Separação Cega de Fontes em Tempo Real Utilizando FPGA

Oswaldo Fratini Filho e Ricardo Suyama
Universidade Federal do ABC, Santo André, Brasil
e-mail: oswaldo.fratini@ufabc.edu.br

Abstract - Independent Component Analysis (ICA) is a versatile statistical tool which has been applied to a variety of applications, and was the first method to effectively solve the Blind Source Separation (BSS) problem. Even though several implementations of ICA algorithms have been proposed in the literature, in applications with stringent requirement, e.g., real-time processing, it is necessary to obtain optimized implementations with a high throughput. In this work, a study about the implementation of a well known ICA algorithm - the FastICA - in FPGA is carried out. The architecture developed using the DSP Builder® library was able obtain a high-processing thoughput (6.89 MSPS) in a low-cost hardware.

Palavras-chave: *BSS, ICA, FastICA e FPGA.*

Introdução

O problema de Separação Cega de Fontes (do inglês, *Blind Source Separation* - BSS) tem sido um tema de pesquisa de grande interesse entre pesquisadores na área de processamento de sinais, em grande parte devido à sua formulação bastante abrangente, o que permite que as ferramentas desenvolvidas para solucionar o problema sejam facilmente aplicáveis em diferentes contextos. O objetivo final no problema de BSS consistem em obter estimativas precisas de sinais (as fontes) a partir de observações que correspondem a misturas desconhecidas destes sinais, conforme ilustrado na figura 1.

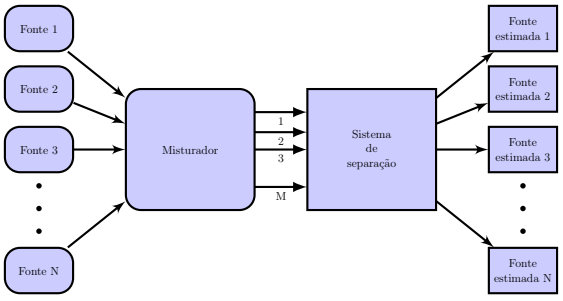


Figura 1 – Estrutura geral do problema de BSS.

No caso mais simples, no qual o processo de mistura é modelado por um sistema linear sem memória, as amostras observadas $x_i(n)$ podem ser modeladas da seguinte maneira:

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

onde $s_i(n)$ denota a n-ésima amostra da fonte i, e a_{ij} denotam os coeficientes da mistura. A mesma equação 1 pode ser representada de forma matricial

$$\mathbf{x}(n) = \mathbf{A}\mathbf{s}(n) \tag{2}$$

Considerando tal modelo de mistura, a estimativa das fontes pode ser realizada por meio de estrutura de separação linear, de maneira que o vetor de sinais estimados é dado por $\mathbf{y}(n) = \mathbf{W}\mathbf{x}(n)$. Evidentemente, se a matriz de mistura é conhecida e inversível, o sistema de separação ideal é dado por $\mathbf{W} = \mathbf{A}^{-1}$. Entretanto, como a matriz $\mathbf{A}$ é desconhecida, é preciso explorar alguma característica estatística da fonte para que seja possível obter a matriz de separação e assim recuperar os sinais originais.

Alguns métodos podem ser utilizados para resolver o problema de BSS, um dos mais conhecidos é o método ICA, que explora a independência estatística entre eles. Intuitivamente, a ideia explorada pelos algoritmos de separação que implementam a ICA é simples: considerando que os sinais das fontes são independentes entre si, busca-se estimar sinais que também sejam independentes entre si. Garante-se que, no caso em que não há ruído e há no máximo uma fonte com distribuição gaussiana, obter a matriz $\mathbf{W}$ buscando estimativas que sejam independentes entre si leva à recuperação das fontes [1].

Um dos mais conhecidos algoritmos que implementam a ICA denomina-se FastICA [2], e seu fluxograma é apresentado na figura 2. Grande parte do sucesso alcançado pelo algoritmo se deve à sua estrutura simples e rápida convergência, obtendo bons resultados em diferentes aplicações práticas.

Em algumas aplicações de interesse, entretanto, há a necessidade de se obter implementações de algoritmos de separação cega de fontes em sistemas embarcados, seja pelas restrições de operação

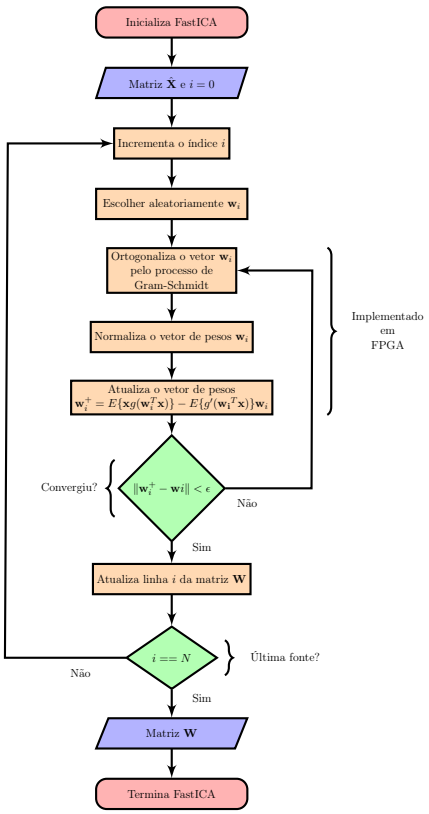


Figura 2 – Fluxograma do algoritmo FastICA

em tempo real do sistema ou mesmo pela necessidade de processamento de um volume massivo de dados, o que também exige que o processamento seja realizado o mais rapidamente possível. Nesse sentido, diferentes trabalhos na literatura exploraram a possibilidade de implementação dos algoritmos em FPGAs [3] [4] [5] [6] [7]. No presente trabalho são apresentadas contribuições a tais estudos, apresentando a implementação e otimização de recursos disponíveis nas FPGAs, inspirados em trabalhos como [8] e [7], visando a implementação dos algoritmos em hardware de baixo custo e que sejam escaláveis para um maior número de fontes.

Grande parte da otimização se refere à minimização de uso dos *variable-precision DSP blocks*, que são os principais recursos do FPGA disponíveis para o processamento digital de sinais, que implementam os multiplicadores acumuladores, assim como o uso de outros recursos como: *Logic Elements* (LE), *Adaptive Logic Module* (ALM) e registradores. A análise do algoritmo FastICA leva em consideração o tempo de processamento de cada iteração, observando-se as restrições de tempo real, a exatidão e acurácia quando utilizada a representação numérica de ponto flutuante e de ponto fixo.

Materiais e métodos

Para análise inicial dos algoritmos, assim com análise dos resultados das simulações dos modelos implementados, foi utilizado o ambiente de simulação MATLAB® . Adicionalmente, foi utilizada a ferramenta de modelagem e simulação Simulink® em conjunto com a biblioteca de blocos lógicos DSP Builder da fabricante de FPGAs Altera® para a implementação dos modelos, seguindo o fluxograma apresentado na Figura 2.

As entradas para simulação de cada algoritmo, ou seja, as misturas das fontes, foram geradas por fontes de sinal geradas com valores entre -1 e 1 seguindo uma distribuição uniforme. As misturas foram pré-processadas sendo submetidas ao processo de centralização e branqueamento.

A fim de avaliar o impacto da redução no número de bits para a representação numérica, foram realizadas simulações de referência considerando representação numérica de ponto flutuante com precisão dupla (64 bits) no MATLAB® . Posteriormente, foram avaliadas diferentes configurações de representação numérica e o impacto na utilização de recursos da FPGA.

Os resultados apresentados correspondem à média de 3000 simulações, nas quais tanto as fontes como a matriz de mistura eram geradas aleatoriamente a cada execução. O desempenho dos algoritmos é medido em termos do erro quadrático médio obtido na estimação das fontes, definido por:

$$EQM = \sum_i E\left\{(\hat{s}_i - s_i)^2\right\} \tag{3}$$

onde $\hat{s}_i$ representa o valor estimado da fonte i obtido pelos métodos de separação.

Resultados

Os resultados das simulações com o FastICA considerando uma representação em ponto flutuante são apresentadas na Figura 3. Conforme esperado, o MSE médio aumenta na medida em que o número de fontes presentes na mistura aumenta, o que dificulta a estimação da matriz de separação. Além disso, observa-se que quanto maior o número de amostras considerado nos cálculos, melhor o desempenho do algoritmo, visto que o algoritmo utiliza médias amostrais para estimar a média estatística dos dados.

Cabe mencionar, no entanto, que aumentar o número de amostras da implementação pode ter um impacto indesejável de aumento de recursos

Figura 3 – Exatidão do algoritmo FastICA em ponto flutuante (64 bits) com diferentes número de amostras e fontes

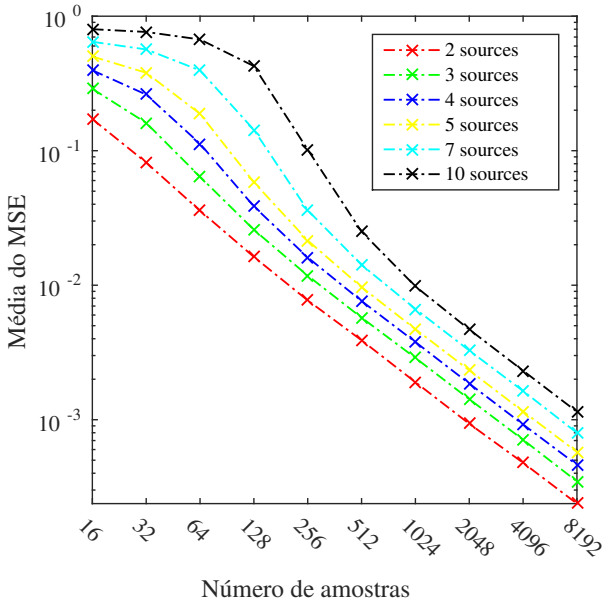
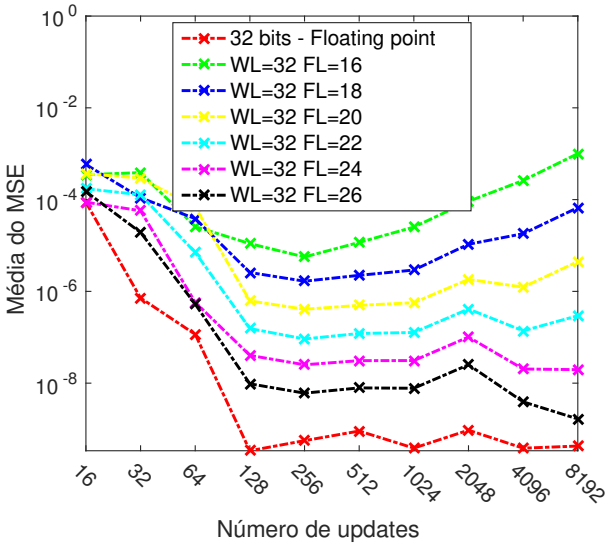


Figura 4 – Exatidão do algoritmo FastICA utilizando diferentes representações númericas quando comparada a implementação em ponto flutuante (64 bits)



utilizados quando a implementação for realizada em FPGAs. Dessa forma, deve haver sempre um compromisso entre a acurácia e a utilização de recursos do sistema embarcado. Em termos de velocidade de convergência, o FastICA exige em torno de 15 iterações para convergir.

Ao utilizar uma representação em ponto fixo, no entanto, o desempenho tende a ser pior, uma vez que os erros acumulados nas operações podem levar a uma degradação significativa na estimação do sistema de separação. Considerando apenas a situação de separação de 2 fontes, foram realizados diferentes experimentos considerando algumas configurações de representação em ponto fixo, e a diferença em relação ao desempenho obtido com representação em ponto flutuante é apresentada na Figura 4. As diferentes curvas correspondem a diferentes configurações de representação, considerando o tamnho da palavra $WL = 32$ e diferentes condições para a parte fracionária dos dados (FL).

Os resultados obtidos indicam que, ao considerar representações com 32 bits, sendo a maior parte deles para a representação da parte fracionária, ambos os algoritmos apresentam pouca degradação em relação à implementação em precisão dupla. Dessa forma, a definição sobre uma ou outra representação deve levar em consideração também aspectos relativos à implementação no hardware.

A implementação em FPGA seguiu o fluxograma apresentado na Figura 2[1], de maneira que pode ser facilmente adaptado para a separação de mais fontes - uma vez que utiliza a abordagem de separação por *deflação* [9]. A quantidade de recursos utilizados da FPGA para cada uma das representações considerada pode ser vista na Tabela 1, onde nota-se que a quantidade de recursos utilizados para a representação em ponto flutuante com 64 bits, como esperado, é bem maior do que nas demais representações.

Cabe destacar que a representação em ponto flutuante de 32 bit apresentou uma redução bastante significativa no uso de blocos DSP da FPGA (bem menor do que no caso de representação em ponto fixo), mas conforme pode-se observar na Tabela 2, o *througput* alcançado é bem inferior ao obtido com a representação em ponto fixo (cerca de 1/3 menor).

Conclusões

Este trabalho teve como objetivo demonstrar a viabilidade, e a implementação do algoritmo FastICA, utilizando uma representação numérica que maximiza o desempenho e atende aos requisitos de aplicações que resolvem o problema de BSS em tempo real.

Para tanto, foi realizada a análise e comparação

[1]Por restrições de espaço, não é possível apresentar o diagrama completo implementado.

Tabela 1 – Recursos do FPGA Cyclone® V SE 5CSEMA4U23C6N utilizados pelo modelo do algoritmo FastICA

Representação Numérica	DSP	LE	ALM	*Registers*
Ponto Flutuante (64 bits)	65 77,38%	11.965 29,91%	13.263 87,87%	10.463 17,33%
Ponto Flutuante (32 bits)	16 19,05%	5.479 13,70%	5.842 38,70%	3.952 6,55%
Ponto Fixo WL=32 FL=26	56 66,67%	1.507 3,77%	1.681 11,14%	1.487 2,46%

Tabela 2 – Desempenho do modelo do algoritmo FastICA em FPGA com 8192 amostras e 13 iterações

Descrição	Máx. freq. (MHz)	Latência (clocks)	Iterações por segundo	*Throughput* (MSPS)
Ponto Flutuante (64 bits)	59,22	8.255	7.173	4,52
Ponto Flutuante (32 bits)	59,26	8.233	7.197	4,53
Ponto Fixo WL=32 FL=26	89,92	8.225	10.936	6,89

de implementações utilizando representação numérica de ponto flutuante com precisão dupla, e sobre o comportamento e escalabilidade de tais implementações, quando necessário aumentar o número de fontes de sinais a serem estimadas.

A implementação do núcleo do algoritmo foi feita utilizando a biblioteca DSP Builder® da Altera® considerando o problema de separação de duas fontes de sinais. O algoritmo FastICA apresentou um bom desempenho em termos do MSE, utilizando a representação em ponto-fixo, situação na qual apresentou o maior *throughput*.

Como perspectivas de continuidade do presente trabalho, pode-se mencionar a implementação do algoritmo FastICA para um maior número de fontes, buscando explorar a escalabilidade presente no modelo proposto, utilizando uma ou mais FPGAs. Além disso, com base nos resultados e discussão realizada, pode-se investigar o uso de recursos de computação heterogênea - envolvendo o uso conjunto de FPGAs e processadores - para a execução dos métodos considerando um maior número de fontes. Por fim, cabe mencionar também a perspectiva de se investigar a implementação de outros métodos para execução em tempo real, como os métodos de separação considerando misturas convolutivas, ou mesmo misturas nas quais o número de fontes é superior ao número de sensores.

Agradecimentos

Os autores agradecem à CAPES e ao CNPq (Proc. 310610/2015-0) pelo apoio financeiro.

Referências

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

[2] A. Hyvärinen and E. Oja, "Independent component analysis: Algorithms and applications," *Neural Netw.*, vol. 13, pp. 411–430, May 2000.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] T. S. Hong and J. Kim, "FPGA implementation of jade ICA algorithm," in *2015 International SoC Design Conference (ISOCC)*, pp. 31–32, Nov. 2015.

[8] C. H. Anis Nordin and H. Szu, "Design of fpga ica for hyperspectral imaging processing," in *SPIE Proceedings*, vol. 4391, p. 444, March 2001.

[9] P. Comon and C. Jutten, *Handbook of Blind Source Separation: Independent Component Analysis and Applications*. Academic Press, 1st ed., 2010.