Single Cycle Nonlinear VLSI Cell for the ICA Algorithm

V. K. Jain
University of South Florida
Tampa, Florida 33620, U.S.A.

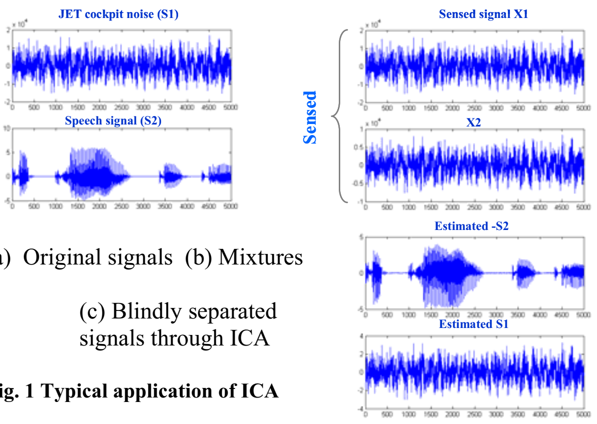
Abstract: This work is motivated by the desire to map the *Independent Component Analysis* (ICA) technique to a coarse-grain parallel-processing chip architecture. As in many other advanced DSP algorithms, the computation of nonlinear functions is critical in this algorithm. We discuss an efficient hardware approach[1] to the computation of such functions for the ICA, some of which are compound and concatenated functions. All of the needed functions are regularized into a single efficient algorithm, and a new result is produced ***every cycle*** – in a pipelined mode – even for a different function every cycle. The underlying principle which makes the combined goals of high-speed and ***multi-functionality*** possible is significance-based polynomial interpolation of ROM tables. Very importantly, the paper uses a key formula for predicting and bounding the worst case arithmetic error. *This theoretical result enables the designer to quickly select the architectural parameters* without the expensive simulations, while guaranteeing the desired accuracy.

I. INTRODUCTION

Independent Component Analysis (ICA) is one among several approaches to blind separation of signals. In theoretical terms, ICA is a method whereby high-dimensional multivariate data is decomposed into its *statistically independent (scalar) components* [1], thereby facilitating source signal acquisition, feature extraction and event classification. Imagine that in a room two people are speaking simultaneously and that there are two microphones which produce time signals denoted by $x_1(t)$ and $x_2(t)$. Each of these received signals is a weighted sum of the speech signals produced by the two speakers denoted by $s_1(t)$ and $s_2(t)$. Then we can express the received signals in terms of the source signals in terms of some weighting coefficients a_{11}, a_{12}, a_{21}, and a_{22} that depend on the microphone characteristics and their distances from the speakers. Clearly, it would be useful to recover the original speech signals from the received signals. More generally, if there are m source signals and m received mixed signals, then their relationship could be expressed as

$$\begin{array}{l} x_1(t) = a_{11}\, s_1(t) + a_{12}\, s_2(t) + \ldots + a_{1m}\, s_m(t) \\ x_2(t) = a_{21}\, s_1(t) + a_{22}\, s_2(t) + \ldots + a_{2m}\, s_m(t) \\ \quad \cdot \quad \cdot\;\cdot\;\cdot \qquad \cdot \qquad\qquad \cdot\;\cdot \\ x_m(t) = a_{m1}\, s_1(t) + a_{m2}\, s_2(t) + \ldots + a_{mm}\, s_m(t) \end{array} \tag{1}$$

or in matrix-vector notation $\underline{x}(t) = \boldsymbol{A}\,\underline{s}(t)$. The ICA technique can be used to estimate $\boldsymbol{A}$ or its inverse $\boldsymbol{W} = \boldsymbol{A}^{-1}$ based on the assumption of their statistical independence, which then allows blind separation of the original signals from their mixtures. As an example, consider the two source signals s_1, s_2 shown in Fig. 1(a), and that the only observable signals are those shown in Fig. 1(b). The question then is whether it is possible and, if 'yes', how to recover the source signals blindly (without the knowledge of the mixing information). The answer is affirmative based upon certain mild assumptions which can be found in [1],[2]. Indeed, the signals estimated by the application of the fast version of ICA, called Fast ICA [2],[3] are shown in Fig. 1(c). Except for a gain factor and sign, they are seen to be excellent replicas of the source signals. The technique is applicable not only to time signals but also to images, and has a wide range of potential applications in industrial and medical fields.



(a) Original signals (b) Mixtures

(c) Blindly separated signals through ICA

Fig. 1 Typical application of ICA

For some applications, ICA analysis on a workstation or a DSP board is adequate, but for others it is essential to have a VLSI chip that would perform the process in real-time. In [4] we had proposed mapping the ICA algorithm to the J-Platform which provides for many of the high speed applications such as FIR filtering of signals and images, Fast Fourier transform, solution of a linear system of equations, and advanced applications like reconstruction of CT images from fan beam projections and RGB to HSI conversion for video. Three very flexible, coarse-grain cells are used to map the ultra high speed objectives. These are the MA_PLUS [11] the Universal NonLinear (UNL) [5]-[8], and the Data fabric [11] cells. This paper discusses the architectural design of that UNL cell (for the computation of nonlinear functions of the ICA, some of which are *compound or concatenated functions*). With our approach all of the needed functions can be regularized into a single efficient algorithm. A new result can be produced ***every cycle*** – in a pipelined mode – even for a different function every cycle. The underlying principle which makes the combined goals of high-speed and ***multi-functionality*** possible is significance-based polynomial interpolation of small ROM tables. Very importantly, the paper uses a key formula for predicting and bounding the worst case arithmetic error. *This result enables the designer to quickly select the architectural parameters* without expensive simulations, while guaranteeing the desired accuracy. The paper emphasizes the

[1] This work was supported in part by NSF Grant No. 0441212

following five functions: (1) First derivative of (approximation to the) negentropy function [2] $g(u) = G'(u) = u\exp(-u^2/2)$; (2) second derivative of (approximation to the) negentropy function [2] $g'(u) = G''(u) = [1-u^2]\exp(-u^2/2)$; (3) $1/x$; (4) $\sqrt{x}$; and (5) $1/\sqrt{x}$. Note that the negentropy function itself $G(u) = -\exp(-u^2/2)$ is not required in fastICA updates, and is therefore not considered in the architecture. Also, note that the first two functions are both *compound and concatenated.* Preliminary estimates indicate that the speed-up for nonlinear function calculation could be a factor of 10 to 30 times as compared to DSP microprocessor boards, since iterative algorithms are typically employed on such boards.

II. NEGENTROPY FUNCTION AND DERIVATIVES

To gain an intuitive feel for the negentropy function (actually an approximation to it [2]), $G(u)$, and its derivatives, see Fig. 2. It will be seen in Appendix A that the third derivative of each function to be computed is also useful, therefore we plot derivatives up to $G^{(5)}(u)$ and even $G^{(6)}(u)$ whose zeros provides the locations of the maxima of $G^{(5)}(u)$. The corresponding mathematical expressions are also given in the figure.

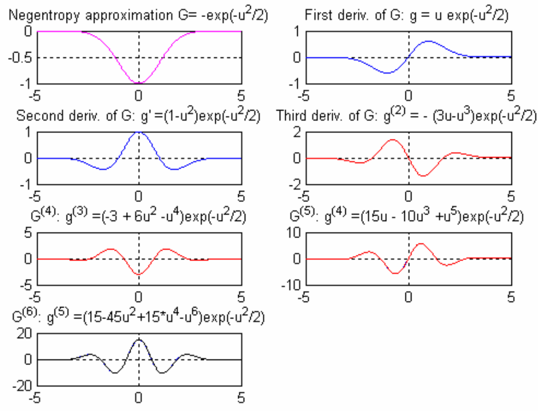


Fig. 2 The nonlinear functions for fastICA and several derivatives

It is useful to remark that the number of cycles needed for the computation of $g'(u) = G''(u) = [1-u^2]\exp(-u^2/2)$ on a DSP board may be estimated as $3+Q+R$ where Q is the number of cycles needed for determining the exponential, and R for norm/denorm.

III. SIGNIFICANCE-BASED COMPUTATION

As in [5], consider that the normalized argument 'x' is drawn from a semi-closed real interval $I_{domain} = [a,b)$ and is represented by a bit vector $\underline{X}$ that is N bits wide. Shown in Fig. 3, this word is segmented into two fields: the upper field consisting of M bits, and a lower field of L bits, such that $M+L=N$. The upper field invokes a 2^m point uniform grid ($2^{m-1}+2^{m-2}$ point in case of square-root and reciprocal-square-root) on the interval I_{domain}, where $m \le M$; call the points on this grid as X_i , $i=0,\ldots,2^m-1$. Now, suppose that the given operand 'x' lies in the i-th interpolation interval $I_i = [X_i , X_{i+1})$. Then, we can use a polynomial of the form $P_i(x) = f_i + yg_i + y_c y h_i$ to approximate $f(x)$ over $I_i = [X_i, X_{i+1})$. Here, y denotes the fractional location of x in the i-th interpolation interval I_i, i.e., $y=(x-X_i)/\Delta$, $\Delta = X_{i+1} - X_i$, the uniform grid interval, and $y_c=1-y$. Note that x is seen to be equal to $X_i+y\,\Delta$, which is in concert with the graphical representation in Fig. 1. Through a slight abuse of notation, we will write $P_i(x)$ also as $P_i(y)$. The use of the "***economic***" form [5],[8],

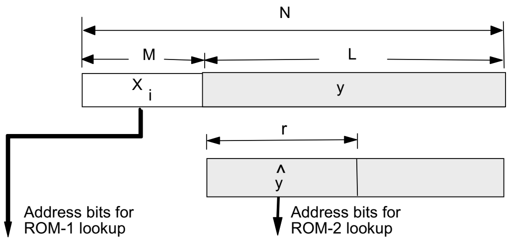


Fig. 3 Input word $\underline{X}$: M MSBs, L LSBs Value $x = X_i + (2^{-M})\,y$

involving the term yy_c rather than y^2 is designed to reduce the contribution of this second order term, thus leading to smaller blocks (on the VLSI chip) for the computation of this term.

In a typical function approximation using the above formula, the terms from left to right tend to be of diminishing importance. The central idea of Jain's 'significance-based computation' is then to recognize the smaller field-widths of the quadratic term arguments and therefore to use lower precision[2], compared to the linear term [5]-[8]. In fact, one of the early circuits Jain et al. developed for 16-bit arguments [8], uses only a single multiplication involving the linear term; the second order term was computed by a data driven shift upon h_i, a shift that approximates multiplication with yy_c. For larger word lengths, however, it becomes necessary to use (a) a ROM table lookup for an approximation of yy_c, addressed by the bits of $\hat{y}$, a reduced precision version of y, and (b) then multiplying this product with h_i on a small multiplier. This small multiplication can be performed in parallel with the main multiplication for the linear term $y * g_i$. The additions can be pipelined, with the consequence that even for a 24 bit argument a new result can be obtained *every cycle*. In fact, potentially, a different desired function (including compound and concatenated ones) for a new argument can be computed, each cycle. This represents a significant advance over other hardware approaches.

IV. COMPUTATION DETAILS

Computation flow: Returning to Fig. 3, the input word $\underline{X}$ is segmented into two fields. The upper field is used for addressing the main ROM, ROM-1, wherein the coefficients bit vector $\underline{b}_i = [\,f_i \;\; g_i \;\; h_i\,]$ is stored as the i-th word. The product yy_c is approximated and stored in a smaller ROM, ROM-2.

Chebyshev-Lagrange Polynomials: We use the Chebyshev-Lagrange polynomial. Thus, given the function $F(y)$ over the interval [0, 1], actually just its values at the roots z_k , the well known Chebyshev-Lagrange polynomial [5] of degree D (with D equal to the number of interpolation roots used minus one) is

$$P_i(y) = \sum_{k=0}^{D} F(z_k) \prod_{j=1, j\neq k}^{D} \frac{y - z_j}{z_k - z_j}$$

For the second order case, D=2, and the D+1 roots are: $z_k = -0.53$, 0, 0.53. Note that for our purposes $F(y) = f(X_i+y_-)$. It is also useful to remark that, in principle, the best polynomial can be generated using the Remez algorithm. However, for relatively fine grids, i.e., when Δ is small, the accuracy gain turns out to be insignificant.

Conversion to the Economic Form: The conversion is

[2] In the case of third order interpolation, used for double precision arguments [7], the third-order term is computed with yet lower precision — without significantly jeopardizing the accuracy of overall computation.

straightforward. If the polynomial obtained is $a_{i0} + a_{i1}\,y + a_{i2}\,y^2$, then its *equivalent economic form* is $f_i = a_{i0}$, $g_i = a_{i1}+a_{i2}$, and $h_i = -a_{i2}$.

Error Criterion: Although several different criteria are often used, each with its own specific mathematical significance, in this paper we use the following dual error criteria:

(a) $\varepsilon_{true} \triangleq \left| f(x) - \hat{P}(x) \right|$, and

(b) $\varepsilon_{qz} \triangleq \left| \tilde{f}(x) - \hat{P}(x) \right|$

where $f(x)$ is the true value of the function in infinite precision, $\hat{P}(x)$ is the value computed through our algorithm, and $\tilde{f}(x)$ is the rounded value of $f(x)$. The latter is often referred to as the '*exactly rounded*' value of the function. We will require that Prob{ $\varepsilon_{qz} > 1$ ulp } = 0, where ulp stands for *'unit in the last place'*.

V. Single-Cycle Architecture

An architecture for a 24 bit argument, multi-function UNL cell [5] is shown in Fig. 4. This multi-function architecture uses two multipliers: M1 of size 22×16 to calculate the linear term $y{*}g_i$, and M2 of size 14×12 to calculate the quadratic term $h_i{*}(yy_c)$. Also ROM-1 is 512×60 bits per function. Nine bits from the MSB's of the input are used for addressing the ROM-1, i.e., m=9. ROM-2 is 1024×10 bits, and is addressed by the leading 9 bits of y. Also, two adders are used in a pipeline mode to enable the generation of a new result ***every cycle***. The function select lines permit calculation of a

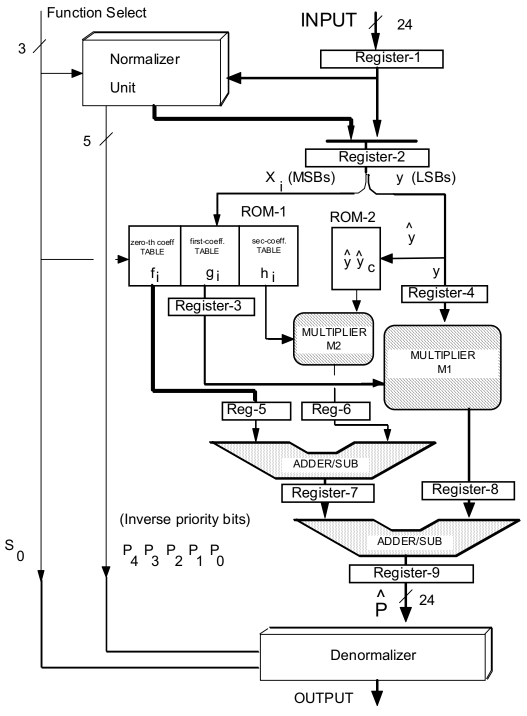


Fig. 3 Architecture of a 24 bit 'single cycle' nonlinear processor

different function, even every cycle. The normalizer unit ensures that the input to the nonlinear core is in conformity with the input ranges for which it is designed. Naturally, a denormalizer is provided at the multiplier output. M1 forms the critical path. If its latency, together with that of its output register, is τ_m, then this is also the cycle time of the processor in a pipelined mode, and the corresponding operating frequency becomes $1/\tau_m$. For recent advances in multipliers, see [9],[10].

Word Lengths:

W	=	Total number of bits in f_i = $I + w$ (I integer bits, w fractional bits)
v	=	Nonzero number of bits in g_i
u	=	Nonzero number of bits in h_i
r	=	Number of bits in $\hat{y}$
S	=	Number of bits in RD($\hat{y}\,\hat{y}_c$) = $2+s$, where s is the number of nonzero bits (the two leading bits are zeros)
A	=	Number of accumulator bits = $s+g+I+a$ (one sign, one guard, I integer bits, a fractional bits)
B	=	Number of output bits = $I_{out} + b$ (I_{out} integer bits);

$I_{out} = I$ for all functions except for cosine function for which $I=1$, but $I_{out} = 0$; also, for Log$_2$ a sign bit is needed so that $B=s+I+b$.

VI. Results for ICA Nonlinear Functions

The following five functions were considered of which the first two are special to the fastICA algorithm: (1) First derivative of (approximation to the) negentropy function [2] $g(u) = G'(u) = u\exp(-u^2/2)$; (2) second derivative of (approximation to the) negentropy function [2] $g'(u) = G''(u) = [1-u^2]\exp(-u^2/2)$; (3) $1/x$, (4) $\sqrt{x}$, and (5) $1/\sqrt{x}$). Note that the first two functions are both compound and concatenated. As stated earlier, all of these are single cycle computations, resulting in an estimated 10 to 30 fold reduction compared to conventional computation of such nonlinear functions. Two architectures were designed as discussed below.

A. 16 bit fixed point

The parameters chosen are W=20; a= 20; r =7; S=7. This results in the following design and the corresponding errors.

```
Input [N W m r]:  [16 20 7 7]
        _b___w___a___r___S_____hmax_____
g_ICA  | 16  20  22  7   9   4.2118e-05 |
gd_ICA | 16  20  22  7   9   9.1553e-05 |
REC    | 14  18  20  7   9   0.00012207 |
SQT    | 16  20  22  7   9   6.1035e-05 |
RQT    |_14__18__20__7___9___0.00073242_|

FUNC    ULP          |MAX ERR|     OK?
g_ICA   1.5259e-05   9.5567e-06     Y
gd_ICA  1.5259e-05   9.9981e-06     Y
REC     6.1035e-05   3.7819e-05     Y
SQT     1.5259e-05   9.7454e-06     Y
RQT     6.1035e-05   4.375e-05      Y
      Design is successful
```

B. 24 bit fixed point

The parameters chosen are W=24; a = 30; r =9; S=10. This results in the following design and the corresponding errors.

```
Input [N W m r]: [24 30 9 10]
        _b___w___a___r___S_____hmax_______
g_ICA  | 24  30  32  10   12   2.6324e-06 |
gd_ICA | 24  30  32  10   12   5.722e-06  |
REC    | 22  28  30  10   12   7.6294e-06 |
SQT    | 24  30  32  10   12   3.8147e-06 |
RQT    |_22__28__30__10___12___4.5776e-05_|
FUNC     ULP           |MAX ERR|      OK?
g_ICA    5.9605e-08   3.4324e-08      Y
gd_ICA   5.9605e-08   3.7827e-08      Y
REC      2.3842e-07   1.3411e-07      Y
SQT      5.9605e-08   3.5972e-08      Y
RQT      2.3842e-07   1.8487e-07      Y
      Design is successful
```

It is important to remark that these architectures are designed based on the error bound derived in Appendix A. Although the notation therein is somewhat involved and the derivation tedious, the final result is quite compact and versatile.

APPENDIX A. THEORETICAL ERROR PREDICTION

This discussion is quite similar to that in [5]. The approximation equation used, including the effect of all quantization steps and finite word lengths of the coefficients is

$$\hat{P} = RD_b\left\{\hat{f} + TR_a\left(\hat{g}*y\right) + TR_a\left(\hat{h}*RD_S\left[\hat{y}\hat{y}_c\right]\right)\right\}$$

where TR denotes truncation and RD roundoff. All errors discussed below are due to operations performed in the above equation. Each error estimate derived below is a reasonably tight upper bound for that particular operation. The sum of all those error bounds then provides an upper bound on the total error.

Error due to Chebyshev Approximation: The error in approximating a function *f* by a third order Chebyshev polynomial T_3 is given by

$$\nu \underline{\underline{\Delta}} \ |f(y) - T_3(y)| \ \le \ \frac{\Delta^3}{192} \ |f'''(\xi)|_{\max}$$

where Δ denotes the small interpolation interval as defined in Section II, and f '''(ξ) the third derivative.

Errors due to round-off in the coefficients:

$$\left|\hat{f}_i - f_i\right| \underline{\underline{\Delta}}\, e_1 \le 2^{-(w+1)} \ \ (\underline{\underline{\Delta}}\,\alpha_1); \quad \left|\hat{g}_i - g_i\right| \underline{\underline{\Delta}}\, e_2 \le 2^{-(w+1)} \ \ (\underline{\underline{\Delta}}\,\alpha_1);$$

$$\left|\hat{h}_i - h_i\right| \underline{\underline{\Delta}}\, e_3 \le 2^{-(w+1)} \ \ (\underline{\underline{\Delta}}\,\alpha_1);$$

Error in the linear term:

$$\hat{g}_i\, y = (g_i + e_2)\, y = g_i\, y + e_2\, y$$

Therefore,

$$\left|\hat{g}_i\, y - g_i\, y\right| \le e_2\, y_{max} \le \alpha_1 \quad (\underline{\underline{\Delta}}\,\beta_1)$$

since $y_{max} \le 1$. Now, because the accumulator truncation error is characterized by a one-sided uniform pdf, it is bounded as $\delta \le 2^{-a}$. As a result

$$TR_a\left(\hat{g}*y\right) \le g_i\, y + \alpha_1 + \delta$$

so that

$$\left|TR_a\left(\hat{g}_i y\right) - g_i y\right| \le = \alpha_1 + \delta \quad (\underline{\underline{\Delta}}\,\varepsilon_1)$$

Error in the second order term: Error in truncation of y to $\hat{y}$ is given by

$$\left|\hat{y} - y\right| \underline{\underline{\Delta}}\, e_4 \le 2^{-r1} \quad (\underline{\underline{\Delta}}\,\xi_1)$$

so that

$$\hat{y}\hat{y}_c = (y + e_4)\left[1 - (y + e_4)\right] \le yy_c + e_4(1 - 2y) - e_4^2$$
$$\approx yy_c + e_4(1 - 2y)$$

Therefore,

$$\left|\hat{y}\hat{y}_c - yy_c\right| = (y + e_4)\left[1 - (y + e_4)\right] \le \xi_1\ \left|1 - 2y\right|_{max} \le \xi_1 \ \ (\underline{\underline{\Delta}}\,\beta_2)$$

because $|1 - 2y|_{\max} = 1$.

The magnitude of the error due to rounding to *S*1 bits is $e_5 \le \zeta_1\ \underline{\Delta}\ 2^{-(S1+1)}$, therefore we can write

$$TR_a\left[\hat{h}_i\, RD_{S1}(\hat{y}\hat{y}_c)\right] = TR_a\left[(h_i + e_3)(\hat{y}\hat{y}_c + \beta_2 + e_5)\right]$$
$$\approx TR_a\left[h_i yy_c + e_3\, yy_c + (\beta_2 + e_5) h_i\right]$$
$$\approx h_i yy_c + e_3\, yy_c + (\beta_2 + e_5)\, h_i + \delta$$

and

$$\left|TR_a\left[\hat{h}_i\, RD_{S1}(\hat{y}\hat{y}_c)\right] - h_i yy_c\right|$$
$$\le \alpha_1 \left|y\, y_c\right|_{max} + (\beta_2 + \zeta_1)\left|h_i\right|_{max} + \delta \quad (\underline{\underline{\Delta}}\,\varepsilon_2)$$

Recognizing that $|yy_c|_{\max} = 1/4$, we have

$$\varepsilon_2 \le \alpha_1/4 + (\xi_1 + \zeta_1)\left|h_i\right|_{max} + \delta$$

Total Error: The magnitude of the error introduced due to the final round-off of *P* to *b* bits is $\theta \le 2^{-(b+1)}$, hence the total error is given by

$$\varepsilon \underline{\underline{\Delta}} \left|f - RD_b(P)\right| \le \nu + \alpha_1 + \varepsilon_1 + \varepsilon_2 + \varepsilon_3 + \theta$$

which can be written in an expanded form as

$$\varepsilon \ \le \ \nu + \alpha_1 + \alpha_2 + \alpha_3/4 + 3\delta + (\xi_1 + \zeta_1)\,|h_i|_{\max}$$
$$+ (\xi_2 + \zeta_2)\,|q_i|_{\max} + \theta$$

Finally, *the total L^∞* is bounded as follows:

$$\boxed{\begin{aligned}&\varepsilon_{total} \underline{\underline{\Delta}} \left|f - \hat{P}\right| \le \Delta^3 f^{(3)}(x)/192 + 9 \times 2^{-(w+3)} + 2^{-(a-1)}\\ &+ \left(2^{-r1} + 2^{-(S_1+1)}\right)|h|_{max} + 2^{-(b+1)}\end{aligned}}$$

The last term arises from *the final roundoff.* Thus

$$L^\infty(Error) \le \varepsilon + 2^{-25} \text{ for the 24 bit design}$$

REFERENCES

[1] A. Hyvärinen, J. Karhunen and E. Oja, *Independent Component Analysis,* NY: John Wiley and Sons, 2001.

[2] A. Hyvärinen and E. Oja, "Independent Component Analysis: algorithms and applications", *Neural Networks*, Vol. 13, pp. 411-430, 2000.

[3] A. Hyvärinen and E. Oja, "A fast fixed-point algorithm for Independent Component Analysis", *Neural Computation*, Vol. 9, Issue 7, pp. 1483-1492, October 1997.

[4] V. K. Jain, S. Bhanja, G. H. Chapman, L. Doddannagari, and N. Nguyen, "A Parallel Architecture for the ICA Algorithm: DSP Plane of a 3-D Heterogeneous Sensor," *Proc. Int. Conf. on Acoustics Speech and Signal Processing*, pp. V-77 to V-80, March 2005.

[5] V. K. Jain, S. Shrivastava, A. D. Snider, D. Damerow, and D. Chester, "Hardware implementation of a nonlinear processor," *Proc. IEEE Int. Symp. on Circuits and Systems*, pp. VI-509 to VI-514, May 1999.

[6] V. K. Jain, and L. Lin, "Floating-point nonlinear DSP coprocessor cell -- Two cycle chip," *Proc. IEEE Workshop on VLSI Signal Processing*, pp. 45-54, Oct. 1996.

[7] V. K. Jain, and L. Lin, "High-speed double precision computation of nonlinear functions," *Proc. Int. Symposium on Computer Arithmetic*, (Ed. Knowles and McAllister), pp. 107-114, July 1995.

[8] V. K. Jain, and L. Lin, "Image processing using a universal nonlinear cell," *IEEE Trans. on Components Packaging and Manufacturing Technology*, pp. 342-349, August 1994.

[9] N. Itoh, et al., "A 600-MHz 54×54-bit multiplier with rectangular-styled Wallace tree", *IEEE Journal of Solid-State Circuits,* pp. 249 - 257, Feb. 2001.

[10] Y. Hagihara, et al., "A 2.7 ns 0.25 µm CMOS 54 × 54b multiplier," *Proc. IEEE Int. Solid State Circuits Conf.*, pp. 296-297, Feb. 1998.

[11] V. K. Jain, and S. Shrivastava, "Rapid system prototyping for high performance reconfigurable computing," *Design Automation for Embedded Systems Jr,* pp. 339-350, August 2000.