We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

2

Real-time Hardware Feature Extraction with Embedded Signal Enhancement for Automatic Speech Recognition

Vinh Vu Ngoc, James Whittington and John Devlin
La Trobe University
Australia

1. Introduction

The concept of using speech for communicating with computers and other machines has been the vision of humans for decades. User input via speech promises overwhelming advantages compared with standard input/output peripherals, such as, mouse, keyboard, and buttons. To make this vision a reality, considerable effort and investment into automatic speech recognition (ASR) research has been conducted for over six decades. While current speech recognition systems perform very well in benign environments, their performance is rather limited in many real-world settings. One of the main degrading factors in these systems is background noise collected along with the wanted speech.

There are a wide range of possible uncorrelated noise sources. They are generally short lived and non-stationary. For example in the automotive environments, noise sources can be road noise, engine noise, or passing vehicles that compete with the speech. Noise can also be continuous, such as, wind noise, particularly from an open window, or noise from a ventilation or air conditioning unit.

To make speech recognition systems more robust, there are a number of methods being investigated. These include the use of robust feature extraction and recognition algorithms as well as speech enhancement. Enhancement techniques aim to remove (or at least reduce) the levels of noise present in the speech signals, allowing clean speech models to be utilised in the recognition stage. This is a popular approach as little-or-no prior knowledge of the operating environment is required for improvements in recognition accuracy.

While many ASR and enhancement algorithms or models have been proposed, an issue of how to implement them efficiently still remains. Many software implementations of the algorithms exist, but they are limited in application as they require relatively powerful general purpose processors. To achieve a real-time design with both low-cost and high performance, a dedicated hardware implementation is necessary.

This chapter presents the design of a Real-time Hardware Feature Extraction System with Embedded Signal Enhancement for Automatic Speech Recognition appropriate for implementation in low-cost Field Programmable Gate Array (FPGA) hardware. While suitable for many other applications, the design inspiration was for automotive applications, requiring real-time, low-cost hardware without sacrificing performance. Main components of this design are: an efficient implementation of the Discrete Fourier Transform (DFT), speech enhancement, and Mel-Frequency Cepstrum Coefficients (MFCC) feature extraction.

2. Speech enhancement

The automotive environment is one of the most challenging environments for real-world speech processing applications. It contains a wide variety of interfering noise, such as engine noise and wind noise, which are inevitable and may change suddenly and continually. These noise signals make the process of acquiring high quality speech in such environments very difficult. Consequently, hands-free telephones or devices using speech-recognition-based controls, operate less reliably in the automotive environment than in other environments, such as in an office. Hence, the use speech enhancement for improving the intelligibility and quality of degraded speech signals in such environments has received increasing interest over the past few decades (Benesty et al., 2005; Ortega-Garcia & Gonzalez-Rodriguez, 1996).

The rationale behind speech enhancement algorithms is to reduce the noise level present in speech signals (Benesty et al., 2005). Noise-reduced signals are then utilized to train clean speech models, and as a result, effective and robust recognition models may be produced for speech recognizers. Approaches of this sort are common in speech processing since they require little-to-no prior knowledge of the operating environment to improve the recognition performance of the speech recognizers.

Based on the number of microphone signals used, speech enhancement techniques can be categorized into two classes, single-channel (Berouti et al., 1979; Boll, 1979; Lockwood & Boudy, 1992) and multi-channel (Lin et al., 1994; Widrow & Stearns, 1985). Single channel techniques utilize signals from a single microphone. Most techniques on noise reduction belong to this category, including spectral subtraction (Berouti et al., 1979; Boll, 1979) which is one of the traditional methods.

Alternatively, multi-channel speech enhancement techniques combine acoustic signals from two or more microphones to perform spatial filtering. The use of multiple microphones provides the ability to adjust or steer the beam to focus the acquisition on the location of a specific signal source. Multi-channel techniques can also enhance signals with low signal to noise ratio due to the inclusion of multiple independent transducers (Johnson & Dudgeon, 1992). Recently, dual microphone speech enhancement has been applied to many cost sensitive applications as it has similar benefits to schemes using many microphones, while still being cost-effective to implement (Aarabi & Shi, 2004; Ahn & Ko, 2005; Beh et al., 2006).

With the focus on the incorporation of a real-time low-cost but effective speech enhancement system for automotive speech recognition, two speech enhancement algorithms are discussed in this chapter. These are Linear Spectral Subtraction (LSS) and Delay-and-Sum Beamforming (DASB). The selection was made based on the simplicity and effectiveness of the algorithms for automotive applications. The LSS works well for speech signals contaminated with stationary noise such as engine and road noise, while the DASB can perform effectively when the location of signal sources (speakers) are specified, for example, the driver. Each algorithm can work in standalone mode or cascaded.

Before discussing these speech enhancement algorithms in detail, common speech preprocessing is first described.

2.1 Speech preprocessing and the Discrete Fourier Transform

2.1.1 Speech preprocessing

Most speech processing algorithms perform their operations in the frequency domain. In these cases, speech preprocessing is required. Speech preprocessing uses the DFT to transform speech from a time domain into a frequency domain. A general approach for processing speech signals in the frequency domain is presented in Figure 1.

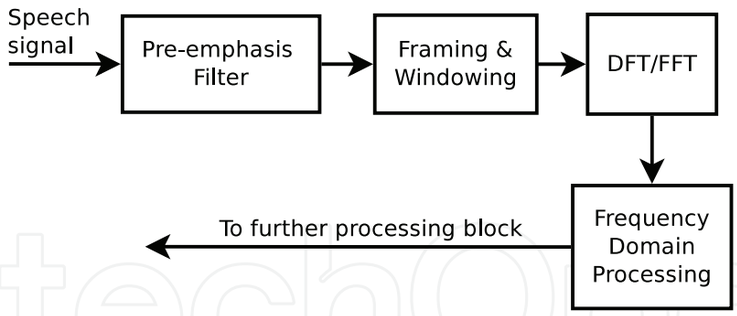


Fig. 1. Block diagram of basic speech processing in the frequency domain

Speech signals, acquired via a microphone, are passed through a pre-emphasis filter, which is normally a first-order linear filter. This filter ensures a flatter signal spectrum by boosting the amplitude of high frequency components of the original signals.
Each boosted signal from the pre-emphasis filter is then decomposed into a series of frames using square sliding windows with frame advances typically being 50% of the frame length. The length of a frame is normally 32ms which has 512 samples at 16Khz sampled rate. To attenuate discontinuities at frame edges, a cosine window is then applied to each overlapping frame. A common window used in speech recognition is the Hamming window.
The framing operation is followed by the application of the DFT, in which time-domain acoustic waveforms of the frames are transformed into discrete frequency representations. The frequency-domain representation of each frame in turn is then used as inputs of the Frequency Domain Processing (FDP) block, where signals are improved by speech enhancement techniques and a speech parametric representation is extracted by the speech recognition front-end.

2.1.2 DFT algorithm

The discrete transform for the real input sequence $x\ \{x(0), x(1), \cdots, x(N-1)\}^T$ is defined as:

$$X(k) \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi kn}{N}}, k = 0, 1, \ldots, N-1. \tag{1}$$

In practice, the above DFT formula is composed of *sine* and *cosine* elements:

$$X_{Re}(k) = \sum_{n=0}^{N-1} x(n) \cos(\frac{2\pi kn}{N}), \tag{2}$$

$$X_{Im}(k) = \sum_{n=0}^{N-1} x(n) \sin(\frac{2\pi kn}{N}), \tag{3}$$

where Re and Im represent the real and imaginary parts of DFT coefficients.
The two formulas (2) and (3) can be implemented directly in FPGA hardware using two MAC (Multiplier and Accumulator) blocks which are embedded in many low-cost FPGA devices. Figure 2 shows the structure of this implementation for either a real or imaginary component. As shown in the figure, the multiplier and the accumulator are elements of one MAC hardware primitive. Therefore, the direct implementation of the DFT formula on FPGA hardware results in a simple design requiring only modest hardware resources. However, it does result in a considerably long latency ($2N^2$ multiplications and $2N^2$ additions).

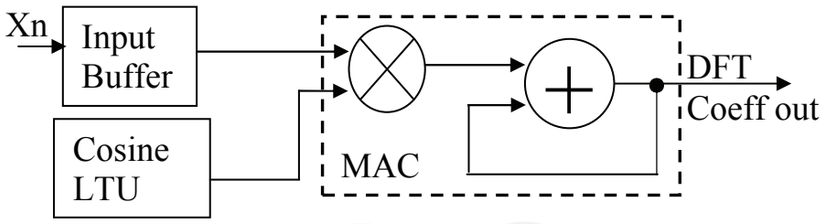


Fig. 2. Hardware structure of direct DFT implementation

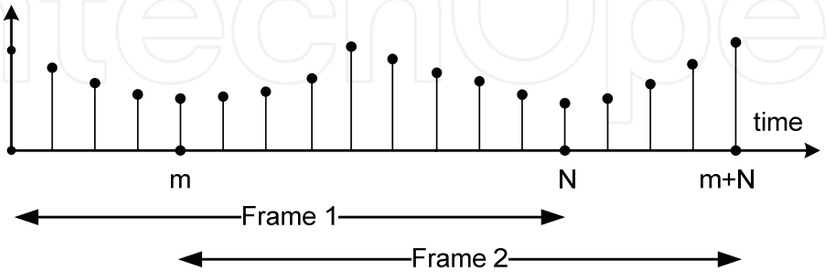


Fig. 3. Overlapped frames

2.1.3 Utilizing overlapping frames property in DFT to reduce latency

Figure 3 shows an example of two overlapped frames: F_1 and F_2. F_2 overlaps $N-m$ samples with the previous frame (F_1). It is expected that computations for those $N-m$ samples from the previous frame (F_1) can be reused for the current frame (F_2). In this way, significant computation, and thus latency, can be saved.

Based on frame F_1 and F_2 mentioned above (Figure 3), the algorithm can be described simply as follows.

In order to utilize the 50% overlapping frames feature, the DFT of Frame F_1 is

$$X_1(k) = \sum_{n=0}^{\frac{N}{2}-1} x(n) e^{\frac{-j2\pi kn}{N}} + \sum_{n=\frac{N}{2}}^{N-1} x(n) e^{\frac{-j2\pi kn}{N}}. \tag{4}$$

Similarly, the DFT of Frame F_2 is

$$X_2(k) = \sum_{n=0}^{\frac{N}{2}-1} x(n+\frac{N}{2}) e^{\frac{-j2\pi kn}{N}} + \sum_{n=\frac{N}{2}}^{N-1} x(n+\frac{N}{2}) e^{\frac{-j2\pi kn}{N}}. \tag{5}$$

In short, formulas (4) and (5) can be respectively inferred as:

$$X_1(k) = A + B, \tag{6}$$

and

$$X_2(k) = C + D. \tag{7}$$

If n in term C is substituted by $n = i - \frac{N}{2}$, then:

$$C = \sum_{i=\frac{N}{2}}^{N-1} x(i) e^{\frac{-j2\pi k(i-\frac{N}{2})}{N}} = e^{j\pi k} \sum_{i=\frac{N}{2}}^{N-1} x(i) e^{\frac{-j2\pi ki}{N}}. \tag{8}$$

By observation, C now clearly has a similar formula to B except for $e^{j\pi k}$ factor. Also, all the samples used in B will be elements of C, as frame F_1 and frame F_2 are 50% overlapped. So the DFT of Frame F_2 is:

$$X_2(k) = e^{j\pi k}B + D. \tag{9}$$

If we generally call term B and D as $X_{half\_old}(k)$ and $X_{half\_new}(k)$ respectively, the DFT of each frame can be presented as:

$$X(k) = e^{j\pi k}X_{half\_old}(k) + X_{half\_new}(k), \tag{10}$$

where the calculation of $X_{half\_old}(k)$ is performed on the $\frac{N}{2}$ overlapped samples that already appear in the previous frame, while that of $X_{half\_new}(k)$ is performed on the $\frac{N}{2}$ new samples in the current frame. Recursively, $X_{half\_new}(k)$ will become $X_{half\_old}(k)$ in the next frame. The expressions $X_{half\_new}(k)$ are computed by the term D formula, with the index running from 0:

$$X_{half\_new}(k) = \sum_{i=0}^{\frac{N}{2}} x(i)e^{\frac{-j2\pi k(i+\frac{N}{2})}{N}}. \tag{11}$$

In practice, term $e^{j\pi k}$ only takes a value of either $+1$ or -1, thus, the computation of $\frac{N}{2}$ overlapped samples can be directly reused. So, only $X_{half\_new}(k)$ needs to be computed, and thus, the DFT computation requirement is reduced by 50%.
Resulting from this saving in computation, a novel simple hardware structure has been developed and compares well to the simpleness of the direct DFT implementation.

2.1.4 Efficient DFT hardware implementation

This section presents an efficient hardware implementation of the previous described overlapping DFT (OvlDFT) algorithm. This algorithm and implementation are subjected to patent (Vu, 2010).
Firstly, assuming that the input samples, $x(i)$, are real, the output of the DFT is symmetric, then as a result, only values k from 0 to $\frac{N}{2} - 1$ are required. Also, as described in the previous section, only X_{half_new} is required to be computed. To simplify the formula, the real (X_{Re}) and imaginary (X_{Im}) parts of X_{half_new} are computed individually, as presented in equation (12) and (13). By doing this, the term $e^{j\pi k}$ in (10) only takes values of either 1 or -1 depending on k.

$$X_{Re}(k) = \sum_{i=0}^{\frac{N}{2}-1} x(i)\cos(\frac{2\pi k(i+\frac{N}{2})}{N}), \tag{12}$$

$$X_{Im}(k) = \sum_{i=0}^{\frac{N}{2}-1} x(i)\sin(\frac{2\pi k(i+\frac{N}{2})}{N}). \tag{13}$$

The structure of the proposed hardware DFT algorithm is shown in Figure 4. In order to achieve the 50% computational saving, additional memory (the RAM block) is required to buffer appropriate results from the computation of the previous frame.
The heart of this hardware structure lies in the novel implementation of this RAM buffer memory. By using RAM blocks which are commonly embedded in low-cost FPGA devices and configured as dual port memory in our proposed hardware structure, the content of

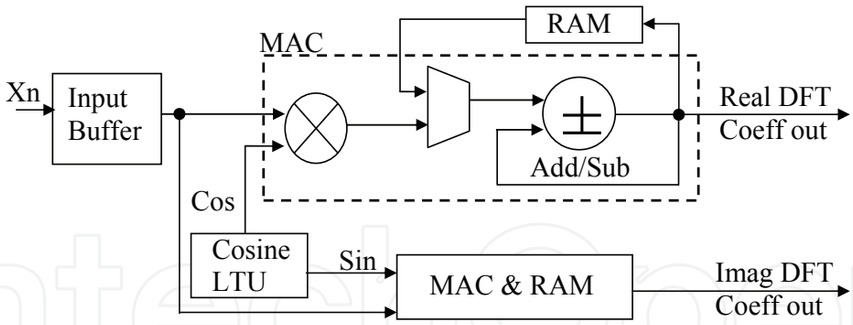


Fig. 4. OvlDFT hardware structure-Components within dashed box belong to one FPGA MAC primitive

the buffer memory slot can be read and written to simultaneously via two different ports as illustrated by the RAM blocks in Figure 4.

As shown in Figure 4, each frame sample in the input buffer is firstly multiplied (MUL block) with a cosine or sine element and then the multiplication results are accumulated (ADD block). After every $\frac{N}{2}$ samples, the current value in the accumulator ($X_{half\_new}(k)$ of current frame) is stored in the RAM at address k (k is the DFT bin index) in order to be used as $X_{half\_old}(k)$ for the upcoming frame.

Simultaneously, the previously stored value in RAM ($X_{half\_old}(k)$, at the same address k) is read via the second port of the dual port RAM and added to (or subtracted from) the current $X_{half\_new}(k)$ in the accumulator by the same ADD block before being replaced by the current value $X_{half\_new}(k)$. The decision between addition or subtraction is dependent on value k, due to the term $e^{j\pi k}$, previously mentioned. The result produced by the ADD element at this time is latched as a DFT coefficient at bin k as follows from Equation (10).

When the next frame arrives, half of the computation of DFT for this new frame is already available from the DFT computation of the previous frame stored in the RAM buffer, hence reducing the calculation latency by half. The process is repeated until the last DFT bin is reached.

The MUL (multiplier) block is a dedicated hardware block common on current FPGA devices. Moreover, the MUL, MUX, and ADD blocks are elements of one primitive MAC block embedded in many low-cost FPGAs. Thus, the proposed hardware architecture can be implemented with simple interconnection and minimum resources on such devices.

The above hardware implementation requires $\frac{N}{2}$ clock cycles to compute each DFT bin. Thus all $\frac{N}{2}$ required frequency bins require only $\frac{N^2}{4}$ clock cycles. If a 50% overlapped frame has 512 samples, with a typical FPGA clock frequency of 100MHz, this represents a latency of 0.16384ms. This is well within the new frame generation rate of 16ms for a 16KHz sample rate. Therefore, the OvlDFT is easily fast enough for speech preprocessing tasks.

2.1.5 Windowing and frame energy computation

In order to reduce spectral leakage, a *window* on the time-domain input signal is usually applied. However, windowing in the time-domain would compromise the symmetry properties utilized by the proposed algorithm and the saved calculation from the previous frame would no longer be valid.

The alternative to applying a window in the time domain is to use convolution to perform the windowing function in the frequency domain (Harris, 1978). Although, as convolution is typically a very time consuming operation, windowing in the frequency domain is

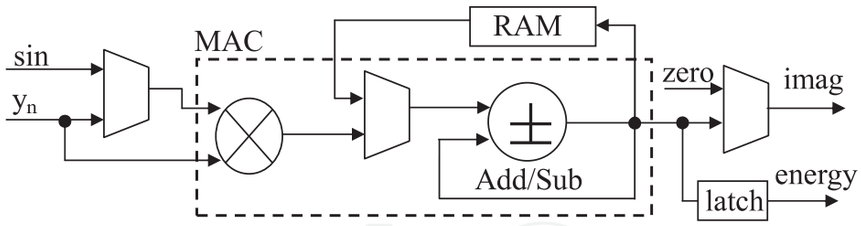


Fig. 5. Modification of the imaginary part of Fig. 4 for energy computation.

only generally used when the window function produces a short sequence of convolution coefficients. Fortunately, this desired property is present in some commonly used window functions such as Hamming and Hann windows. The Hann window produces three values (-0.25, 0.5, and -0.25) which can be easily implemented in the convolution processing by shift registers instead of the more expensive multipliers.

Furthermore, the frame energy power required by the MFCC block (discussed later) can be computed with almost no cost by modifying the first DFT bin calculation phase in the OvlDFT design. In contrast, the energy value must be calculated separately if a normal time-domain window is applied, hence, the OvlDFT algorithm will result in further resource savings when used in a MFCC hardware design.

The frame energy is computed follows. In the DFT computation, the imaginary part of the first frequency component is always zero. This can be exploited to compute the frame energy with a modest amount of additional hardware. The imaginary part of the proposed DFT implementation is modified to embed the frame energy computation by adding two multiplexers and a latch, as shown in Fig. 5.

When the first frequency component of a frame is computed, the input frame sample is fed to the imaginary MAC instead of the sine value (sin). Thus, the input sample will be squared and accumulated in the MAC. Consequently, the final output of this imaginary MAC, when calculating the first component, is the energy of the frame while the actual imaginary part of the first frequency component is tied to zero. For other components, the normal procedure described in Section 2.1.4 is performed.

This method of frame energy computation can only be used in conjunction with frequency-domain windowing. If windowing is performed in the time domain, the frame will be altered, and thus, the frame energy will not be computed correctly.

2.2 Linear spectra subtraction

2.2.1 Algorithm

In an environment with additive background noise $r(n)$, the corrupted version of the speech signal $s(n)$ can be expressed as:

$$y(n) = s(n) + r(n). \tag{14}$$

Following the preprocessing procedure, the captured signal is framed and transformed to the frequency domain by performing the discrete Fourier transform (DFT) on the framed signal $y(n)$:

$$Y(i,\omega) = S(i,\omega) + R(i,\omega), \tag{15}$$

where i is the frame index.

Before the spectral subtraction is performed, a scaled estimate of the amplitude spectrum of the noise $\left|\hat{R}(i,\omega)\right|$ must be obtained in a silent (i.e. no speech) period. An estimate of the amplitude spectrum of the clean speech signal can be calculated by subtracting the spectrum

of the noisy signal with the estimated noise spectrum:

$$\left|\hat{S}(i,\omega)\right|^{\gamma} = \left|Y(i,\omega)\right|^{\gamma} - \alpha(i,\omega)\left|\hat{R}(i,\omega)\right|^{\gamma}, \tag{16}$$

where γ is the exponent applied to the spectra, with $\gamma = 1$ for amplitude spectral subtraction and $\gamma = 2$ for power spectral subtraction. The frequency-dependent factor, $\alpha(i,\omega)$, is included to compensate for under-estimating or over-estimating of the instantaneous noise spectrum. Should the subtraction in Equation (16) give negative values (i.e. the scaled noise estimate is greater than the instantaneous signal), a flooring factor is introduced. This leads to the following formulation of spectral subtraction:

$$\left|\hat{S}_t(i,\omega)\right|^{\gamma} = \left|Y(i,\omega)\right|^{\gamma} - \alpha(i,\omega)\left|\hat{R}(i,\omega)\right|^{\gamma},$$

and

$$\left|\hat{S}(i,\omega)\right|^{\gamma} = \begin{cases} \left|\hat{S}_t(i,\omega)\right|^{\gamma} & \left|\hat{S}_t(i,\omega)\right|^{\gamma} > \beta\left|Z(i,\omega)\right|^{\gamma}, \\ \beta\left|Z(i,\omega)\right|^{\gamma} & \text{otherwise,} \end{cases} \tag{17}$$

where $|Z(i,\omega)|$ is either the instantaneous noisy speech signal amplitude or the noise amplitude estimate, β is the noise floor factor ($0 < \beta \ll 1$). Common values for the floor factor range between 0.005 and 0.1 (Berouti et al., 1979).
The enhanced amplitude spectrum $\left|\hat{S}(i,\omega)\right|$ is recombined with the unaltered noisy speech phase spectrum to form the enhanced speech in the frequency domain and ready to be fed to the further speech processing blocks.

2.2.2 Linear spectral subtraction implementation

A generalized hardware implementation of the spectral subtraction derived directly from the previous description is shown in Figure 6.

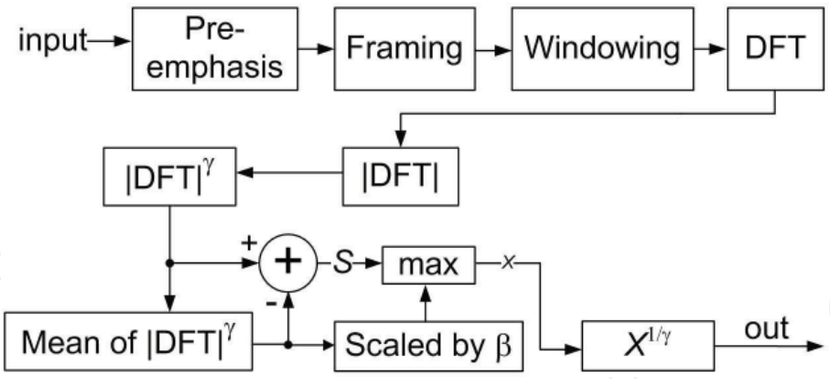


Fig. 6. The block diagram of the generalized implementation of spectral subtraction.

The estimated noise is calculated from the first N frames and stored in an internal buffer by the *Mean of* $|DFT|^{\gamma}$ block. The essence of the spectral subtraction technique occurs through subtracting the stored estimated noise from the subsequent magnitude spectrum for each frame as stated in Equation (17). The result of this subtraction is then compared with a scaled version of the average noise magnitude (known as the noise floor), with the larger of the two chosen as the output, denoted by $|X|$.
To recover the normal magnitude level, $|X|$ is raised to the power of $1/\gamma$. The output of this block, out, is ready to be used as the magnitude part of the enhanced signal to be fed into the speech recognition engine.

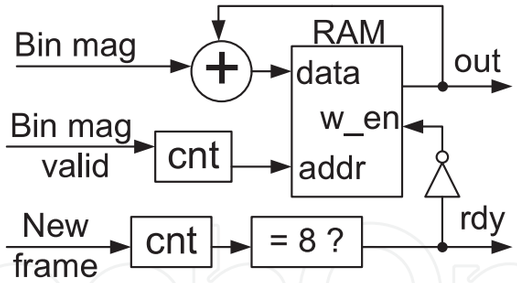


Fig. 7. Noise calculation block

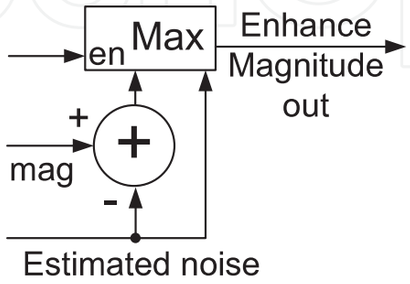


Fig. 8. Noise subtraction block

Algorithm refinement for in-car speech enhancement

For cost-effective automotive speech enhancement, we make the assumption that the noise characteristics $\left|\hat{R}(i,\omega)\right|$ can be accurately estimated during a silent period before the speech, for example 8 frames; and $\left|\hat{R}(i,\omega)\right|$ remains unchanged during the entire speech. Therefore, we can set $\alpha(i,\omega) = 1$ for all the values of i and ω. We also use the noise estimate for the calculation of the noise floor, that is $|Z(i,\omega)| = \left|\hat{R}(\omega)\right|$.

Typically, the parameters γ and β are set to optimize the signal-to-noise ratio (SNR). However, for the best speech recognition performance optimization, we may choose these two parameters differently from their common values (Kleinschmidt, 2010; Kleinschmidt et al., 2007).

It has been shown that magnitude spectral subtraction provides better speech recognition accuracy than power spectral subtraction (Whittington et al., 2008). Therefore, $\gamma = 1$ is selected for our implementation. One important benefit of this selection is that the resource requirement of the implementation is significantly reduced because the need for resource-intensive square and square root operations is avoided.

With $\gamma = 1$, experiments using floating-point software (Whittington et al., 2009) have been used to determine the optimal value of β on part of the AVICAR database (Lee et al., 2004). It has been shown that maximum recognition accuracy can be obtained by setting $\beta = 0.55$ and that the performance is only marginally worse (approximately 0.1%) if we set $\beta = 0.5$. Therefore, $\beta = 0.5$ was selected for the implementation because of its simplicity.

Efficient noise estimation and subtraction

An inefficient design covering the steps to estimate the noise and apply noise subtraction can result in significant additional hardware resources due to the requirement of a complex control flow and data buffering. To achieve low hardware resource usage, a pipeline design is proposed. The design requires no control mechanism as the data is processed in an orderly fashion due to the simple pipeline structure.

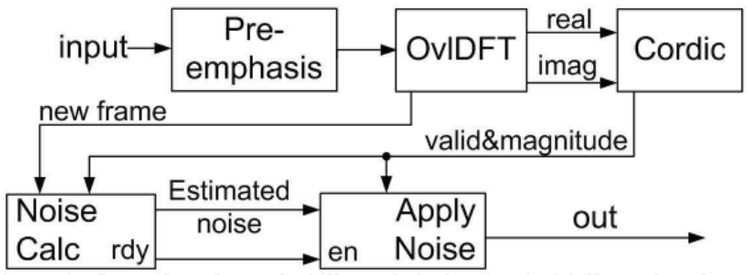


Fig. 9. The block diagram for the FPGA design

The first 8 frames of the input signal are used to compute the estimated noise magnitude spectrum by the structure shown in Figure 7. From frame 9, the noise subtraction is applied by a cascaded structure, as shown in Figure 8.
The magnitude valid pulse, *Bin_mag_valid*, drives an 8-bit counter as the memory location in RAM for the current sample. Concurrently, to perform the noise calculation, the magnitude value is accumulated and stored to the same memory location as long as the signal *rdy* is not set. If *rdy* is set, the same counter will function as the read address to access the estimated noise in the RAM buffer, thus eliminating the need for the complex feedback control from the subsequent block, the *Noise Subt* block.
Similarly, the *New frame* pulse, indicating a new frame, drives a frame counter during noise estimation. When the frame counter reaches 8, the *rdy* signal will be set, indicating the end of the noise estimation process. Signal *rdy* is also used to disable the frame counter and the RAM writing function.
The noise subtraction is applied by a structure shown in Figure 8 and the cascaded wiring in Figure 9. The subtraction result is compared with the associated estimated noise scaled by $\beta = 0.5$. To perform this, the *MAX* block simply re-interprets the estimated noise signal by moving the fraction point of the value one bit to the left, eliminating a shift register.

The proposed structure of the overall system.
The structure of the proposed FPGA implementation of spectral subtraction is shown in Figure 9, with the detail is described below.
The input signal first passes through the speech preprocessing block. In addition to the DFT real and imaginary components, a pulse output signal, *new frame*, is generated to indicate that a frame has been processed. The DFT coefficients are then fed to the Cordic block (a core supplied by Xilinx (Xilinx, 2010) to produce the magnitude of each coefficient.
The essence of the spectral subtraction technique occurs through the *Noise Calc* and the cascaded *Noise Subt* blocks which estimate the noise and perform noise subtraction respectively, as detailed in Section 2.2.2.

2.3 Dual-channel array beam-forming
2.3.1 Algorithm
Beamforming is an effective method of spatial filtering that differentiates the desired signals from noise and interference according to their locations. The direction where the microphone array is steered is called the look direction.
One beamforming technique is the delay-and-sum beamformer which works by compensating signal delay to each microphone appropriately before they are combined using an additive operation. The outcome of this delayed signal summation is a reinforced version

of the desired signal and reduced noise due to destructive interference among noises from different channels.

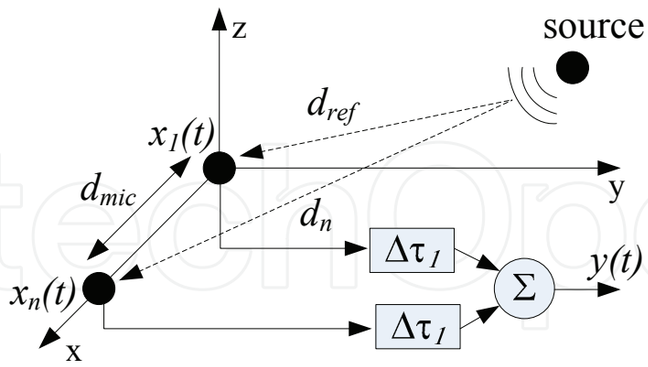


Fig. 10. Dual-microphone delay-and-sum beamforming

As illustrated in Figure 10, consider a desired signal received by N omni-directional microphones at time t, in which each microphone output is an attenuated and delayed version of the original signal $a_n s(t-\tau_n)$ with added noise v_n, is given by:

$$x_n(t) = a_n s\left(t-\tau_n\right) + v_n\left(t\right). \tag{18}$$

In the frequency domain, the array signal model is defined as:

$$\mathbf{X}\left(\omega\right) = S\left(\omega\right)\mathbf{d} + \mathbf{V}\left(\omega\right), \tag{19}$$

where $\mathbf{X} = [X_1(\omega),\ X_2(\omega),\ \cdots,\ X_N(\omega)]^T$, $\mathbf{V} = [V_1(\omega), V_2(\omega),\ \cdots,\ V_N(\omega)]^T$. The vector $\mathbf{d}$ represents the array steering vector which depends on the actual microphone and source locations.

For a source located near the array, the wavefront of the signal impinging on the array should be considered a spherical wave and the source signal is said to be located within the near-field of the array instead of a planar wave commonly assumed for a source located far from the array. In the near field, $\mathbf{d}$ is given by (Bitzer & Simmer, 2001):

$$\mathbf{d} = [a_1 e^{-j\omega\tau_1}, a_2 e^{-j\omega\tau_2}, ..., a_N e^{-j\omega\tau_N}]^T, \tag{20}$$

$$a_n = \frac{d_{ref}}{d_n}, \tau_n = \frac{d_n - d_{ref}}{c}, \tag{21}$$

where d_n and d_{ref} denote the Euclidean distance between the source and the microphone n, or the reference microphone, respectively, and c is the speed of sound.

To recover the desired signal, each microphone output is weighted by frequency domain coefficients $w_n(\omega)$. The beamformer weights are designed to maintain the beam at the look direction to be constant (e.g. $\mathbf{w}^H\mathbf{d} = 1$). For a dual-microphone case, the beamformer output is the sum of each weighted microphone:

$$Y(\omega) = \sum_{n=1}^{2} w_n^*(\omega) X_n(\omega). \tag{22}$$

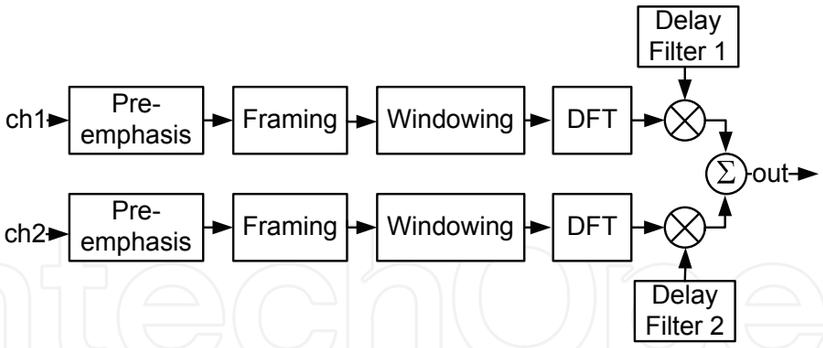


Fig. 11. General diagram of the DASB

The beamformer output $Y(\omega)$ is enhanced speech in the frequency domain and is ready to be fed to the following speech processing blocks. In digital form, the whole process of DASB can be summarized in Figure 11 where the delay filters are defined by the weighting coefficients $w_n(\omega)$.
For fixed microphone positions, the array steering vector d and therefore the weighting coefficients $w_n(\omega)$ will be fixed. Hence, $w_n(\omega)$ can be pre-computed and stored in read-only memory (ROM) to save real-time computation.

2.3.2 Dual-channel array beam-forming implementation

In this section, the duplicated processing sections of the general DASB structure shown in Figure 11 are identified and some efficient sharing mechanisms are proposed.

Sharing between two input channels

The sharing of one hardware block for both input channels can be achieved with a novel and simple modification to the *OvlDFT* structure presented previously (Vu, Ye, Whittington, Devlin & Mason, 2010).
As all the intermediate computations between segments of *OvlDFT* are stored in RAM, the computation of the second input channel can be added by simply doubling the memory space of the Input Buffer as well as the RAM blocks to convert them to "ping-pong" buffers, as illustrated in Figure 12. With the double size buffer, data for Channel 1 and Channel 2 can be located on the lower and upper half of the memory respectively. When a segment of Channel 1 is finished, the Input Buffer's address is increased, and the most significant bit of each memory address will be automatically set so that the second half of the memory is examined. Thus, Channel 2 will then be processed automatically.
Assuming the speech input is a sequence of N real samples, only $\frac{N}{2}$ frequency bins are needed. The output of the system will be sequences of $\frac{N}{2}$ DFT coefficients of the first channel, followed by an equivalent sequence of the second channel.

Delay filter sharing

In the frequency domain, the process of filtering is simply the multiplication of the DFT coefficients of the input signal with the corresponding delay filter coefficients. Delay filter coefficients are pre-computed and stored in read only memory (ROM).
As discussed previously, the overlapping DFT produces DFT coefficients of the two channels alternatively in one stream. Thus, to make the structure simple and easy to implement, the coefficients of the two delay filters are stored in one block of ROM; one filter is located in the lower half of address space while the other is located in the upper half. These filter coefficients

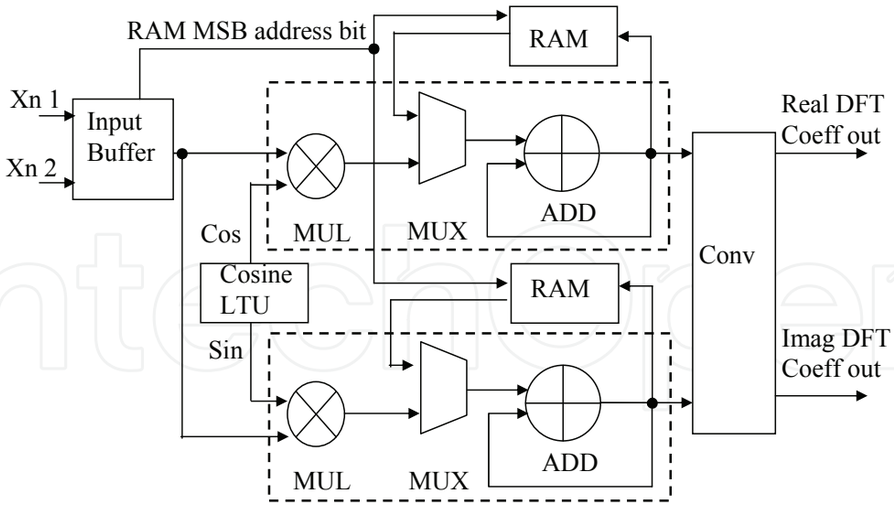


Fig. 12. Dual channel overlapping DFT hardware structure

can be read independently by the most significant bit of the ROM address, which changes automatically when the address is increased.

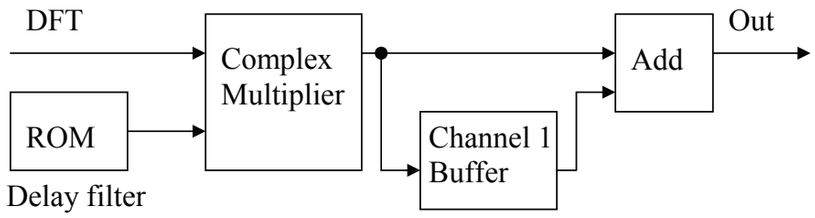


Fig. 13. DASB Delay Filter Diagram

Figure 13 shows the diagram of the Delay filter used for both channels. The product of the filter coefficient (from the lower half of the ROM) and the corresponding DFT coefficient (from the sequence of channel 1) is buffered at the same address of the Channel 1 Buffer block memory. When the DFT coefficients of Channel 2 are calculated and multiplied with filter coefficients from the upper half of the ROM, the product will be added to the Channel 1 delay filter product (already stored in the buffer) to produce the final DASB output.

FPGA implementation

The FPGA design consists of three main blocks as illustrated in Figure 14.

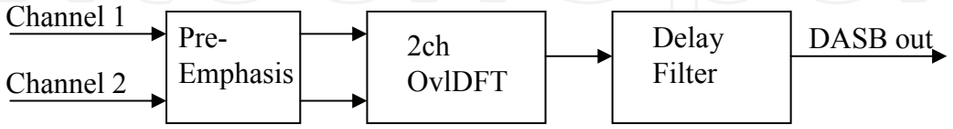


Fig. 14. FPGA design diagram of DASB

The first block is the pre-emphasis filter. The common practice for setting this pre-emphasis filter is given by $y(i) = x(i) - 0.97x(i-1)$, where $x(i)$ and $y(i)$ are the ith input and output samples, respectively. Its implementation requires a delay block, a multiplier and an adder. The second block is the dual-channel overlapping frame DFT as presented in Section 2.3.2 with Hann windowing. The Input Buffer using dual-port BlockRAM is configured as a circular

buffer. Two input channels are multiplexed so that they are stored into the same circular buffer at the lower and upper memory location, respectively.
The third block is the delay filter as presented in Section 2.3.2 and shown in Figure 13. As there is a large time gap between any two DFT coefficients, only one MAC primitive is used to perform the complex multiplication through 4 clock cycles. This provides further saving of hardware resources.
The FPGA design of the DASB can easily process dual 16 bit inputs at 16 KHz sample rate in real-time with the master clock as low as 8.2 MHz.

3. Speech recognition feature extraction front-end

The speech recognition front-end transforms a speech waveform from input devices, such as a microphone, to a parametric representation which can be recognized by a speech decoder. Thus, the front-end process, known as feature extraction, plays a key role in any speech recognition system. In many systems, the feature extraction front-end is implemented using a high-end floating-point processor, however, this type of implementation is expensive both in terms of computer resources and cost.
This section discusses a new small footprint *Mel-Frequency Cepstrum Coefficients* front-end design for FPGA implementation that is suitable for low-cost speech recognition systems. By exploiting the overlapping nature of the input frames and by adopting a simple pipeline structure, the implemented design only utilizes approximately 10% of the total resources of a low-cost and modest-size FPGA device. This design not only has a relatively low resource usage, but also maintains a reasonably high level of performance.

3.1 Mel-frequency cepstrum coefficients

Following the speech preprocessing and enhancement, the signal spectrum is calculated and filtered by F band-pass triangular filters equally spaced on the Mel-frequency scale, where F is a number of filters. Specifically, the mapping from the linear frequency to the Mel-frequency is according to the following formula:

$$Mel(f) = 1127 \ln(1 + \frac{f}{700}). \tag{23}$$

The cepstral parameters are then calculated from the logarithm of the filter banks amplitude, m_i, using the discrete cosine transform (DCT) (Young et al., 2006):

$$c_k = \sqrt{\frac{2}{F}} \sum_{i=1}^{F} m_i \cos\left[\frac{\pi k}{F}(i - 0.5)\right]. \tag{24}$$

where index k runs from 0 to $K-1$ (K is the number of cepstral coefficients).
The higher order cepstral coefficients are usually quite small so that there is a large variation of cepstral coefficients between the low-order and high-order coefficients. Therefore, it is handy to re-scale the cepstral coefficients to achieve similar magnitudes. This is done by using a lifter scheme as follows (Young et al., 2006):

$$c'_k = (1 + \frac{L}{2} \sin \frac{\pi k}{L}) c_k, \tag{25}$$

where c'_k is the rescaled coefficient for the c_k value.

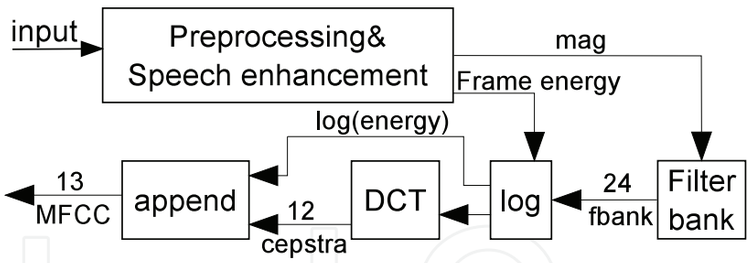


Fig. 15. Block diagram of the overall FPGA design

An energy term is normally appended with the cepstra. The energy (E) is computed as the logarithm of the signal energy, that is, for speech frame $\{y(n), n = 0, 1, \cdots, N-1\}$.

$$E = \log \sum_{n=0}^{N-1} y^2(n). \tag{26}$$

Optionally, time derivatives, Delta and Acceleration Coefficients, can be added to the basic static parameters which can greatly enhance the performance of a speech recognition system. The delta coefficients are computed using the following regression formula

$$d_t = \frac{\sum_{\theta=1}^{\Theta} \theta(c_{t+\theta} - c_{t-\theta})}{2\sum_{\theta=1}^{\Theta} \theta^2}, \tag{27}$$

where d_t is a delta coefficient at time t computed in terms of the corresponding static coefficients $c_{t-\Theta}$ to $c_{t+\Theta}$, and the value of Θ is the window size. Acceleration coefficients are obtained by applying the same formula to the delta coefficients.

3.2 MFCC front-end implementation

In many applications, such as an in-car voice control interface, low power consumption is important, but low cost is vital. Therefore, the design will first attempt to save resources and then reduce latency for low-power consumption (Vu, Whittington, Ye & Devlin, 2010).

3.2.1 Top-level MFCC front-end design

The new front-end design consists of 5 basic blocks as illustrated in Fig. 15, which has 24 Mel-frequency filter banks and produces 39 observation features: 12 cepstra coefficient and one frame energy value, plus their delta and accelerator time derivatives.
The core MFCC blocks include: *Filter-bank*, Logarithm, *DCT* block (combining DCT and the lifter steps), and Append-Deltafy (computing Delta and Accelerator time derivatives) blocks are described in later sections.

3.2.2 A note on efficient windowing by convolution

As noted previously, the speech preprocessing with OvlDFT performs windowing by convolution with an embedded frame energy computation. Although the circular convolution is simple, significant hardware resources are specifically required to compute the first and the last frequency bins.
Each of the other output frequency bins depend on three input components: the previous bin, itself and the following bin. However, the first frequency bin requires the last frequency bin to compute the circular convolution and via versa. This incurs an additional hardware resource cost for buffering and control.

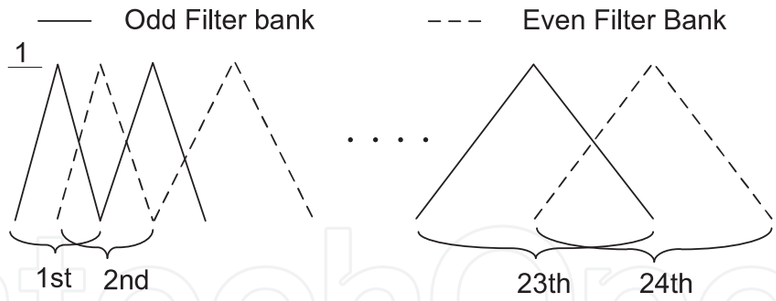


Fig. 16. Triangular Mel frequency filter bank

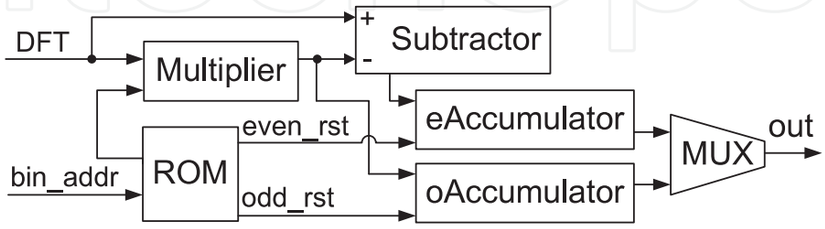


Fig. 17. Block diagram for Mel frequency bank calculation

These hardware resource costs can be saved if band-limiting is applied. Very low and very high frequencies might belong to regions in which there is no useful speech signal energy. As a result, a few frequency bins at the beginning and the end of the frequency range can be rejected without a significant loss of performance. Thus hardware used for the additional buffering and control can be saved.
In related work, Han et al. proposed an MFCC calculation method involving half-frames (Han et al., 2006). However, in their method, the windowing is performed in the time domain and the Hamming window is applied to the half-frames instead of the full-frames in the original calculation. As the method presented here applies the window function on the full-frames, in theory, the output of this method should have a smaller error from the original calculation than the method of Han et al.

3.2.3 Mel filter-bank implementation

The signal spectrum is calculated and filtered by 24 band-pass triangular filters, equally spaced on the Mel-frequency scale. Dividing the 24 filters into 12 odd filters and 12 even filters as shown in Figure 16 leads to a simplification in the required hardware structure.
As the maximum magnitude of each filter is unity and aligned with the beginning of the next filter (due to the equal separation in the Mel-frequency scale), the points of the even filter banks can be generated by subtracting each of the odd filter bank samples from 1. Thus, only the odd-numbered filters need to be calculated and stored, leading to the saving of significant memory space.
More specifically, if the weighted odd power spectrum, E_{odd}, is calculated first then the weighted even power spectrum, E_{even}, can be easily computed as:

$$E_{even} = X_k(1 - W_{odd}^k) = X_k - E_{odd}, \tag{28}$$

where X_k is the power of the frequency bin k; and W_{odd}^k is the associated weight values from the stored odd filter.

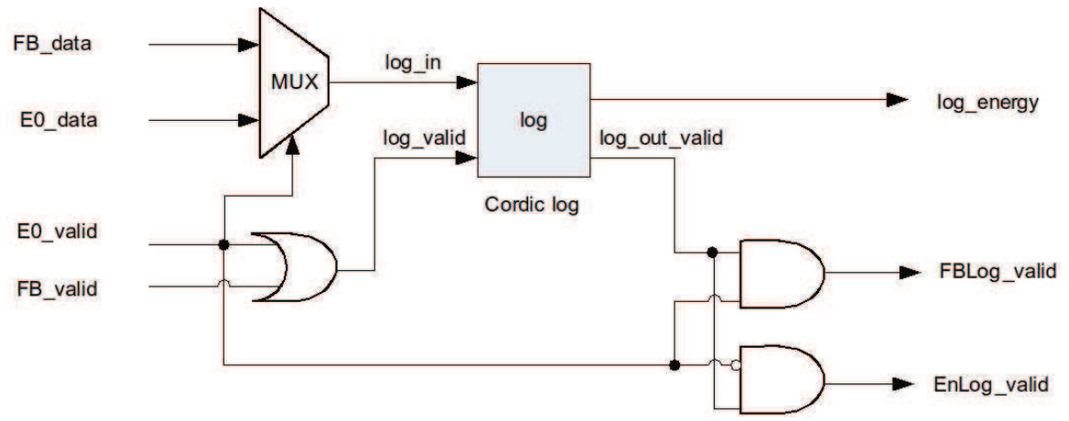


Fig. 18. Sharing scheme for logarithm calculation

The above observation leads to efficient implementation of the filter bank algorithm shown in Figure 17. The data from speech preprocessing is processed in a pipelined fashion through the Multiplier blocks. The Multiplier block multiplies each data sample with the odd filter value at the corresponding sample location (according to the frequency bin address, *bin_addr*) producing E_{odd}, while E_{even} is an output of the *Subtractor* block. These products are then added to the value in the odd and even accumulators (*oAccumulator* and *eAccumulator* blocks) successively. The resulting either odd or even filter-bank data values are then merged into the *out* stream by the multiplexer (*MUX*).
The ROM stores the frequency bin address where the accumulators need to be reset in order to start a fresh calculation. The same process is repeated until 24 filterbank values have been calculated.
Equation (28) was also investigated by Wang et al. (Wang et al., 2002), although, without the distinction of the odd and the even filter, where a complex Finite State Machine (FSM) for control is required as described in (Wang et al., 2002). This complex FSM normally generates a long latency as well as requiring significant hardware resources.
In contrast, the work presented here results in a much simpler pipeline implementation (with only 1 multiplier, 1 ROM and 3 adders/subtractors) and thus saves more hardware resources. Furthermore, this implementation runs in a pipeline fashion with a much smaller latency; it requires only $N + 4$ (where N is the number of frequency bins) clock cycles to compute any number of filters.

3.2.4 Logarithm calculation

Two different data points in the MFCC design, triangle filter banks and frame energy output, are required to perform the logarithm. Figure 18 shows a structure of sharing one logarithm block to compute both data streams alternatively.
A multiplexer is needed to select if either the incoming energy data or filter bank data are to be processed by the *log* block which is implemented by using CORDIC logic core provided by Xilinx (Xilinx, 2010).
From this block, the logarithmic operation is applied to the input data so long as valid signals are active high. Log_valid is activated if either E0_valid or FB_valid signals are high. If the logarithmic value is available at the output, then the log_out _valid signal will go high. Energy_valid indicates when the logarithmic value of the E0 energy is available at the output. This signal will only be high when both log_out_valid and E0_valid are true. Similarly,

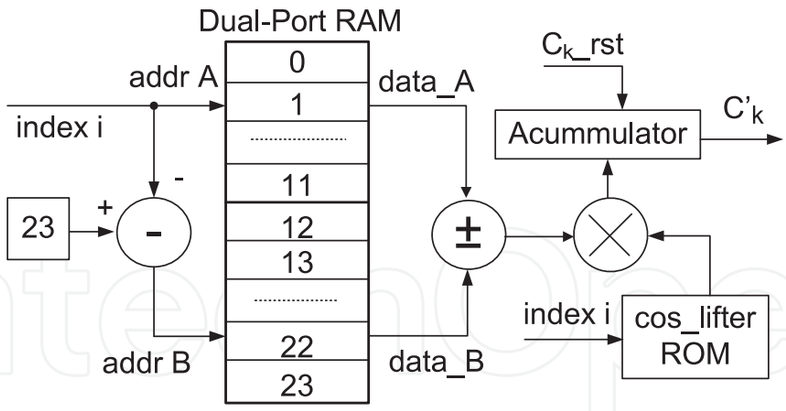


Fig. 19. Block diagram for cepstra and lifter calculation

FBLog_valid indicates if the logarithmic value of the filter bank coefficient is available at the output.

3.2.5 Cepstra and lifter computation

The Mel-Frequency Cepstral Coefficients (MFCCs) are calculated from the log filter bank's amplitude m using the DCT defined in Equation (24). The cosine values are multiplied with the constant $\sqrt{\frac{2}{F}}$ (F is 24 in this example) and are stored in a ROM, prior to the summation operation. This yields the following equation:

$$c_k = \sum_{i=1}^{24} m_i \, r_i, \tag{29}$$

where $r_i = \sqrt{\frac{2}{F}} \cos\left[\frac{\pi k}{F}(i - 0.5)\right]$.
Due to the symmetry of the cosine values, the summation can be reduced to a count from 1 to 12 according to the following formula:

$$\begin{aligned} c_k &= \sum_{i=1}^{12} \left[m_i r_i + (-1)^{i-1} m_{25-i} r_i \right] \\ &= \sum_{i=1}^{12} r_i \left[m_i + (-1)^{i-1} m_{25-i} \right]. \end{aligned} \tag{30}$$

As discussed previously, it is advantageous to re-scale the cepstral coefficients to have similar magnitudes by using the lifter in Equation (25). A separate calculation of this lifter formula is essential, although it requires time and resources. However, by combining the lifter formula with the pre-computed DCT matrix, r_i, the lifter can be calculated without any extra time or hardware cost. Thus, r_i now becomes:

$$r_i = \sqrt{\frac{2}{F}} \cos\left[\frac{\pi k}{F}(i - 0.5)\right] (1 + \frac{L}{2} \sin\frac{\pi k}{L}). \tag{31}$$

The block diagram of cepstra and lifter computation is presented in Fig. 19. A dual-port RAM is first filled with the 24 log filter bank amplitudes. Then, two symmetrical locations are read from the RAM via two independent ports. The RAM data outputs are added to,

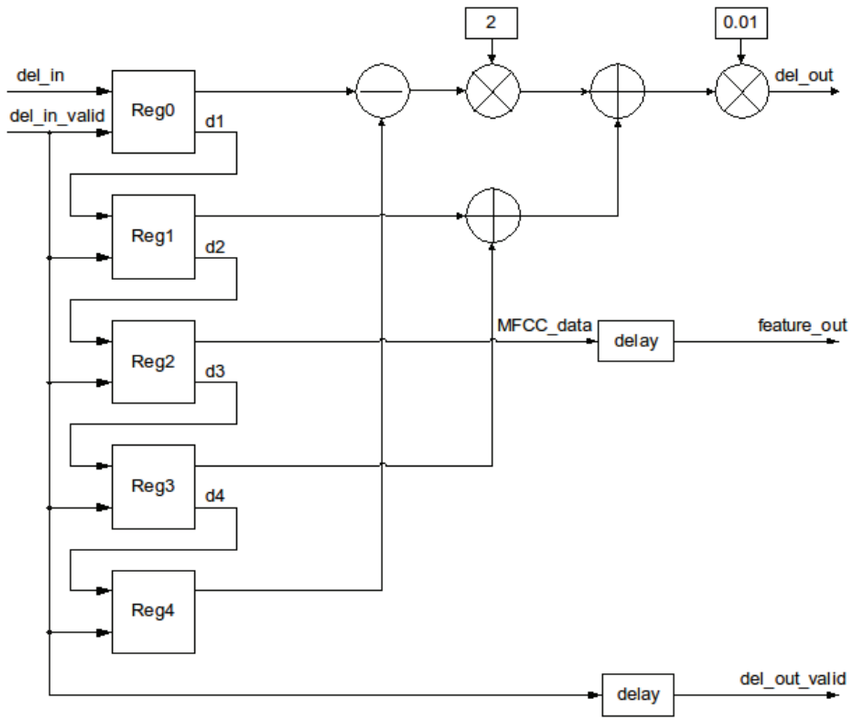


Fig. 20. Deltafy block diagram

or subtracted from, each other respectively. The computation results and the constant values from the ROM are then processed by a MAC which performs a multiplication and accumulates the resulting values. The accumulator of the MAC is reset by C_{k_rst} at the beginning of every new c'_k computation.

3.2.6 Deltafy block

In this work, both delta and accelerator coefficients have a window size of 2, so they can share the same formula (Equation 27) and the same hardware structure. With the window size having a value of 2 ($\Theta = 2$ in Equation 27), the derivative d_t of element c_t is calculated by the following formula

$$d_t = \frac{(c_{t+1} - c_{t-1}) + 2 \times (c_{t+2} + c_{t-2})}{10}. \tag{32}$$

Figure 20 shows the corresponding hardware structure of both delta and accelerator computation. The thirteen elements of MFCC data (12 DCT coefficient appended by one frame energy value) are shifted from register Reg0 to Reg4. When all four registers have one valid element, the signal del_in_valid enables the derivative calculation for the element in Reg2 only, performing the computation shown in Equation 32.
The del_out_valid signal then enables the MFCC data and its derivative to be available at the output. Then, the MFCC elements in each register are shifted forward by one register and the above process is repeated.
The accelerator coefficients are computed by cascading the same hardware after the delta computation hardware.

4. Performance evaluation

Having constructed the new OvlDFT hardware design and integrated it into a speech recognition feature extraction front-end with two embedded speech enhancement techniques, it is necessary to validate their performance. For this work there are four aspects of interest: (i) is the OvlDFT is an effective DFT processing block; (ii) how well does the fixed-point hardware MFCC feature extraction front-end match a floating-point software equivalent; (iii) do the embedded speech enhancement techniques improve speech recognition performance; and (iv) how effective in terms of hardware resource usage and hardware processing are these implementations.

4.1 Testing and resource usage of FPGA design

The small footprint fixed-point hardware MFCC feature extraction with embedded speech enhancement design has been implemented on a Xilinx Spartan 3A-DSP 1800 development board. As this is a low-cost and modest-size FPGA device the, the design's resource utilization can illustrate the advantages of this hardware implementation. The development of the FPGA design was conducted block by block, based on equivalent floating-point MATLAB implementation. Each block was tested after it was completed to ensure correct operation before the next block was developed.

To verify key sections and the complete design, test data signals were fed into the system with the output data passed to a computer for analysis. This output was then compared with that from a floating-point model of the system. To determine relative quantization error range, both the FPGA and the floating-point model outputs were converted back into the time domain.

Testing of the OvlDFT design

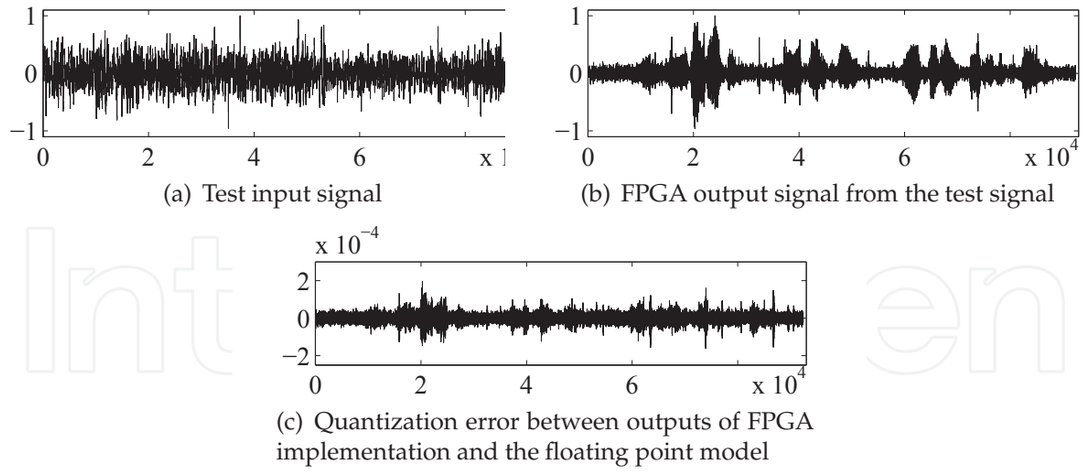
To test the OvlDFT, the same speech files from the AVICAR database (Lee et al., 2004) were fed to the OvlDFT and a floating-point Matlab DFT. The power spectrum of the both versions was then compared side by side. The comparison result showed that the two output data sets are identical to the fourth or fifth digit following the decimal point. This experiment was repeated using other AVICAR speech files used in the later speech recognition experiments with corresponding results.

Testing of the LSS design

Figure 21 shows an example of the input (speech from the AVICAR, AF2_35D_P0_C2_M4.wav file), corresponding output, and quantization error of the hardware system compared to the floating point model output. It can be seen that the enhanced output is much cleaner than the inputs and that the quantization error is of the order of 10^{-4}. This test was repeated with similar AVICAR speech files used in the later speech recognition experiments and resulted in a consistent quantization error.

Testing of the DASB design

Two speech files from microphone 2 and 6 of the AVICAR (AF2_35D_P0_C2 records) were chosen for Channel 1 and Channel 2 respectively. Figure 22 shows the test inputs and output of the fixed-point FPGA system and the difference between the FPGA output and that of the floating-point model. Here it can be seen that the enhanced output is clean and the error is within the range of $\pm 10^{-4}$. This test was repeated with a range of AVICAR data sets used later in the performance experiments with all cases exhibiting a consistent error of $\pm 10^{-4}$.

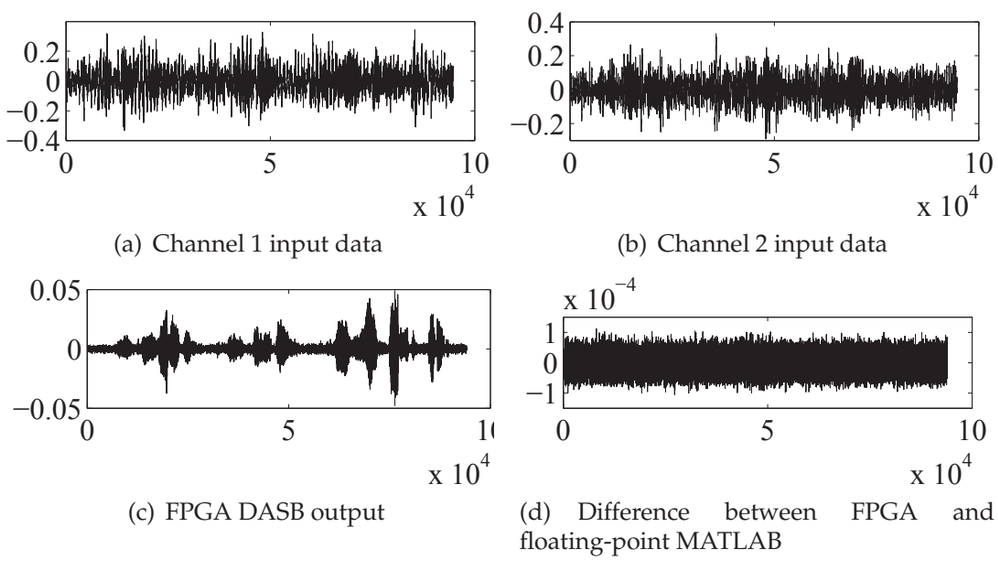


(a) Test input signal

(b) FPGA output signal from the test signal

(c) Quantization error between outputs of FPGA implementation and the floating point model

Fig. 21. An example of input and output signals of the LSS FPGA design



(a) Channel 1 input data

(b) Channel 2 input data

(c) FPGA DASB output

(d) Difference between FPGA and floating-point MATLAB

Fig. 22. The input, output and FPGA quantization error of DASB on the test files

Testing of the MFCC design

To test the quantization error of the MFCC design, a comparison of the fixed-point FPGA output was made with the output from the equivalent floating-point Hidden Markov Model Toolkit (HTK) (Young et al., 2006). The configurations of both HTK and the FPGA design are: 512 samples per 50% overlapped frame, 50Hz-7950Hz cut-off frequency, 24-filter filter bank, 12 cepstra coefficients and lifted by a parameter $L = 22$. Using the same speech input, the quantization error is 10^{-3} which is still consistent over the AVICAR test set.

FPGA resource usage

Table 1 shows the resource usage of the MFCC front-end with embedded DASB and then LSS speech enhancement. With the LSS applied first, the hardware is slightly larger due to the application on both channels. However, due to the low working clock rate, the hardware

Resources	Available	Enhanced MFCC resources	Usage
Slices	16640	3401	20.44%
BRAMs	84	28	33.33%
Multiplier	84	12	14.28%

Table 1. Resource usage on Spartan-3A DSP 1800 device of the MFCC feature extraction design with embedded speech enhancement

resource can be share between the two channels, thus, the additional hardware is mainly input/output buffer generated using BRAM blocks.
The FPGA resource utilisation for the design at present is only around 14% to 33%, thus only a modest portion of the target FPGA resources have been utilised, giving significant space for other future designs, such as, the implementation of a speech recognition decoder.
The MFCC implementation with speech enhancement processes data in pipeline. This pipeline requires only a 4.1Mhz clock, which is the required clock of the slowest component (the OvlDFT), to process a 16KHz sample rate speech in real-time. Hence, if a significant higher clock was used, say 100Mhz, the resulting spare processing capacity could be applied to addition tasks, such as, enabling input from a large microphone array.

4.2 Recognition performance

Validation of speech enhancement performance in this context can only be measured through statistical analysis of speech recognition rates for various enhancement scenarios, including the no enhancement case, using data sets containing a variety of speakers. Experiments for this work were conducted using the phone numbers task of the AVICAR database (Lee et al., 2004).

AVICAR database

AVICAR is a multi-channel Audio-Visual In-CAR speech database collected by the University of Illinois, USA. It is a large, publicly available speech corpus designed to enable low-SNR speech recognition through combining multi-channel audio and visual speech recognition. For this collection, an array of eight microphones was mounted on the sun visor in front of the speaker who was positioned on the passenger's side of the car. The location of the speaker's mouth was estimated to be 50 cm behind, 30 cm below and horizontally aligned with the fourth microphone of the array (i.e. 58.3cm in a direct line). The microphones in the array are spaced 2.5 cm apart. Utterances for each speaker were recorded under five different noise conditions which are outlined in Table 2 (Lee et al., 2004)

Condition	Description
IDL	Engine running, car stopped, windows up
35U	Car travelling at 35 mph, windows up
35D	Car travelling at 35 mph, windows down
55U	Car travelling at 55 mph, windows up
55D	Car travelling at 55 mph, windows down

Table 2. AVICAR noise conditions

The speech recognition experiments involved passing sets of the AVICAR speech waveforms through the hardware feature extraction unit (incorporating the OvlDFT and embedded speech enhancement) followed by the HTK speech decoder. This was repeated for each of the various enhancement scenarios in turn, as well as the no enhancement case to provide a

baseline reference. All speech recognition results quoted below are word correction (in %), calculated as:

$$WordCorrection = \frac{N-D}{N}.100\% \tag{33}$$

Where:

- N represents the total number of words in the experiment;
- D the number of correct words omitted in the recogniser output;

Performance experiments

To evaluate the designs, A baseline speech recognition is first set up for comparison. For this work, the HTK software is used as a recognition engine for both of the baseline and the FPGA system.

In these experiments, the baseline used HCopy supplied by HTK as the speech recognition front-end, while the the FPGA design produces the MFCC features which are then fed directly to the HTK recognition engine. In both cases, the HTK recognition engine uses an acoustic model trained by HTK tools from a Wall Street Journal corpus with 16 Gaussians per state. To simplify the evaluation steps, the continuous speech phone numbers task (i.e. digit sequences) has been used with the following grammar:

```
$digit = one | two | three | four | five |
  six | seven | eight | nine | oh | zero;
  ( SENT-START <$digit> SENT-END )
```

There are about 60 sentences in the test set of each noise condition. Each sentence is a speech of 10 digit phone number, thus, there are around 600 digits in total to be recognised for each noise condition. For the speech recognition experiments of the LSS enhancement alone, speech file from microphone 4 (central to the speaker) were used. While for evaluation of other scenarios which all include DASB, speech files from microphone 2 and 6 (equal distant on either side of the speaker) were used.

This experiment was designed to provide an indicative measure of the speech recognition performance of the hardware design. What is important here is to show that there is an improvement in speech recognition performance using hardware speech enhancement techniques, not the absolute value of the speech recognition performance. To conduct such a test, a huge speech database and complex language model would be required with test conducted across a wide range of scenarios, this is beyond the scope of this work.

	IDL	35U	35D	55U	55D	Average
Baseline	88.0	67.8	56.1	58.8	29.0	59.9
FPGA LSS	90.8	70.7	58.6	64.4	47.0	66.3

Table 3. Word correction of FPGA LSS-MFCC design

The Linear Spectral Subtraction scenario demonstrates clear improvement over the no enhancement baseline case under all noise conditions. Although the improvement is a rather modest 2-3% for the lower noise conditions, becoming a more substantial 18% for the noisiest condition, 55MPH, windows down (Table 3).

The Delay-Sum Beamforming scenario provides a substantial improvement over the baseline of between 17-20% for all but the lowest noise (idle) condition where the improvement is still over 5% (Table 4). The DASB also provides greater recognition improvement than the Linear Spectral Subtraction alone for all cases apart from the noisiest condition, where the improvement is basically the same for both techniques.

	IDL	35U	35D	55U	55D	Average
Baseline	88.0	67.8	56.1	58.8	29.0	59.9
FPGA DASB	93.6	86.1	74.0	78.1	46.5	75.6

Table 4. Word correction of FPGA DASB-MFCC design

	IDL	35U	35D	55U	55D	Average
Baseline	88.0	67.8	56.1	58.8	29.0	59.9
FPGA DASB-LSS	94.2	87.8	77.5	81.4	62.3	80.6

Table 5. Word correction of FPGA DASB-LSS-MFCC design

	IDL	35U	35D	55U	55D	Average
Baseline	88.0	67.8	56.1	58.8	29.0	59.9
FPGA LSS-DASB	94.6	88.4	78.3	82.7	62.3	81.2

Table 6. Word correction of FPGA LSS-DASB-MFCC design

Cascading the two enhancement techniques results in even greater performance improvement than either scenario operating alone. As shown in Table 5 and 6, the improvement is over 20% in all but the lowest noise case which has an improvement of more than 6%. While the recognition rate for highest noise case, 55MPH windows down, has more than doubled. The order in which the hardware enhancement blocks cascaded doesn't seem to exhibit any significant difference for this test. While the recognition performance of the Linear Spectral Subtraction followed by Delay-Sum Beamforming is between 0 and 1.3% higher than the reverse case this difference is likely to be within the limitation of the test.

5. Discussion and conclusions

In this chapter, a small footprint FPGA hardware implementation of a MFCC feature extraction front-end with embedded speech enhancement has been presented. The two speech enhancement techniques were chosen because of their simpleness and effectiveness for the example in-car application.

By exploiting the overlapping nature of the input frames and other redundancy in data and control processes, the design has achieved a modest hardware resource usage on a low-cost FPGA. The patented OvlDFT is a key to this small hardware utilization, and along with other optimization has resulted in only about 20% utilisation of a Spartan 3A DSP 1800 device.

In addition, the design is able to work in real time with a clock of only 4.1 Mhz which is very much slower than a typical FPGA clock of 100 Mhz. These two factors illustrate how little of the FPGA's speech processing potential is actually used, leaving significant room for addition designs, such as, a hardware speech decoder. Furthermore, with the sharing of speech preprocessing hardware between the MFCC feature extraction and the two speech enhancement designs, the embedded speech enhancement feature is provided almost for free. Speech recognition experiments using noisy files from the AVICAR database indicates the speech enhancement provides clear improvement in recognition performance, particularly for cases where the speech enhancement techniques are combined.

However, it should be noted that this speech recognition test is rather limited in scope. In that, it only uses the AVICAR phone numbers task and microphones in limited positions. To gain a more accurate measurement of speech recognition performance, comprehensive experiments would be required using many speech databases and vocabularies. To do this requires considerable time and processing power which was beyond the range of this work where the

focus was on the hardware design. Also, tests using different microphone positions relative to the speaker should be conducted as microphone position may impact on performance.
In conclusion, the real-time hardware feature extraction with embedded signal enhancement for automatic speech recognition design has been demonstrated to be effective and equivalent in performance to a comparable software system. Furthermore, it exhibits characteristic suitable for application in low-cost automotive applications, although it may also be used in other noisy environments.

6. Acknowledgment

The authors gratefully acknowledge the Cooperative Research Centre for Advance Automotive Technologies (AutoCRC) for their partial support of this work.

7. References

Aarabi, P. & Shi, G. (2004). Phase-based dual-microphone robust speech enhancement, *IEEE Transactions on Systems, Man, and Cybernetics, Part B* 34(4): 1763–1773.

Ahn, S. & Ko, H. (2005). Background noise reduction via dual-channel scheme for speech recognition in vehicular environment, *IEEE Transaction on Consumer Electronics* 51(1): 22–27.

Beh, J., Baran, R. H. & Ko, H. (2006). Dual channel based speech enhancement using novelty filter for robust speech recognition in automobile environment, *IEEE Transaction on Consumer Electronics* 52(2): 583–589.

Benesty, J., Makino, S. & Chen, J. (2005). *Speech Enhancement*, Springer.

Berouti, M., Schwartz, R. & Makhoul, J. (1979). Enhancement of speech corrupted by acoustic noise, *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 208–211.

Bitzer, J. & Simmer, K. U. (2001). Superdirective microphone arrays, *in* M. S. Brandstein & D. B. Ward (eds), *Microphone Arrays*, Springer, chapter 2, pp. 19–38.

Boll, S. (1979). Suppression of acoustic noise in speech using spectral subtraction, *Acoustics, Speech and Signal Processing, IEEE Transactions on* 27(2): 113 – 120.

Han, W., Chan, C.-F., Choy, C.-S. & Pun, K.-P. (2006). An efficient mfcc extraction method in speech recognition, *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, pp. 4 pp.–.

Harris, F. (1978). On the use of windows for harmonic analysis with the discrete fourier transform, *Proceedings of the IEEE* 66(1): 51–83.

Johnson, D. H. & Dudgeon, D. E. (1992). *Array Signal Processing: Concepts and Techniques*, Simon & Schuster.

Kleinschmidt, T. (2010). *Robust Speech Recognition using Speech Enhancement*, PhD thesis, Queenslan University of Technology. `http://eprints.qut.edu.au/31895/1/Tristan_Kleinschmidt_Thesis.pdf`.

Kleinschmidt, T., Dean, D., Sridharan, S. & Mason, M. (2007). A continuous speech recognition evaluation protocol for the avicar database, *1st International Conference on Signal Processing and Communication Systems*, Gold Coast, Australia, pp. 339–344.

Lee, B., Hasegawa-johnson, M., Goudeseune, C., Kamdar, S., Borys, S., Liu, M. & Huang, T. (2004). Avicar: Audio-visual speech corpus in a car environment, *in Proc. Conf. Spoken Language, Jeju, Korea*, pp. 2489–2492.

Lin, Q., Jan, E.-E. & Flanagan, J. (1994). Microphone arrays and speaker identification, *Speech and Audio Processing, IEEE Transactions on* 2(4): 622 –629.

Lockwood, P. & Boudy, J. (1992). Experiments with a nonlinear spectral subtractor (nss), hidden markov models and the projection, for robust speech recognition in cars, *Speech Commun.* 11(2-3): 215–228.

Ortega-Garcia, J. & Gonzalez-Rodriguez, J. (1996). Overview of speech enhancement techniques for automatic speaker recognition, *Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on*, Vol. 2, pp. 929 –932.

Vu, N., Whittington, J., Ye, H. & Devlin, J. (2010). Implementation of the mfcc front-end for low-cost speech recognition systems, *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pp. 2334 –2337.

Vu, N., Ye, H., Whittington, J., Devlin, J. & Mason, M. (2010). Small footprint implementation of dual-microphone delay-and-sum beamforming for in-car speech enhancement, *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pp. 1482 –1485.

Vu, N. (2010). Method and device for computing matrices for discrete Fourier transform (DFT) coefficients, *Patent No. WO/2010/028440*.

Wang, J.-C., Wang, J.-F. & Weng, Y.-S. (2002). Chip design of mfcc extraction for speech recognition, *Integr. VLSI J.* 32(1-3): 111–131.

Whittington, J., Deo, K., Kleinschmidt, T. & Mason, M. (2008). Fpga implementation of spectral subtraction for in-car speech enhancement and recognition, *Signal Processing and Communication Systems, 2008. ICSPCS 2008. 2nd International Conference on*, pp. 1–8.

Whittington, J., Deo, K., Kleinschmidt, T. & Mason, M. (2009). Fpga implementation of spectral subtraction for automotive speech recognition, *Computational Intelligence in Vehicles and Vehicular Systems, 2009. CIVVS '09. IEEE Workshop on*, pp. 72–79.

Widrow, B. & Stearns, S. D. (1985). *Adaptive Signal Processing*, Prentice-Hall.

Xilinx (2010). Cordic v4.0 product specification.
URL: *http://www.xilinx.com/support/documentation/ip_documentation/cordic_ds249.pdf*

Young, S. J., Evermann, G., Gales, M. J. F., Hain, T., Kershaw, D., Moore, G., Odell, J., Ollason, D., Povey, D., Valtchev, V. & Woodland, P. C. (2006). *The HTK Book, version 3.4*, Cambridge University Engineering Department, Cambridge, UK.

Speech Technologies

Edited by Prof. Ivo Ipsic

ISBN 978-953-307-996-7
Hard cover, 432 pages
Publisher InTech
Published online 23, June, 2011
Published in print edition June, 2011

This book addresses different aspects of the research field and a wide range of topics in speech signal processing, speech recognition and language processing. The chapters are divided in three different sections: Speech Signal Modeling, Speech Recognition and Applications. The chapters in the first section cover some essential topics in speech signal processing used for building speech recognition as well as for speech synthesis systems: speech feature enhancement, speech feature vector dimensionality reduction, segmentation of speech frames into phonetic segments. The chapters of the second part cover speech recognition methods and techniques used to read speech from various speech databases and broadcast news recognition for English and non-English languages. The third section of the book presents various speech technology applications used for body conducted speech recognition, hearing impairment, multimodal interfaces and facial expression recognition.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen