Implementation of a Blind Source Separation Algorithm in a Heterogeneous Computing Architecture

Oswaldo F. Filho[1] and Ricardo Suyama
Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas, Universidade Federal do ABC, Santo André, SP, Brazil

Abstract

The statistical method Independent Component Analysis (ICA) has been successfully used to solve the problem of Blind Source Separation (BSS). However, in applications that require real-time operation, it is of great interest as such techniques are implemented in embedded systems, for which it is necessary to obtain a balance between performance and the cost required for its implementation. Therefore, this paper proposes a heterogeneous computing architecture basaed on a Field-Programmable Gate Array (FPGA) Cyclone IV, present in the DE0-Nano Kit. The architecture is composed of a subsystem that implements the algorithm FastICA, implemented using the DSP Builder library, and the Nios II microprocessor. In order to reduce the computational complexity, a fixed point representation was used, which allowed a reduction in the number of Embedded Multipliers and Logic Elements used, thus maximizing the operating frequency. The algorithm implemented in the proposed architecture achieved a performance comparable to other proposals in the literature, but with a reduced implementation cost and the flexibility introduced by the Nios II microprocessor.

Keywords: BSS, ICA, FastICA, FPGA, Heterogeneous Computing.

1. Introduction

Blind Source Separation (BSS) methods have been successfully applied in different areas, such as Telecommunications [1], Biomedical Signal Processing [2] and Audio Processing [3], and several methods were proposed and studied in the literature [4]. Nevertheless, when one considers applications with a real-time processing requirement or with a vast amount of information to be processed (e.g., big data applications), not only the method itself but also how they are implemented becomes extremely relevant. In this

[1]E-mail Corresponding Author: oswaldo.fratini@ufabc.edu.br

context, some works in the literature have been focused specificaly on the implementation of well-known signal processing algorithms in special embedded hardware, such as GPUs and FPGAs, to boost the processing power and meet application requirements.

FPGA-based implementations of BSS algorithms, such as the FastICA method [5], have been proposed in the literature [6] [7] [8] [9] [10], and emphasize the processing gain obtained with an architecture specially designed for the algorithm.

In this paper proposes a new implementation of the FastICA algorithm exploring an heterogeneous computing architecture, which confers a high degree of flexibility to the system, i.e., it would be possible to easily expand the system to deal with a larger amount of signals. In addition to that, the optimization in the design was able to minimize the use of embedded multipliers, rather scarce in the EP4CE22F17C6 (a low-cost Altera Cyclone IV product line), Logic Elements (LE) and memory blocks.

The paper has been structured as follows. In Section 2, a review of the FastICA algorithm is presented. In Section 3, the FPGA implementation of the FastICA core is discussed in details, followed by some simulations results in Section 4. Finally, in Section 5, we present the concluding remarks.

2. FastICA Algorithm for BSS

The BSS problem can be stated as follows: Let a set of M sensors capture a set of N signals or interest (the sources), with $M \geq N$. Each capted signal x represents a mixture with different sources s, which can be mathematically given by:

$$\begin{aligned} x_1(n) &= a_{11}s_1(n) + a_{12}s_2(n) + \ldots + a_{1N}s_N(n) \\ x_2(n) &= a_{21}s_1(n) + a_{22}s_2(n) + \ldots + a_{2N}s_N(n) \\ &\vdots \\ x_M(n) &= a_{M1}s_1(n) + a_{M2}s_2(n) + \ldots + a_{MN}s_N(n) \end{aligned} \tag{1}$$

where a_{ij} represents the mixing coefficients. This equation can be presented in matrix form as:

$$\mathbf{x} = \mathbf{A}\mathbf{s} \tag{2}$$

The goal of BSS algorithms is to estimate the source signals $\mathbf{s}$ solely based on the observed signals $\mathbf{x}$, which, in this case, can be achieved by adjusting a matrix $\mathbf{W}$ that should have a structure similar to that of $\mathbf{A^{-1}}$.

Then, the estimated signals $\mathbf{y}$ would be given by:

$$\mathbf{y} = \mathbf{W}\mathbf{x} \tag{3}$$

The Independent Component Analysis (ICA) [4] it is one of the main methods used to solve the BBS problem. This method requires that the source signals have two statistical characteristics: be statistically independents and don't have a gaussian distribution. This method estimates $\mathbf{W}$ such that the estimated signals $\mathbf{y}$ be as mutually independent as possible. Alternatively, one can also employ a different, estimating the sources one-by-one from the mixtures: in this case, as implemented in the FastICA algorithm used in the present paper, one can correctly extract the sources trying to estimate a source that be as nongaussian as possible [11].

In order to use the FastICA algorithm, which implements the ICA concept, it is necessary to preprocess the mixtures $\mathbf{x}$ to ensure that signals are zero mean and uncorrelated, performing the following steps:

1. Centering:
$$\mathbf{m} = \mathbf{x} - E\{\mathbf{x}\} \tag{4}$$
2. Whitering:
$$\hat{\mathbf{x}} = \mathbf{E}\mathbf{D}^{-1/2}\mathbf{E}^T\mathbf{x} \tag{5}$$

where $\mathbf{E}$ and $\mathbf{D}$ correspond to the matrices with the eigenvectors and eigenvalues of the autocorrelation matrix of the observed signals.

In FastICA, the statistical independence between the estimated sources $\mathbf{y}$, is achieved by maximizing the non-gaussianity of the estimates. To this end, the Negentropy [11] $J_{Negentropy}(y)$ of the estimated signals is approximated by a nonlinear function $G(\cdot)$, such that the algorithm can be summarized as follows:

1. Randomly initialize $\mathbf{w}$
2. Estimates new weights of $\mathbf{w}^+ = E\{\mathbf{x}g(\mathbf{w}^T\mathbf{x})\} - E\{g'(\mathbf{w}^T\mathbf{x})\}\mathbf{w}$
3. Normalize weight vector $\mathbf{w} = \mathbf{w}^+/||\mathbf{w}^+||$
4. If not converged, back to item 2

where the nonlinear function $G(u) = \frac{1}{4}u^4$, and $g(u) = u^3$ and $g'(u) = 3u^2$ denote its first and second order derivatives.

Since in this version of the FastICA algorithm we have an extracting vector $\mathbf{w}_i$ for each source to be estimated, it is necessary to guarantee that these

vectors do not converge to the same solution, i.e., extract the same source, which can be done by an orthogonalization procedure of these vectors. In the present paper, we employed the Gram-Schmidt orthogonalization method.

3. FPGA Implementation

In the implementation of the FastICA algorithm, we chose a fixed point representation to minimize the required number of FPGA resources, thus increasing operation frequency and pipelines throughput.

In order to determine the best fixed point representation and number of samples to be processed in each iteration of the FastICA algorithm, simulations were performed using MATLAB with Fixed-Point Designer and Signed Fixed-Point (sfi) numeric object to processing the mixtures $\mathbf{x}$ created using sources $\mathbf{s}$ with samples assuming values between -1 and +1 with uniform distribution and a mixture matrix $\mathbf{A}$ randomly generated, for different number of bits for the signed integer part and fractional part of the representation. After this first analysis, it was decided that a word with 24 bits - with 16 bits for the fractional part and 128 samples by unit - should be used.

For the circuit design, in this work we used the Simulink with DSP Builder Blockset Library from Altera to generate the model and simulations. The HDL is generated by the integration with Quartus II software, also from Altera.

In our design, special attention was given to the amount of Logic Elements (LE), Embedded Multipliers (DSP 9x9) and Memory Blocks used to implement the algorithm, since a reduction in their use would lead to more processing pipelines, thus increasing the processing power of the system.

The FastICA algorithm was implemented in a subsystem, which is shown in figure 1.

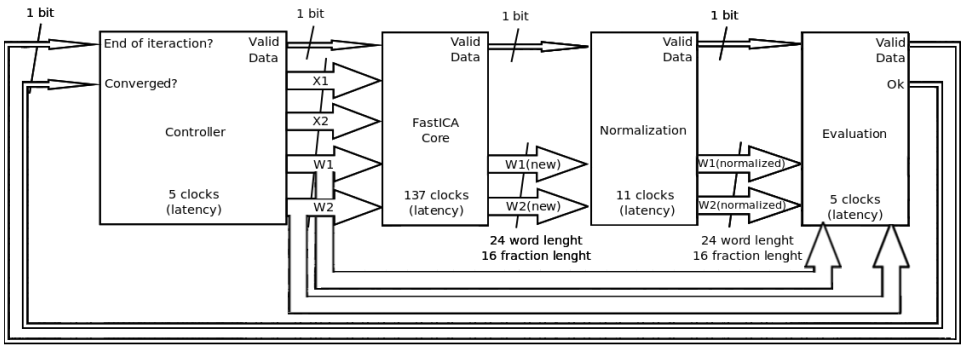

Figure 1: FastICA subsystem blocks diagram.

The FastICA subsystem was divided in 4 blocks: Controller, FastICA Core, Normalization and Evaluation. In total, each iteration of the subsystem corresponds to 137 clocks of latency in processing.

Another important block is Controller, that is responsible for the control of the FastICA subsystem and allows the integration with Nios II processor using Qsys system integration tool.

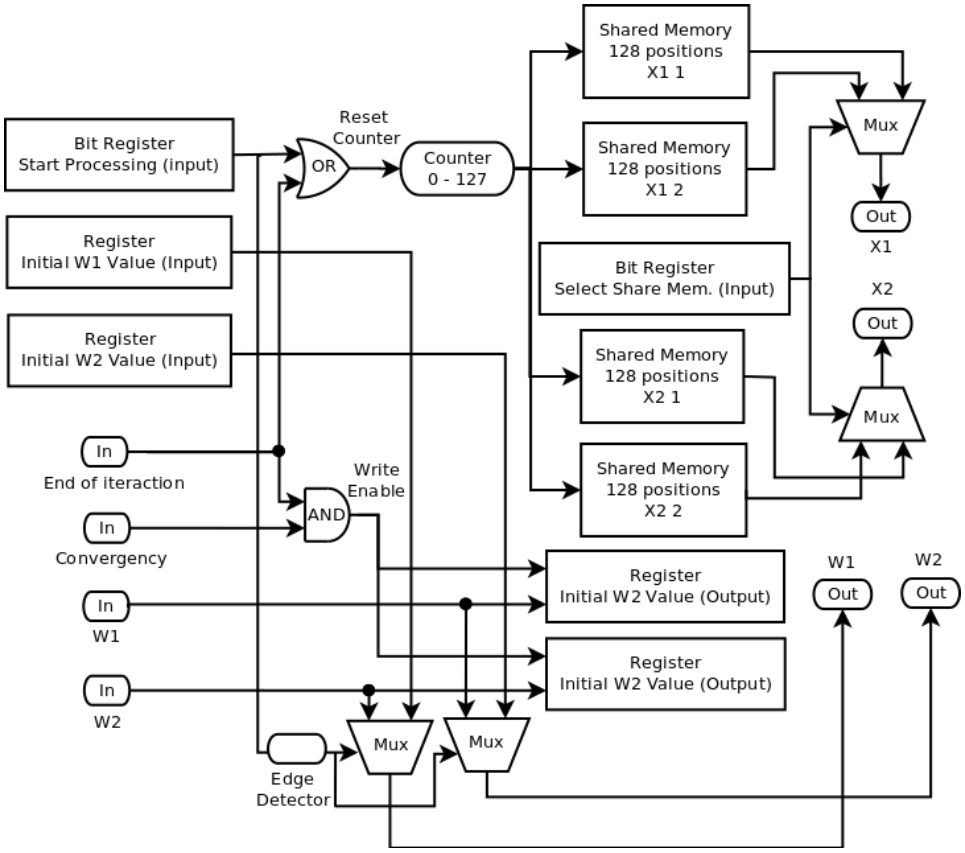

Figure 2: FastICA subsystem controller block and interfaces.

If it is necessary to change the interface to a specific BSS application, one can simply change the Nios II program language, which has access to all registers and shared memories of the FastICA blocks. Table 1 shows the available FPGA resources and how many resources were used to implement the FastICA Subsystem and the NIOS II processor.

The performance of the FastICA subsystem obtained with this FPGA

implementation is presented in Table 2, in which it is possible to observe that the total throughput is larger than 7MHz.

Table 1: Resources

Description	LE	Memory Kbits	DSP 9x9
EP4CE22F17C6	22,320	594	132
FastICA Subsystem	3,121	66.597	105
Nios II	11,215	143.929	7

Table 2: FastICA subsystem performance

Description	Value
Maximum operation frequency	58.59 MHz
Processing time per iteration	158 clocks
Unit average processing time	1058 clocks
Multipliers latency	25 clocks
Throughput	7.088 MHz samples per seconds

In a related work [10], the authors used a floating point numerical representation, a Altera FPGA Stratix II with 2 times more resources than Altera Cyclone IV and do not provide information about resources usage. Nevertheless, comparing the FastICA core processing time of the both works, it is possible to observe that our implementation is about 6 times faster.

4. Simulations

In order to assess the performance of the architecture, simulations were carried out considering a scenario in which two sources are mixed according to the linear model presented before. The results for 10,000 runs are shown in table 3. The mean was calculated without considering outliers - mainly due to convergence issues of FastICA algorithm.

It can be noted that the algorithm attains a low MSE and a low Amari index value, indicating that the sources are correctly estimated in most of the trials, despite the fact that the algorithm is implemented in fixed-point representation. A typical result is shown in figure 3, where the true sources, the observed signals and the recovered signals are shown - in this case, it is clear that the system was able to invert the action of the mixing system.

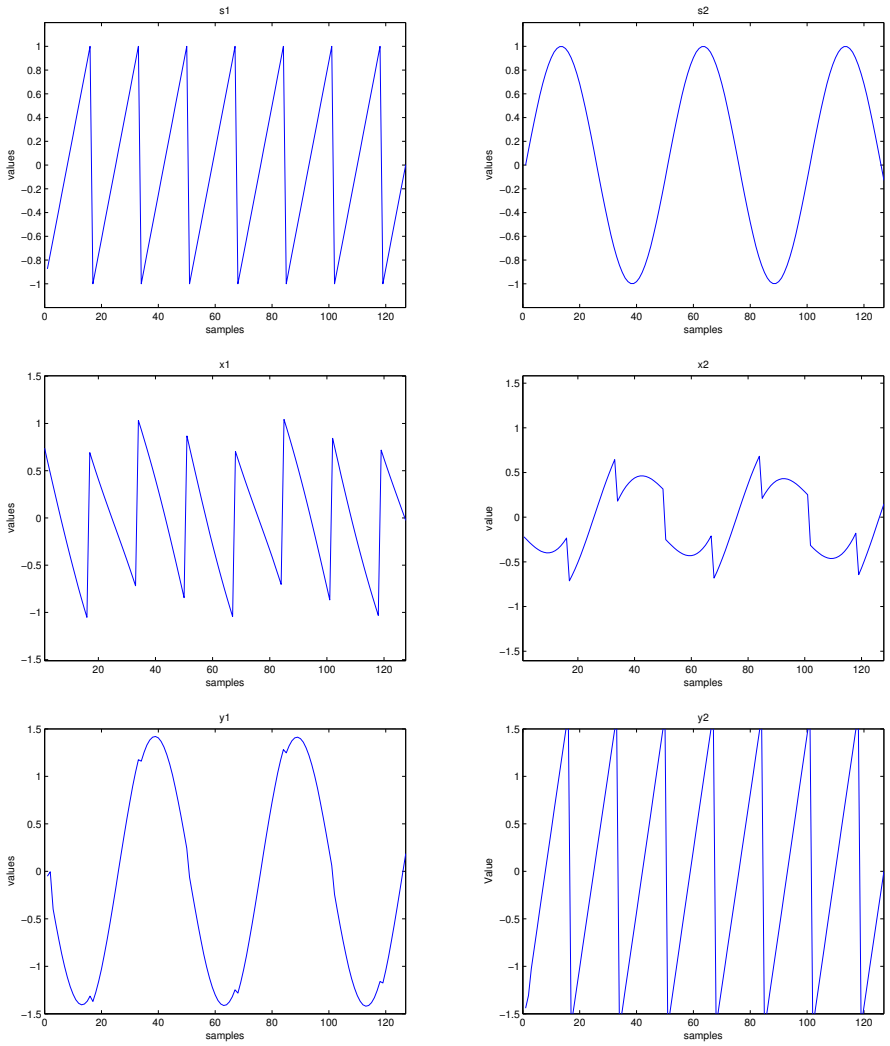

Figure 3: Original sources, mixtures and estimated sources of a simulation.

Table 3: FastICA algorithm performance with fixed point

Description	Minimum	Maximum	Mean
Number of iterations	3	12	6.693
MSE Est. Source 1	0.683 x 10^{-9}	0.449	0.082
MSE Est. Source 2	0.430 x 10^{-9}	0.454	0.083
Amari index	0.003	0.808	0.273

5. Conclusions

The premise of this work was to obtain an optimized implementation of the FastICA algorithm in embedded harware, using a low-cost FPGA development kit. The proposed architecture was able to obtain a high throughput, making it suitable for applications with real-time requirements. Perspectives for future work include extending the architecture to deal with more sources, and implementing other BSS algorithms in FPGA and SoCs.

Acknowledgments. The authors acknowledge CAPES and CNPq.

References

[1] Y. Du, Y. Fang, N. Wu, and K. Yen, "Performance analysis for complex fastica algorithms in MIMO OFDM systems," in *Future Computer and Communication (ICFCC), 2010 2nd International Conference on*, vol. 2, pp. V2–782–V2–786, May 2010.

[2] S. Cerutti and C. Marchesi, *Blind Source Separation: Application to Biomedical Signals*, pp. 379–409. Wiley-IEEE Press, 2011.

[3] J. S. Calderón-Piedras, Á. D. Orjuela-Cañón, and D. A. Sanabria-Quiroga, "Blind source separation from single channel audio recording using ICA algorithms," in *2014 XIX Symposium on Image, Signal Processing and Artificial Vision*, pp. 1–5, Sept. 2014.

[4] P. Comon, "Independent component analysis, a new concept?," *Signal Process.*, vol. 36, pp. 287–314, Apr. 1994.

[5] A. Hyvrinen and E. Oja, "A fast fixed-point algorithm for independent component analysis," *Neural Computation*, vol. 9, pp. 1483–1492, 1997.

[6] H. Du and H. Qi, "An FPGA implementation of parallel ICA for dimensionality reduction in hyperspectral images," in *Geoscience and Remote Sensing Symposium, 2004. IGARSS '04. Proceedings. 2004 IEEE International*, vol. 5, pp. 3257–3260, 2004.

[7] C. Charoensak and F. Sattar, "A single-chip FPGA design for real-time ICA-based blind source separation algorithm," in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pp. 5822–5825, 2005.

[8] H. Du, H. Qi, and X. Wang, "A parallel independent component analysis algorithm," in *Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on*, vol. 1, pp. 8 pp.–, 2006.

[9] K.-K. Shyu and M.-H. Li, "FPGA implementation of fastica based on floating-point arithmetic design for real-time blind source separation," in *Neural Networks, 2006. IJCNN '06. International Joint Conference on*, pp. 2785–2792, 2006.

[10] K. K. Shyu, M. H. Lee, Y. T. Wu, and P. L. Lee, "Implementation of pipelined fastica on FPGA for real-time blind source separation," *IEEE Transactions on Neural Networks*, vol. 19, pp. 958–970, June 2008.

[11] A. Hyvärinen, J. Karhunen, and E. Oja, *Independent Component Analysis*. Wiley, 2001.