10.2478/v10048-011-0022-1

MEASUREMENT SCIENCE REVIEW, Volume 11, No. 4, 2011

Implementation of Fast-ICA: A Performance Based Comparison Between Floating Point and Fixed Point DSP Platform

Dinesh Patil, Niva Das[1], Aurobinda Routray[2]

Indian Institute of Technology, Electrical Engineering, 721302, Kharagpur, India, dinesh.ptl@gmail.com
[1]I.T.E.R, Electronics and Communication Engineering, SOA University, 751030, Bhubaneswar, India, nivadas@gmail.com
[2]Indian Institute of Technology, Electrical Engineering, 721302, Kharagpur, India, aroutray@iitkgp.ac.in

The main focus of the paper is to bring out the differences in performance related issues of Fast-ICA algorithm associated with floating point and fixed point digital signal processing (DSP) platforms. The DSP platforms consisting of TMS320C6713 floating point processor and TMS320C6416 fixed point processor from Texas Instruments have been chosen for this purpose. To study the consistency of performance, the algorithm has been subjected to three different test cases comprising of a mixture of synthetic signals, a mixture of speech signals and a mixture of synthetic signals in presence of noise, respectively. The performance of the Fast-ICA algorithm on floating point and fixed point platform are compared on the basis of accuracy of separation and execution time. Experimental results show insignificant differences in the accuracy of separation and execution time obtained from fixed point processor when compared with those obtained from floating point processor. This clearly strengthens the feasibility issue concerning hardware realization of Fast-ICA on fixed point platform for specific applications.

Keywords: blind source separation, fast-ICA, floating point implementation, fixed point implementation

1. INTRODUCTION

BLIND SOURCE separation (BSS) involves separating a number of unknown sources from a set of observed mixture of sources. The problem of BSS arises in diverse fields like image processing, biomedical signal processing, speech processing [1-4][18], etc., where independent component analysis (ICA) [5] methods have been successfully applied. Some applications involving speech, acoustic noise, biomedical signals, etc., require real time processing. Fast-ICA, a commonly used ICA algorithm based on fixed point iteration is suitable for real time operation because of its faster convergence speed [6].

The present work investigates the feasibility of integrating Fast-ICA algorithm into high end consumer devices. For DSP chips embedded in handheld devices like mobile phones and pagers, fixed point implementation is preferred due to its lower cost and lower power consumption ability as compared to the floating point case. Few papers report issues related to real time implementation of ICA algorithms. Charonsek and Sattar [7] propose a design method for an ICA based BSS algorithm on FPGA platform to suit the real time environment. Shyu and Li [9] have demonstrated the implementation of Fast-ICA algorithm on FPGA using floating point arithmetic. The main focus of this paper is on the reduced execution time and high accuracy achievable through the use of hierarchical design procedure and 32 bit floating point format. The hardware realization of a source separation algorithm known as DUET (Degenerate Unmixing estimation Technique) has been proposed in [8]. DUET employs time-frequency masking methods to separate an arbitrary number of sources from just two mixtures and is reported to be computationally inexpensive. The paper highlights the performance related issues associated with implementation of DUET on floating point and fixed point DSP processors.

In this paper the ICA algorithm has been tested on a 32-bit fixed point as well as on a 32-bit floating point DSP processor. The major manufacturers in the DSP industry are Texas Instruments (TI), Analog Devices and Motorola. TI and Analog Devices offer both fixed point and floating point DSP families while Motorola offers fixed point DSP families. We selected TI for our experimentation as its products are most widely used. The performance of the algorithm on both cases has been evaluated for its accuracy of separation and execution time. While accuracy of separation is judged by the coefficient of correlation between original source and separated source, execution time is the time taken by the processor to execute the algorithm.

The organization of the paper is as follows: Section II brings out the problem under consideration. Section III gives an overview of the Fast-ICA algorithm. Section IV outlines approaches and considerations for implementation of Fast-ICA on a DSP processor. The issues involved while migrating from the floating point to fixed point platform have also been discussed in this section. Section V reports performance of the Fast-ICA algorithm on both floating point and fixed point processors. Section VI is the conclusion of the work.

2. PROBLEM FORMULATION

The basic BSS model employing linear instantaneous mixing and possessing equal number of sources as that of sensors ($n = m$) may be expressed as:

$$\mathbf{x}_k = \mathbf{A}_k \mathbf{s}_k \tag{1}$$

where, $\mathbf{A}_k$ is a $n \times n$ mixing matrix, $\mathbf{s}_k = [s_{1k}, s_{2k}, ..., s_{nk}]^T$, represents a vector of source signals, $\mathbf{x}_k = [x_{1k}, x_{2k}, ..., x_{nk}]^T$, represents a vector of sensor signals, at a particular time instant k.

The problem is to estimate the source signals $\mathbf{s}_k$ from the observed mixtures $\mathbf{x}_k$ by using a separating system or matrix $\mathbf{W}_k$ with no a priori information on either the distribution of sources or the mixing structure. The estimated sources are represented as:

$$\mathbf{y}_k = \mathbf{W}_k \mathbf{x}_k \tag{2}$$

where $\mathbf{y}_k = \begin{bmatrix} y_{1k} & y_{2k} & \cdots & y_{nk} \end{bmatrix}^T$ is an estimate of $\mathbf{s}_k$ and $\mathbf{W}_k$ is the separating matrix at time instant *k*.
Equation (2) may be expanded as:

$$\mathbf{y}_k = \mathbf{W}_k \mathbf{x}_k = \mathbf{W}_k \mathbf{A}_k \mathbf{s}_k = \mathbf{G}_k \mathbf{s}_k$$

Where, $\mathbf{G}_k$ is known as a global system matrix having dimension $(n \times n)$. In the ideal sense $\mathbf{G}_k$ should be a generalized permutation matrix, each row and each column of which contain only one non-zero element.

3. FAST-ICA ALGORITHM

Fast-ICA algorithm is one of the most popular methods used to solve problems in BSS. It is easy to use and more reliable as it does not depend upon any user defined parameters. Experimental results establish the fact that it outperforms most of the ICA algorithms in convergence speed. The algorithm uses a fixed point iteration scheme to find the local maxima of the cost function:

$$J_{\mathbf{G}} = \sum_{i=1}^{n} E\left\{\mathbf{G}\left(\mathbf{w}^T \mathbf{x}\right)\right\}$$

where G is a non-linear function and E stands for expectation.

The cost function to be maximized can be based on mutual information, likelihood, and some approximation of non-Gaussianity or some variations of these properties. A widely used contrast function is based on kurtosis and may be expressed as:

$$J(\mathbf{w}) = E\left\{\left(\mathbf{w}^T \mathbf{x}\right)^4\right\} - 3\|\mathbf{w}\|^4 + F\left(\|\mathbf{w}\|^2\right) \tag{3}$$

where:

$$kurt\left(\mathbf{w}^T \mathbf{x}\right) = E\left\{\left(\mathbf{w}^T \mathbf{x}\right)^4\right\} - 3\left[E\left\{\left(\mathbf{w}^T \mathbf{x}\right)^2\right\}\right]^2 = E\left\{\left(\mathbf{w}^T \mathbf{x}\right)^4\right\} - 3\|\mathbf{w}\|^4$$

under the constraint ||**w**|| =1 and *F* is a penalty factor due to the constraint. The learning rule has the form:

$$\mathbf{w}(k+1) = \mathbf{w}(k) \pm \mu(k)\left[\mathbf{x}(k)\left(\mathbf{w}(k)^T \mathbf{x}(k)\right)^3 - 3\|\mathbf{w}(k)\|^2 \mathbf{w}(k) + f\left(\|\mathbf{w}(k)\|^2\right)\mathbf{w}(k)\right] \tag{4}$$

where **x**(k) is the observation sequence, μ(k) is the learning rate sequence and f is the derivative of *F*/2. The expectations are removed by instantaneous values. The terms inside the square bracket are the gradients of kurtosis (first two terms) and the gradient of $F\left(\|\mathbf{w}\|^2\right)$ (third term). The gradient of $F\left(\|\mathbf{w}\|^2\right)$ has the form (scalar × **w**) as long as this is a function of $\left(\|\mathbf{w}\|^2\right)$ only. Positive sign before the bracket means finding the local maxima and negative sign before the bracket means finding the local minima. The fixed points **w** of the learning rule in (4) are obtained by taking the expectations and equating the change in weight **w** to zero, which may be expressed as:

$$E\left\{\mathbf{x}\left(\mathbf{w}^T \mathbf{x}\right)^3\right\} - 3\|\mathbf{w}\|^2 \mathbf{w} + f\left(\|\mathbf{w}\|^2\right)\mathbf{w} = 0 \tag{5}$$

As the third term in (5) can be written in the form (scalar × **w**), the final form of (5) is

$$\mathbf{w} = scalar \times \left(E\left\{\mathbf{x}\left(\mathbf{w}^T \mathbf{x}\right)^3\right\} - 3\|\mathbf{w}\|^2 \mathbf{w}\right) \tag{6}$$

The scalar term in the above equation is insignificant and the effect can be eliminated with normalization.

In the basic Fast-ICA method, the observed mixture signals are preprocessed and then whitened before being subjected to the separation algorithm. The mathematical relationships can be described as:

$$\mathbf{x} = \mathbf{A}\mathbf{s}, \mathbf{v} = \mathbf{V}\mathbf{x} \text{ so that, } E\left[\mathbf{v}\mathbf{v}^T\right] = \mathbf{I},$$

where, **x** is the observed signal, **s** is the source signal, **V** is the whitening matrix, **A** is the mixing matrix and **v** is the whitened signal.

Here, $\mathbf{v} = \mathbf{V}\mathbf{A}\mathbf{s}$, where $\mathbf{B} = \mathbf{V}\mathbf{A}$ is an orthogonal matrix, i.e., $E\left[\mathbf{B}\mathbf{B}^T\right] = \mathbf{I}$. The objective is to determine **B** responsible for separating the independent signals.

Fixed Point Algorithm for ICA: The steps of Fast-ICA for separating one independent component are shown.

1. Pre-whiten the observed data **x** to obtain **v**.
2. Take a random initial vector $\mathbf{w}(0)$ and normalize it to unity, i.e., $\mathbf{w}(0) = \mathbf{w}(0)/\|\mathbf{w}(0)\|_2$, and set $j = 1$.
3. Set $\mathbf{w}(j) = E\left\{\mathbf{v}\left(\mathbf{w}(j-1)^T \mathbf{v}\right)^3\right\} - 3\mathbf{w}(j\text{-}1)$. The expectation operator can be estimated using a large number of samples.
4. Set $\mathbf{w}(j) = \mathbf{w}(j)/\|\mathbf{w}(j)\|_2$, i.e., normalize **w**(*j*).
5. If $\left|\mathbf{w}^T(j)\mathbf{w}(j-1)\right|$ is not close to 1, then set $j = j+1$ and repeat step 3. Otherwise output vector **w**(*j*).
6. Using **w**(*j*), one of the separated signals is given by $\mathbf{s}(k) = \mathbf{w}^T(j)\mathbf{v}(k)$, $k = 1,2,\ldots$.

To estimate n independent components, the above algorithm is run for n times.

To ensure that different independent components are estimated each time, an orthogonalizing projection is added inside the recursive loop given above. Each $\mathbf{w}_i \ (i = 1, 2, \ldots)$ vector found in the above process is a column vector of the orthogonal matrix **B**. Thus independent components are estimated one by one by projecting the current solution $\mathbf{w}(j)$ on the space orthogonal to the columns of matrix **B** previously found. Define $\tilde{\mathbf{B}}$ as a matrix whose columns are the columns of matrix **B**. The projection operation is added to the beginning of step 4 above, which now becomes:

$$\mathbf{w}(j) = \mathbf{w}(j) - \tilde{\mathbf{B}}\tilde{\mathbf{B}}^T\mathbf{w}(j), \text{ then}$$

$$\text{Set} \quad \mathbf{w}(j) = \frac{\mathbf{w}(j)}{\|\mathbf{w}(j)\|_2}$$

4. TESTING

The Fast ICA algorithm has been tested for both synthetic signals and audio signals on two platforms:

- TI 6713 - 32 bit floating point platform
- TI 6416 - 32 bit fixed point platform.

The mixing matrices are generated randomly.

The performance of the algorithm is compared for floating point and fixed point implementations in terms of execution speed and accuracy of separation.

4.1. Floating point implementation

The TMS320C6713 was selected as the target processor [10], [17] for developing the algorithm. The device is a 32 bit processor based on the high performance very long instruction word (VLIW) architecture. Operating at 225 MHz, the C6713 delivers up to 1350 million floating point operations per second (MFLOPS), 1800 million instructions per second (MIPS) and with dual fixed/floating point multipliers up to 450 multiply-accumulate operations per second (MMACS). It has a 264 KB system on chip memory consisting of 4 KB Level 1 program cache, 4 KB Level 1 data cache and 256 KB Level 2 memory cache. Further details on the DSK and chip are available on the TI website [10].

The signals are downloaded to the real time floating point platform. Before being subjected to the separation algorithm, the signals are mixed using the randomly generated mixing matrix and then whitened on the DSK itself.

4.2. Fixed point implementation

The TMS320C6416 was selected as the target processor for the fixed point implementation. The device is a 32 bit processor based on the second generation high performance, very long instruction word (VLIW) architecture. With performance up to 5760 million instructions per second (MIPS) at a clock rate of 720 MHz, the C6416 possesses the operational flexibility of high speed controllers and the numerical capability of array processors. It can produce four 16-bit multiply-accumulates (MACs) per cycle for a total of 2880 multiply-accumulate operations per second (MMACS) or eight 8-bit MACs per cycle for a total of 5760 MMACs. The C6416 also has a 1056 KB system on chip memory consisting of 16 KB L1 program cache, 16 KB L1 data cache, and 1024 KB L2 cache. Full details on the DSK and chip are available on the TI website [11].

A fixed point implementation equivalent to the floating point system was carried out in C using the built in routines of the code composer studio v3.1 IDE [12] in the first case. This is the same as emulating the floating point program on the fixed point processor (C6416). Although the accuracy of separation is high, the execution time is large.

To reduce the execution time alongside maintaining an appreciable level of separation, in the second case, the fixed point code is derived through manual fixed point programming. The manual code optimization involves steps like [13], [15] :

- Replacement of floating point variables by fixed point ones, encoded as integers.
- Selection of appropriate fixed point format for scaling to avoid overflows and to reduce loss of precision.
- Implementation of arithmetic operations like addition, subtraction, multiplication, division, square root, shifting, truncation and change of exponent, etc., in fixed point arithmetic using a series of preprocessor macros [13].

The following subsections highlight the basic concepts of fixed point arithmetic used in developing the algorithm on the fixed point platform [15-16].

4.2.1. Fixed point representation

A fixed point number can be thought of an integer multiplied by a two's power with negative exponent, i.e., $QN\left(= Q \times 2^{-N}\right)$, where Q is the mantissa and N is the exponent. An alternative notation for fixed point representation is:

$M \bullet N$, where M is the number of integer bite and N is the number of fractional bits

The range of a fixed point number is defined by the integer part, e.g., $16 \bullet 16(\text{signed})$: range is [-32768,32767]. Similarly the precision of a fixed point number is the smallest difference between two successive numbers, e.g., $16 \bullet 16$: precision is $1/2^{16}$.

4.2.2. Fixed point arithmetic rules [16]

The rules for doing some basic arithmetic operations on fixed point integers are listed below:

- Conversion from real to fixed point numbers
  - Multiply by 2^N and round to nearest integer
  - $(\text{int})(R * (1 << N) + (R \geq 0 ? 0.5 : -0.5))$

  Conversion from fixed point number to real number
  - Cast to real and divide by 2^N
  - $(\text{float})F / (1 << N)$

- Conversion from/to integers
  - Shift N bits up or down (scaling by 2^N)
  - $F = I << N,\ I = F >> N$
- Addition (+) and Subtraction (-)
  -Same as adding and subtracting integers.
  -To perform the operation $c = a + b$, first convert a and b to have the same exponent and then add the mantissa.
- Multiplication $(a * b)$
  - Multiply as integers and divide the result by 2^N
  -For multiplication, the intermediate result from $(a * b)$ is in $2M \bullet 2N$ $(Q2N)$ format
  -Store the intermediate result in double sized integer format, e.g., for 32 bit fixed point numbers; the intermediate result is stored in a 64 bit integer.
  - $(\text{int})((INT64)a * (INT64)b) >> N$
- Division (a / b)
  - Multiply by 2^N and divide by b (as integers)
  -As the intermediate result is expected to produce overflow, store it in 64 bit integer.
  - $(((INT64)a << N)/b)$

5. EXPERIMENTAL RESULTS & DISCUSSION

The experimental results have been demonstrated for three different test cases:

- Synthetically generated signals with no noise added during mixing.
- Sound signals generated by musical instruments
- Synthetically generated signals with noise added during mixing.

The following three implementations of the Fast-ICA algorithm have been tested for the above cases.

- Implementation on floating point processor TI 6713
- Floating point program emulated (migration using inbuilt method) on TI 6416.
- Implementation of fixed point algorithm (code optimization through manual fixed point programming) on TI 6416.

Test Case 1 comprises of three signals, i.e., a sine wave having frequency of 800Hz, a square wave having frequency of 700Hz and a saw-tooth wave having frequency of 600Hz generated synthetically using MATLAB code. The signals were sampled at 10 KHz and mixed using the mixing matrix given by:

A= [0.0891 0.3906 -0.3408; -0.8909 -0.6509 0.8519; 0.4454 0.6509 -0.3976]

Fig.1, Fig.2 and Fig.3 show the source signals, mixed signals and the separated signals as a result of implementation on floating point DSP (C6713), respectively. Fig.4 shows the separated signals as a result of the floating point emulation on fixed point DSP platform (C6416). Fig.5 shows the separation results when Fast-ICA was coded using fixed point arithmetic and implemented on fixed point processor (C6416). Fig.6 presents a comparison of execution time required for number of samples in the form of bar-graph for three different implementations. Table 1 reflects the separation results in terms of correlation coefficients between the original source and the separated source after implementation on the DSP.

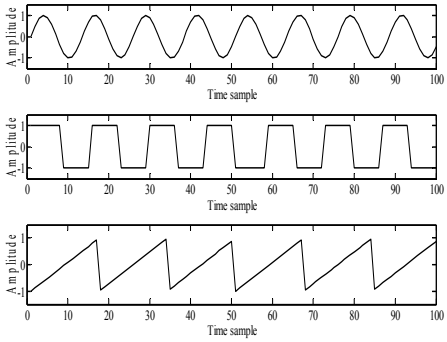

Fig.1. Source signals

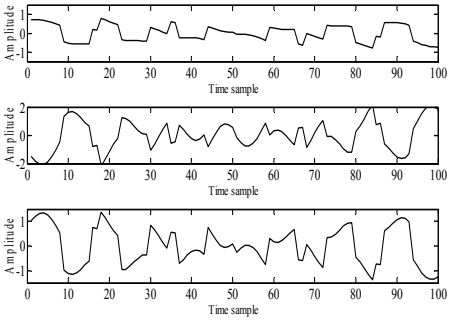

Fig.2. Mixed signals

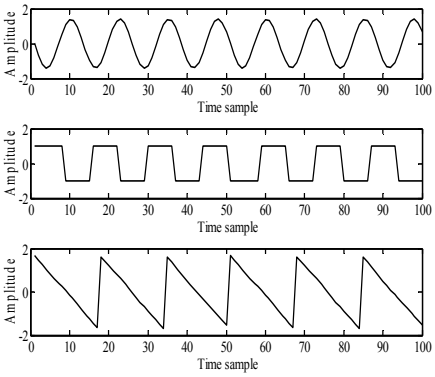

Fig.3. Separated signals by Fast-ICA on Floating-point 6713 DSP

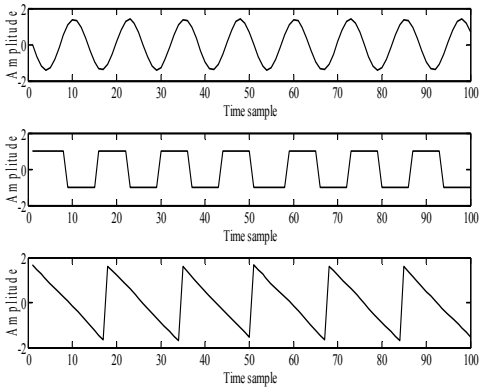

Fig.4. Separated signal by Fast-ICA with floating point emulation on Fixed-point 6416 DSP

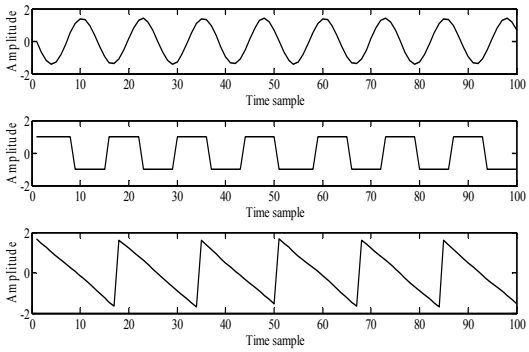

Fig.5. Separated signal by Fast-ICA on Fixed-point 6416 DSP with optimization

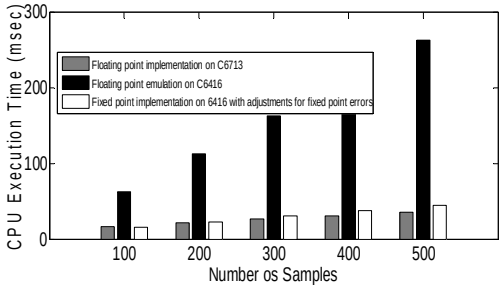

Fig.6. Number of samples vs. Execution Time

Table 1. Correlation coefficient comparison

Implementation Platform	Correlation Coefficient		
	Sine	Square	Saw-tooth
Floating-point Implementation on 6713	-1.0000	1.0000	-0.9997
Floating-point emulation on 6416	-1.0000	1.0000	-0.9997
Fixed-point Implementation on 6416 with optimization	1.0000	1.0000	0.9997

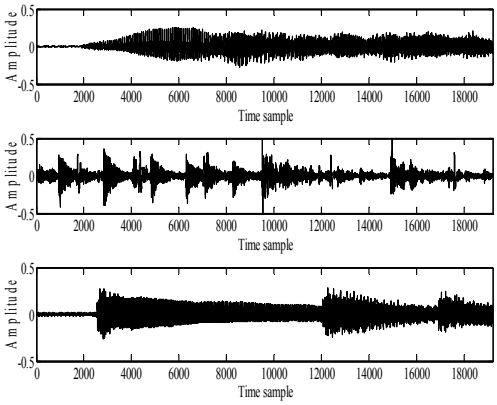

Fig.7. Sound Source signals

Test Case 2 comprises of sound signals generated from three musical instruments, i.e., violin, drums and piano recorded at a sampling frequency of 8 KHz to serve as source signals. 19200 numbers of samples have been considered for experimentation. The samples of the signals were mixed in the same way for all implementations using the mixing matrix **A** as specified for *test case 1*.

Fig.7, Fig.8 and Fig.9 show the original sound signals, mixed sound signals and the separated sound signals after implementation on the floating point DSP (6713), respectively. Fig.10 shows the separated signals as a result of floating point program emulated (without any optimization) on the fixed point DSP (6416). Fig.11 presents the results of optimized Fast-ICA implemented on a fixed point DSP (6416).

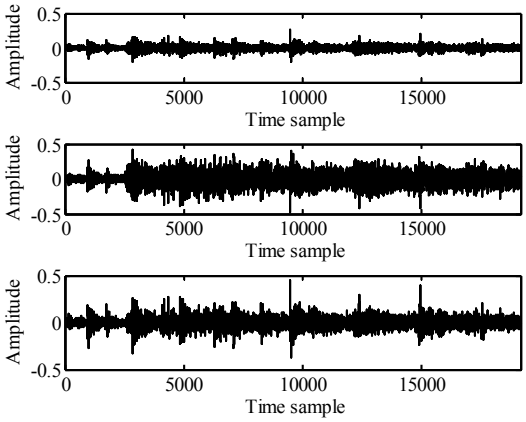

Fig.8. Mixed sound signals

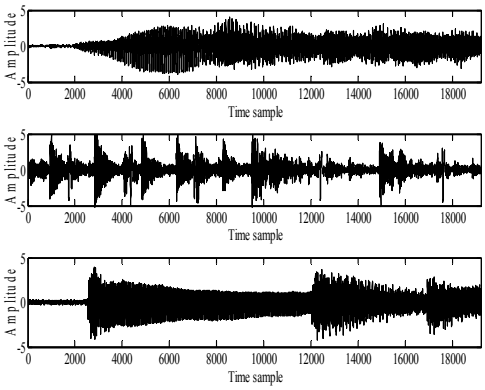

Fig.9. Separated signal by Fast-ICA on Floating point 6713 DSP

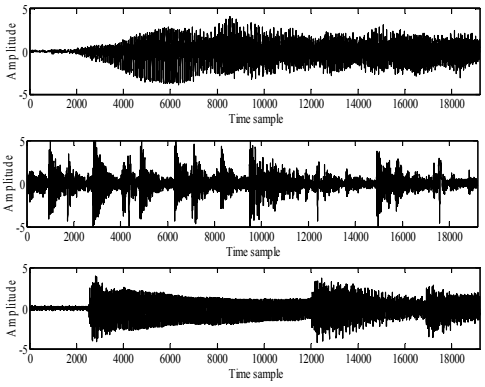

Fig.10. Separated signal by Fast-ICA on Fixed-point 6416 DSP without optimization

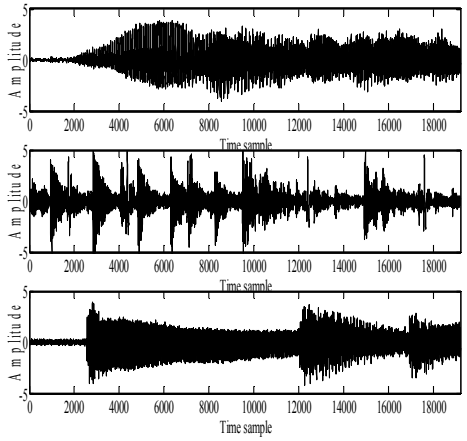

Fig.11. Separated signal by Fast-ICA on Fixed-point 6416 DSP with optimization

Table 2 and Table 3 display the correlation coefficients for the separated sound signals and the execution time required for 19200 numbers of samples for three different implementations, respectively.

Test case 3 comprises of three synthetically generated signals as specified in *test case 1*, exposed to additive Gaussian noise of 27dB during the mixing stage. The same mixing matrix **A** was used for the mixing purpose. Three different implementations have been done as was done for the first two test cases.

Table 2. Correlation coefficient comparison

Implementation Platform	Correlation Coefficient		
	Signal 1	Signal 2	Signal 3
Floating-point Implementation on 6713	-0.9987	-0.9984	-0.9999
Floating point emulation on 6416	-0.9987	-0.9984	-0.9999
Fixed-point Implementation on 6416 with optimization	-0.9976	0.9984	-0.9997

Table 3. Comparison of CPU Execution Time for sound signal

Implementation Platform	Execution Time (sec)
Floating-point Implementation on 6713	1.8490
Floating-point emulation on 6416	13.7221
Fixed-point Implementation on 6416 with optimization	1.6297

Fig.12 presents the results of implementation on the floating point DSP (6713). Fig.13 displays the separation results of the optimized fixed point Fast-ICA implemented on a fixed point DSP (6416). Table 4 shows the performance indices [14] obtained as a result of implementation of Fast-ICA on floating point DSP (6713) and implementation of Fast-ICA in optimized fixed point form on fixed point DSP (6416). The execution times for 100 numbers of samples for both implementations are almost the same as those of Table 1.

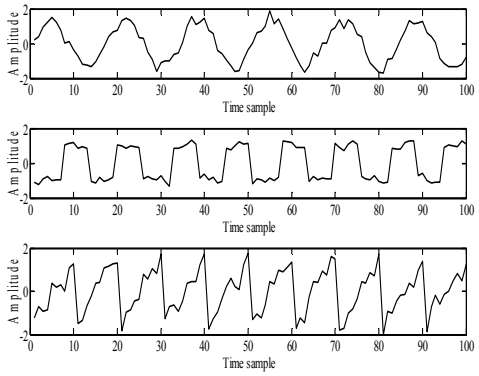

Fig.12. Separated signal by Fast-ICA on Floating-point 6713 DSP under 27dB Gaussian noise

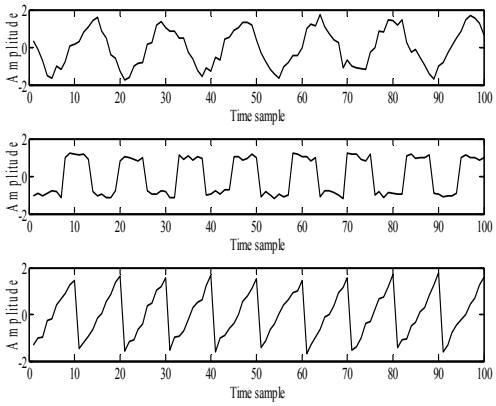

Fig.13. Separated signal by Fast-ICA on Fixed point 6416 DSP with optimization under 27dB Gaussian noise

Table 4. Performance index (under SNR 27 dB)

Implementation Platform	Performance Index
Floating-point Implementation on 6713	0.1134
Fixed-point Implementation on 6416 with optimization	0.1409

6. CONCLUSION

This paper presented a comparative study of a fixed point algorithm implemented on a fixed point platform with respect to another floating point processor. The accuracy and speed were found to be acceptable. In addition, the fixed point processor needs less space and consumes less power. More work needs to be done in this direction to embed these codes in portable consumer devices, without further deterioration of energy efficiency.

REFERENCES

[1] Cichocki, A., Kasprzak, W., Amari, S. (1996). Neural network approach to blind separation and enhancement of images. In *Signal Processing VIII: Theories and Applications*, Vol. 1, 579-582.

[2] Makeig, S., Bell, A., Jung, T.-P., Sejnowski, T.J. (1996). Independent component analysis in electro-encephalographic data. In Mozer, M. et al. (eds.) *Advances in Neural Information Processing Systems 8.* Cambridge, MA: MIT Press, 145-151.

[3] Weinstein, E., Feder, M., Oppenheim, A.V. (1993). Multichannel separation by decorrelation. *IEEE Trans. Speech and Audio Processing*, 1, 405-413.

[4] Karhunen, J., Hyvarinen, A., Vigario, R., Hurri, J., Oja, E. (1997). Application of neural blind separation to signal and image processing. In *Proceedings of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP'97).* Munich, Germany, 131-134.

[5] Common, P. (1994). Independent component analysis, A new concept? *Signal Processing*, 36, 287-314.

[6] Hyvarinen, A., Oja, E. (1997). A fast fixed point algorithm for independent component analysis. *Neural Computation*, 9 (7), 1483-1492.

[7] Charoensak, C., Sattar, F. (2005). A single-chip FPGA design for real-time ICA-based blind source separation algorithm. In *Proceedings of the IEEE International Symposium on Circuits and Systems,* Vol. 6, 5822-5825.

[8] Harte, N., Hurley, N., Fearon, C., Rickard, S. (2005). Towards a hardware realization of time frequency source separation of speech. In *Proceedings of the IEEE European Conf. on Circuit Theory and Design,* Vol. 1, 71-74.

[9] Shyu, K., Li, M. (2006). FPGA Implementation of FastICA based on Floating point arithmetic design for real-time blind source separation. In *Proceedings of the IEEE Int. Joint Conf. on Neural Networks,* 2785-2792.

[10] Texas Instruments Inc. (2001, revised 2005). *Floating-Point Digital Signal Processors* (manual, sprs1861). Houston, Texas.

[11] Texas Instruments Inc. (2005). *Fixed-Point Digital Signal Processor* (manual, sprs146n). Houston, Texas.

[12] Texas Instruments Inc. (2000). *Code Composer Studio User's Guide* (manual, spru328b). Houston, Texas.

[13] Ivancescu, G. (2007). *Fixed point arithmetic and tricks*. Retrieved August 12, 2011, from http://x86asm.net/articles/fixed-point-arithmetic-and-tricks/

[14] Amari, S., Cichocki, A., Yang, H. (1995). Recurrent neural networks for blind separation of sources. In *Proceedings of the Int. Symp. Nonlinear Theory Applications.* Las Vegas, NV, 37-42.

[15] Yates, R. (2009). *Fixed point arithmetic: An Introduction*. Retrieved August 12, 2011, from http://www.digitalsignallabs.com

[16] Lauha, J. (2006). *The neglected art of Fixed point arithmetic*. Retrieved August 12, 2011, from http://jet.ro/files/The_neglected_art_of_Fixed_point_arithmetic_20060913.pdf

[17] Ingole, P.V., Sapkal, A.K., Sarate, G.G., Hirekhan, S.R. (2011). Implementation of signal generator (DSP) using TMS 320C6713 DSK. *International Journal of Computer Science & Emerging Technologies,* 1 (2), 95-98.

[18] Krishnaveni, V., Jayaraman, S., Arvind, S., Hariharasudhan, V., Ramdoss, K. (2006). Automatic identification and removal of ocular artifacts from EEG using wavelet transform. *Measurement Science Review*, 6, 45-57.

Received May 16, 2011.
Accepted September 9, 2011.