HARDWARE IMPLEMENTATION OF DISTRIBUTED SPEECH RECOGNITION SYSTEM FRONT END

Ahmad A. Al Sallab, Dr. Hossam Fahmy, Prof. Dr. Mohsen Rashwan

Electronics and Communications Department, Faculty of Engineering, Cairo University
Cairo, Egypt
ahmad.elsallab@gmail.com

ABSTRACT

*Modern speech recognition applications are heading towards embedded systems and hand-held devices. **D**istributed **S**peech **R**ecognition (DSR) system architecture emerged to address this kind of applications. Most of the existing implementations of this system are presented in software fashion, with little consideration to the end product platform in which the system will be deployed. In this paper, an optimized hardware implementation of the front end part of the DSR specified in the basic ETSI Aurora standard ETSI ES 201 108 is presented in FPGA platform prototype, with consideration of migration to structured ASIC in case of mass-production. Main design issues and tips are highlighted. Results are presented in terms of hardware resources utilization, comparison of some basic system components to third party reference designs and compliance to the Aurora standard.*

1. INTRODUCTION

Human-machine interaction is likely to take place in natural language in future embedded systems and mobile devices. Speech enabled car navigation; natural language e-Learning applications and home automation are among those applications. This inherits all the embedded systems design constraints to the speech recognition domain, like limited hardware, memory, power consumption and cost, which creates the need to re-architecture the already existing speech recognition systems.

Early attempts generated the client-server **N**etwork **S**peech **R**ecognition (NSR) architecture [5], where the voice signal is encoded with normal speech coding techniques at the client *Front end* and sent via voice channel to the sever *Back end*, where all the recognition process takes place after decoding the voice signal. Communication between the two parts is to take place via wired/ wireless channel. This architecture provides light front end mobile terminal, which reduces its cost, hardware and power consumption. However, this scheme suffered two main problems: First, ordinary speech encoding/decoding techniques do not preserve many characteristics needed for the speech recognition, where speech codes care about perceptual quality rather than improving the recognition results, which degrades the Word Error Rate (WER) at the back end [5]. Second, encoding the voice signal makes it exposed to the channel errors during transmission, especially in case of un-reliable wireless channel [5].

Distributed speech recognition (DSR) architecture emerged to tackle the above problems of NSR. Multi-modal applications are those applications where interaction between user and the computer may take place in keyboard strokes, voice commands or even hand writing. This requires voice and data channels. DSR reduces the two channels to only single data channel over which voice and data can be transmitted, by sending parameterized representation of the speech signal (the features vector) instead of coding the voice signal directly as in NSR. This has two advantages: First, the speech signal is not directly encoded and transmitted, which protects it against channel errors, and improves the WER significantly over unreliable channels. Second, special encoding and framing algorithms are used, that focus on lowering the bit rate (4.8 kbps) rather than preserving the perceptual quality.

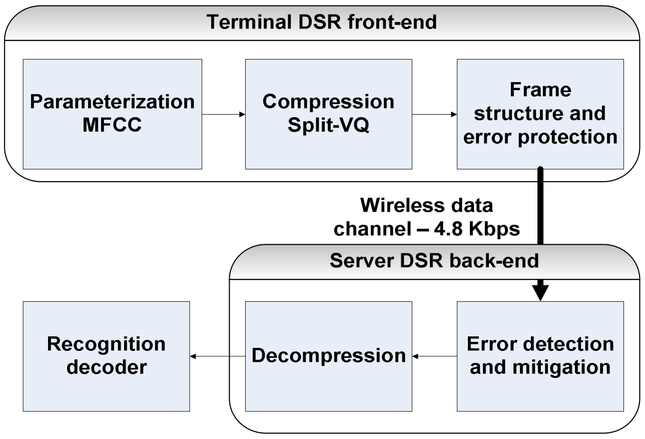

Figure 1 DSR System Architecture

2. ETSI AURORA STANDARD OVERVIEW

The STQ-Aurora DSR group at the **E**uropean **T**elecommunications **S**tandards **I**nstitute (ETSI) has published four standard specifications featuring the front end side of the DSR. The main algorithms specified are: features extraction (Mel-Frequency Cepstral Coefficients- MFCC), the compression split-vector quantization and the framing and error protection algorithms. The front end algorithm block diagram specified in [3] is shown in Figure 2. Three sampling rates configurations are supported (8, 11 and 16 kHz), where the frame length, frame rate and frame overlapping varies according to the configured sampling rate. For the three configurations, the input speech frame shift interval is 10 ms. The features vector (14 features) is compressed into 7 indices using split vector quantization algorithm. The compressed speech frame is then formatted in a multi-frame packet format, where the target data rate out of the front end is 4.8 kbps.

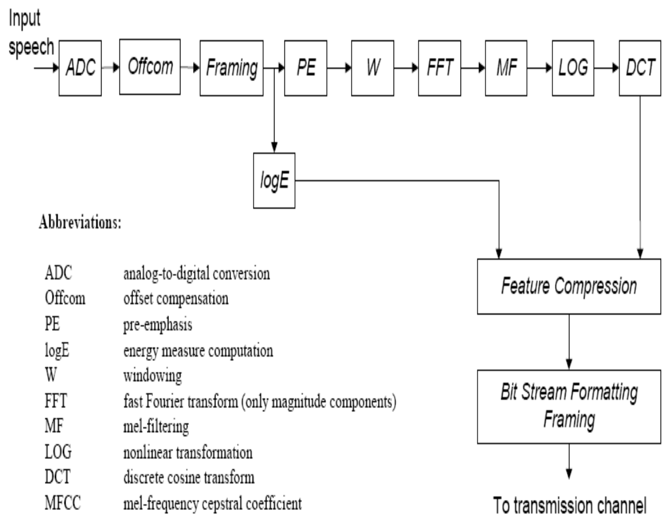
Input speech
ADC
Offcom
Framing
PE
W
FFT
MF
LOG
DCT
logE
Feature Compression
Bit Stream Formatting Framing
To transmission channel
Abbreviations:
ADC analog-to-digital conversion
Offcom offset compensation
PE pre-emphasis
logE energy measure computation
W windowing
FFT fast Fourier transform (only magnitude components)
MF mel-filtering
LOG nonlinear transformation
DCT discrete cosine transform
MFCC mel-frequency cepstral coefficient

Figure 2 Block diagram of the Front end algorithm in Aurora Standard ETSI ES 201 108

In the following, we will present a proposal for a hardware implementation of the front end part of the basic standard.

3. HARDWARE IMPLEMENTATION

In our design, **F**ield **P**rogrammable **G**ate **A**rray (FPGA) design style is chosen for prototyping and migration to structured **A**pplication **S**pecific **I**ntegrated **C**ircuit (ASIC) is chosen for mass production. The customization capabilities offered by this style gives flexibility in optimizing the hardware utilization and area in the target chip, which reduces the final product cost. In addition, this style is characterized by its low power consumption, which makes it suitable for hand-held battery powered devices.

The design proposed here is either optimized for memory resources or for processing time. As appears in Figure 2, the algorithm has many complex components, like Hamming window, LogE, FFT, LOG, DCT and many others. Those components contain complex non-linear trigonometric, logarithmic and other complex functions that can be either calculated on the fly, or stored in a Look-up table (LUT), like the Hamming window factors or the trigonometric functions, which requires extra memory. Some hardware optimized numerical algorithms exist to calculate such functions, like the **CO**rdinate **R**otation **D**igital **C**omputer (CORDIC) [7], which was used extensively in the memory optimized solution to calculate complex functions instead of storing their results in a LUT, which comes on the expense of extra processing time. On the contrary, the time optimized solution chooses to store the look-up table (LUT) of such functions, which reduces the required processing time.

The idea of having two solutions comes from the relaxed time constraint on the system, where the frame rate specified in [3] is 10 ms. The net processing time available for the front end part is only 9.16 ms after removing the header and CRC overheads, which is relatively relaxed time compared to nowadays chip frequencies, hence, giving room for optimizing hardware by using hardware optimized, but time consuming algorithms like CORDIC. On the other hand, if time is critical for the user of the chip, the other time optimized option is also available.

In the next sub-sections, some main components of the system will be discussed.

2.1 CORDIC Core

The CORDIC algorithm is used extensively in this design, due to two main reasons; first, its hardware implementation is highly optimized, where it utilizes only adders/subtractors, shift registers and one look-up table. This simple hardware can perform a lot of complex functions, ranging between trigonometric, hyperbolic and linear functions, which are the three types of the algorithm. Second, the accuracy of the result is high in small number of iterations, and simple convergence constraints.

The main target of the algorithm is to rotate an input complex vector by certain angle; this is called the *rotation mode*. The other mode is the *vectoring mode*, where it is required to align the input vector to the x-axis. The combination of the three types with the two modes of the algorithm can give a very wide range of complex functions.

A simple, configurable hardware is presented in [7]. Table 1 and Table 2 show the usage of the CORDIC in our system, and the corresponding configuration. Table 1 shows the usage of CORDIC in both time and memory optimized solutions, while Table 2 shows the extra usages in the memory optimized solution only. For more information about how the functions are calculated, please refer to [7].

Usage	Type	Mode
FFT magnitude	Trigonometric	Vectoring
LogE	Hyperbolic	Vectoring
Hardware divider	Linear	Vectoring
Non-linear tranform	Hyperbolic	Vectoring

Table 1 CORDIC usage in both time and memory optimized solutions

Usage	Type	Mode
Hamming window factors	Trigonometric	Rotation
FFT twiddle factors	Trigonometric	Rotation
DCT cosine	Trigonometric	Rotation

Table 2 CORDIC usage in memory optimized solution only

In time optimized solution, only two CORDIC cores are needed, the first to calculate LogE feature, and the other one to be re-used between the FFT magnitude, the natural logarithm of the Mel-filter output and hardware divider. In memory optimized solution, a CORDIC core is needed for the

Hamming window, and another one for the LogE feature, and the last one to be re-used between FFT, Mel-output natural logarithm, hardware divider and DCT.

2.2 Hamming window

The Hamming window equation is [3]:

$$s_w(n) = \left\{0{,}54 - 0{,}46 \times \cos\left(\frac{2\pi(n-1)}{N-1}\right)\right\} \times s_{pe}(n),\ 1 \le n \le N$$

Where N is the speech frame length in samples, n is the sample order, S_{pe} is the pre-emphasis filter result and S_w is the window filtered sample.

In the memory optimized solution, CORDIC module [7] is used to calculate the cosine in the above equation with every new sample, and hence the window factor is calculated.

In the time optimized solution, only half of the window is stored in a LUT ROM, and the rest is deduced from it.

2.3 Fast Fourier Transform (FFT)

The basic FFT equation is [3]:

$$bin_k = \left|\sum_{n=0}^{FFTL-1} s_w(n) e^{-jnk\frac{2\pi}{FFTL}}\right|, \quad k = 0,\dots,FFTL-1.$$

Where bin_k is the magnitude of the resulting FFT coefficients, and FFTL is the length of the FFT result vector.

Split radix-2 algorithm was used for FFT calculation [1] [2] and [6]. In the memory optimized solution of FFT, shown in Figure 3, the twiddle factor complex multiplication involved in the butterfly operations is interpreted as vector rotation, since multiplication by a complex exponential is equivalent to rotating the multiplied complex vector by the argument of the exponential. Here the hardware optimized CORDIC core [7] is utilized as shown in the architecture below. This optimized core reduces the hardware utilization and memory requirements.

On the other hand, the time optimized solution tends to store the twiddle factors in a LUT ROM of length equals to the length of the FFT vector, and performing complex multiplication of the FFT vectors and the complex exponential, which requires extra memory and hardware multipliers. However, instead of storing all the twiddle factors, only the first quadrant values of the cosine function is stored, where the rest of the wave can be deduced from it. Sine values of the complex exponential can be deduced from the cosine values. This reduces the memory requirements by 75%. The time optimized architecture is similar to the one in Figure 3, with substitution of the CORDIC core with LUT ROM of the twiddle factors.

2.4 Discrete Cosine Transform (DCT)

The DCT basic equation is [3]:

$$C_i = \sum_{j=1}^{23} f_j \times \cos\left(\frac{\pi \times i}{23}(j-0{,}5)\right), \qquad 0 \le i \le 12$$

Where C_i is the 13 dimension result vector of the DCT, and f_j is the result of the non-linear transformation after the Mel-filter banks output.

Same discussion about the Hamming window goes here, where the memory optimized solution calculates the cosine values using CORDIC core [7], while the time optimized solution stores the cosine values in a LUT ROM.

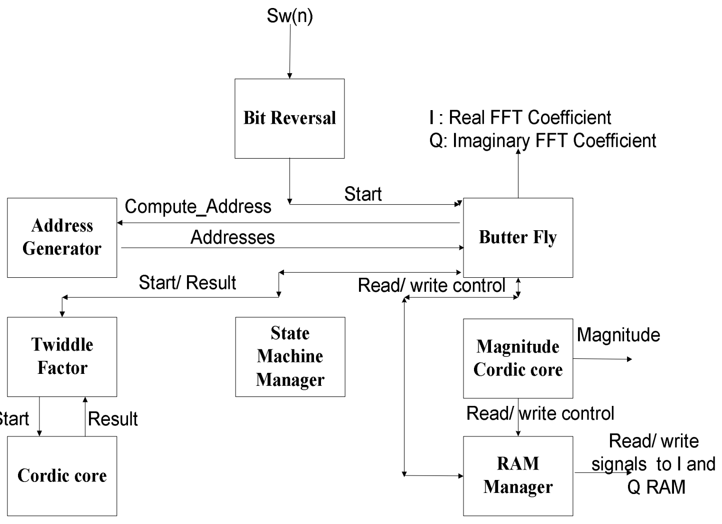

Figure 3 Memory optimized FFT architecture

2.5 Memory manager

This component is implementation specific, and not mentioned in [3]. It controls the access to the system RAM and ROM. The RAM memories used in the system are:

- *Input samples RAM*: this memory is managed in a circular buffer fashion to manage the frame overlapping requirements. It has size of N (speech frame length).
- *I RAM:* this memory is used to store the real part of the intermediate FFT radix-2 stages. It holds the real input and output of the FFT. It has size of FFTL (the FFT length).
- *Q RAM:* same as I RAM, but for the imaginary part of the intermediate FFT radix-2 stages.

The last two memories are re-used in the Mel-filter and DCT components after the FFT is finished. The system has some ROM memories to hold some constant values:

- *Mel-filter banks centre frequencies (25 entries)*
- *Quantization tables (14 tables, each of length = 64 entries, except for tables 12 and 13, which have 256 entries)*

In addition, the time optimized solution needs extra ROM:

- *Hamming Window factors (N/2 length, where N is the speech frame length)*
- *Twiddle factors (FFTL/4 length, where FFTL is the length of the FFT result vector)*
- *DCT factors (24 DCT factors)*

4. SYSTEM EVALUATION

The evaluation of the presented design will be held on three axes: First, the system hardware utilization and time performance will be presented. Second, some of the main system components that are usually used in benchmarking are compared to third party reference designs and other designs. And at last, the compliance of the system output is validated against the official standard test vectors provided by the ETSI.

3.1 Hardware utilization and Processing time performance

Table 3 shows the result of synthesising the design on the 10K gates Altera FPGA device Cyclone III EP3C10U256C8, for both memory and time optimized solutions, with the three configurations specified in the Aurora standard [3].

Solution	Configuration	Logic gates	Total Memory
Memory optimized	8 kHz	9,980 (97%)	4.47 Kbytes
	11 kHz	9,980 (97%)	4.75 Kbytes
	16 kHz	9,722 (94%)	6.45 Kbytes
Time optimized	8 kHz	8,160 (79%)	4.87 Kbytes
	11 kHz	8,160 (79%)	5.18 Kbytes
	16 kHz	8,160 (79%)	7.19 Kbytes

Table 3 Hardware utilization of the memory and time optimized solutions over the three configurations

Table 4 shows the frame processing time performance as a percentage of the net allowed frame time after removing the header and CRC overheads (9.16 ms). A chip frequency of 100 MHz (after synthesis) is assumed.

	8 kHz	11 kHz	16 kHz
Memory optimized	1.3856%	1.4459%	2.7021%
Time optimized	0.8563%	0.9166%	2.005%

Table 4 processing time performance of the memory and time optimized solutions

Table 4 shows that the time optimized solution gives better results in terms of hardware utilization of logic gates and time performance. On the other hand, the memory optimized solution is better in terms of memory usage, but not so far from the time optimized solution.

3.2 Comparison to other designs

In this section the FFT and CORDIC modules are evaluated against the Altera reference designs and other designs. Altera reference designs are available from Altera to be used as IP core (called MegaCore function or Mega function), with documentation available on Altera web site, www.altera.com.

3.2.1 FFT vs. other design

The FFT component is usually used to benchmark most of **D**igital **S**ignal **P**rocessing (DSP) systems. Table 5 shows the comparison between Altera's reference design of the FFT (Burst data flow architecture, 256 points, single output, 16 bits signal width) [12], other designs and the FFT of our system, in the time optimized solution. All designs have 16 bits fixed point signal length, and 256 point FFT length.

	FPGA	Logic Elements	Memory (Bits)	Multipliers (18X18)	Clock Cycle Count
Altera [12]	Cyclone III EP3C10F256C6	1,463	9,472	4	1628
Design in [9]	Stratix II EP2S15F672C3	6702	20480	48	43
Design in [10]	Stratix II EP2S60F1020C4	1334	--	--	--
Front end FFT	Cyclone III EP3C10F256C6	998	9,232	4 (18X18)	256

Table 5 FFT comparison to other designs

The results in Table 5 show that the FFT design presented here outperforms Altera reference design in terms of hardware utilization of logic elements and memory bits, while they both utilize 4 18x18 multipliers. In terms of time performance, the local FFT design takes only 256 cycles to finish, while the reference design takes 1628 cycles, which means that the reference design takes 6.36 times as that of the local FFT design.

The design in [9] is very efficient in terms of clock cycles count, however, this comes on the expense of hardware, memory and multipliers resources usage.

The enhanced performance of the FFT design proposed here is due to the following reasons:

- The optimized usage of memory, especially in the LUT of the sine and cosine factors as discussed in 2.3.
- Fixing the internal signals lengths to 16 bits (less than 18) optimizes the usage of the embedded multipliers (18 x 18) on the FPGA.
- Using the dual-port RAM feature of the used FPGA enables simultaneous memory access during FFT stages, which improved the time performance by 50%.
- Using split-radix FFT algorithm with only *one* butterfly core and iterating on it highly reduced the hardware utilization.
- Pipelining between the FFT components (bit-reversal, butterfly, twiddle factors and address generator) improved the time performance.

3.2.2 CORDIC vs. Altera's reference design

Table 6 shows the comparison between Altera's CORDIC reference design [11] and our implementation of the CORDIC hardware on Cyclone FPGA devices.

	Clocks	Logic elements
Altera's Reference CORDIC	16	963
Front end CORDIC	16	399
Ratio (Reference/ Local design)	1	2.41

Table 6 CORDIC benchmarking against Altera's reference design

The results in Table 6 show that the local CORDIC design takes the same clock cycles to finish as the reference design. However, in terms of logic elements utilization, the reference CORDIC design uses logic elements about 2.4 times as the local CORDIC design.

3.3 Compliance to the ETSI Aurora Basic Standard

The ETSI provides reference high level C-Code together with reference test vectors of 8 ms of continuous speech, which represents about 813 speech frame, to test proprietary implementations against them to prove compliance to the Aurora standard. The design presented here was tested against those results. Table 7 shows average fixed point error between the reference and the **S**ystem **U**nder **T**est (SUT), with 8 and 16 kHz configurations. The word length is 16 bits.

	8 kHz	16 kHz
Average fixed point error	0.002621	0.004883

Table 7 Average fixed point error in 8 and 16 kHz configurations

Table 7 shows that the system under test output is correct to the third decimal place. Note that, the above average error is the error between the features after the quantization block in both reference and under test systems.

5. RELATED WORK

A similar implementation of the front end module in Aurora ETSI system is presented in [8]. Note that; the design in [8] is only the features extraction part, so the comparison held here does not include the rest of quantization and framing components of the front end client.

	FPGA	Logic Elements	DSP units
Proposed design	Cyclone III EP3C10U256C8	7,221	46
Design in [8]	Stratix EP1S20F484C5	18,340	52

Table 8 Comparison between proposed front end design and the one in [8]

The usage of algorithms with low hardware resources utilization like CORDIC reduced the hardware resources and DSP units in the proposed design in this paper. Also, limiting the fixed point length of the internal signals to 16 bits enabled using the embedded multipliers and DSP MAC units on the used FPGA (18 bits width). Other platform dependent optimizations like using the dual-port RAM capability; DSP MAC units improved the utilization. Finally, re-use of some components and using single processing cores and iterating on them in many complicated operations like FFT, Mel filtering and DCT optimized the resources utilization. The design in [8] is more concerned with re-usability, so some of the platform capabilities might not be exploited as the one proposed here, which is more customized.

6. CONCLUSION

In this paper, we presented a hardware solution to implement the front end part of a distributed speech recognition system, taking the front end of the ETSI Aurora basic standard as a reference. The hardware platform chosen for implementation is FPGA for prototyping and structured ASIC for mass production.

Two solutions were presented: memory optimized and time optimized. Hardware optimized algorithms like CORDIC were used in the design, especially in the memory optimized solution to reduce the ROM needed to store constant values and calculate it instead. Results show that the design can fit in a low cost Cyclone III 10 K gates FPGA. Two main components were used to obtain system benchmarks against corresponding reference designs provided by Altera and other designs, which are FFT and CORDIC components. The result of the comparison is highly in the favour of our system in terms of hardware resources or time performance.

Finally, compliance to the reference standard being implemented (the basic ETSI Aurora Standard) is proved by comparing the system final output to the reference standard output over 8 ms of continuous speech. The result of comparison shows compliance between fixed point and reference outputs to the third decimal place.

REFERENCES

[1] Xudding Huang, Alex Acero, Hsiao-Wuen Hon, " Spoken Language Processing, A guide to Theory, Algorithm, and System Development ", Prentice Hall, 2001.

[2] Sen M Kuo, Bob H Lee and Wenshun Tian, " Real-Time Digital Signal Processing, Implementations and Applications ", Second Edition, John Wiley and Sons, 2001.

[3] European Telecommunications Standard Institute, ETSI, " Speech Processing, Transmission and Quality Aspects (STQ); Distributed speech recognition; Front-end feature extraction algorithm; Compression algorithms ", Aurora Standard, ETSI ES 201 108 V1.1.3 (2003-09).

[4] David Pearce, " Enabling New Speech Driven Services for Mobile Devices: An overview of the ETSI standards activities for Distributed Speech Recognition Front-ends ", in *Proc. AVIOS 2000*, The Speech Applications Conference, San Jose, CA, USA, May 22-24, 2000.

[5] Dmitry Zaykovskiy, " Survey of the Speech Recognition Techniques for Mobile Devices ", in *Proc. SPECOM 2006*, 11-th International Conference on Speech and Computer, St. Petersburg, Russia, 25-29 June 2006.

[6] Steven G. Johnson and Matteo Frigo, " A modified split-radix FFT with fewer arithmetic operations ", IEEE Transactios, Signal Processing, vol. 55, pp. 111-119, Jan. 2007.

[7] Ray Andraka, " A survey of CORDIC algorithms for FPGA based computers ", in Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays, California, USA, 1998.

[8] V. Rodellar-Biarge, C. Gonzalez-Concejero, E. Martinez De Icaya, A. Alvarez-Marquina and P. Gómez-Vilda, " Hardware reusable design of feature extraction for distributed speech recognition ", Proceedings of the 6th conference on Applications of electrical engineering, Istanbul, Turkey, 2007.

[9] Jesús García; Juan A. Michell; Gustavo Ruiz; Angel M. Burón, " FPGA realization of a Split Radix FFT processor ", Proceedings of SPIE--The International Society for Optical Engineering, Vol 6590, May 2007.

[10] C. González-Concejero, V. Rodellar, A. Álvarez-Marquina, E. Martínez de Icaya and P. Gomez-Vilda, " A portable hardware design of a FFT algorithm ", Latin American Applied Research, vol. 37 no.1, March 2007.

[11] Altera, " CORDIC reference design ", Altera Application Note, AN: 263, www.altera.com/products/ip/dsp/ipm-index.jsp, June 2005.

[12] Altera, " FFT MegaCore function User Guide ", Altera MegaCore documentation and user guides, UG-FFT-7.0, www.altera.com/products/ip/dsp/ipm-index.jsp, November 2008.