32 Bit Single Cycle Nonlinear VLSI Cell for the ICA Algorithm

Vijay K. Jain
University of South Florida
Department of Electrical Engineering
Tampa, FL 33620
jain@eng.usf.edu

Earl E. Swartzlander, Jr.
University of Texas at Austin
Dept. of Electrical & Computer Engineering
Austin, TX 78712
eswartzla@aol.com

Abstract: *The Independent Component Analysis (ICA) technique is amenable to a coarse-grain parallel-processing chip architecture. However, the computation of nonlinear functions is critical in this algorithm. An efficient hardware approach is presented here for the computation of such functions*[1]*, some of which are compound and concatenated. All of the needed functions are regularized into a single algorithm so a new result produced on each cycle even if the function changes from one cycle to the next. The worst case arithmetic error is predicted and bounded. This enables the designer to quickly select the architectural parameters without expensive simulations, while insuring the desired accuracy. A design is presented for the 32 bit fixed point case.*

Key Words: Nonlinear cell, UNL, significance based computation, ICA.

I. INTRODUCTION

Independent Component Analysis is an attractive approach to the blind separation of signals. In theoretical terms, ICA is a method whereby high-dimensional multivariate data are decomposed into their *statistically independent (scalar) components* [1]-[3], thereby facilitating source signal acquisition, feature extraction and event classification.

The paper is organized as follows. Section 2 provides the motivation for hardware implementation of the ICA algorithm and the need for fast computation of the nonlinear functions used therein. Section 3 discusses the specific functions particular to ICA. Section 4 reviews the underlying principles that make the single-cycle architecture possible. Computation details are discussed in Section 5 and the corresponding architecture is shown in Section 6. Design results for all of the nonlinear functions needed by the ICA algorithm are given in Section 7.

2. MOTIVATION FOR HARDWARE IMPLEMENTATION

For some applications, ICA analysis on a workstation or a DSP board may be adequate, but for many others it is essential to have a VLSI chip that performs the process in real-time. Mapping the ICA algorithm to the J-Platform is proposed in [4],[5]. This platform provides the functions needed for many of high speed applications. Three cells are used. These are the MA_PLUS cell [11] the Universal NonLinear (UNL) cell [6]-[8], and the data fabric cell[11].

This paper discusses the architectural design of the UNL cell (for the computation of nonlinear functions in the ICA). With this approach all of the needed functions can be regularized into a single efficient algorithm. The underlying principle which makes the combined goals of single-cycle throughput and multi-functionality possible is significance-based polynomial interpolation using small ROM tables.

The paper emphasizes the following five functions: (1) the first derivative of an approximation to the negentropy function [2] $g(u) = G'(u) = u\ \exp(-u^2/2)$; (2) the second derivative of the approximation to the negentropy function [2] $g'(u) = G''(u) = [\ 1-u^2]\exp(-u^2/2)$; (3) the reciprocal $1/x$; (4) the square root $\sqrt{x}$: and (5) the reciprocal square root $1/\sqrt{x}$. Note that the negentropy function itself $G(u) = -\exp(-u^2/2)$ is not required in fast ICA updates and is therefore not considered in the architecture. Also, note that the first two functions are both compound and concatenated. Finally, the designs are based on approximations to negentropy, rather than kurtosis because of the advantages that this choice offers [2], [3]. Preliminary estimates indicate that the speed-up for nonlinear function calculation could be a factor of 10 or more as compared to DSP microprocessor boards, since iterative algorithms are typically employed on such boards.

3. NEGENTROPY FUNCTION AND DERIVATIVES

The approximation to the negentropy function [2], $G(u)$, and its derivatives, are shown in Fig. 1. Since the third derivative of each function is also useful, derivatives up to $G^{(7)}(u)$ whose zeros provide the locations of the maxima of $G^{(6)}(u)$ and the corresponding mathematical expressions are shown. 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.

[1] This work was supported in part by NSF Grants No. 0741338 and No. 0551621.

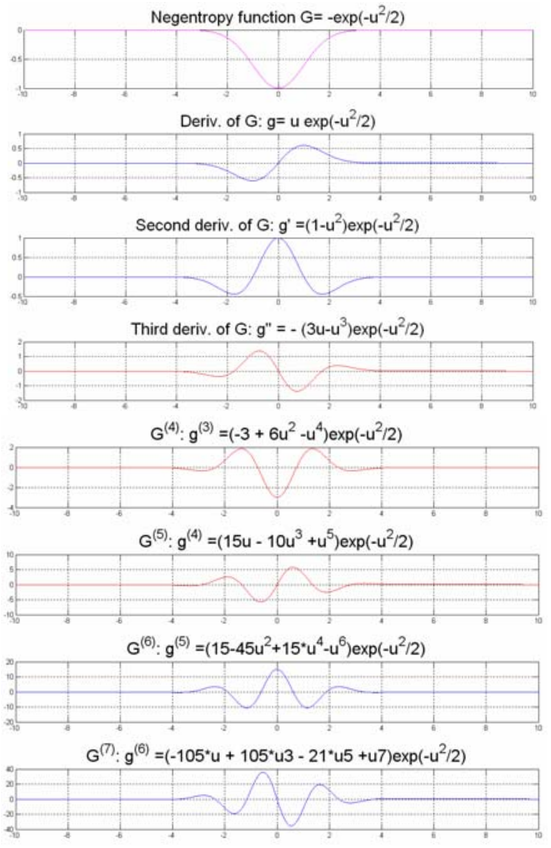

Fig. 1 The nonlinear functions for fast ICA and several derivatives

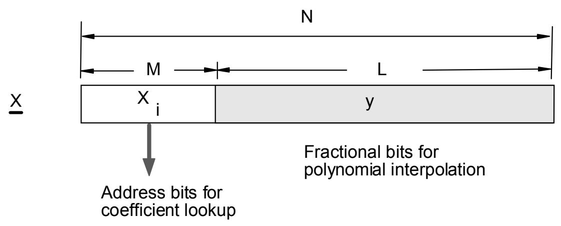
N
M
L
X
X
i
y
Fractional bits for polynomial interpolation
Address bits for coefficient lookup

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

4. SIGNIFICANCE-BASED COMPUTATION

As in [6] and [10], 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. As shown in Fig. 2, 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 ($2^{m-1}+2^{m-2}$ 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 operand 'x' lies in the i-th interpolation interval $I_i = [X_i\, , X_{i+1})$.

Then, a polynomial of the form shown below can be used to approximate $f(x)$ over $I_i = [X_i, X_{i+1})$.

$$\begin{aligned}\overline{P}3(x) &= a_{i0} + a_{i1}\,x + a_{i2}\,x^2 + a_{i3}\,x^3 \\ &\underline{\underline{\Delta}}\; P_3(y) = F_i + G_i\,y + H_i\,y^2 + Q_i\,y^3 \\ &= f_i + g_i\,y + h_i\left(y\,y_c\right) + q_i\left(y^2 y_c\right)\end{aligned}$$

Here, y denotes the fractional location of x in the i-th interpolation interval I_i, i.e., $y = (x\text{-}X_i\,)/\Delta$, where $\Delta = X_{i+1}\text{-}\, X_i$, the uniform grid interval, and $y_c = 1\text{-}y$. Note that $x = X_i + y\Delta$, which is in concert with the graphical representation in Fig. 2. Through a slight abuse of notation, $P_i(x)$ is written also as $P_i(y)$. The use of the "economized form" in the last line [7]-[10], involving the term yy_c and y^2y_c rather than y^2 and y^3 is designed to reduce the contributions of the second and third order terms, thus leading to smaller blocks (on the VLSI chip) for the computation of these terms. In typical function approximations using the above formula, the terms from left to right tend to be of rapidly diminishing importance. The central idea of 'significance-based computation' is then to recognize the smaller field-widths of the quadratic and cubic term arguments and therefore to use much lower precision arithmetic compared to the linear term [7]-[10]. The relationships between the polynomial coefficients a_i and the f_i, g_i, h_i, and q_i coefficients of the economized form are as follows:

$$f_i = a_{i0}\,;\; g_i = a_{i1} + a_{i2} + a_{i3}\,;\; h_i = -\left(a_{i2} + a_{i3}\right);\; q_i = -\,a_{i3}\,;$$

$$\Leftrightarrow$$

$$a_{i0} = f_i\,;\; a_{i1} = g_i + h_i\,;\; a_{i2} = -h_i + q_i\,;\; a_{i3} = -q_i\,;$$

5. COMPUTATION DETAILS

Computation flow: Returning to Fig. 2, the input word $\underline{X}$ is segmented into two fields; the upper field is used for addressing the coefficient ROM, ROM-1, wherein the coefficients bit vector $\underline{b}_{\,i}$ = [$\underline{f_i}$ $\underline{g_i}$ $\underline{h_i}$ $\underline{q_i}$] is stored as the i-th word. The computations are then performed based on the above third order economized polynomial, as shown in the flow diagram of Fig. 3. Note that the products yy_c and y^2y_c are approximated and stored in ROM-2 and ROM-3, respectively.

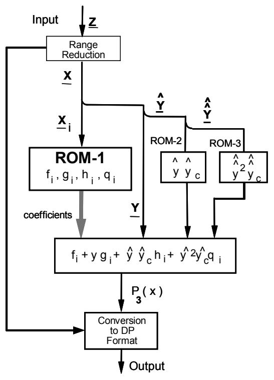

Fig. 3 Computation flow diagram

Chebyshev-Lagrange Polynomials: The well known Chebyshev-Lagrange polynomials are used. Given the function *F(y)* over the interval [0, 1], more specifically, its values at the roots z_k, the Chebyshev-Lagrange polynomial [6], [10], [12] of degree *D* 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 third order case *D*=3, and the *D*+1 roots are

$$z_i = \pm\sqrt{\frac{1}{2}}\sqrt{1 \pm \sqrt{\frac{1}{2}}} \ .$$

Note that $F(y) = f(X_i + y)$. In principle the best polynomial could be generated using the Remez algorithm [6]. However, for relatively fine grids, i.e., when Δ is small, the benefit can turn out to be very insignificant. As discussed earlier, conversion to the economized form is straightforward.

Error Criterion: Although several different criteria are often used, each with its own specific mathematical significance, in this paper the L^∞ error criteria is used: $Sup\left| f(x) - \hat{P}(x) \right|$, where $f(x)$ is the true value of the function in infinite precision, and $\hat{P}(x)$ is the value computed through our algorithm.

Design Objective: The design objective is to keep the L^∞ error to less than or equal to one ulp (unit in the last place).

6. SINGLE-CYCLE ARCHITECTURE

An architecture for a 32 bit fixed-point argument is shown in Fig. 4. This multi-function architecture uses three multipliers: M1 of size 28 by 20 [13]-[14] to calculate the linear term $y{*}g_i$, M2 of size 12 by 10 to calculate the quadratic term $h_i{*}(yy_c)$, and M3 of size 2 by 8 to calculate the cubic term $q_i{*}(y^2 y_c)$. Thirteen bits from the MSB's of the input are used for addressing ROM-1; ROM-2 is addressed by the leading 11 bits of *y*; ROM-3 is addressed by the leading 7 bits of *y*. Also, three adders are used in a pipelined mode to enable the generation of a new result every cycle. The function select lines permit calculation of a different function, even every cycle. The multiplier 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 [13]-[16]. The latency, however, is four clock cycles – measured from the output of the input register to the cell's output register.

Due to a lack of space, the details of selection of coefficient field widths are not provided here. The widths of $\underline{f}_i$, $\underline{g}_i$, $\underline{h}_i$, and $\underline{q}_i$ are in a diminishing order. As such, only a limited number of bits need be stored in ROM-1 and used for computations. In general, this selection represents tradeoffs between the sizes of the VLSI macros and the desired accuracy (typically, the error budget is a single ulp).

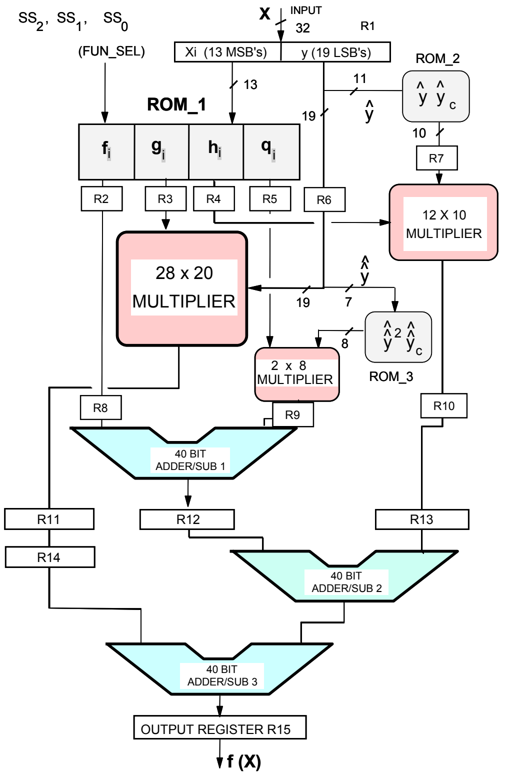

Fig. 4 Architecture of 32 bit multi-function single-cycle nonlinear processor

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

7. RESULTS FOR ICA NONLINEAR FUNCTIONS

The following five functions were considered, of which the first two are specific to the fast ICA 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 fold reduction compared to conventional computation of such nonlinear functions.

The parameters chosen are $W = 40$, $a = 42$; $r1 = 13$; $S1 = 10$; $r2 = 7$ and $S2 = 8$. This results in the following design and the corresponding errors.

Input [N W m r1 s1 r2 s2] : [32 40 12 13 10 7 8]

```
 FUNC      b  w  a    hmax        qmax
g(ICA)   |31 40 42 1.6066e-010 1.7764e-015
gd(ICA)  |31 40 42 3.4925e-010 3.4242e-015
REC      |32 39 41 4.6566e-010 7.1054e-015
SQRT     |32 40 42 2.3283e-010 8.8818e-016
REC-SQRT|32 39 41 6.9849e-010 1.7764e-014
```

```
 FUNC     ULP             |MAX ERR|  OK?
g_ICA    | 4.6566e-010  1.433e-010    Y
gd_ICA   | 4.6566e-010  1.7283e-010   Y
REC      | 2.3283e-010  1.9287e-010   Y
SQRT     | 2.3283e-010  1.5454e-010   Y
REC-SQRT| 2.3283e-010  2.2968e-010   Y

         Design is successful
```

ROM Sizes: For the design, ROM-1 is unavoidably large with a depth of 2^{12} (2^{13} for reciprocal and reciprocal-square-root) and a width of 82 bits for each function. ROM-2 is quite small with a depth of 2^{11} and a width of 11 bits. ROM-3 is still smaller with a depth of 2^7 and a width of 8 bits. Clearly RAMs could be used instead, in which case the nonlinear functions in the cell could be changed at start up for a particular application. The design meets the desired objective of 1 ulp, or less, errors.

8. CONCLUSIONS

Several advanced algorithms in medical and commercial applications require high-speed, high-accuracy computation of nonlinear functions. An efficient approach has been presented for the computation of multiple nonlinear functions in a VLSI coprocessor so that accurate results can be obtained in a single clock cycle for 32 bit fixed point words, even for compound and concatenated functions.

The polynomial based interpolation on a grid on the interval of interest, in contrast to global polynomial approximation [11], [12], yields considerable benefits in terms of high accuracy and small VLSI macros. The principle of significance based computation, together with an economized form third order polynomial, further reduces the sizes of these macros. For example, the third degree term requires only a small 2 by 8 bit multiplier, rather than a full 32 by 32 multiplier. An important contribution of the paper is the use of an error bound which enables the design parameters to be selected without long simulations.

REFERENCES

[1] 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 and L. Doddannagari, "A Highly Reconfigurable Computing Array: DSP Plane of a 3-D Heterogeneous SoC," *Proc. IEEE Int. System on a Chip Conf.*, pp. 243- 246, Sept. 2005.

[5] 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.

[6] M. J. Schulte and E. E. Swartzlander, Jr., "Hardware design for exactly rounded elementary functions," *IEEE Trans. on Computers, Special issue on Computer Arithmetic*, pp. 964-973, August 1994.

[7] 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.

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

[9] 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.

[10] V. K. Jain, "Double precision nonlinear cell for fast independent component analysis algorithm," *Proc. SPIE Defense and Security Symposium*, Vol. 6231, pp. OK-1 to OK-12, April 2006.

[11] I. Koren and O. Zinaty, "Evaluating elementary functions in a numerical coprocessor based on rational approximations," *IEEE Trans. on Computers*, pp. 1030-1037, Aug. 1990.

[12] P. T. P. Yang, "Table-lookup algorithms for elementary functions and their error analysis," *Proc. 10-th Symposium on Computer Arithmetic*, pp. 232-236, June 1991.

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

[14] N. Itoh, *et al.*, "A 600-MHz 54×54-bit multiplier with rectangular-styled Wallace tree," *IEEE J. Solid-State Circuits*, pp. 249-257, Vol. 36, Feb. 2001.

[15] S. K. Hsu, S K. Mathew, M. A. Anders, B. R. Zeydel, V. G. Oklobdzija, R. K. Krishnamurthy and S. Y. Borkar, "A 110 GOPS/W 16-bit multiplier and reconfigurable PLA loop in 90-nm CMOS," *IEEE J. Solid-State Circuits*, pp. 256-264, Vol. 41, Jan. 2006.

[16] S. R. Vangal, Y. V. Hoskote, N. Y. Borkar and A. Alvandpour, "A 6.2-GFLOPS floating-Point multiply-accumulator with conditional normalization," *IEEE J. Solid-State Circuits*, pp. 2314-2323, Vol. 41, Oct. 2006.