FPGA HARDWARE FOR SPEECH RECOGNITION USING HIDDEN MARKOV MODELS

José L. Gómez-Cipriano, Roger P. Nunes, Dante A.C. Barone

Informatics Institute
Federal University of Rio Grande do Sul, Porto Alegre – Brazil
jgomez@inf.ufrgs.br

ABSTRACT

Some speech recognition applications, like speaker verification, dialog recognition or the speech to text transcription could require real time processing and a good precision. Other applications such as toys, automotive vehicles or portable machines still could aggregate the portability and low-power requirements, in addition to physical compactness. These requirements could require a hardware solution for the speech recognition problem.

The current work proposes an architecture using hardware based in FPGAs, optimizing the pre-processing and parameter extraction for performing efficient speech recognition.

1. INTRODUCTION

Most of existing speech recognition systems (SRS) use a software designed for personal computers. The added program works continuously in an operating system (Windows, Linux, etc.), requiring a computer with a compatible sound board. However, a disadvantage of such systems in some applications is the requirement of a personal computer: it is not economically viable to use a full computerized system in the control of a washing machine or a TV device, for example. A specific hardware may be a solution. Moreover, a hardware speech recognition system, when the datapath is specified based in the speech recognition algorithms, could execute operations faster than a general purpose processors implementation [1] [2].

A system with these features could be used in the experimentation of different stages of speech recognition technology, reducing the physical space used by a speech recognition software running in a PC.

The goal of this work is to obtain functional blocks for a low weight portable speech recognition system of isolated words, using FPGAs [3]. In the next sections each component system, based in FPGAs, is shown.

2. THE PRE-PROCESSING SYSTEM

The input of this system is the voice signal, filtered by an anti-aliasing filter and converted to digital format with an A/D converter. The voice signal was recorded in RAW format, using 16 bits and a sampling rate of 11025Hz.

In figure 1, the block diagram of the pre-processing proposed is shown. This hardware system has a pre-emphasis functional block, a frame division function and a windowing block.

\[ S(n) \rightarrow \text{pre-emphasis} \rightarrow \tilde{S}(n) \rightarrow \text{frame separation} \rightarrow Q_1(n) \rightarrow \text{windowing} \rightarrow W(n) \times Q_1(n) \]

**Figure 1:** Pre-processing of a digital speech signal \( S(n) \).

2.1. The Pre-emphasis Function

The digital voice signal \( S(n) \) is filtered by a pre-emphasis filter:

\[ \tilde{S}(n) = S(n) - a_{\text{pre}} S(n-1) \]  

where \( \tilde{S}(n) \) is the output of the filter [6]. The pre-emphasis coefficient \( a_{\text{pre}} \) was approximated by the rational number 15/16, because the simplicity of division by 16 reducing the hardware complexity.

**Figure 2:** Datapath of the pre-emphasis function.
Some modifications were made in equation 1, to furnish a fixed point implementation:
\[
S(n) = S(n) - \frac{15}{16} S(n-1) = S(n) - S(n-1) + \frac{S(n-1)}{16} \tag{2}
\]
In figure 2, the datapath of this function is shown. A division circuit, a 20 bits adder, a 16 bits subtractor, an accumulator and two registers were used.

For external data, 16 bits were used. However a 20 bits datapath was used for internal operations in order to obtain a better precision.

The division by 16 is obtained making a 4-bits shift right and introducing four zeros in the more significant bits position, allowing a one clock pulse division.

2.2. The Frame Separation Function

After pre-emphasis, the voice signal must be separated in overlapping frames \(i=0, ..., T-1\), where \(T\) is the number of frames that the voice signal could be divided. Each frame is formed by 252 voice samples (22ms aprox.) making the frame size near to 20ms, in order to guarantee stationarity [4]. Moreover, the number of samples (252) allows to make a frame composition by three equal size blocks (84 samples, each block), necessary for the pipeline designed in the frame overlapping.

Because of the frame size and overlap, each sample falls into the first third of one analysis frame, the second third of the previous analysis frame and the final third of the two previous analysis frames. After 84 samples, when one of the three frames is completed, the relationship of the three analyzed frames rotates cyclically. As a result the Hamming window presented both the problem of producing the cosine-based values and rearranging the segments of the window upon completing a frame.

An algorithm that allows the optimal hardware implementation of frame separation and overlapping was developed.

The algorithm operates in the next manner: In order to obtain the frames, the samples after the pre-emphasis are first segmented in blocks \(j=1...T\), where \(T\) is the number of blocks that could be formed from the voice signal. Figure 3 shows this blocks segmentation. Each block \(j\) has 84 samples, with no overlap between the blocks. Each frame is composed by 3 successive blocks (252 samples).

A internal dual-port RAM memory, that allows read/write parallel operations, was used for the temporal storage of 252 samples. This memory was divided in three memory segments \(M_0, M_1, M_2\), where each segment has 84 samples.

Figure 4 shows how the frame separation is made. Each new block \(j\) begins to be stored in the segment memory \(M_j\), where the samples of the two previous blocks are stored. Each block is multiplied by multiplied by the correspondent values of the Hamming window, \(w(n)\), \(n=0...251\), before storing in memory. The final frame multiplied by the Hamming window is stored in an output register.

In each time period, only 28 samples of the new block are stored. It represents the third part of the 84 samples of the block. In parallel, the content of the segment memory that has the samples of the last block is read. In the next period time, the final 28 samples of the new block are stored. At the same time, the segment memory that corresponds to the last block is also read.

2.3. The Windowing Function

The Hamming window, has the equation [6]:
\[
w(n) = 0.54 - 0.46 \cdot \cos\left(\frac{2\pi \cdot n}{N_s - 1}\right), 0 \leq n \leq N_s - 1 \tag{3}
\]
where \(N_s\) is the number of samples of each window and \(n\) is the sample being evaluated. The window size is 252 samples (22ms aprox.), in order to simplify the Hamming window and the frame separation implementation. In figure 5, the windowing datapath join with the frame division block is shown.

A ROM memory stores the first 126 values of the Hamming window (\(n=0...125\), corresponding to the first two quads of the cosine function. The window values for other points are calculated using cosine function periodicity properties. Hamming memory addressing is made by an up/down binary counter.
3. THE MEL-CEPSTRA PARAMETERS EXTRACTION SYSTEM

In figure 6, a diagram for the mel-cepstra parameters extraction is shown. First, the spectral energy is computed for windowed sequential frames. After this, the energy in each of the 27 channels of a triangular bandpass filters set is calculated. Finally, the discrete cosine transform (DCT) of the log energy is computed.

A simple FFT processor was first implemented, using known circuits [2].

In figure 7, the hardware for triangular filters function is shown. The values of the triangular filters are stored in a ROM. A three stage pipeline is used. The output of this bank are 27 energy values.

A logarithm processor using the CORDIC algorithm [5] was implemented in order to obtain the log energy value required for this system. Figure 8 shows the logarithm processor implementation, formed by a scaling circuit, a hardware implementation of the CORDIC algorithm and a control unit. The logarithm processor uses 20 clock pulses for the calculus of a logarithm value.

The outputs of the triangular filters (Filters_Value signal) are scaled in order to obtain numbers between 0.5 and 1, that allows the calculus of the CORDIC algorithm for logarithm function. The radix obtained by the scaling is used to get the final result of the logarithm operation.

A DCT pipelined memory optimized algorithm [6] was also implemented. In figure 9, the DCT architecture is shown. A ROM stores cosine values for the DCT calculus. A three stages pipeline was used.
4. THE VITERBI PROCESSOR

![Datapath of Viterbi processor.](image)

![Addressing circuit for the Viterbi processor.](image)

Figure 10: Datapath of Viterbi processor.

Figure 11: Addressing circuit for the Viterbi processor.

A Viterbi decoder for speech recognition was also implemented [7]. The goal was the use in left-right Hidden Markov models. The vector quantization stage that makes the link between the Mel-Cepstra extraction block and the Viterbi processor is not implemented yet. For testing the system, the quantization was made in software. In figure 10, the datapath for the Viterbi processor is shown. The addressing circuit for the Viterbi processor is shown in figure 11.

5. EXPERIMENTAL RESULTS

The functions explained above were implemented using the Maxplus II tool, in order to use them with FPGAs. They implementation in Matlab and in C was also made, in order to compare behavioral simulations with hardware results. Tests were made with an isolated word small vocabulary for industrial control of elevators.

In table 1, a comparison between the implementation using speech recognition in hardware vs. software implementation is shown.

<table>
<thead>
<tr>
<th>Stage</th>
<th>HW</th>
<th>SW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter Extraction and Pre-processing</td>
<td>2085</td>
<td>110000</td>
</tr>
<tr>
<td>Recognition with Viterbi decoding</td>
<td>3,75</td>
<td>50000</td>
</tr>
</tbody>
</table>

Table 1: Hardware vs. software results.

6. CONCLUSIONS

The speed characteristics of a dedicated circuit, the physical space, the flexibility of the VHDL description and the potential of FPGA systems make this design usable for the development of new applications. Hardware description language specification opens the possibility to synthesize such systems in different circuit technologies, generating application specific integrated circuits. The system proposed will be used in problems that require a small vocabulary and a limited speaker number.

Many optimizations were made, to obtain a low latency and a low use of memory.

7. REFERENCES


