand-alone blocks and the complete system synthesis are very similar. Concerning power dissipation, it can be noticed that the consumption of the pre-processing stage is larger than the feature extraction one.

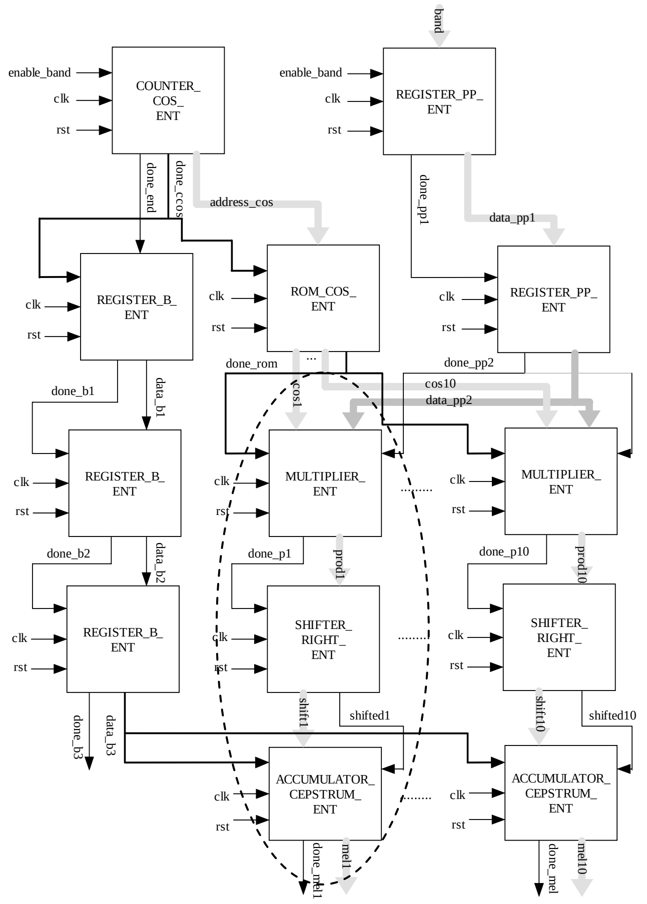

Fig. 6. Mel Cesptrum Coefficients Block Diagram

	LE (18,430)	Pins (362)	Memory bits (1,669,248)	DSP (80)
Pre-processing	10,030	53	4,096	2
Feature extraction	6,033	72	48	42
Complete System	18,340	29	4,321	52

Table 2. Resources demanded

	F.Max (MHz)			Power(mw)		
	clk1	clk2	clk	Internal	I/O	Total
Pre-processing	269.47	99.74		502.74	9.43	512.17
Feature extraction			103.31	348.99	20.27	369.26
Complete system	244.34	97.64	102.63	2,346.82	81.13	2,427.95

Table.3. Time and power

4 Conclusions

In this paper, a parameterized speech feature-extraction design oriented to reusability as a core for the implementation of the server front-end for the Standard Aurora has been presented. The design may be used as part of a SoC to implement all parts of the standard. A re-scaling in the parameters will affect to RAM memory sizes and read/write accesses. ROM memories size and contents are to be calculated off-line. The performance of the system can be considered sufficient for the application. Nevertheless, it can be improved using pre-designed library blocks from the design tool, at the cost of loosing design portability.

Acknowledgments

This work is funded by grant TEC2006-12887-C02-00 from Plan Nacional de I+D+i, Ministry of Education and Science, Project HESPERIA (http.//www.proyecto-hesperia.org) from the Programme CENIT, Centro para el Desarrollo Tecnológico Industrial, Ministry of Industry, Spain and Project IBIOL (CCG06-UPM/INF-28) from Autonomic Government of Madrid, Spain.

References:

[1] Uve Hansmann, Lothar Merk, Martin S. Nicklous, Thomas Stober. "Persuasive Computing: the mobile world". Springer Verlag 2003.

[2] http://www.etsi.org/aurora/

[3] ETSI ES 2002 050, V1.1.5. "Speech Processing Transmission and Quality Aspects (STQ); Distributed Speech Recognition; Advanced front-end feature extraction algorithms, Compression Algorithms". *European Telecommunications Standard Institute*, January 2007.

[4] Bashir M. Al-Hasmini Ed. "Systems-on-chip: Next Generation Electronics". IEEE Circuits, Devices and Systems, Series 18, 2006.

[5] J Michel Keating and Pierre Bricand, "Reuse Methodology Manual: For System-on-a-Chip Designs". *Third Edition. Kluwer Academic* Publishers, 2002.

[6] J. R. Deller, J. G. Proakis and J. H. L. Hansen "Discrete-Time Processing of Speech Signals", Mc Millan, NY, 1993.

[7] C. Gonzalez-Concejero, V. Rodellar, A, Alvarez-Marquina and P. Gómez-Vilda, "A portable hadware design of a FFT algorithm", *Latin American Applied Research*, Vol. 37, pp. 79-82, 2007.

[8] J. N. Mitchell, "Computational multiplication and division using binary logarithms", *IRE Transactions on Electronic Computers*, pp. 512-517, August 1962.

[9] http://www.altera.com