Real-Time Signal Reconstruction from Short-Time Fourier Transform Magnitude Spectra Using FPGAs

Mouhcine Chami, Joseph Di Martino, Laurent Pierron, El Hassan Ibn Elhaj

Real-Time Signal Reconstruction from Short-Time Fourier Transform Magnitude Spectra using FPGAs

M. Chami*, J. Di Martino, L. Pierron**, E. Ibn Elhaj***
*INPT, Rabat, Morocco
**INRIA Nancy - Grand Est / Loria-UHP, Vandœuvre-lès-Nancy, France
E-mail: chami@inpt.ac.ma

***Abstract*— Real Time implementation of new frequential domain synthesizer algorithm based on the Nawab approach for signal reconstruction from short-time Fourier transform magnitude spectra is proposed in this paper. A register-transfer-level (RTL) synthesizer based on our algorithm was designed and simulated using VHDL as the hardware description language in respecting real time delay. The implemented RTL model was verified by comparing its performances with those obtained from a Python language implementation of the same synthesizer. We prove in particular that the proposed algorithm can be implemented in real time with a sampling frequency up to 60 kHz. Finally a real time implementation using an Altera DE2 development kit, generating good quality audio signals, was implemented.**

***Index Terms*— Short-Time Fourier Transform, Magnitude-only reconstruction, Python, RTL, VHDL, Real Time systems, Altera Kit Development.**

I. Introduction

THE short-time Fourier Transform (STFT) is a widely used time-frequency representation for signals such as speech and music [1]. The magnitude spectrum of discrete time signal $x(n)$ is typically obtained from the STFT, which is defined as

$$X(mL,\varpi) = \sum_{n=-\infty}^{\infty} x(n)w(n-mL)e^{-j\varpi n} \qquad (1)$$

where w is the analysis window, L is the analysis step size, and m is the index of the frames of the STFT. This parameter is selected so as to ensure a degree of time overlap between adjacent short-time sections. The magnitude spectrum for the short-time Fourier transform spectrum (STFTM) of $x(n)$ is deduced from (1):

$$|X(mL,\varpi)| = |\sum_{n=-\infty}^{\infty} x(n)w(n-mL)e^{-j\varpi n}| \qquad (2)$$

Generally, the STFTM is not reversible, since the time-domain signal cannot be uniquely determined from its STFTM only. This problem has been studied since 1980 specifically by Nawab et al. [2] and Griffin et al. [3]. In 2005 and 2007, Beauregard, Zhu et al. proposed iterative methods which gave good results measured by the signal-to-error ratio (SER) defined in (3) [4] [1] [5].

This study was supported in part by the INRIA Euro-Mediterranean 3+3 M06/07 Larynx and M09/02 Oesovox projects

$$\text{SER} = 10\log_{10}\frac{\sum_{m=-\infty}^{\infty}\int_{\varpi=-\pi}^{\pi}|X(mL,\varpi)|^2 d\varpi}{\sum_{m=-\infty}^{\infty}\int_{\varpi=-\pi}^{\pi}[|X(mL,\varpi)|-|X'(mL,\varpi)|]^2 d\varpi} \qquad (3)$$

where $X'(n)$ is the estimated or reconstructed signal.

Nawab, Griffin and Zhu-Beauregard methods are all iterative and therefore can not be easily implemented in real-time. The main objective of this paper is to present an efficient real time signal reconstruction algorithm from STFTM [6] presented by authors Di Martino and Pierron (D&P). The great advantage of our method is to use a non iterative process, for signal reconstruction, with equivalent or better SER results than those already published [1][4]. To the best of our knowledge this study is the first one presenting an effective real time implementation dedicated to signal reconstruction from STFTM. Based on these investigations, our synthesizer is designed and implemented using VHDL as design entry and simulation language [7]. The VHDL design is verified by comparing the VHDL simulation results with performance obtained from Python-language simulations [8].

The obtained result encourages us to implement this algorithm in hardware and specially in Field Programable Gate Array (FPGA) [9].

This paper is organized as follows. Section II presents Di Martino and Pierron spectral synthesizer. The implementation in Python language of this synthesizer with the associated SER results are exhibited in section III. Section IV is devoted to the VHDL development of all the operating blocks and modules of our synthesizer. And finally, the implementation in FPGA for real time processing is explained in the next sections.

II. D&P spectral synthesizer

The proposed spectral synthesizer is based on the Nawab algorithm [2]. The algorithm uses the scaled Hamming window (4) cited also in [1], where L is the analysis step size, N is the window length, $a = 0.54$ and $b = -0.46$.

TABLE I

PERFORMANCE OF OUR SYNTHESIZER

N	L	SER (dB)														
		SR = 8 KHz			SR = 11 KHz			SR = 16 KHz			SR = 22 KHz			SR = 44 KHz		
		Female	Male	Music	Female	Male	Music	Female	Male	Music	Female	Male	Music	Female	Male	Music
256	16	19.09	18.48	16.95	20.84	19.37	16.73	20.07	15.87	14.65	17.88	13.30	15.70	12.61	12.54	14.16
	32	18.92	18.43	17.71	20.67	19.21	17.44	19.98	16.16	15.33	18.20	13.92	16.32	13.68	13.58	15.23
	64	14.70	14.26	13.83	14.97	14.42	13.90	14.80	12.95	12.51	13.98	11.76	13.23	11.74	11.97	12.41
512	32	12.80	12.61	15.77	15.48	15.23	16.61	18.12	18.01	16.10	19.94	18.84	16.61	17.94	13.33	15.72
	64	12.80	12.58	16.16	15.33	15.15	17.05	17.94	17.84	16.70	19.88	18.75	17.26	18.04	13.94	16.25
	128	12.13	11.81	13.59	13.59	13.01	13.70	14.23	14.04	13.45	14.65	14.14	13.84	13.90	11.71	13.22
1024	64	8.95	8.80	13.26	10.30	10.05	14.44	12.65	12.64	16.16	15.29	15.11	16.53	19.93	18.85	16.56
	128	9.52	9.52	13.74	10.75	10.55	14.89	12.68	12.60	16.76	14.98	15.09	17.02	19.87	18.84	17.31
	256	9.89	10.05	12.45	10.92	10.72	13.06	11.83	11.80	13.68	13.37	12.95	13.54	14.65	14.31	13.77

$$H(n) = \begin{cases} \frac{2\sqrt{\frac{L}{N}}}{\sqrt{4a^2+2b^2}}(a + b\cos(\frac{\pi(2n+1)}{N})), & \text{if } 0 \leq n < N \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

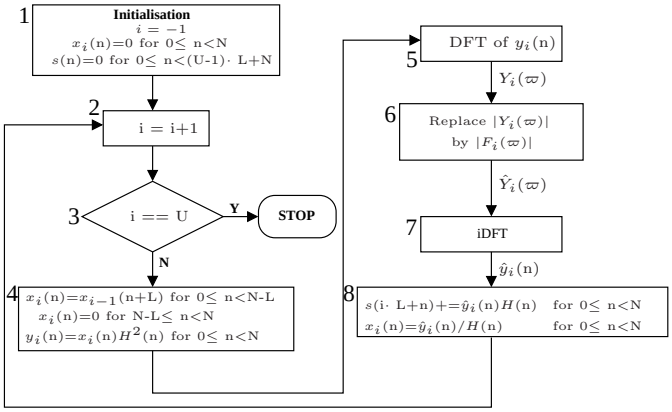

Fig. 1. Description scheme of the proposed spectral synthesizer algorithm

The proposed algorithm is implemented using the following steps :

1) At the initialization process i = -1, the working signal x_i(n) is set to zero for all N samples, the reconstructed signal s(n) is set to zero for n ∈ [0, (U-1)L+ N[. Where N is the DFT size, U is the number of frames and L is the analysis / synthesis step size.
2) i is incremented
3) if i equals U the computational process stops.
4) else the signal x_i(n) is the result of left shifting the previous signal x_{i-1}(n) by L steps. The L last elements of x_i(n) are set to zero. The extrapolated signal y_i(n) is the result of multiplying x_i(n) by H^2(n).
5) The DFT of y_i(n) is calculated which gives $Y_i(\varpi)$.
6) The modulus of $Y_i(\varpi)$ is replaced by the i^{th} known spectrum $|F_i(\varpi)|$.
7) The inverse Fourier transform of this new complex spectrum gives the new estimate of $\hat{y}_i$(n).
8) The synthesized signal s(n) is obtained by adding the values of $\hat{y}_i$(n) using an overlapping technique OLA (overlap-add) [10]. Finally, The new signal x_i(n) is obtained with dividing $\hat{y}_i$(n) by H(n) and the process iterates to step 2.

The essential difference between the Nawab extrapolation algorithm with the proposed approach lies mainly in two points: first the inner loop used for the extrapolation of the L samples has been removed and on the other hand, contrary of Nawab algorithm, which reconstructs the re-synthesized signal by concatenation of each L extrapolated samples, the proposed algorithm uses the classic OLA technique for reconstruction. Therefore our approach is not iterative and can be implemented in real time as we shall show in the following sections.

III. PYTHON IMPLEMENTATION FOR D&P SYNTHESIZER

A. Python language

Python associated with the Numerical package is a generic language, extremely portable and efficient enough for image or signal processing [8][11]. Python has the flexibility of Perl, associated with the numerical power and ease of use of MATLAB, but available as an open source environment. The source code is generally small when compared to compiled languages by several reasons: high-level data types and operations, no type declarations (dynamic typing), automatic memory management, and command blocks marked by indentation. In C/C++, equivalent data structures and functionalities with the same optimization would cost considerable more programming time. There is also a great native set of libraries implemented in C/C++ (built-in), for Python, that practically discard the process of compilation / correction / recompilation (except for API extensions of the language). These characteristics generate a high productivity gain [12].

B. D&P synthesizer reference program result

We evaluate the performance of our Python implementation of the proposed synthesizer by an extensive experience using a set of 100 audio test samples for each Sampling Rate including male speech (36 samples), female speech (44 samples) and music (20 samples). The mean duration of our samples is 2 s. The original Sampling Rate (SR) for all sounds is 16 000 Hz and a free audio converter generates different SR from 8 000 Hz to 44 000 Hz for a global audio set.

For each SR, different sets (N, L) are used by the reference program, where N is the DFT size and L is the new samples estimated at each step. The average SER computed by (3) is shown in Table I. The results show that for SR= 44 000 Hz the best sets are (1024,64) and (1024,128), for SR= 22 000 Hz and SR= 16 000 Hz the best sets are (512,32) and (512,64)

and for SR= 11 000 Hz and SR= 8 000 Hz the best sets are (256,32) and (256,16).

The main important result that can be deduced from this study is that whatever the sampling frequency it exists an optimal couple (N*,L*) giving acceptable SER results ($>$ 16 dB) for the proposed algorithm (underlined SER in Table I).

Note that using a smaller L means less extrapolated values per step and an increase in the amount of computation.

C. Performance and complexity comparison

All synthesizers algorithms for signal reconstruction from short-time Fourier transform magnitude spectra needs one DFT and one iDFT for each iteration so the computational complexity increases with iterations. D&P algorithm uses one iteration so that our method is well suited for real time implementation. The VLSI implementation of this new algorithm will be detailed in the next sections.

IV. Synthesizer Implementation

Here, we present the detailed VLSI design of D&P algorithm [6]. This VLSI implementation is performed using VHDL as design and simulation entry.

During the synthesizer design, there is a tradeoff between the implementation complexity (hardware area) and the processing delay. The complexity can be reduced by reusing the hardware components as much as possible. However, since, at any time, a hardware component can perform the function for one specific module only, different modules have to use it one after another, which results in longer synthesis delay. Providing a separate copy of the same hardware component to any module that needs it, can significantly reduce the delay since these modules can operate in parallel, at the price of larger complexity. In this paper, the synthesizer is designed to minimize the delay in order to respect real-time use and high sampling frequency.

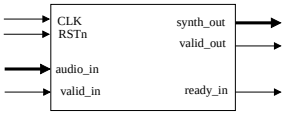

Fig. 2. Synthesizer Top level

The block diagram of the synthesizer is shown in Fig. 2 and Fig. 3. The major components include a Modulus module, a Cos_Sin module, a DFT/iDFT module, a memory block and the operational control logic circuits. It is assumed that the received L samples are used for calculating there Short-Time Fourier Transform Magnitude Spectra in the Modulus block. The modulus is multiplied by set of (cos, sin) of the Short-Time Fourier Transform argument spectra produced by the Cos_Sin block. The result of an IDFT (inverse DFT) is used for calculating firstly the synthesized speech and secondly the update of the working signal x. Fig. 4 shows the synthesizer progress in terms of blocks activations and in time. The Cos-Sin module and iDFT synthesizer top level performed in different time slots. So, they can use only one DFT/iDFT submodule for surface optimizing.

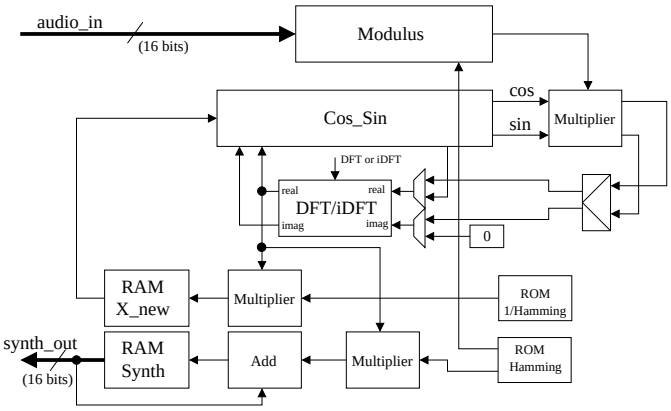

Fig. 3. Synthesizer block diagram

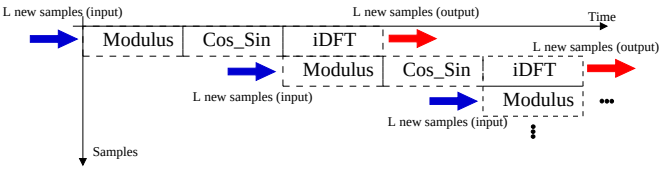

Fig. 4. Synthesizer progress in functions and in time

A. DFT / iDFT architecture

Discrete Time Fourier transform provides frequency domain representation for a signal. FFT is an important algorithm to calculate Discrete Fourier Transform (DFT). The discrete Fourier transform of N complex samples $x(k), k = 0, 1, \ldots, N-1$ is defined as [13]

$$F(r) = \sum_{k=0}^{N-1} x(k) W^{rk}, r = 0, 1, \ldots, N-1 \qquad (5)$$

where $W = exp(-2\pi j/N)$

The FFT is factored into Radix-4 Butterfly operations. When an odd power of two is required, a small radix-2 "follower" stage performs the final iteration. The radix-2 stage does not require a full complex rotator so its cost is minimal.

1) Radix-4 FFT: From the implementation point of view, DFT computation is highly inefficient; therefore, a divide-and-conquer algorithm has been proposed to improve computation efficiency. Among the approaches followed for the FFT computation, radix-4 time-decimation has been widely used for a number of practical applications. This approach is adopted in the present work. The algorithm consists in the decomposition of the computation in 4x4 multiplicative and additive processing, named 4x4 butterflies or dragonflies that can be expressed as equation (6) and shown at Fig. 5, where $x^*(k)$ is the new value calculated from $x(k)$.

$$\begin{bmatrix} x^*(k) \\ x^*(k+4^p) \\ x^*(k+2\cdot 4^p) \\ x^*(k+3\cdot 4^p) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} W_N^{0q} x(k) \\ W_N^{1q} x(k+4^p) \\ W_N^{2q} x(k+2\cdot 4^p) \\ W_N^{3q} x(k+3\cdot 4^p) \end{bmatrix} \quad (6)$$

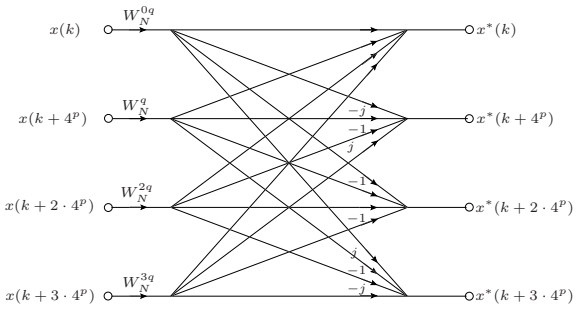

Fig. 5. Basic butterfly computation in a radix-4 FFT algorithm

In order to re-use the dragonfly computational block, it must be established the addressing index (q and p) for each block as stated in (7-8).

$$q = [R/4^{4-C}]4^{4-C} \quad (7)$$

$$p = REM(R, 4^{4-C})4^{C+1} + [R/4^{4-C}] \quad (8)$$

where C is the column index (C=0,1,... 3), R is the R-dragonfly (R=0,1,2, ... 127) at stage C and the REM function returns a numeric value that is the remainder of first argument divided by second argument.

2) Radix-2 FFT: The algorithm consists in the decomposition of the computation in 2 multiplicative and additive processing, named 2x2 butterflies that can be expressed as (9) and shown at Fig. 6. Where k varies from 0 to N/2. Radix-4 can be used like Radix-2.

$$\begin{bmatrix} x^*(k) \\ x^*(k+\frac{N}{2}) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x(k) \\ W_N^k x(k+\frac{N}{2}) \end{bmatrix} \quad (9)$$

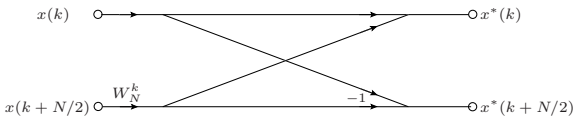

Fig. 6. Basic butterfly computation in a radix-2 FFT algorithm

3) 512-FFT architecture and implementation: The engine proposed contains one memory double-bank RAM (Real part and imaginary part). The input data are stored using bit reversal method. The coefficients are stored in an internal ROM. The main block for the FFT computation uses one radix-4 block. For the 512 FFT size the radix4 are used like radix-2 in last iteration (This method reduces the architecture by 750 slices). The overall algorithm computation is supervised by a sequencer with a finite-state machine (FSM) as shown in Fig. 7. The iFFT uses the same FFT engine with sign inversion of the imaginary input and imaginary output.

The FFT core is described under VHDL and simulated in Modelsim in order to verify its functionality. Results were

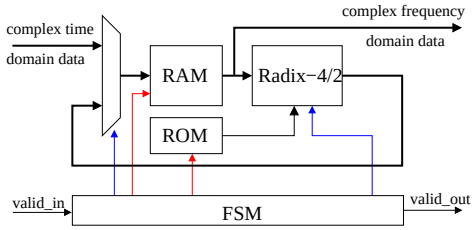

Fig. 7. FFT / iFFT engine

compared with Python simulations under the same conditions to validate the algorithm performance.

Time-computation performance of the FFT core can be estimated by the clock cycle number, required to compute a full 512-point input signal. For a 50 MHz master clock, the 512-point FFT computation time is 491,6 μs. The engine block represents a Linear calculation Error (LE) less than 0.0625, where the Linear calculation Error is given by (10).

$$LE = MAX_i(|Result\_python(i) - Result\_VHDL(i)|) \quad (10)$$

B. Cordic engine architecture

CORDIC (Coordinate Rotation Digital Computer) is an iterative algorithm used for calculating magnitude and phase of complex numbers [14][15]. This algorithm is considered as more suitable for hardware implementation as it does not require complex multiplications. CORDIC algorithm rotates the complex number successively such that the imaginary part approaches to zero. At this point the real part gives the approximate magnitude. This is achieved by multiplying the complex number by a succession of constant values. The multiplying factors are in the powers of 2, so the multiplications can be implemented as shifts and adds during the iterations of the algorithm. The bloc diagram of the Cordic engine is shown in Fig. 8.

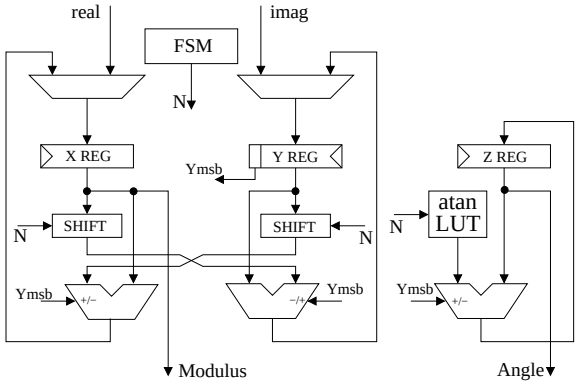

Fig. 8. Basic processor for Cordic iterations

C. Modulus

For each frame with N samples, the operation performed in this module is presented in (11).

$$\begin{array}{ll} x_{ham} = & H * x \\ (X,Y) = & DFT(x_{ham},0) \\ Modulus = & MEC(X,Y) \end{array} \quad (11)$$

where H presents the scaled hamming window presented in (4), x is the N samples for the treated frame, MEC is the Magnitude Estimation using Cordic engine and finally $Modulus$ is the result for this module.
Fig. 9 describes the Modulus architecture. The new L audio samples are shifted in last addresses of RAM input. The RAM input is multiplied by Hamming window and introduced in the FFT engine. The pipelined Cordic ([15]) amplitude block (downloaded from opencores.org) are used for calculate the modulus of spectrum produced by the FFT engine. The system is supervised by a sequencer with a finite-state machine (FSM) as shown in Fig. 9.

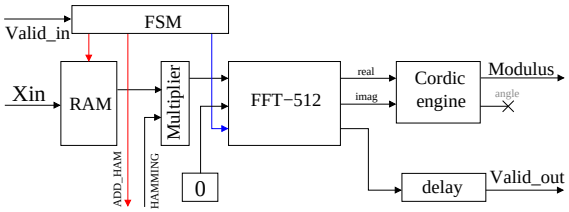

Fig. 9. Modulus FFT Block

The same process is used to validate the algorithm performance. The modulus block represents a Linear calculation Error (LE) less than 0.5723.

D. Cos_Sin module

Fig. 10 describes the Cos_Sin block architecture. The X_new $N - L$ samples are shifted in last addresses of X_old RAM. The data of this RAM is multiplied by the square of Hamming window and introduced and sent to FFT engine. The pipelined Cordic ([15]) algorithm permits to generate the polar angle of the complex domaine data produced by the FFT engine. Using this angle, the cos/sin table (ROM cos/sin) gives the corresponding Cosinus and Sinus. The system is supervised by a sequencer with a finite-state machine (FSM) as shown in Fig. 10.

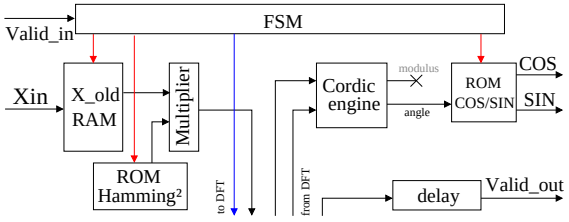

Fig. 10. Block diagram of Cos/sin FFT engine

TABLE II
PERFORMANCE OF D&P SYNTHESIZER COMPARING WITH PYTHON MODEL

Sp.	SER (dB) with python	SER (dB) with VHDL
1	18.6463	18.4644
2	16.9182	15.9239

TABLE III
PERFORMANCE OF D&P SYNTHESIZER IMPLEMENTATION

Selected Device	Altera Cyclone II (2C35)	%
Number of Logic Elements	23,031	69%
Total registers	7,634	23%
Total memory bits	364,544	75%
Embedded Multiplier 9 bits	70	100%
Maximum internal frequency	59 MHz	–

E. synthesis of D&P audio synthesizer

The architecture of our synthesizer presented in Fig. 2 is implemented and validated by simulating it with Modelsim. Time-computation performance of the new synthesizer can be estimated by the clock cycle number, required to compute a full 64 samples of audio signal. For a 50 MHz master clock, the 64-point synthesizer computation time is 1064.6 μs. So the synthesizer permits the reconstruction of at least 60116 new samples in one second. Consequently, the ADC/DAC can work with sampling rate lower than 60.116 kHz. The delay of the first sample output is 1562 μs.

Two speech tests are introduced like stimuli to the design and a specific Python program is used for calculating the SER of the reconstructed signal. The results described in Table II show that the Python simulation gives the best results due to linear error produced by the VHDL treatment.

The analysis and synthesis results using Quartus II are presented in Tab. III. The implementation presents high efficiency in surface.

V. REAL-TIME IMPLEMENTATION IN ALTERA FPGA

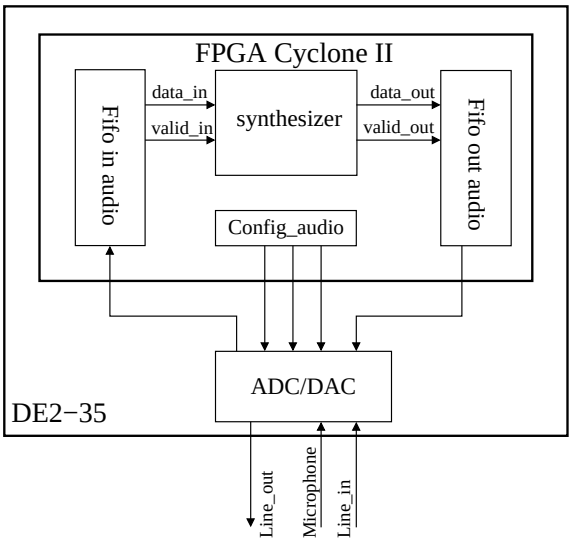

Fig. 11. Block diagram of real time implementation in DE2-35

The final implementation of our synthesizer is done using the development and education board "ALTERA DE2-35". The card contains 24-bit CD-quality audio CODEC with line-in, line-out, and microphone-in jacks accepting sampling frequency with 8 to 96 KHz. In this application, 32 Khz Sample Rate are used. The overall block diagram of the complete system is shown in Fig. 11. To verify the performance of the new synthesizer design on hardware, the VHDL code synthesized (sof file) is downloaded into the Target FPGA device (Cyclone II 2C35) and the complete system is reset. The experimental set up for this system consists in introducing audio from the microphone or audio player. The audio outputs are listened through headphones. When listening, as predicted by the simulation tests, the audio re-synthesized signals are of good quality, without interruption or artifact.

VI. Conclusion

In this study, we propose a new real-time spectral synthesizer based on the iterative sequential Nawab approach. A framework for an FPGA-based signal reconstruction has been presented. The simulation results verify the correctness of the design with small differences in SER due to linear error calculation. The results also show that the proposed algorithm can be implemented in real time with a sampling frequency up to 60 kHz.

References

[1] X. Zhu, G. T. Beauregard, and L. Wyse, "Real-time signal estimation from modified short-time fourier transform magnitude spectra," *IEEE Transactions on audio, speech and language processing*, vol. 15, no. 5, pp. 1645–1653, July 2007.

[2] S. H. Nawab, T. F. Quatieri, and J. S. Lim, "Signal reconsctruction from short-time fourier transform magnitude," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-31, no. 4, pp. 986–998, August 1983.

[3] D. W. Griffin and J. S. Lim, "Signal estimation from short-time fourier transform," *IEEE Transactions on acoustics, speech and signal processing*, vol. ASSP-32, no. 2, pp. 236–243, April 1984.

[4] G. T. Beauregard, X. Zhu, and L. Wyse, "An efficient algorithm for real time spectrogram inversion," in *In Proc. 8th Int. Conf. Digital Audio Effects (DAFX-05*, Septembre 2005, pp. 116–221.

[5] D. W. Griffin and J. S. Lim, "Speech synthesis from short-time fourier transform magnitude and its application to speech processing," vol. 9, Mars 1984, pp. 61–64.

[6] J. Di Martino and L. Pierron, "Synthétiseur numérique audio amélioré," patent no. 10/02674, INRIA, Université Henri Poincaré Nancy 1, Tech. Rep., 2010.

[7] A. Rushton, *VHDL for Logic Synthesis*. John Willy & Sons, 1998.

[8] M. C. Brown, *Python*. McGraw-Hill, 2001.

[9] S. Kilts, *Advanced FPGA Design (Architecture, Implementation, and Optimization)*. John Willy & Sons, 2007.

[10] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*. Englewood Cliffs, N.J., Prentice-Hall, Inc., 1975.

[11] M. Lutz and D. Ascher, *Learning Python*. O'Reilly & Associates, 1998.

[12] A. Silva, R. Lotufo, R. Machado, and A. Saude, "Toolbox of image processing using the python language," in *Image Processing, International Conference on*, vol. 3, no. 2, 2003, pp. 1049–1052.

[13] W. Cochran and al., "What is the fast fourier transform?" *Proceedings of the IEEE*, vol. 55, no. 10, pp. 1664 – 1674, october 1967.

[14] J. Volder, "The cordic trigonometric computing technique," *IRE Transaction on Electronic Computers*, vol. EC-8, no. 3, pp. 330–334, sept 1959.

[15] Y. Hu, "Cordic-based vlsi architectures for digital signal processing," *Signal Processing Magazine, IEEE*, vol. 9, no. 3, pp. 16 –35, jul 1992.