Fixed-Point Hestenes SVD Algorithm for Computing Eigen Faces

Tapan Pradhan, *Member, IEEE,* Bibek Kabi, *Member, IEEE,* Aurobinda Routray, *Member, IEEE,* and Gonnabhaktula Anirudh

Abstract—The present paper discusses the development of fixed-point Hestenes' singular value decomposition (SVD) algorithm of symmetric matrices in computing eigen faces for image processing applications. Steps for developing fixed-point algorithm from floating-point version are discussed. Quantization noise propagation model for computationally intensive sections of the algorithm are derived and validated. The results are compared with the double precision floating-point version of the algorithm. Accuracy is measured in terms of number of accurate fractional bits, signal-to-quantization-noise-ratio (SQNR), orthogonality and factorization errors. Results show that the constant fixed-point format performs better than variable one in terms of number of accurate fractional bits when the variables have close ranges. Variable fixed-point format implementation exhibits an overall better performance. A case study has been presented where variable fixed-point Hestenes' SVD algorithm with three different wordlengths (WLs) is used as an image processing tool to compute eigen faces from one male and one female face database.

Keywords—Constant fixed-point format, covariance matrix, eigen faces, error propagation, fixed-point SVD, signal-to-quantization-noise-ratio (SQNR), variable fixed-point format, wordlength (WL).

I. Introduction

PREDOMINANTLY in all signal and image processing applications covariance matrix is used as an efficient feature descriptor. Several signal and image processing algorithms are based on subspace based method involving computation of eigenspace (singular values and singular vectors) of the covariance matrix. SVD plays an important role in finding singular triplet of these matrices. Principal component analysis (PCA), pattern recognition in signal and image processing are few examples where SVD is primarily used [1], [2], [33] and [34].

SVD of a dense matrix $A_{m\times n}$ can be stated as

$$A = U\Sigma V^T \tag{1}$$

where $U_{m\times m}$ and $V_{n\times n}$ are orthogonal (or unitary) matrices and their columns are called left and right singular vectors respectively. Σ is a diagonal matrix and contains all singular values along its diagonal in a non-increasing order. For a symmetric matrix $m = n$, U and V span the same vector space. Hence computation of either U or V is sufficient. For symmetric matrices, eigenvalue decomposition and singular value decomposition are closely related as follows [4] and [6]:
Suppose A is a symmetric matrix, with eigenvalues λ_i and orthonormal eigen vectors u_i so that $A = U\Lambda U^T$ is an eigenvalue decomposition of A, with $\Lambda = diag[\lambda_1\lambda_2...\lambda_n], U = [u_1u_2...u_n]$ and $UU^T = I$. Then an SVD of symmetric matrix A is, $A = U\Sigma V^T$, where diagonal elements of Σ i.e. $\sigma_i = abs(\lambda_i)$ and $v_i = sign(\lambda_i).u_i$ where $sign(0) = 1$. For symmetric positive definite matrices eigenvalue decomposition (EVD) and SVD leads to the same decomposition.

There are several algorithms for SVD as stated in literature [3]-[6]. Classical Jacobi algorithm for SVD is known to be accurate among all. However, being a two-sided version it is one of the slowest algorithms. Fixed-point format of Jacobi SVD algorithm has been implemented using CORDIC (COordinate Rotation DIgital Computer) and is faster as compared to its floating-point counter part [9]. Hestenes' algorithm is a variant of one-sided Jacobi's algorithm and is discussed in [7]-[8]. Being a one-sided version the time of computation is lesser than two-sided Jacobi's algorithm. To implement this algorithm in embedded platform and reconfigurable hardware, development and analysis of fixed-point version is necessary. Although numerical linear algebra algorithms like SVD and other signal processing algorithms are designed with floating-point data types, they are finally implemented in fixed-point architectures to reduce cost, chip size, power consumption and latency [32] and [35]. With the wide spread application of embedded computing in the last ten years, fixed-point algorithms have gained considerable importance. Therefore, we have taken up a case study for the development of fixed-point Hestenes' algorithm. We have developed the fixed-point version of the algorithm in constant and variable fixed-point format. A fixed-point format of a number can be represented as mQn, Q$m.n$ or Qn, where m is the number of integer bits or integer wordlength (IWL) and n is the number of fractional bits or fractional wordlength (FWL). In constant fixed-point format representation, definition of IWLs is same for all variables and remains static throughout the program. This type of representation is suitable for algorithms in which variables have close dynamic ranges. In variable fixed-point format representation, definition of IWLs are assigned to variables based on their statistics and remain same throughout the program. Both the fixed-point formats are implemented in C/C++ and SystemC environments. We have also evaluated the numerical accuracy of the fixed-point algorithm using bit-true (fixed-point) simulation and analytical based approach. Signal-to-quantization-noise-ratio (SQNR), number of accurate fractional bits, orthogonality and factorization errors are used to evaluate accuracy from bit-true simulation results. Quantization noise source and error propagation model for computationally intensive parts of the algorithm have been

xxxxxxx xxxxxxx
xxxxx xxxxxx
Manuscript received April xx, xxxx; revised December xx, xxxx.

TABLE I
HESTENES' ALGORITHM FOR SVD

$V = I$	$\dots$
repeat until convergence	$c = \frac{1}{\sqrt{1+t^2}}, s = t.c$
for $p = 1 : (n-1)$	end
for $q = (p+1) : n$	J=I
$\alpha = A(:,p)^T A(:,p)$, $\beta = A(:,q)^T A(:,q)$	$J(p,p) = c, J(q,q) = c, J(p,q) = s, J(q,p) = -s$
$\gamma = A(:,p)^T A(:,q)$	$A = A.J$, $V = V.J$
if $\gamma = 0$	endfor
$c = 1; s = 0$	endfor
else	for $k = 1 : n$
$\zeta = \frac{(\beta-\alpha)}{2\gamma}$	$\sigma_i = \|\|A(:,k)\|\|$
$t = \frac{sign(\zeta)}{(\|\zeta\|+\sqrt{1+\zeta^2})}$	end
contd $\dots$	$U = AV\Sigma^{-1}$

studied analytically. In a case study this fixed-point SVD algorithm has been utilized as an image processing tool to extract the dominating feature vectors or eigen faces of male and female subjects. Errors with respect to double precision floating-point algorithm are also presented.

The rest of the paper is organised as follows. Section II discusses Hestenes' algorithm for floating-point platforms. Conversion of floating-point to fixed-point procedure along with various fixed-point formats have been discussed in section III. Section IV presents the numerical accuracy evaluation using fixed-point simulation and analytical approach. Section V presents a case study of computing eigen faces using fixed and double precision floating-point Hestenes' SVD algorithms. Section VI discusses generalization of the fixed-point SVD algorithms in Tensor space. Fixed-point simulation results are stated and analyzed in section VII.

II. FLOATING-POINT HESTENES' ALGORITHM

A. Floating-point representation

Nowadays almost all floating-point processors use IEEE 754 data format to represent real numbers using finite precision arithmetic. According to this format real number is represented using sign, exponent and mantissa (SEM) bits. For a single precision 32-bit wordlength, 1-bit is reserved for sign, 8-bit for exponent and the remaining 23-bit for mantissa. A math co-processor is used to convert data in this format. C/C++, MATLAB and other programming environments have support for floating-point data type.

B. Hestenes' Algorithm

Hestenes' algorithm is based on one-sided Jacobi's algorithm. A symmetric matrix of size $n \times n$ is used to generate an orthogonal rotation matrix V so that the transformed matrix $A' = AV = W$ has orthogonal columns. Now if we normalize each non-null column of matrix W to unity, we get the relation $W = AV = U\Sigma = U \cdot diag(\sigma_1, \sigma_2, ..., \sigma_n)$, $A = U\Sigma V^T, V = \prod_{i=1}^{n} J_i$ and singular values of A are $\sigma_i = \parallel a'_i \parallel_2$, where $A' = [a'_1 a'_2 \dots a'_n]$ (Table I).

III. FLOATING TO FIXED-POINT CONVERSION

A. Fixed-point representation

A fixed-point number is represented as an integer multiplied by a two's power with negative exponent 'e'. The exponent 'e'

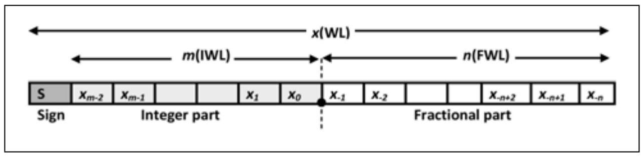

Fig. 1. Fixed-point representation

is considered as the number of digits after virtual binary point to represent fractional part. This fictitious binary point is also known as Q point. Thus the fixed-point format can be termed as mQn, Q$m.n$ or Qn, where m is the number of integer bits and n is the number of fractional bits. In SystemC fixed-point data is represented using the following attributes [21].

$$< wordlength, integer\ wordlength, \\ quantization\ mode, overflow\ mode > \tag{2}$$

Wordlength, integer wordlength are the parameters which influence the performance of fixed-point algorithms. The WordLength (WL) is the total number of bits required to represent a fixed-point number and depends on the architecture of the processing platform. The Integer WordLength (IWL) and Fractional WordLength (FWL) are the number of bits to the left and right of the virtual binary point respectively. Finite wordlength effects such as overflow and quantization are two serious concerns while converting a program from floating-point to fixed-point format [22]. These occur due to insufficient IWL and FWL respectively. Quantization mode includes rounding and truncation. Overflow or underflow is protected by means of saturation or wraparound process. Fig. 1 shows an example of a signed fixed-point number representation using $(m+n+1)$-bit, where m is the IWL and n is the FWL. The range for such a fixed-point number can be estimated as

$$-2^m \leq Range \leq (2^m - 2^{-n}) \tag{3}$$

Dynamic ranges and resolutions for the Q notation with different wordlengths are shown in Table II. It is well known that the fixed-point format have better precision than floating-point because the fixed-point format can represent numbers with more significant bits than floating-point with the same wordlength [23]. There are various representation of fixed-point numbers such as unsigned, signed and 2's compliment format. In this paper, we have used signed fixed-point format.

TABLE II
DYNAMIC RANGES AND RESOLUTIONS FOR DIFFERENT FIXED AND FLOATING-POINT NUMBER FORMATS

Type	Min	Max	Resolution
Fixed-point			
24-bit (Q23)	-1	0.999999999	1.1920×10^{-7}
32-bit (Q31)	-1	0.999999999	4.6566×10^{-10}
64-bit (Q63)	-1	0.999999999	1.0842×10^{-19}
Floating-point (normalized)			
32-bit	-3.4028×10^{38}	3.4028×10^{38}	1.1755×10^{-38}
64-bit	-1.7783×10^{308}	1.7783×10^{308}	2.2251×10^{-308}

TABLE III
FIXED-POINT FORMATS OF DIFFERENT ARITHMETIC OPERATIONS

Operations	Fixed-point formats of the operands	Fixed-point format of the result
Addition/Substraction	$a(m_1, n_1)$, $b(m_2, n_2)$	$IWL_{result} = max(m_1, m_2)$
Multiplication	$a(m_1, n_1)$, $b(m_2, n_2)$	$IWL_{result} = m_1 + m_2 + 1$
Division	$a(m_1, n_1)$, $b(m_2, n_2)$	$IWL_{result} = m_1 + n_2 + 1$

B. Fixed-point arithmetic

Hestenes' algorithm involves basic arithmetic operations like addition, subtraction, multiplication, division and square root. Different fixed-point formats of operands and results are shown in Table III.

C. Different fixed-point formats

Hestenes' algorithm has been implemented in two fixed-point formats mentioned below.

1) Constant
2) Variable

1) Constant fixed-point format: Since ANSI C or C++ do not support fixed-point data types, a new fixed-point data type along with fixed-point arithmetic operations have been constructed. When a floating-point algorithm is converted to fixed-point format using constant fixed-point format representation, all variables with float or double data types are converted to new fixed-point data type 'Fixed'. The new data type 'Fixed' for all variables are assigned with a single definition of IWL which remain same throughout the program. Optimum IWL is selected corresponding to minimum error. Being an iterative process selection of optimum IWL is expensive. This format can be useful for algorithms with less number of variables having close dynamic ranges.

SystemC based fixed-point data types used for this format are listed below [16].

```
sc_fxtype_params param(WL, IWL, q_mode, o_mode);
sc_fxtype_context c(param);
sc_fix_fast var;
```

The first two syntaxes are assigned at the beginning of the program with the information of WL, IWL, quantization and overflow mode. The last syntax is used to declare the variables.

2) Variable fixed-point format: When variables have different ranges, it would be insignificant to apply constant fixed-point format. This provides motivation to represent our fixed-point algorithm in variable format. Range estimation plays a significant role in estimating optimum integer wordlengths for variables. These IWLs remain same throughout the program. We have constructed fixed-point data type for variable type representation in C/C++. SystemC supports variable fixed-point data type listed below [16].

```
sc_fixed_fast < WL, IWL, q_mode, o_mode > var;
```

a) Conversion Process: To convert any floating-point algorithm to constant fixed-point format, float and double data types are converted to 'Fixed' data type using operator overloading characteristics of C/C++ and SystemC. Constant IWL definition is assigned either in the header file (C/C++) or in the beginning of the program (SystemC). Selection of optimum IWL is an iterative process and is done at a minimum error.

For variable fixed-point format, the conversion of floating-point algorithm into fixed-point format involves range estimation, computation of optimum integer wordlength and simulation of fixed-point algorithm. The development steps of variable fixed-point format is shown in Fig. 2.
It can be observed that IWLs of this format are computed from statistics (sum, number of data, mean, standard deviation, maximum, minimum and kurtosis) of the variables obtained from range estimation process. The floating-point algorithm is then converted to fixed-point format with optimum integer wordlengths. The fixed-point algorithm is then tested for desired accuracy. Implementation of fixed-point algorithms in ASIC, FPGA, etc. requires wordlength optimization [19] subject to accuracy constraint. However, wordlength optimization step is not required for uniform wordlength processors (DSP and ARM platforms). In this paper, the fixed-point Hestenes' algorithm is simulated using 24, 32 and 64-bit wordlengths.

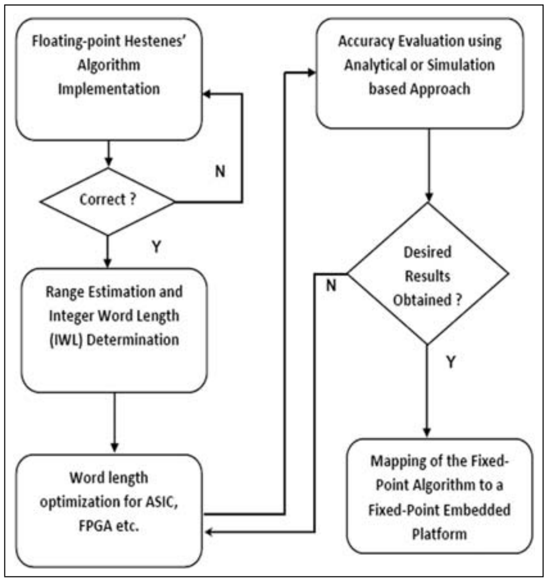
Floating-point Hestenes' Algorithm Implementation
Correct ?
N
Y
Range Estimation and Integer Word Length (IWL) Determination
Word length optimization for ASIC, FPGA etc.
Accuracy Evaluation using Analytical or Simulation based Approach
Desired Results Obtained ?
N
Y
Mapping of the Fixed-Point Algorithm to a Fixed-Point Embedded Platform

Fig. 2. Block diagram of floating-point to variable fixed-point conversion process

b) Range estimation: Range estimation is one of the important steps in fixed-point algorithm development [12]. Ranges of the variables are computed to find the optimum integer wordlength for fixed-point conversion. There are mainly two approaches for range estimation: analytical approach [14], [15] and statistical approach [17], [10] and [11]. Analytical approach does not require simulation and hence it is faster. On the other hand they involve more conservative wordlengths. Simulation based approaches do not require any model. A slightly modified algorithm is able to track and update the statistics of the variables. Operator overloading characteristics of C/C++ are used in this case. A new class has been defined to compute the current results and track the variables using linked list. When simulation is completed, statistics of the variables are used to compute the optimum integer wordlengths. After the range estimation is complete a search algorithm is applied to find out the optimum wordlength for less quantization error. A useful survey of such search algorithms are discussed in [20]. Floating-point to fixed-point conversion, used in this paper utilizes a simulation based statistical approach to find out the optimum integer wordlength. Since the wordlength is fixed in case of different DSP and ARM platforms, computed IWLs decide the Q formats of the variables. For a computed IWL=8, the FWL for a 32-bit system will be 23 (taking sign bit as an extra bit). Once the IWL and FWL (or Q$m.n$) are computed, the results are used to develop the bit true fixed-point format of the algorithm in SystemC or other platforms.

IV. Numerical Accuracy Evaluation of Fixed-point Hestenes' Algorithm

Accuracy of any fixed-point algorithm is important as it affects the selection of optimized wordlength. Acceptability of fixed-point algorithm is also governed by its accuracy and performance (latency, power consumption and area). There are two approaches to evaluate the accuracy i) simulation based and ii) analytical.

A. Fixed-point simulation based approach

Numerical accuracy and application performance of a fixed-point specification can be directly obtained from bit-true simulation. Large number of samples or input data is used to run the algorithm for number of times to obtain better accuracy. Total execution time for such fixed-point simulation is computed using (4) [19].

$$T_s = \tau N_{smp} N_{op} N_i \tag{4}$$

where, τ is simulation time for one sample or input data, N_{smp} is number of samples, N_{op} is number of operations in the algorithm description and N_i is number of times the accuracy is evaluated. In this work, simulation of fixed-point Hestenes' algorithm is performed in SystemC 2.2 and C/C++ platform using operator overloading characteristics of object oriented language. The simulation time is significantly larger in both the cases. Accuracy of fixed-point algorithm can be quantified in terms of signal-to-quantization-noise-ratio (SQNR) [13], number of accurate fractional bits, orthogonality and factorization errors [(5)-(8)].

$$\begin{aligned} SQNR(dB) &= 10\log_{10}\frac{E(\mid f_{float}\mid^2)}{MSE} \\ &= 10\log_{10}\frac{E(\mid f_{float}\mid^2)}{E(\mid f_{float}-f_{fixed}\mid^2)} \end{aligned} \tag{5}$$

where MSE is mean-square-error.

Number of accurate fractional bits is given by,

$$\begin{aligned} &\text{Number of accurate fractional bits} \\ &= -\log_2[\text{abs}(\max(f_{fixed}-f_{float}))] \end{aligned} \tag{6}$$

where $[\text{abs}(\max(f_{fixed}-f_{float}))]$ represents maximum absolute error between floating-point and fixed-point results. f_{fixed} and f_{float} are fixed and floating-point results obtained.

Orthogonality (O_{error}) and factorization errors (F_{error}) are also taken into consideration for accuracy evaluation of the fixed-point Hestenes' algorithm.

Orthogonality and factorization errors are given by,

$$O_{error} = \parallel I_n - U^{\mathrm{T}}U \parallel_2 \tag{7}$$

$$F_{error} = \parallel A - U\Sigma V^{\mathrm{T}} \parallel_2 \tag{8}$$

where, U is the orthogonal matrix, each column of which corresponds to singular vectors, A is the symmetric matrix and Σ is the singular value matrix.

B. Analytical Approach

While simulation based approaches lead to large execution time, analytical methods consume much less time. However, they are applicable to limited class of systems [18]. In this section an analytical approach is adopted to estimate the quantization error for computationally intensive parts involving square, square root and division operations in Hestenes' algorithm. The model is constructed with quantization noise source and its propagation through the arithmetic operations to the final output.

1) Quantization Noise Model:

a) Quantization noise source: Quantization noise arises when data are represented or stored using finite precision arithmetic. During this process excess bits are eliminated and the data is approximated using finite number of bits or wordlength. In an algorithm these errors propagate through fixed-point arithmetic operations and are accumulated at the output. Quantization noise is assumed to be uncorrelated with the signals and also uncorrelated among themselves. There are mainly two types of quantization errors truncation and rounding. In case of truncation all bits after b bits are discarded. Hence for a positive number the quantized value is always less than the actual value resulting in a negative error. Rounding is similar to truncation except the case that a 1 is added with the LSB to round up or down. Mean and variance of various quantization modes are presented in Table IV. Ranges of different relative errors during various quantization modes are shown in Table V [24], [25] and [26].

In Table IV and V q is quantization step and is equal to $q = 2^{-b}$, k is the number of bits eliminated from β-bits during quantization and b is the number of bits for fractional part.

TABLE IV
MEAN AND VARIANCES FOR QUANTIZATION MODES

Quantization modes	Conventional rounding	Convergent rounding	Truncation
Mean	$\frac{q}{2}(2^{-k})$	0	$\frac{q}{2}(1-2^{-k})$
Variance	$\frac{q^2}{12}(1-2^{-2k})$	$\frac{q^2}{12}(1+2^{-2k+1})$	$\frac{q^2}{12}(1-2^{-2k})$

TABLE V
RANGE OF RELATIVE ERRORS

Type of Quantization error	Number representation	Range of relative error
Truncation	Positive	$-(2^{-b}-2^{-\beta}) \leq \epsilon_t \leq 0$
	Sign magnitude	$0 \leq \epsilon_t \leq (2^{-b}-2^{-\beta})$
Rounding	All numbers	$-\frac{1}{2}(2^{-b}-2^{-\beta}) \leq \epsilon_r \leq \frac{1}{2}(2^{-b}-2^{-\beta})$

b) Quantization noise propagation: During a floating to fixed-point conversion process, depending on the FWL assigned to each variable some bits are eliminated due to rounding or truncation. This elimination of bits under various modes results in quantization noise. This noise propagates and produces unavoidable error in the output. Therefore, if a propagation model is developed for various arithmetic operations, it can serve as a basis for evaluation of fixed-point specification accuracy and this could further help in optimizing wordlengths for fixed-point architectures [19]. We consider an operation f with two inputs x and y and $z = f(x,y)$ as the output. The floating-point values of the inputs and output are x, y and z with their fixed-point values as $\hat{x}$, $\hat{y}$ and $\hat{z}$ after quantization. The quantization noises associated with inputs and output are $b_x = \hat{x}-x$, $b_y = \hat{y}-y$ and $b_z = \hat{z}-z$. The propagation model can be developed using Taylor series [19].

$$\begin{aligned}\hat{z} &= f(\hat{x},\hat{y}) \\ &= f(x,y) + (\hat{x}-x)\frac{\partial f(x,y)}{\partial x} + (\hat{y}-y)\frac{\partial f(x,y)}{\partial y} \\ &+\frac{1}{2}[(\hat{x}-x)^2\frac{\partial^2 f(x,y)}{\partial x^2} + 2(\hat{x}-x)(\hat{y}-y)\frac{\partial^2 f(x,y)}{\partial x \partial y} \\ &\quad +(\hat{y}-y)^2\frac{\partial^2 f(x,y)}{\partial y^2}]\end{aligned} \tag{9}$$

Computationally intensive operations

Square operation

The quantization noise propagation model for square operation using Taylor series is,

$$\begin{aligned}f(\hat{x}) &= f(x) + 2(\hat{x}-x)\frac{\partial f(x)}{\partial x} + \frac{1}{2}[2(\hat{x}-x)^2\frac{\partial^2 f(x)}{\partial x^2}] \\ &= f(x) + 2(\hat{x}-x)x + (\hat{x}-x)^2\end{aligned} \tag{10}$$

The propagation model for square can also be derived easily as,

$$\hat{z} = \hat{x}\hat{y} \Rightarrow \begin{cases} \hat{z} = (x+b_x)(x+b_x) \\ = x^2 + 2b_x x + b_x^2 \end{cases}$$

The fixed-point simulation results for square operation shown in Table VI can be validated as follows,

TABLE VI
QUANTIZATION ERROR FOR SQUARE OPERATION

Floating-point values	Fixed-point format	Values after quantization	Ranges
$z = x^2$ $x = 3.75$ $z = 14.0625$	$sc\_fixed < 8, 2, SC\_RND > x$ $sc\_fixed < 8, 4, SC\_RND > z$	$\hat{x} = -0.25$ $\hat{z} = 0.0625$	$-2 \leq x \leq 1.984375$ (x is rounded to -0.25)

TABLE VII
QUANTIZATION ERROR FOR SQUARE ROOT OPERATION

Floating-point values	Fixed-point format	Values after quantization	Ranges
$z = \sqrt[2]{x}$ $x = 3.7$ $z = 1.9235$	$sc\_fixed < 8, 3, SC\_RND > x$ $sc\_fixed < 8, 2, SC\_RND > z$	$\hat{x} = 3.6875$ $\hat{z} = 1.92028$	$-4 \leq x \leq 3.96875$ (so x is rounded to 3.6875)

$$\hat{z} = \hat{x}^2 \Rightarrow \begin{cases} \hat{z} = x^2 + 2b_x x + {b_x}^2 \\ = 14.0625 - 15 - 15 + 16 \\ = 0.0625 \end{cases}$$

Square root operation

The noise propagation model for square root operation is derived using binomial series as,

$$\hat{z} = \sqrt[2]{\hat{x}} \Rightarrow \begin{cases} \hat{z} = \sqrt{x+b_x} \\ = x^{\frac{1}{2}}(1+\frac{b_x}{x})^{\frac{1}{2}} \\ = x^{\frac{1}{2}} + \frac{b_x}{2x^{\frac{1}{2}}} - \frac{b_x^2}{8x^{\frac{3}{2}}} + \frac{b_x^3}{16x^{\frac{5}{2}}} - \ldots \end{cases}$$

The fixed-point simulation results shown in Table VII are validated as,

$$\hat{z} = \sqrt[2]{\hat{x}} \Rightarrow \begin{cases} \hat{z} = \sqrt{x+b_x} \\ = x^{\frac{1}{2}}(1+\frac{b_x}{x})^{\frac{1}{2}} \\ = (3.7)^{\frac{1}{2}}(1+\frac{3.6875-3.7}{3.7})^{\frac{1}{2}} \\ = 1.92028 \end{cases}$$

If we increase the terms in w $(-\frac{b_x^2}{8x^{\frac{3}{2}}} + \frac{b_x^3}{16x^{\frac{5}{2}}} - \ldots)$ in the above derivation, we observe that the terms do not affect the value ($\hat{z}$) as they are quite small in magnitude.

Division

The noise propagation model for division can be expressed using α_1, α_2 and w as,

$$\hat{z} = \frac{\hat{x}}{\hat{y}} \Rightarrow \begin{cases} \hat{z} = \frac{(x+b_x)}{(y+b_y)} \\ = \frac{y^2(x+b_x)}{y^2(y+b_y)} \\ = \frac{xy(b_y+y)}{y^2(y+b_y)} + \frac{b_x y(b_y+y)}{y^2(y+b_y)} - \frac{b_y x(b_y+y)}{y^2(y+b_y)} + \frac{xb_y^2 - b_x b_y y}{y^2(y+b_y)} \\ = \frac{x}{y} + \frac{1}{y}b_x - \frac{x}{y^2}b_y + \frac{xb_y^2 - b_x b_y y}{y^2(y+b_y)} \\ = f(x,y) + \alpha_1(\hat{x}-x) + \alpha_2(\hat{y}-y) + w \end{cases}$$

TABLE VIII
DIVISION

Floating-point values	Fixed-point format	Values after quantization	Ranges
$z = \frac{x}{y}$ $x = 2.5$ $y = 3.75$ $z = 0.6667$	$sc\_fixed < 8, 2, SC\_RND > x$ $sc\_fixed < 8, 2, SC\_RND > y$ $sc\_fixed < 8, 6, SC\_RND > z$	$\hat{x} = -1.5$ $\hat{y} = -0.25$ $\hat{z} = 6$	$-2 \leq x, y \leq 1.984375$ (so x is rounded to -1.5 and y to -0.25)

The above derivation can be simplified as,

$$\hat{z} = \frac{\hat{x}}{\hat{y}} \Rightarrow \begin{cases} \hat{z} = \frac{(x+b_x)}{(y+b_y)} \\ \hat{z} - \frac{x}{y} = \frac{(x+b_x)}{(y+b_y)} - \frac{x}{y} \\ \hat{z} = \frac{x}{y} + \frac{yb_x - xb_y}{y(y+b_y)} \\ = f(x,y) + u \end{cases}$$

The fixed-point simulation results for division shown in Table VIII can be validated as follows,

$$\hat{z} = \frac{\hat{x}}{\hat{y}} \Rightarrow \begin{cases} \hat{z} = f(x,y) + \alpha_1(\hat{x} - x) + \alpha_2(\hat{y} - y) + w \\ = 0.6667 - 1.0667 - 0.7111 + 5.6889 \\ = 6 \end{cases}$$

Example from Hestenes' algorithm

The above models are used to derive noise propagation model for a computationally intensive step of Hestenes' algorithm (Table I) given by,

$$\zeta = \frac{(\beta - \alpha)}{2\gamma} \tag{11}$$

The quantization noises corresponding to α, β and γ are b_α, b_β and b_γ respectively. The noise propagation model for (11) can be derived and validated (Table IX) as,

$$\hat{\zeta} \Rightarrow \begin{cases} = \frac{\hat{\beta} - \hat{\alpha}}{2\hat{\gamma}} \\ = \frac{1}{2}(\frac{\hat{\beta}}{\hat{\gamma}} - \frac{\hat{\alpha}}{\hat{\gamma}}) \\ = \frac{1}{2}[(\frac{\beta}{\gamma} + \frac{b_\beta}{\gamma} - \frac{\beta b_\gamma}{\gamma^2} + \frac{\beta {b_\gamma}^2 - b_\beta b_\gamma \gamma}{\gamma^2(\gamma + b_\gamma)}) - \\ (\frac{\alpha}{\gamma} + \frac{b_\alpha}{\gamma} - \frac{\alpha b_\gamma}{\gamma^2} + \frac{\alpha {b_\gamma}^2 - b_\alpha b_\gamma \gamma}{\gamma^2(\gamma + b_\gamma)})] \\ = \frac{1}{2}[(0.150 + 2.365 \times 10^{-5} + 1.868 \times 10^{-6} + \\ 3.179 \times 10^{-10}) - (10.1880 - 1.729 \times 10^{-5} + \\ 0.00012 + 1.365 \times 10^{-9})] \\ = -5.01905 \end{cases}$$

2) Output Noise Power and SQNR: During a fixed-point conversion process, the output error due to quantization is a random variable and hence treated as noise. This noise not only results from the input quantization, but also occurs due to cast operation. The output noise power has been analytically derived for one computationally expensive step in Hestenes algorithm (11). The quantization noise corresponding to output ($\hat{\zeta}$) is given by

$$b_\zeta = \hat{\zeta} - \zeta \tag{12}$$

From (5) it is clear that noise power corresponding to the output noise b_ζ is equal to its second order moment $(E({b_\zeta}^2))$. The second order moment is calculated from the mean and variance shown in Table IV. The output noise power obtained is 2.3449×10^{-8}

SQNR can be found out analytically using the output noise power expression.

$$SQNR_{ana} \Rightarrow \begin{cases} = 10\log_{10} \frac{E(|\zeta_{float}|^2)}{E(|\zeta_{float} - \zeta_{fixed}|^2)} \\ = 10\log_{10}(\frac{|-5.019014|^2}{2.3449 \times 10^{-8}}) \\ = 90.3112 \end{cases}$$

where $E(|\ \zeta_{float}\ |^2)$ is the signal power i.e. square of the floating-point value of ζ from Table IX and $E(|\ \zeta_{float} - \zeta_{fixed}\ |^2)$ is the noise power i.e. the second order moment of noise b_ζ $(E({b_\zeta}^2))$, calculated analytically. For both cases (fixed-point simulation and analytical) the signal power remains same, whereas, the noise power (P_n) in the first case is calculated directly from the square of the mismatch of ζ and $\hat{\zeta}$ (13), whereas in the second case the noise power $((E({b_\zeta}^2)))$ is calculated using Table IV. The fixed-point simulation result of ζ and $\hat{\zeta}$ are shown in Table IX.

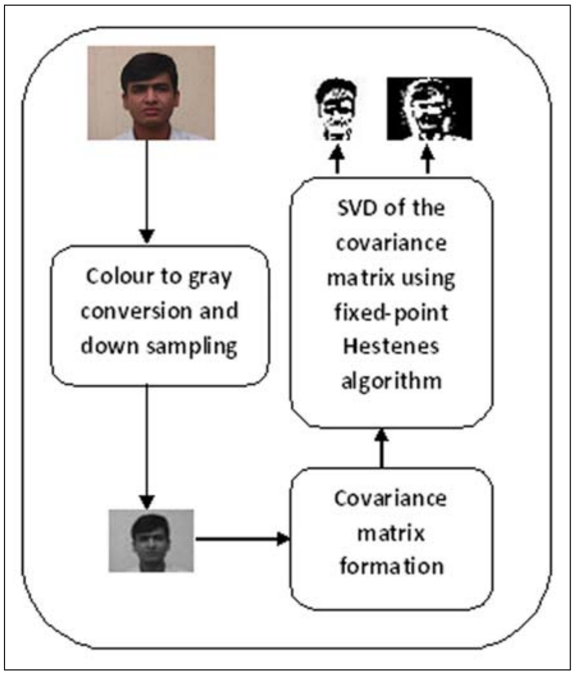
Colour to gray conversion and down sampling
Covariance matrix formation
SVD of the covariance matrix using fixed-point Hestenes algorithm

Fig. 3. Block diagram showing the process of eigen face computation

$$P_n = (\zeta - \hat{\zeta})^2 \tag{13}$$

where ζ is the floating-point value. Therefore, SQNR under fixed-point simulation is,

$$SQNR_{sim} \Rightarrow \begin{cases} = 10\log10(\frac{E(|\zeta|^2)}{P_n}) \\ = 102.8863 \end{cases}$$

TABLE IX
EXAMPLE FROM HESTENES' ALGORITHM

Floating-point values	Fixed-point format	Values after quantization	Ranges
$\alpha = 24158.666016$	$sc\_fixed < 32, 29, SC\_RND > \alpha$	$\hat{\alpha} = 24158.625$	$-268435456 \leq \alpha \leq 268435455.99999$
$\beta = 355.693909$	$sc\_fixed < 32, 29, SC\_RND > \beta$	$\hat{\beta} = 355.75$	$-268435456 \leq \beta \leq 268435455.99999$
$\gamma = 2371.279541$	$sc\_fixed < 32, 30, SC\_RND > \gamma$	$\hat{\gamma} = 2371.25$	$-536870912 \leq \gamma \leq 536870911.99999$
$\zeta = -5.019014$	$sc\_fixed < 32, 19, SC\_RND > \zeta$	$\hat{\zeta} = -5.01905$	$-262144 \leq \zeta \leq 262143.99999$

The relative error of estimation for SQNR is given as

$$E_r = \left| \frac{SQNR_{ana} - SQNR_{sim}}{SQNR_{sim}} \right| = 0.1222 \qquad (14)$$

V. Fixed-point Hestenes' SVD algorithm for computation of eigen faces

The fixed-point Hestenes' SVD algorithm could be used for signal and image processing applications in embedded platforms. In a case study we have used the algorithm to compute eigen faces from an Indian Face Database [31]. Five images each of size 640×480 have been collected separately from one male and one female Indian Face Database. In each case images have been down sampled (50×38) and converted to gray scale images. These images are used to construct the covariance matrix. Fixed-point Hestenes' algorithm decomposes the covariance matrix into singular triplets (Fig. 3). Singular vectors thus formed are the eigen faces (Fig. 8 and 10) and could be used in pattern recognition, image compression and other image processing applications. Errors associated with these eigen faces are also shown in Fig. 9 and 11. Results of double precision floating-point algorithm has been considered as the reference.

VI. Generalization of Fixed-point Tensor SVD

Using the inherent elegance of matrix formulations, SVD has played an important role in advances being made in the area of digital signal processing, systems and control. An increasing variety of signal and image processing issues involve manipulation of data using higher-order vectors, tensors, multidimensional arrays, etc. With immense development in the fields of higher-order statistics for multivariate stochastic processes, higher-order tensors are gaining more importance [27]. One of the solution for tensor SVD has been discussed in [28], where the 3-d array is transformed to 2-d and then SVD is applied to this 2-d array. A hybrid scheme by combining Genetic algorithm and Nelder-Mead method has been proposed in [30] for SVD of multidimensional arrays (tensors). The concept of positive singular value decomposition has been discussed in [29]. Fixed-point development of tensor SVD algorithm would help in real-time applications using embedded platforms like FPGAs, ARM, etc. The fixed-point Hestenes' algorithm in constant and variable static formats can be used for solving tensor SVD as discussed in [28] (3-d array is transformed to 2-d and then SVD is applied to the 2-d array).

VII. Results and Discussion

The advantage of floating-point over fixed-point is its large dynamic range and constant relative accuracy. However, the precision is better in fixed-point format. Fixed-point algorithm does not require extra hardware or power and hence it is cheaper to implement. Fixed-point processing is faster as it involves integer arithmetic. However, limited wordlength and dynamic range affects the accuracy.

Accuracy of fixed-point algorithm has been evaluated in terms of number of accurate fractional bits, SQNR, orthogonality and factorization errors. Fig. 4 shows the variation of accurate fractional bits (6) for a 5×5 matrix with different condition numbers. As mentioned earlier, the constant fixed-point format of the algorithm exhibits better accuracy due to close dynamic ranges of the variables. However, the variable fixed-point format implemented in SystemC shows better SQNR (5) trend compared to others proving overall efficiency [Fig. 5]. Orthogonality and factorization errors [(7), (8)] are also found to be negligible in such an implementation [Fig. 6-7]. Thus it is clear that the results obtained from SystemC based implementation of variable fixed-point Hestenes' algorithm approaches the results of floating-point version. Constant fixed-point SystemC implementation is also in close vicinity with the accuracy of variable format implementation. In the case study discussed in section V, covariance matrix is formed from a set of five images of size 50×38, down sampled from a set of color images of size 640×480 and finally converted to gray scale images with pixel values between 0–255. Fixed-point SVD algorithm decomposes this covariance matrix into singular triplets. The singular vectors thus obtained are the eigen faces (Fig. 8 and 10). The fixed-point results with 32 and 64-bit show the minimum error trend (Fig. 9 and 11). Fifth singular values as well as vectors are tiny and hence the corresponding eigen faces appear to be black compared to others.

VIII. Conclusion

This paper discusses the development of fixed-point Hestenes' algorithm in different formats and their accuracy evaluation using bit true simulation and analytical approach. Fixed-point Hestenes algorithm has also been applied in an image processing algorithm to compute eigen faces. It is observed that constant fixed-point format algorithm performs better because of close dynamic ranges of the variables. However variable fixed-point format implementation exhibits an overall better performance. Noise propagation model for square, square root, division and a computationally intensive part of Hestenes' algorithm are derived and validated. Accuracy is evaluated based on number of accurate fractional

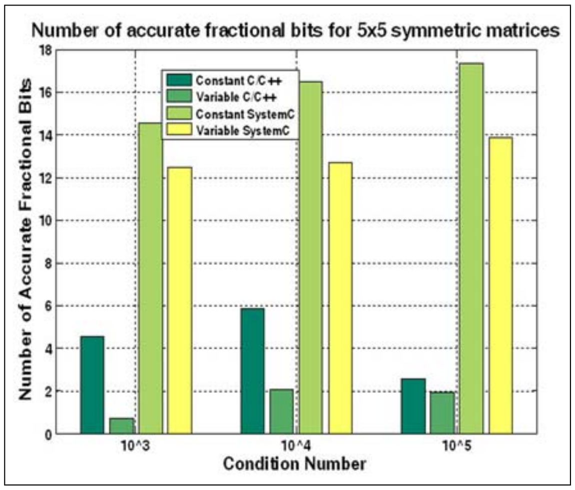

Fig. 4. Variation of number of accurate fractional bits with condition numbers

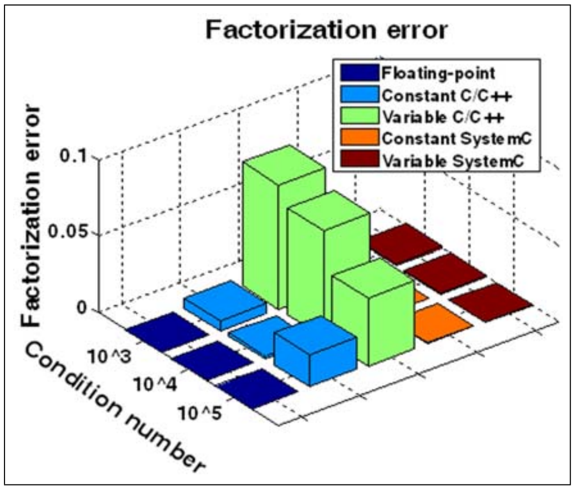

Fig. 7. Factorization error in floating and fixed-point formats

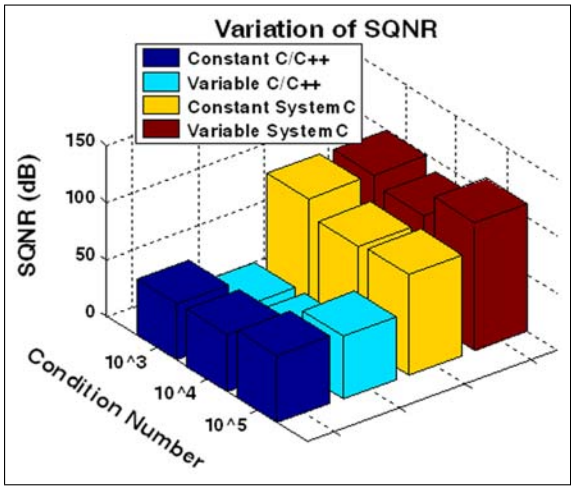

Fig. 5. Variation of SQNR with condition numbers

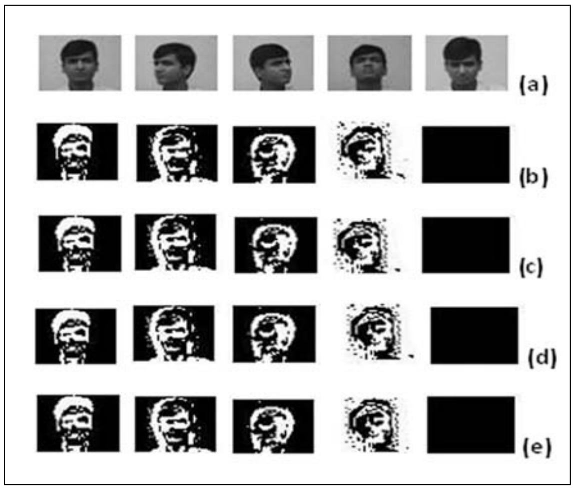

Fig. 8. (a) Male images and eigen faces using (b) Double precision floating-point (c) 24-bit, (d) 32-bit and (e) 64-bit fixed-point

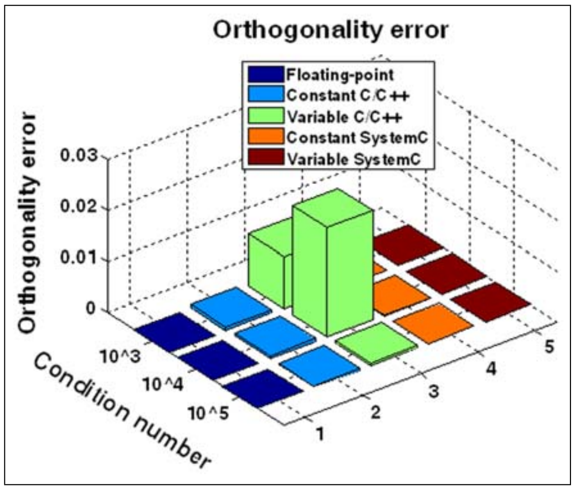

Fig. 6. Orthogonality error in floating and fixed-point formats

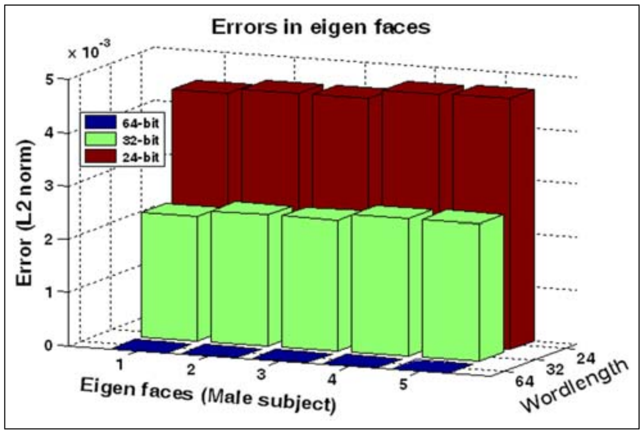

Fig. 9. Error in male eigen faces

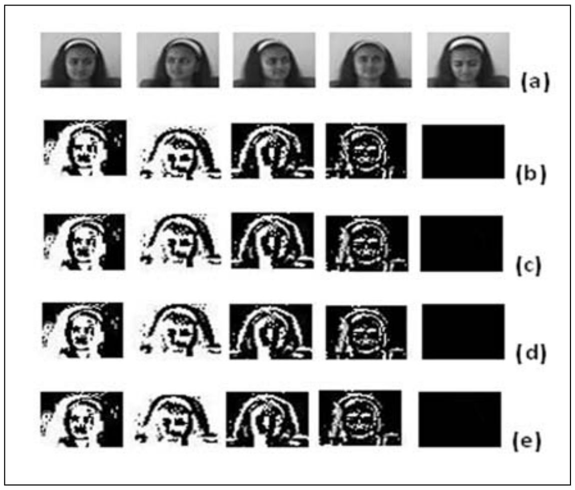

Fig. 10. (a) Female images and eigen faces using (b) Double precision floating-point (c) 24-bit, (d) 32-bit and (e) 64-bit fixed-point

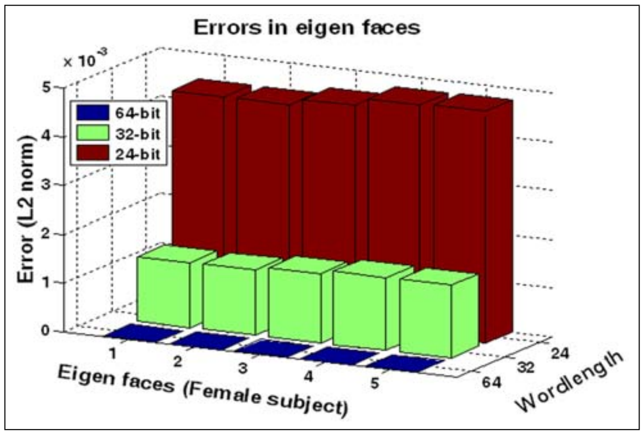

Fig. 11. Error in female eigen faces

bits, SQNR, orthogonality and factorization errors. Eigen face computation uses variable fixed-point SVD algorithm. The error trend shows that 32-bit and 64-bit wordlength are better for this image processing application. Error using 24-bit wordlength is also under considerable limit.

It is evident that this algorithm can be implemented in fixed-point architectures for embedded applications.

REFERENCES

[1] S. Udomhunsakul, "Noise Reduction using Adaptive Singular Value Decomposition," *International Journal of Circuits, Systems and Signal Processing, NAUN*, Vol. 7, Issue 2, 2013, pp. 91–100.

[2] M. Turk and A. Pentland, "Eigenfaces for Recognition", *Journal of Cognitive Neuroscience*, vol. 3, no. 1, 1991.

[3] G. Golub and C. F. V. Loan, "Matrix Computations", *The Johns Hopkins University Press*, 3rd ed. Baltimore, USA, 1996.

[4] B. N. Datta, "Numerical Linear Algebra and Applications", *SIAM*, 2010.

[5] T. Pradhan, A. Routray, and B. Kabi, "Comparative Evaluation of Symmetric SVD Algorithms for Real-Time Face and Eye Tracking," *Matrix Information Geometry, Springer Berlin Heidelberg*, 2013. pp 323–340.

[6] J. W. Demmel, "Applied Numerical Linear Algebra", *SIAM*, Philadelphia, 1997.

[7] M. R. Hestenes, "Inversion of Matrices by Biorthogonalization and Related Results", *SIAM*, vol. 6, no. 1, Mar. 1958, pp. 51-90.

[8] G. Svensson, "A Block-Hestenes Method for the SVD", hem.passagen.se/-gohel/num/hest.ps, Sept. 1989.

[9] P. M. Szecówka and P. Malinowski, "CORDIC and SVD Implementation in Digital Hardware", *Proc. Int. Conf. Mixed Design of Integrated Circuits and Systems (MIXDES)*, 2010, pp. 237242.

[10] S. Kim and W. Sung, "Fixed-Point simulation Utility for C and C++ Based Digital Signal Processing Programs", *Proc. Twenty-Eighth Annual Asilomar Conference on Signals, Systems, and Computers*, pp. 162-166, Oct. 1994.

[11] S. Kim, K. Kum, and W. Sung, "Fixed-Point Optimization Utility for C and C++ Based Digital Signal Processing Programs", *IEEE Trans. Circuits Syst. II*, vol. 45, pp. 14551464, Nov. 1998.

[12] K. Kum, J. Kang and W. Sung, "AUTOSCALER For C: An Optimizing Floating-Point to Integer C Program Converter For Fixed-Point Digital Signal Processors", *IEEE Trans. Circuits Syst. II*, vol. 47, pp. 840848, Sept. 2000.

[13] S. Kim and W. Sung, "A Floating-point to Fixed-point Assembly Program Translator for the TMS 320C25", *IEEE Trans. Circuits Syst. II*, vol. 41, pp. 730739, Nov. 1994

[14] H. Keding, M. Willems, M. Coors, and H. Meyr, "FRIDGE: A fixed-point design and simulation environment", in *Proc. Design Automation and Test in Europe*, 1998, pp. 429-435.

[15] M. Coors, H. Keding, O. Luthje and H. Meyr, "Design and DSP implementation of fixed-point systems", *EURASIP J. App. Signal Process.*, 2002.

[16] S. A. Edwards, "Using Program Specialization to Speed SystemC Fixed-Point Simulation", *Proc. of the ACM SIGPLAN Symposium on Part. Evaluation and Semantics-Based Program Manipulation*, 2006, pp. 21-28.

[17] Z. Nikolić, H. T. Nguyen, G. Frantz, "Design and Implementation of Numerical Linear Algebra Algorithms on Fixed Point DSPs", *EURASIP J. Adv. Signal Process.*, 2007.

[18] D. Menard, R. Rocher and O. Sentieys, "Analytical Fixed-Point Accuracy Evaluation in Linear Time-Invariant Systems", *IEEE Trans. Circuits Syst.I*, vol. 55, no. 10, pp. 31973208, Nov. 2008.

[19] D. Menard, P. Scalart and O. Sentieys, "Analytical approach for Numerical accuracy estimation of Fixed-point systems based on smooth operations", *IEEE Trans. Circuits Syst.I*, vol. 59, Nov. 2012.

[20] K. Han and B. L. Evans, "Transforming floating-point algorithms to fixed-point implementations", *VDM Verlag*, Germany, 2009.

[21] IEEE STD 1666-2005 IEEE Standard SystemC Language Reference Manual, http://standards.ieee.org/getieee/1666/download/1666-2005.pdf.

[22] B. W. Bomar, "Finite Wordlength Effects," in *The Digital Signal Processing Handbook*, V. K. Madisetti and D. B. Williams, Eds. CRC Press, 1998.

[23] J. Janhunen, T. Pitkanen, Olli Silven and M. Juntti, "Fixed- and Floating-point processor comparison for MIMO-OFDM detector", *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 8, pp. 15881598, Dec. 2011.

[24] A. V. Oppenheim and R. W. Schafer, "Digital signal processing", *New Jersey: Prentice-Hall*, 1975.

[25] S. K. Mitra, "Digital Signal Processing", *McGraw-Hill*, 3rd ed. New York, USA, 2005.

[26] M. Abbas, O. Gustafsson and H. Johansson, "On the Fixed-point implementation of fractional delay filters on the Farrow structure", *IEEE Trans. Circuits Syst.I*, vol. 60, 2013.

[27] L. D. Lathauwer, B. D. Moor, J. Vandewalle, "A Multilinear Singular Value Decomposition", *SIAM Journal on Matrix Analysis and Applications archive*, vol. 21, no. 4, pp. 1253 - 1278, Mar. May 2000.

[28] N. E. Mastorakis, "Singular Value Decomposition in Multidimensional Arrays", *Int. J. of Syst. Sci.*, vol.27, no.7, pp.647-650, July 1996.

[29] N. E. Mastorakis, "Positive Singular value Decomposition", *Recent Advances in Signal Processing and Communications (dedicated to the father of Fuzzy Logic, L. Zadeh)*, WSEAS-Press, pp.7-17, 1999.

[30] N. E. Mastorakis, "The Singular Value Decomposition (SVD) in Tensors (Multidimensional Arrays) as an Optimization Problem. Solution via Genetic Algorithms and method of Nelder-Mead", *Proceedings of the 6th WSEAS Int. Conf. on Systems Theory and Scientific Computation*, pp.7-13, August 2006.

[31] V. Jain, A. Mukherjee, The Indian Face Database, http://vis-www.cs.umass.edu/~vidit/IndianFaceDatabase/ , 2002.

[32] A. Saeed, M. Elbably, G. Abdelfadeel, and M. I. Eladawy. "Efficient FPGA implementation of FFT/IFFT Processor," *International Journal of Circuits, Systems and Signal Processing, NAUN*, Vol. 3, Issue 3, 2009, pp. 103–110.

[33] K. Ramirez-Gutierrez, D. Cruz-Perez, J. Olivares-Mercado, M. Nakano-Miyatake, H. Perez-Meana, "A Face Recognition Algorithm using

Eigenphases and Histogram Equalization," *International Journal of Computers, NAUN*, vol.5, no.1, pp 34–41, 2011.

[34] G. Aguilar-Torres, K. Toscano-Medina, G. Sanchez-Perez, M.Nakano-Miyatake, H. Perez-Meana, "Eigenface -Gabor algorithm for feature extraction in face recognition," *International Journal of Computers, NAUN*, Vol. 3, pp. 20–30, Jan. 2009.

[35] Y. Takizawa, S. Yatono and A. Fukasawa, "High performance digital signal processing system for wideband mobile communications," *International Journal of Mathematics and Computers in Simulation, NAUN*, Vol. 1, pp 146–149.