Received December 23, 2019, accepted January 2, 2020, date of publication January 13, 2020, date of current version January 21, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2966044

FPGA Implementation of $L_{1/2}$ Sparsity Constrained Nonnegative Matrix Factorization Algorithm for Remotely Sensed Hyperspectral Image Analysis

MOSTAFA GUDA, SAFA GASSER, MOHAMED S. EL-MAHALLAWY, (Member, IEEE), AND KHALED SHEHATA

Department of Electronics and Communication Engineering, Arab Academy for Science, Technology and Maritime Transport, Cairo 11799, Egypt

Corresponding author: Mostafa Guda (mostafa.guda@student.aast.edu)

ABSTRACT Remotely sensed hyperspectral images provide data of the earth's surface components. The data provided is collected through airborne devices such as satellites with the capability to collect large amounts of data to be sent to ground stations for processing. The main disadvantage of this scenario is the limited bandwidth connection between the airborne devices and the ground station on Earth which affects the information sending and real time processing. A possible solution is to include an on-board data processor. Field-Programmable Gate Arrays (FPGAs) are excellent target platform that allows the design reconfigurability, powerful computing and high performance levels. One of the most commonly used techniques in hyperspectral data analysis is linear spectral unmixing. In the last decade, $L_{1/2}$ sparsity constrained Nonnegative Matrix Factorization (NMF), a linear spectral unmixing algorithm, and its extensions have been heavily studied to unmix the hyperspectral images and recover their material spectra. $L_{1/2}$ regularizer is proven to have much better results in terms of sparsity and accuracy than other regularizers yet, to the best of our knowledge, has not been implemented. In this paper, we present an FPGA design for the $L_{1/2}$ sparsity constrained NMF ($L_{1/2}$-NMF) algorithm. The proposed design is tested on both synthetic and real data sets and implemented on Altera Family FPGAs. Implementation results show that the proposed design successfully unmixes the data with maximum frequency of 52.6 MHz and a speedup factor of 3.9 for the synthetic data set and a frequency of 104.32 MHz and a speedup factor of 1.14 for the real data set. The implementation results are compared to the simulation results and ground truth signatures using Spectral Angular Distance (SAD) measure. Calculations show that the implementation results have comparable SAD values to the simulation results.

INDEX TERMS Field-programmable gate arrays (FPGAs), hyperspectral unmixing, $L_{1/2}$ nonnegative matrix factorization algorithm (NMF).

I. INTRODUCTION

The launch of earth observation satellites has enhanced the field of remote sensing considerably in the past few years. Remotely sensed images cover many applications such as mineral exploration, military surveillance and environmental monitoring [1]. A common problem that surfaces is the existence of mixed pixels in the captured image. This problem occurs due to the low spatial resolution of the sensor and the captured surface variability when capturing a hyperspectral image [2]. This problem is handled via spectral unmixing algorithms. The spectral unmixing algorithms aim to decompose a mixed pixel into a collection of material spectra named end-members and each end-member's contribution in the pixel which is named the fractional abundance [3]. The idea of spectral unmixing is explained more in figure 1 [4].

Mainly, two types of spectral mixture models are used to describe the mixture in a pixel: linear and non linear mixing models [5]. In the literature, most of the existing approaches that solve the unmixing problem are based on the linear mixing model [4]. The linear mixing model based approaches are classified into geometrical and statistical approaches [6]. The geometrical approaches are based on the fact that the pixel observations of a hyperspectral image are confined within

The associate editor coordinating the review of this manuscript and approving it for publication was Xian Sun.

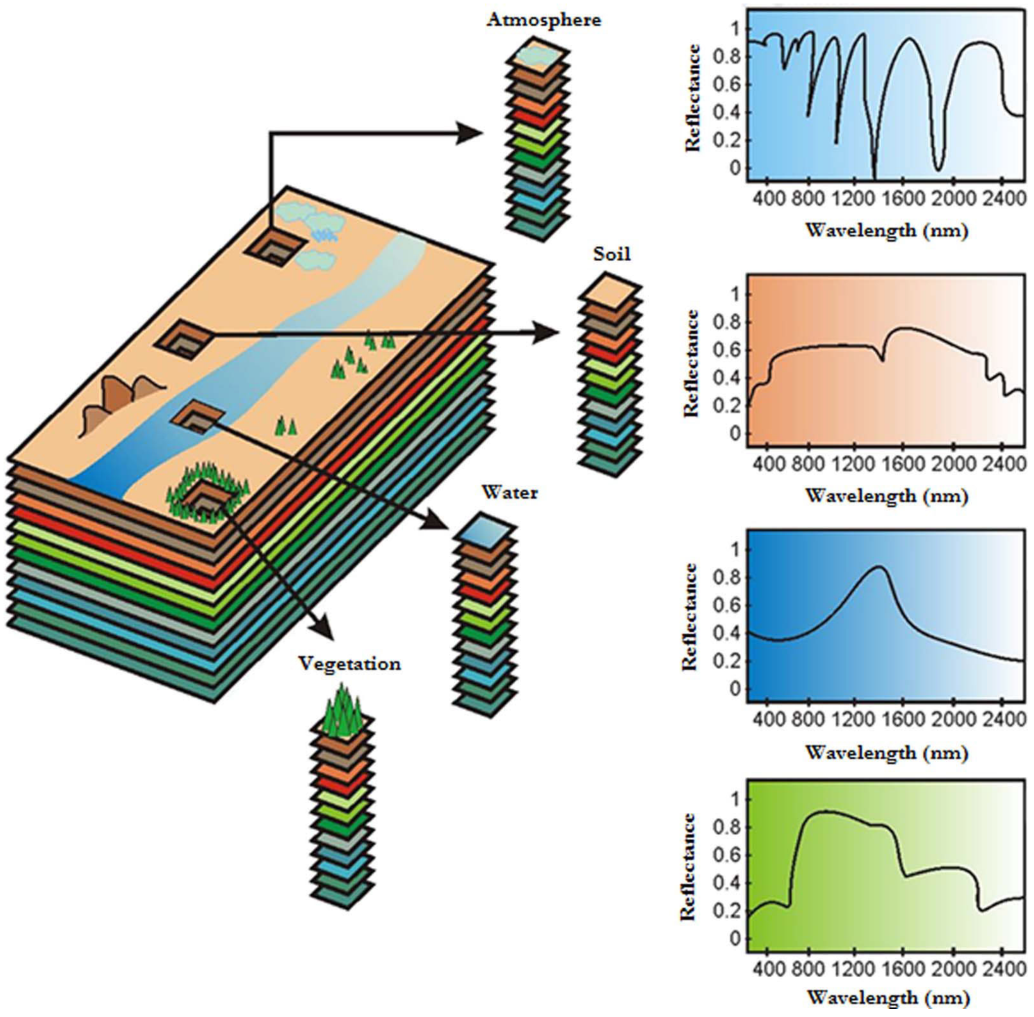


FIGURE 1. Explanation of spectral unmixing concept [4].

a simplex and its vertices represent the end-members [7]. Finding the simplex vertices is the solution to the unmixing problem.

In [8], the authors introduce the Pixel Purity Index (PPI) algorithm which aim to find the purest pixel through repeated projection onto a random unit vector and finding the end-members through automatic unmixing [9]. In [10], the author introduces the N-FINDR algorithm which targets finding the maximum volume simplex inscribed within the data set through non-linear inversion. In [11], the authors propose Automated Morphological End-member Extraction (AMEE) which tries to find the pure pixels in the data through combining the spatial and spectral information of the data. The authors then use mathematical morphology, a technique applied to the spectral domain while reserving the spatial characteristics of the data. The authors provide Spectral Angular Distance (SAD) results that are comparable to the SAD values of the PPI and N-FINDR algorithm. In [12], the authors introduce the Vertex Component Analysis (VCA) algorithm. VCA algorithm, like other geometrical based approaches, assumes the presence of pure pixels in the data and repeatedly projects the data onto a a subspace orthogonal to the subspace spanned by the already determined end-members. The algorithm stops when all end-members are found. The VCA algorithm provides results better than PPI algorithm and better than or similar to N-FINDR algorithm. VCA algorithm has less complexity than PPI and N-FINDR algorithm. In [13], the authors introduce the Orthogonal Bases Algorithm (OBA) which utilize the same concepts of the N-FINDR algorithm while replacing the volume matrix calculation with Gram-Schmidt iterative orthogonalization. The authors provide SAD results that are comparable to PPI, N-FINDR and VCA algorithms. Despite providing good results in extracting end-members from the hyperspectral image, the geometrical based approaches depend on the assumption of having a pure pixel in the data. This assumption can not always be true due to the low resolution of the data and the highly mixed featured of the target image [1].

The statistical based approaches overcome the weakness of the geometrical based approaches. The statistical based approaches do not assume the presence of a pure pixel of each end-member in the data. The statistical based algorithms utilize the statistical features of the data. One of the most common algorithms based on the statistical approach is Iterated Constrained End-member (ICE) [14]. The authors in [14] handle the unmixing problem as an optimization problem and propose an objective function to be minimized. Most recently, the authors in [15] proposed an unmixing method based on Gaussian Mixture Model (GMM), super-pixel segmentation (SS) and low-rank representation (LRR). The authors provide SAD results comparable to SAD results in [16]–[18]. In [19], the authors provide a comparison of VCA, PPI and Optical Real-time Adaptive Spectral Identification System (ORASIS) algorithm [20] for mineralogical unmixing of hyperspectral data. ORASIS algorithm is a collection of stepwise algorithms working together to produce a set of end-members that may not be included in the data set. The four stages of ORASIS algorithm are prescreener, basis selection, end-member selection and unmixing. The authors use a highly mixed synthetic data with maximum purity of 0.48%, AVIRIS dataset of Cuprite, Nevada and Hyperion dataste of the Dost-Bayli area, Ardabil. The authors show that ORASIS outperforms both VCA and PPI algorithms.

One of the most widely used statistical approaches to solve the unmixing problem is Non-negative Matrix Factorization (NMF) [21], [22]. NMF approximates the data as the multiplication between two non-negative matrices. The first matrix contains the end-members and the second matrix contains the fractional abundances [23]. NMF algorithm results lack high accuracy because of the non-convexity of its objective function. Researchers in the literature add regularizing terms to the objective function to force certain constraints and improve the accuracy of the NMF algorithm. In [1], the authors introduce the Minimum Volume Constrained NMF algorithm (MVC-NMF) which exploits the fact that the simplex volume determined by the end-members is the minimum among all possible simplexes that circumscribe the data scatter space. The authors compare their SAD results to VCA algorithm and prove that MVC-NMF algorithm outperforms VCA algorithm. In [24], the authors propose a modified version of the MVC-NMF algorithm where they apply the first stage of (ORASIS) algorithm in order to reduce the large data set size. This modification overcomes the shortcoming of MVC-NMF in dealing with large data. The authors test the modified technique on both a set of noisy synthetic data and Hyperion image of Dost-Bayli located in the Ardabil province in northwestern Iran. In [25], the authors introduce Constrained NMF (CNMF). The authors add a regularization term to enforce smoothness constraint over the end-members matrix. The authors compare CNMF SAD results to that of NMF and CNMF has better SAD values. In [26], the authors introduce Piecewise Smooth NMF with Sparseness Constraint (PSNMFSC). Piecewise smoothness corresponds to smooth variation of the data while sparseness is a property of hyperspectral data where each pixel is a mixture of some end-members of the total number of end-members in the scene. The authors compare the SAD results of PSNMFSC to VCA and MVC-NMF algorithms. The experiments show that PSNMFSC has the best results compared to VCA and MVC-NMF. In [6], the authors introduce Abundance Separation and Smoothness Constrained NMF (ASSNMF). The abundance separation constraint minimizes the mutual information between the abundance distributions of different end-members while the abundance smoothness constraint is added based on the fact that minimum abrupt changes happen and variations occur within the image. The authors compare ASSNMF algorithm with four other algorithms namely CNMF, PSNMFSC, MVCNMF and VCA. The authors find ASSNMF to have superior SAD results. In [27], the authors introduce double abundance characteristics constrained NMF (DAC^2NMF). The authors measure the smoothness levels of each pixel pair according to the similarities between them by taking advantage of the spectral information of the data. The authors also avoid incorrect smoothness constraints by assigning zero smoothness level to the pixels not similar to the observed pixel. The authors also add a separation constrain to prevent over-smooth results. The authors compare DAC^2NMF algorithm to MVC-NMF, ASSNMF, $L_{1/2}$-NMF [3] and GLNMF [28]. The SAD results of DAC^2NMF algorithm are comparable to the four aforementioned algorithms.

Many authors focus on the sparsity based constraints and its extensions. Sparseness refers to using a few end-members out of the whole present number of end-members in the scene to represent the pixel. In [29], the author add a sparseness constraint based on L_1 norm to utilize the sparse distribution of data. In [30], the authors propose a new method to calculate the sparseness measure and integrate it in the NMF cost function named NMF-SMC. The authors compare their SAD results to those of VCA and MVC-NMF. The results show that NMF-SMC outperforms both VCA and MVC-NMF algorithms. In [31], the authors introduce NMF with Data-Guided Constraint (DGC-NMF). The authors propose estimating the abundance map of the data by using unconstrained NMF as a first step. The authors use both $L_{1/2}$ and L_2 regularizers to control the sparsity over the abundances. The authors use the $L_{1/2}$ regularizer to control sparsity over pixels with high sparsity levels. The authors use the L_2 regularizer to control sparsity over pixels with low sparsity levels. The authors compare the DGC-NMF algorithm to VCA, unconstrained NMF, L_2-NMF and $L_{1/2}$-NMF algorithms. The authors show that DGC-NMF has comparable SAD results to the previously mentioned algorithms. In [32], the authors introduce a linear hyperspectral unmixing method based on L_1-L_2 sparsity and Total Variation (TV) regularization (L_1-L_2SUnSAL-TV). This algorithm forces strong sparsity through calculating the difference between the first and second norms. The algorithm also incorporates spatial correlation information by adding a TV constraint to force spatial smoothness. The authors compare (L_1-L_2SUnSAL-TV) to six other methods among them are Sparse Unmixing model via variable Splitting and Augmented Lagrangian (SUnSAL), collaborative

SUnSAL (CLSUnSAL) and Sparse unmixing using Spectral a Priori Information (SUnSPI). The authors show that (L_1-L_2SUnSAL-TV) outperforms the other algorithms in comparison. In [33], the authors introduce bilateral filter regularized L_2 sparse NMF (BF-L_2 SNMF).The authors add a L_2 regularizer to enforce sparseness. The authors also add a bilateral filter regularizer to utilize the correlation information between the abundance vectors. The authors show that BF-L_2 SNMF SAD results are comparable and sometimes better than VCA, $L_{1/2}$-NMF and GLNMF algorithms. $L_{1/2}$ regularizer is better in sparsity representation than L_1 and L_2 regularizers [3], [34]. In [3], the authors introduce the $L_{1/2}$-NMF algorithm. The authors add an $L_{1/2}$ regularizer to the objective function to enforce sparsity over the abundances. The authors compare the $L_{1/2}$-NMF algorithm SAD results to VCA, MVC-NMF, L_1-NMF and PSNMFSC algorithms. The authors show that $L_{1/2}$-NMF outperforms the algorithms in comparison. In [28], the authors introduce Graph-regularized $L_{1/2}$-NMF algorithm (GLNMF). The authors add graph regularization constrain where a nearest neighbour graph is built to incorporate the internal structure information of the data. The authors compare the SAD results of GLNMF algorithm to VCA, MVC-NMF and $L_{1/2}$-NMF. The authors show that GLNMF results are superior and more noise tolerant. In [2], the authors introduce iterative half-thresholding $L_{1/2}$-NMF (HT$L_{1/2}$-NMF). The HT$L_{1/2}$-NMF takes advantage of the a priori known sparseness information and integrate it in optimization of the algorithm to adaptively adjust the regularization parameter through while the algorithm is iterating. The authors compare the SAD results of HT$L_{1/2}$-NMF to VCA, L_1-NMF and $L_{1/2}$-NMF algorithms. The authors show that HT$L_{1/2}$-NMF provides more accurate and much sparser results.

A problem that arises with hyperspectral unmixing is the limited bandwidth of the channel between the satellite and the earth station [35]. One solution to this problem is the addition of an on-board processing resource to reduce the size of the data transmitted to the Earth station [36]. An advantage of on-board processing is instead of sending the whole hyperspectral image, the resulting features (end-members and abundances), which are much smaller in size, are transmitted to the ground station. In recent years, many researchers contributed to the implementation of the hyperspectral unmixing algorithms on different target platforms. In [37], the authors implement the VCA algorithm on NVIDIA Fermi GPU. The authors state that the design is faster than the MATLAB and C versions of the algorithm. GPUs have a significant disadvantage in on-board processing as they are limited in temperature, memory and energy consumption [37]. In [38] the authors implement a modified version of the VCA algorithm (MVCA) on Nvidia GPU using Open Computing Language (OpenCL). The proposed MVCA implementation has a speedup factor of 115 when compared to a sequential implementation of the VCA algorithm. In [39], the authors use a systolic array-based FPGA implementation of the PPI algorithm on a Xilinx Virtex-II XC2V6000-6 FPGA. In [40], the authors implement the VCA algorithm on a low cost Xilinx Zynq board with a Zynq-7020 System on Chip (SoC) FPGA based on the Artix-7 FPGA programmable logic. The authors in [36] implement the NFINDR algorithm on Virtex-4 XC4VFX60 FPGA. The authors state that the implementation of the NFINDR algorithm is superior in computational time compared to an equivalent C software version. In [41] the authors implement the Hysime algorithm used for estimating the number of end-members in the hyperspectral image [42]. The Hysime algorithm is implemented on a Virtex-7 XC7VX690T FPGA. In [43] the authors implement Fast Automatic Target Detection algorithm (Fast-ATGP) on a Xilinx Virtex-7 XC7VX690T FPGA.

Field-Programmable Gate Arrays (FPGAs) are the most common and most suitable resource for on-board processing of hyperspectral unmixing algorithms [41]. FPGAs have smaller size and weight while having also lower power consumption compared to other platforms like Graphical Processing Units (GPUs) and multicore processors [44], [45]. FPGAs are also suitable for being used in satellite payload as we currently have FPGAs with increased resistance to space ionizing radiations [46]. FPGAs also provide the ability of changing their functionality through reconfiguration [41], [47].

Despite outperforming the PPI, NFINDR and VCA algorithms [3], $L_{1/2}$ NMF algorithm is not yet implemented on any target platform. This paper presents, to the best of our knowledge, the first FPGA based architecture $L_{1/2}$ NMF algorithm. The paper provides real-time synthesis results, comparison with MATLAB simulation results and ground truth spectra.

The rest of the paper is organized as follows: in section II we discuss the mathematical approaches of the linear mixing model (LMM) and the $L_{1/2}$-NMF algorithm. In section III we introduce the proposed FPGA implementation of the $L_{1/2}$-NMF algorithm. In section V we discuss the implementation and simulation results. Finally we conclude our paper in section VI.

II. MIXING MODELS AND $L_{1/2}$-NMF ALGORITHM

A. MIXING MODELS

Mixing models are classified to linear and non linear models [5]. The linear mixing model (LMM) is valid when end-members follow a discrete distribution where each end-member does not interfere with other end-members [48]. Under the linear mixing model, it is assumed that the end-members are mixed linearly as stated in equation 1

$$X = AS + E \tag{1}$$

where $X \in R^{L\times N}$ is the hyperspectral image, L is the number of bands and N is the number of pixels. $A \in R^{L\times K}$ is the end-members signature matrix and K is the number of estimated end-members in the image. Each column of the matrix A represents an end-member spectrum. $S \in R^{K\times N}$ is the fractional abundances matrix. On the other hand, the non linear mixing models add a non linear part to the linear mixing model in equation 1 as follows;

$$X = AS + \mu(A, S, b) + E \tag{2}$$

The term $\mu(.)$ defines an additive non linear term that depends on the end-member matrix A, the abundance coefficients in S and additional non linearity coefficients b which adjusts the amount of non linearity in the pixel.

B. $L_{1/2}$-NMF ALGORITHM

The L$_{1/2}$-NMF algorithm follows the LMM. The only known term in equation 1 is the hyperspectral image X. In order to find A and S we perform L$_{1/2}$-NMF to minimize the distance between X and AS. Many methods can be used to measure such difference. The algorithm uses the Euclidean distance method and it is given as follows:

$$f(A, S) = \frac{1}{2}\|X - AS\|_2^2 + \lambda\|S\|_{1/2} \tag{3}$$

where

$$\|S\|_{1/2} = \sum_{k,n=1}^{K,N} s_n(k)^{1/2} \tag{4}$$

and

$$\lambda = \frac{1}{\sqrt{L}} \sum_{l} \frac{\sqrt{N} - \|x_l\|_1/\|x_l\|_2}{\sqrt{N-1}} \tag{5}$$

$\|S\|_{1/2}$ represents the sum of the element-wise square root of the elements of the matrix. λ is a factor that weighs the contribution of the L$_{1/2}$ regularization term. In order to solve the cost function in equation 3 multiplicative iterative algorithm is used to estimate A and S. When applied, the values in A and S are updated as follows:

$$A \leftarrow A.*XS^T./ASS^T \tag{6}$$

$$S \leftarrow S.*A^TX./\left(A^TAS + \frac{\lambda}{2}S^{-\frac{1}{2}}\right) \tag{7}$$

where $(.)^T$ represents matrix transpose and.* and./ represent element-wise multiplication and division, respectively. $S^{-\frac{1}{2}}$ is the element-wise inverse square root of S.

In order to ensure the full additivity constraint for the fractional abundances is achieved, both matrices X and A are augmented with a row of constants as follows:

$$X_f = \begin{bmatrix} X \\ \delta 1_N^T \end{bmatrix} \quad A_f = \begin{bmatrix} A \\ \delta 1_K^T \end{bmatrix} \tag{8}$$

where δ is a positive factor that controls the achievement of full additivity constraint. δ value is in the range between 10 and 20 to balance between convergence rate and accuracy of the estimation. Another important problem is dividing by zero in the term $S^{-\frac{1}{2}}$. Any zero value is replaced by a small value to avoid trivial results [3].

The algorithm stops the optimization process if one of two conditions is met. The first condition is a set number of iterations. The second condition is the distance between the current iteration and the previous one is less than a certain value ϵ. Figure 2 shows the flow chart of the L$_{1/2}$-NMF algorithm. The L$_{1/2}$-NMF algorithm can be summarized as follows [3]

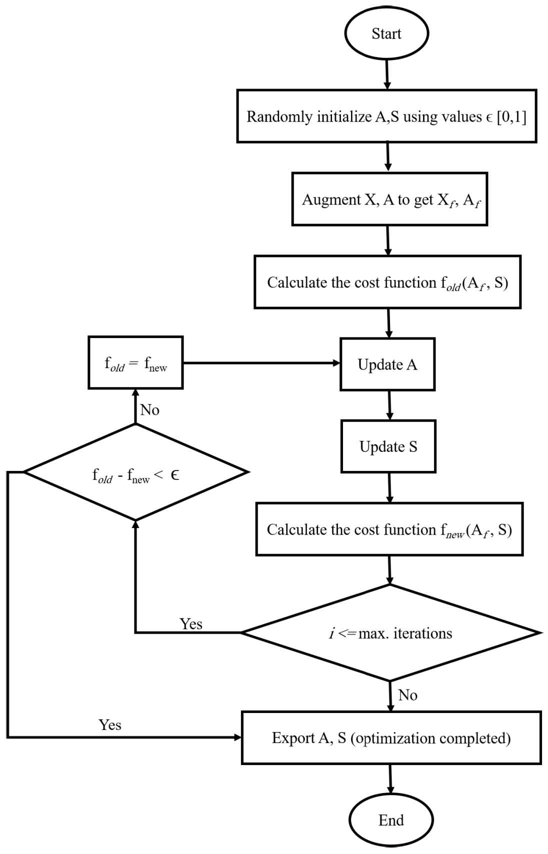

Start
Randomly initialize A,S using values ∈ [0,1]
Augment X, A to get X_f, A_f
Calculate the cost function f_old(A_f, S)
f_old = f_new
Update A
No
Update S
f_old - f_new < ϵ
Calculate the cost function f_new(A_f, S)
Yes
i <=max. iterations
No
Yes
Export A, S (optimization completed)
End

FIGURE 2. L$_{1/2}$-NMF algorithm flow chart.

L$_{1/2}$-NMF Algorithm

```
1: Estimate the number of end-members (k) using preprocessing algorithm. The authors used Hysime algorithm [42].
2: Estimate the weight parameter $\lambda$ according to the sparsity measure over X using equation 5.
3: Initialize A and S where A,S $\in$ [0, 1].
4: Rescale each column of S to unit norm.
5: **While** i < max. iterations **do**
   (a) $f_{old} = f(A_f, S)$ using equations 3 and 8
   (b) Update A using equation 6
   (c) Update S using equations 7 and 8
   (d) $f_{new} = f(A_f, S)$
   (e) **if** $f_{old} - f_{new} > \epsilon$ then
          $i = i + 1$
          $f_{old} = f_{new}$
          **Goto** (b)
       **else**
          **break**
       **end while**
```

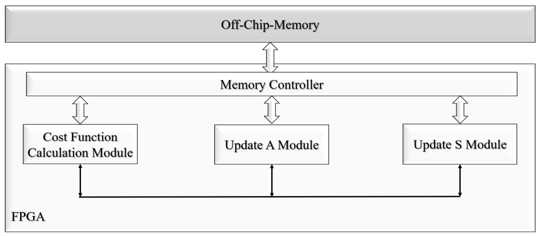


FIGURE 3. Hardware architecture of L$_{1/2}$-NMF algorithm.

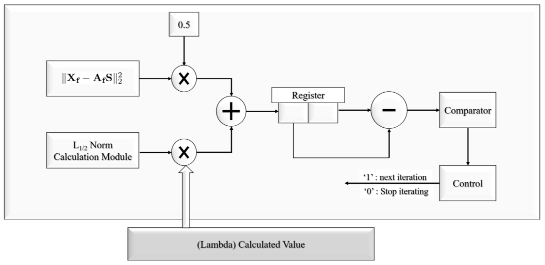


FIGURE 4. Cost function module architecture.

III. PROPOSED FPGA IMPLEMENTATION OF THE L$_{1/2}$-NMF ALGORITHM

This section describes the detailed implementation of the L$_{1/2}$-NMF Algorithm. As shown in Figure 3, the proposed hardware architecture of the L$_{1/2}$-NMF Algorithm contains an off-chip memory to save the augmented hyperspectral image, memory controller to select the data from the required addresses and the system main modules namely Cost Function Calculation Module, Update A Module and Update S Module implementing equations 3, 6 and 7 respectively.

A. COST FUNCTION CALCULATION MODULE

Figure 4 shows the architecture of the Cost Function Calculation Module. We divide the cost function to two main sub-blocks. The first sub-block is represented by $\|X_f - A_f S\|_2^2$. The second sub-block represents the L$_{1/2}$ norm calculation of the abundances matrix S represented by $\|S\|_{1/2}$. After calculating the first term, we multiply the resulting value by half then add the result to the multiplication of the second term by λ. The value of λ is calculated on an external processor and passed to the FPGA. The resulting value of the cost function calculation is saved in a shift register. While the algorithm is iterating, the value of the previous iteration is shifted to the second location inside the register. We take the difference between the current iteration and the previous one then pass the result to a comparator. The comparator compares the subtraction result with the predefined value ϵ to check the first stopping condition. If the subtraction value is greater than the value of ϵ then the iterations counter is incremented and the algorithm starts the next iteration. If the subtraction value is less than or equal to the value of ϵ then the algorithm stops iterating.

As shown in Figure 5, we start the calculation of the cost function by implementing the multiplication of the augmented matrix A$_f$ by the abundances matrix S. Both matrices are initialized and implemented as RAM blocks. Two control blocks are used to control the read,write processes, the addresses generation and the multiplication process. Control block 1 controls the writing addresses generation and the read and write enables. Control block 2 is used to generate the required reading addresses, the multiplication enable and the accumulation enable. The multiplication process here is a matrix multiplication process which is divided into a multiplier and an accumulator in our proposed design. We perform the element-by-element multiplication and pass the result to the accumulator until the vector-by-vector multiplication is completed.

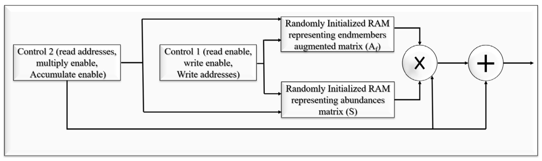

Control 2 (read addresses, multiply enable, Accumulate enable)
Control 1 (read enable, write enable, Write addresses)
Randomly Initialized RAM representing endmembers augmented matrix (A$_f$)
Randomly Initialized RAM representing abundances matrix (S)

FIGURE 5. Stage 1 of calculating the norm of $X_f - A_f S$.

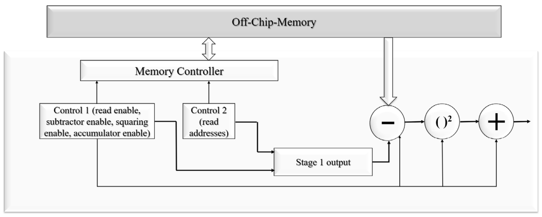

Off-Chip-Memory
Memory Controller
Control 1 (read enable, subtractor enable, squaring enable, accumulator enable)
Control 2 (read addresses)
Stage 1 output
$()^2$

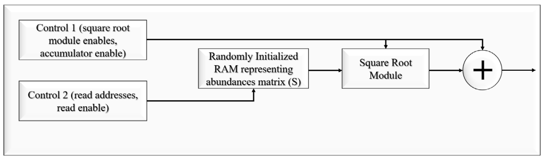
FIGURE 6. Stage 2 of calculating the norm of $X_f - A_f S$.

Figure 6 shows the next stage of calculating the cost function. The output of stage 1 is subtracted from the corresponding value in the augmented matrix X$_f$. The resulting subtraction value is then squared and passed to an accumulator. This process continues until all values are covered to calculate the first term of the cost function equation $\|X_f - A_f S\|_2^2$. In addition to the previously mentioned modules, we have two control blocks. The first block controls and synchronizes the subtraction, squaring, and accumulation processes with each output of the first stage. It also has the read enable sent to the off-chip memory. The second block generates the required addresses to read from the off-chip memory.

The second term of the cost function equation $\|S\|_{1/2}$ is implemented as shown in Figure 7. The L$_{1/2}$ term is implemented as in equation 4. The square root module is implemented in three blocks as shown in Figure 8. Altera floating point square root IP is used along with two converters from fixed point to floating point and vice versa. The square root module needs 40 clocks to perform the operation.

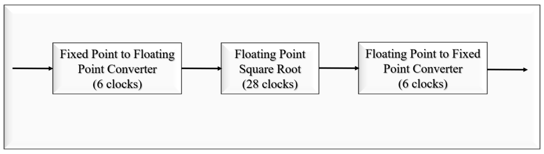
B. UPDATE A MODULE

In this section, we illustrate the hardware implementation of the module given by equation 6. As shown in Figure 9, equation 6 can be broken down to numerator and denominator terms. Each of the terms is handled separately then the



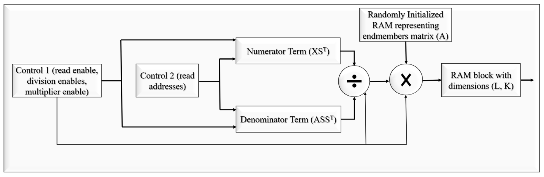
Control 1 (square root module enables, accumulator enable)
Control 2 (read addresses, read enable)
Randomly Initialized RAM representing abundances matrix (S)
Square Root Module
+

FIGURE 7. $L_{1/2}$ norm calculation module architecture.



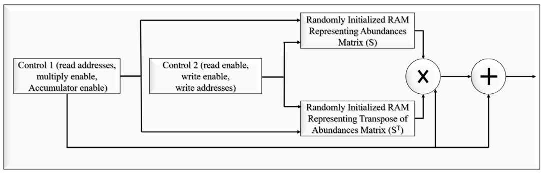
Fixed Point to Floating Point Converter (6 clocks)
Floating Point Square Root (28 clocks)
Floating Point to Fixed Point Converter (6 clocks)

FIGURE 8. Square root module architecture.



Control 1 (read enable, division enables, multiplier enable)
Control 2 (read addresses)
Numerator Term (XS^T)
Denominator Term (ASS^T)
Randomly Initialized RAM representing endmembers matrix (A)
÷
X
RAM block with dimensions (L, K)

FIGURE 9. Update a module architecture.



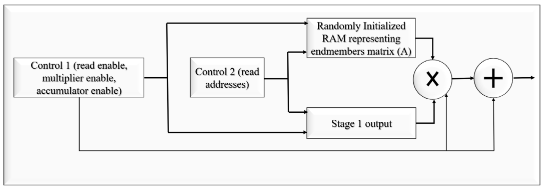
Control 1 (read addresses, multiply enable, Accumulator enable)
Control 2 (read enable, write enable, write addresses)
Randomly Initialized RAM Representing Abundances Matrix (S)
Randomly Initialized RAM Representing Transpose of Abundances Matrix (S^T)
X
+

FIGURE 10. Stage 1 of denominator term calculation.

element-by-element division and multiplication operations are performed at the end. The numerator term is divided by the denominator term and the output is multiplied using and element-by-element multiplier by the end-members matrix A. Two control blocks are used for generation of required reading addresses, read enable, division and multiplication enables. The numerator term values are divided by the denominator term values in an element-by-element operation. Next, the division result is multiplied by the corresponding value in the end-members matrix A. The resulting values are saved into A RAM block with the same dimensions of the end-members matrix A.

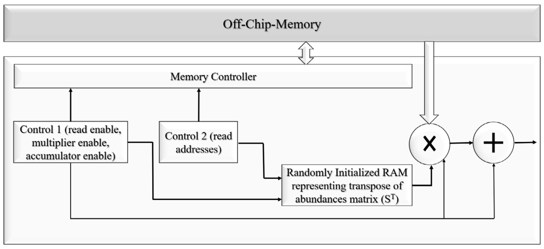
We start by implementing the matrix multiplication (SS^T) as shown in Figure 10. We use two RAM blocks representing the abundances Matrix and its transpose. In addition, a control block is used for read and write control signals, write addresses and read enable. Another control block is used to generate the read addresses, multiplication and accumulation enables.

Next, we multiply the result of stage 1 by the matrix A as done is stage 1. The result of stage 1 is saved in a memory element and accessed through control signals. Both matrix multiplication processes are broken down to a multiplier and an accumulator as shown in Figure 11.



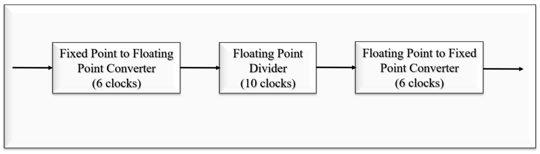
Control 1 (read enable, multiplier enable, accumulator enable)
Control 2 (read addresses)
Randomly Initialized RAM representing endmembers matrix (A)
Stage 1 output
X
+

FIGURE 11. Denominator term calculation architecture.



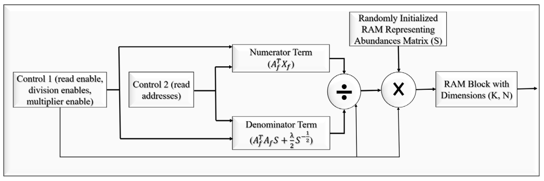
Off-Chip-Memory
Memory Controller
Control 1 (read enable, multiplier enable, accumulator enable)
Control 2 (read addresses)
Randomly Initialized RAM representing transpose of abundances matrix (S^T)
X
+

FIGURE 12. Numerator term calculation architecture.



Fixed Point to Floating Point Converter (6 clocks)
Floating Point Divider (10 clocks)
Floating Point to Fixed Point Converter (6 clocks)

FIGURE 13. Element-by-element divider module architecture.



Control 1 (read enable, division enables, multiplier enable)
Control 2 (read addresses)
Numerator Term ($A_f^T X_f$)
Denominator Term ($A_f^T A_f S + \frac{\lambda}{2} S^{-\frac{1}{2}}$)
Randomly Initialized RAM Representing Abundances Matrix (S)
÷
X
RAM Block with Dimensions (K, N)

FIGURE 14. Update S module architecture.

As shown in Figure 12, The numerator term is implemented in the same manner as the denominator using the off-chip memory, a randomly initialized memory element, a control block with control signals for multiplier, accumulator, read and write processes and another control block for reading addresses.

Next we implement the element-by-element division of the numerator and the denominator terms. For this module we use a floating point divider as the fixed point divider outputs are quotient and remainder while the floating point divider output is a rational number. As shown in Figure 13, the divider module contains a fixed point to floating point converter, the floating point divider and a floating point to fixed point converter.

C. UPDATE S MODULE

This section illustrates the hardware implementation of the module given by equation 7. The matrices X representing the image and A representing the end-members are augmented as in equation 8. As in the previous module in section III-B, we break down the equation to a numerator term and

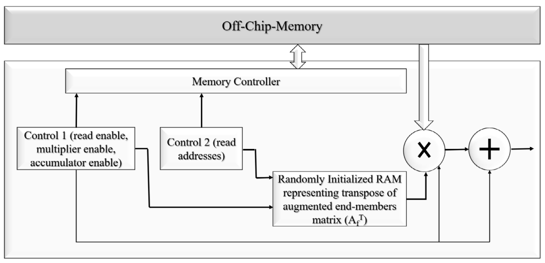

Off-Chip-Memory
Memory Controller
Control 1 (read enable, multiplier enable, accumulator enable)
Control 2 (read addresses)
Randomly Initialized RAM representing transpose of augmented end-members matrix (A_f^T)
X
+

FIGURE 15. Numerator term calculation architecture.

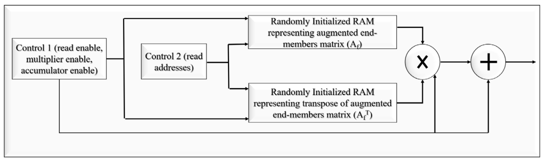

Control 1 (read enable, multiplier enable, accumulator enable)
Control 2 (read addresses)
Randomly Initialized RAM representing augmented end-members matrix (A_f)
Randomly Initialized RAM representing transpose of augmented end-members matrix (A_f^T)
X
+

FIGURE 16. Stage 1 of implementing the denominator first term.

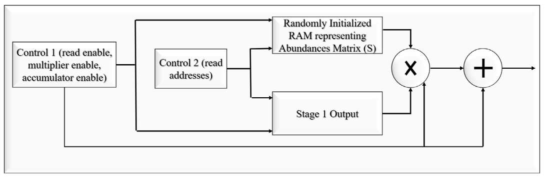

Control 1 (read enable, multiplier enable, accumulator enable)
Control 2 (read addresses)
Randomly Initialized RAM representing Abundances Matrix (S)
Stage 1 Output
X
+

FIGURE 17. Calculation of the denominator first term.

denominator terms as shown in Figure 14. The resulting values of the numerator and denominator terms are passed to the element-by-element divider. The divider result is multiplied by the abundances matrix S and the result is saved into a RAM block with the same dimensions as the abundances matrix S. Two control blocks are used for generation of required reading addresses, read enable, division and multiplication enables. The resulting value are saved into A RAM block with the same dimensions of the abundances matrix S.

We start by implementing the matrix multiplication between the transpose of the augmented end-members matrix A_f^T and the augmented matrix X_f as shown in Figure 15. As previously mentioned, the matrix multiplication is broken down to a multiplier and an accumulator.

The denominator has two terms. The matrix multiplication $A_f^T A_f S$, is implemented through two stages. Figure 16 shows the first stage implementing the matrix multiplication $A_f^T A_f$. The end-members are saved in two RAM blocks with dimensions L by K and K by L. Two control blocks are used for required addresses generation and control signals generation. The matrix multiplication is broken down to a multiplier and an accumulator as in the previous modules.

The output of stage 1 is saved into a RAM block and multiplied by the abundances matrix S. As shown in Figure 17, the matrix multiplication is performed on two stages as a multiplier and an accumulator. Control blocks are needed to generate control signals for the multiplier and

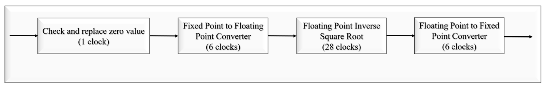

Check and replace zero value (1 clock)
Fixed Point to Floating Point Converter (6 clocks)
Floating Point Inverse Square Root (28 clocks)
Floating Point to Fixed Point Converter (6 clocks)

FIGURE 18. Inverse square root module architecture.

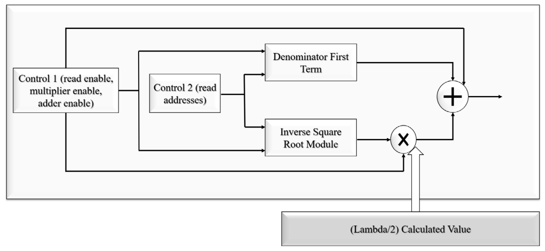

Control 1 (read enable, multiplier enable, adder enable)
Control 2 (read addresses)
Denominator First Term
Inverse Square Root Module
X
+
(Lambda/2) Calculated Value

FIGURE 19. Denominator term calculation architecture.

accumulator enables, read and write enables and generating reading addresses.

The second term of the denominator is implemented through two stages. In the first stage, we calculate the element-wise inverse square root of the abundances matrix S. Before calculating the inverse square root, the input value is tested. If the value is zero then it is replaced by a very small value. The inverse square root module is implemented in 3 blocks. Altera floating point inverse square root IP is used along with two converters from fixed point to floating point and from floating point to fixed point. The inverse square root module needs 49 clocks to perform the operation as shown in Figure 18.

Finally, the inverse square root module output is multiplied by the value $\frac{\lambda}{2}$ then added to the first term of the denominator. The aforementioned multiplication and addition processes are element-by-element operations. Figure 19 shows the implementation of the denominator term. Control blocks are added for control signals generation to synchronously enable the multiplier and the adder, read and write enables and required addresses generation.

IV. COMPLEXITY ANALYSIS

In this section, we discuss the computational complexity of the $L_{1/2}$-NMF algorithm relative to some other algorithms present in the literature. As we mentioned in section I, the $L_{1/2}$-NMF algorithm outperforms NFINDR, PPI and VCA algorithms [3]. However, the $L_{1/2}$-NMF algorithm has higher complexity than NFINDR, PPI and VCA algorithms. This is due to the fact that the $L_{1/2}$-NMF algorithm computes both the end-members and the abundances matrices while NFINDR, PPI and VCA algorithms compute only the end-members matrix. In general, statistical based approaches have higher complexity than geometrical based approaches [4]. Table 1 shows the computational complexity of the NFINDR,PPI, VCA and $L_{1/2}$-NMF algorithms.

where L is the number of bands, K is the number of end-members to be detected and N is the number of pixels present

TABLE 1. Computational complexity of NFINDR,PPI, VCA and $L_{1/2}$-NMF algorithms.

Algorithm	Complexity
NFINDR	$K^{\eta+1}N$ [12]
PPI	2KsN [12]
VCA	$2K^2N$ [12]
$L_{1/2}$-NMF	$LKN + (KN)^2$ [3]

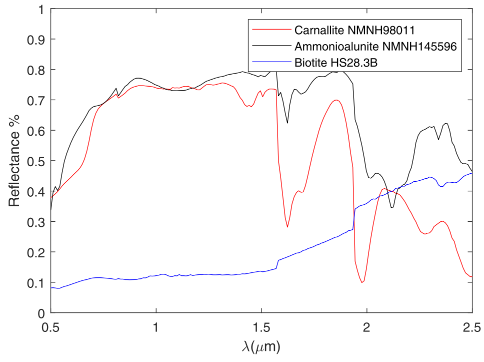


FIGURE 20. Reflectances of carnallite NMNH98011, ammonioalunite NMNH145596 and biotite HS28.3B.

in the scene, $2.3 < \eta < 2.9$ and s is the number of skewers for the PPI algorithm.

V. EXPERIMENTAL RESULTS

A. SIMULATION AND IMPLEMENTATION TOOLS

The hardware implementation of the $L_{1/2}$-NMF algorithm described in section III is designed using Mentor Graphics FPGAdv 8.1. For the synthetic data set, the simulations are performed using ModelSim SE PLUS 6.3a tool. The proposed design is synthesized using Intel Quartus II 15.0 web edition. Altera Cyclone V device number 5CGXFC9E7F35C8 [49] is chosen as the target hardware platform. For the real data set, the simulations are performed using ModelSim 10.1c tool and the proposed design is synthesized using Intel Quartus Prime 15.1 standard pro edition. Altera Arria 10 device number 10AS048E1F29E1SG [50] is chosen as the target hardware platform. This is due to the limitation of the previously used target platform for the synthetic data set experiment. The chosen FPGA for the real data set experiment has more resources and is of a newer technology. The Arria 10 FPGA uses 20 nm technology while Cyclone V FPGA uses 28 nm technology. The newer technology allows the FPGA to operate at higher frequency and to have more resources which we show later in section V-C.5. The implementation results are compared with simulation results performed on MATLAB R2018a with Windows 10 operating system on Intel Core^{TM} i7-4790 CPU@3.6 GHZ and 16 GB memory.

B. DATA SETS

1) SYNTHETIC HYPERSPECTRAL DATA SET

We test the proposed implementation of the $L_{1/2}$-NMF algorithm on synthetic data prepared as mentioned in [12].

FIGURE 21. Subimage of the samson hyperspectral data set. [51]

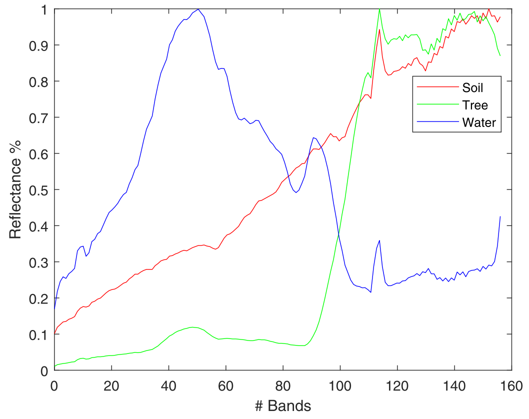


FIGURE 22. Reflectances of soil, water and tree.

A simulated scene of dimensions 188 × 400 is generated satisfying equation 1. In section III, we show that the proposed architecture of the $L_{1/2}$-NMF algorithm depends on memory blocks in many steps of the design. Since this is the first implementation of the $L_{1/2}$-NMF algorithm, we choose to test our proposed design on a smaller scene with smaller number of end-members to avoid memory overflow and validate our results compared to our simulation results. Three spectral signatures are selected from the U.S. Geological Survey (USGS) digital spectral library [52]. The selected spectra are Carnallite NMNH98011, Ammonioalunite NMNH145596 and Biotite HS28.3B as shown in Figure 20.

Each value in the dataset is converted in MATLAB to binary representation. The binary value is of length 64 bits with 15 bits representing the integer part and 48 bits representing the fraction part. The choice of 64 bits representation is to provide more accuracy during calculations especially while truncating the outputs of the multipliers and dividers. Inputs and outputs data type is a standard logic vector 64 bits. Intermediate signals and RAM addresses are of an unsigned type.

2) REAL HYPERSPECTRAL DATA SET

In this section, we use real hyperspectral data set to test the proposed FPGA hardware implementation of the

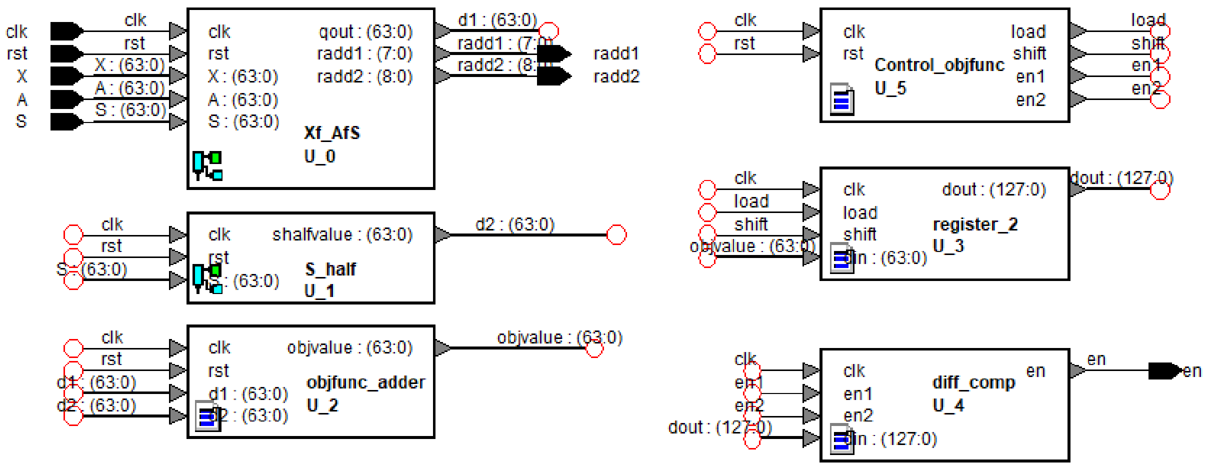


FIGURE 23. Cost function calculation module implementation.

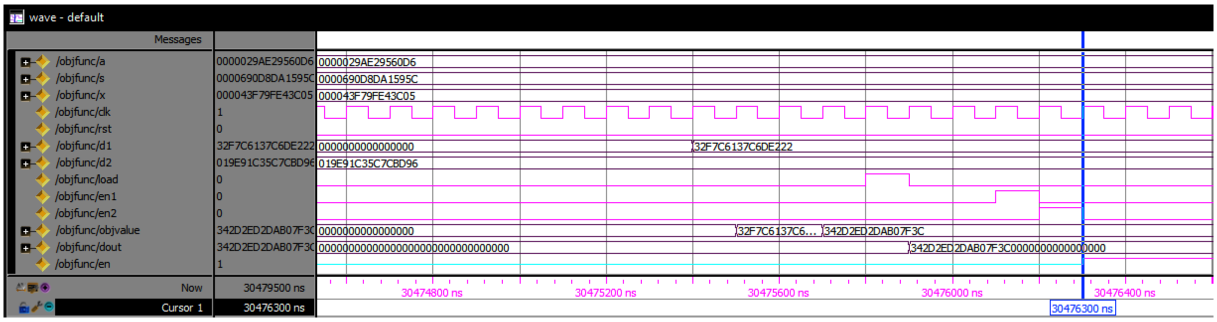

```
wave - default
Messages
/objfunc/a          0000029AE29560D6   0000029AE29560D6
/objfunc/s          0000690D8DA1595C   0000690D8DA1595C
/objfunc/x          000043F79FE43C05   000043F79FE43C05
/objfunc/clk        1
/objfunc/rst        0
/objfunc/d1         32F7C6137C6DE222   0000000000000000   32F7C6137C6DE222
/objfunc/d2         019E91C35C7CBD96   019E91C35C7CBD96
/objfunc/load       0
/objfunc/en1        0
/objfunc/en2        0
/objfunc/objvalue   342D2ED2DAB07F3C   0000000000000000   32F7C6137C6...   342D2ED2DAB07F3C
/objfunc/dout       342D2ED2DAB07F3C   00000000000000000000000000000000   342D2ED2DAB07F3C0000000000000000
/objfunc/en         1
Now        30479500 ns    30474800 ns   30475200 ns   30475600 ns   30476000 ns   30476400 ns
Cursor 1   30476300 ns    30476300 ns
```

FIGURE 24. Simulation results of cost function calculation module.

$L_{1/2}$-NMF algorithm. The data set[1] is acquired by the SAMSON instrument hyperspectral sensor. Figure 21 shows the region of 95 × 95 pixels of the Samson hyperspectral data set.

This data set consists of three end-members representing water, soil, and tree. Figure 22 shows the spectra of the three end-members. The image consists of 156 channels covering the wavelengths from 401 nm to 889 nm.

As previously mentioned for the synthetic data set, each value in the dataset is converted in MATLAB to binary representation. The binary value is of length 64 bits with 15 bits representing the integer part and 48 bits representing the fraction part. The choice of 64 bits representation is to provide more accuracy during calculations especially while truncating the outputs of the multipliers and dividers. Inputs and outputs data type is a standard logic vector 64 bits. Intermediate signals and RAM addresses are of an unsigned type.

C. IMPLEMENTATION RESULTS

1) COST FUNCTION CALCULATION MODULE IMPLEMENTATION RESULTS

Figure 23 shows the implementation of the Cost Function Calculation module. We calculate the first term in equation 3 $\|X_f - A_f S\|_2^2$ in the first unit "Xf_AfS". The second term $\|S\|_{1/2}$ is calculated in the second unit "S_half". The resulting values from the first and second units are passed to the "objfunc_adder" unit. The first value is multiplied by half and the second value is multiplied by λ then the values are added to calculate the cost function value. The cost function value is stored inside the shift register "register_2". While the algorithm is iterating, the value is shifted inside the register and the new iteration value is stored in its place. Both values are subtracted and the result is compared with the threshold value ϵ in the "diff_comp" unit. The "diff_comp" unit also has a counter for the number of iterations to which counts up to the maximum number of iterations in the second stopping condition. The "control_objfunc" controls the save and shift processes in the register.

Figure 24 shows the simulation results of the Cost Function Calculation module. The cost function value "objvalue" is calculated and stored in the register. The cost function value is then subtracted from the previous value and compared to the threshold value ϵ. The result is the "en" signal indicating the need for more iterations or the end of the optimization.

2) UPDATE A MODULE IMPLEMENTATION RESULTS

Figure 25 shows the implementation of the Update A module. As previously explained in section III-B, The numerator term is calculated in "XSTsaved" unit and the denominator term is calculated in the "ASSTsaved" unit. Each element in the

[1]Available here: http://lesun.weebly.com/hyperspectral-data-set.html

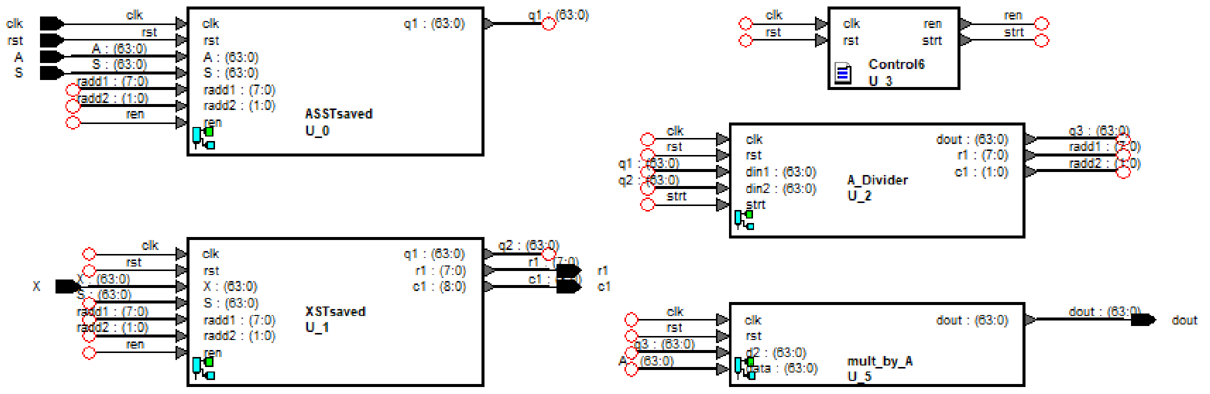


FIGURE 25. Update a module implementation.

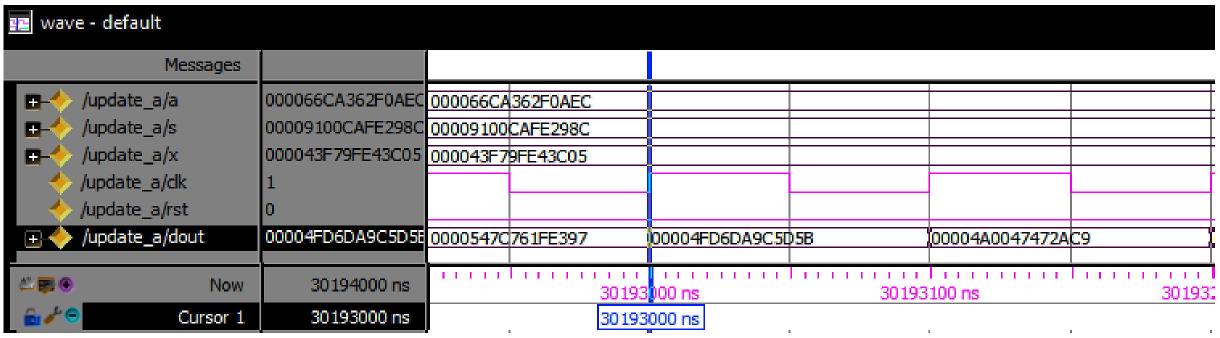

```
wave - default
Messages
/update_a/a      000066CA362F0AEC   000066CA362F0AEC
/update_a/s      00009100CAFE298C   00009100CAFE298C
/update_a/x      000043F79FE43C05   000043F79FE43C05
/update_a/clk    1
/update_a/rst    0
/update_a/dout   00004FD6DA9C5D5B   0000547C761FE397   00004FD6DA9C5D5B   00004A0047472AC9
Now        30194000 ns          30193000 ns     30193100 ns     30193:
Cursor 1   30193000 ns     30193000 ns
```

FIGURE 26. Simulation results of update a module.

numerator term is divided by its corresponding element in the denominator term inside the "A_Divider" unit. The division result is then multiplied by the corresponding element in the end-members matrix A inside the "mult_by_A" unit and stored in a RAM block.

Figure 26 shows the simulation results of the Update A module. As previously mentioned, after updating the end-members matrix values, the resulting values are stored inside a RAM block. These resulting values are passed to the other two main modules; Cost Function Calculation module and Update S module; to complete the optimization steps. The updated end-members matrix also replaces the old one in the next iteration. After the algorithm completes the optimization, the values of the end-members matrix are exported to a text file and read inside MATLAB software to compare the implementation and simulation results.

3) UPDATE S MODULE IMPLEMENTATION RESULTS

Figure 27 shows the implementation of the Update S module. As previously explained in section III-C, the denominator terms are calculated in "AfTAfS_saved" and "S_neg_half_saved" units. Each value from the "S_neg_half_saved" unit is multiplied by $\frac{\lambda}{2}$ and added to the corresponding resulting value from the "AfTAfS_saved" unit inside the "S_adder" unit. The resulting value from "AfTXf_saved" is divided by the corresponding value resulting from the "S_adder" unit using the "S_divider" unit. Finally, the resulting value from the "S_divider" unit is multiplied by the corresponding value in the abundances matrix stored in the "RAM_3_400_controlled" unit. The final values are stored in a RAM block for updating and reuse.

Figure 28 shows the simulation results of the Update S module. As previously mentioned, the denominator terms are calculated while the numerator term is calculated. Next, the numerator term is divided by the denominator term then multiplied by the abundances matrix. The resulting values are stored in a RAM block to update the old values in the abundances matrix and to be passed to the other modules to continue the optimization process.

4) SYNTHESIS RESULTS

The proposed hardware implementation of the $L_{1/2}$-NMF algorithm for the synthetic data set is synthesized using Intel Quartus II 15.0 web edition. For the real data, the proposed design is synthesized using Quartus Prime 15.1 standard pro edition which supports newer FPGA families. Table 2 summarizes the resources used for the proposed hardware implementation of the $L_{1/2}$-NMF algorithm for both synthetic and real data sets.

We implement our FPGA design for the synthetic data set on Altera Cyclone V 5CGXFC9E7F35C8 FPGA. The proposed FPGA design of the $L_{1/2}$-NMF algorithm occupies 9127 logic elements, 17133 registers, around 1.26 Mb of memory and 192 DSP blocks. The proposed design operates

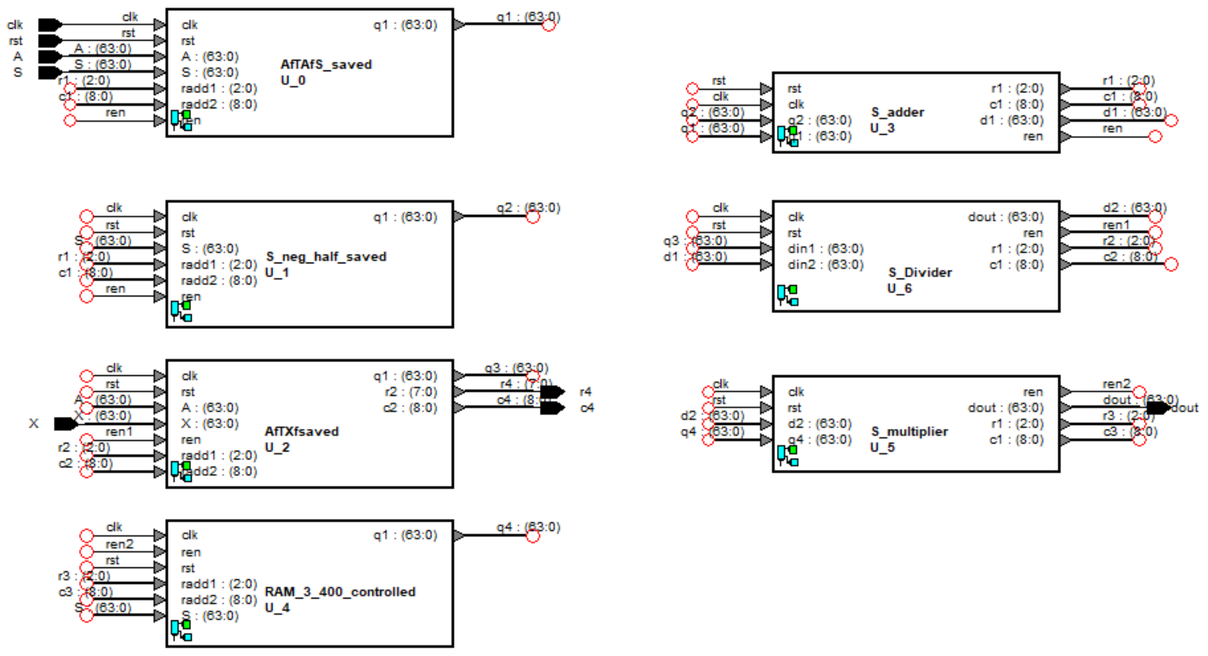


FIGURE 27. Update S module implementation.

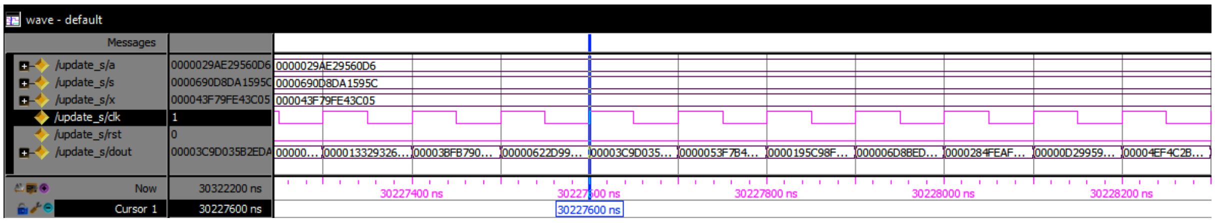


FIGURE 28. Simulation results of update S module.

TABLE 2. Place and route results.

Synthetic Data						Samson Real Data					
Resource	Available	Cost Function Module	Update A Module	Update S Module	Overall System	Resource	Available	Cost Function Module	Update A Module	Update S Module	Overall System
Logic Utilization	113560	2241	2494	4174	9127	Logic Utilization	183590	16873	587174	49339	124474
Total Registers		4192	4286	8297	17133	Total Registers		34320	124391	99374	258479
Total Block Memory Bits	12492800	3110850	416064	652113	1259715	Total Block Memory Bits	29306880	3110850	4670784	9337443	17119077
Total DSP Blocks	342	28	53	103	192	Total DSP Blocks	1368	32	55	103	190
Total Pins	616	195	278	278	310	Total Pins	416	216	279	279	321
Max. Freq. MHz		51.33	52.81	53.45	52.6	Max. Freq. MHz		105.73	103.07	105.91	104.32
Iteration Run Time(msec)		4.46	4.29	4.26	8.72	Iteration Run Time(msec)		35.85	36.79	36.02	72.81

at 52.6 MHz maximum frequency. Our tests show that the simulation run time for the algorithm on MATLAB is 52.11 seconds while the implementation run time on FPGA is 13.33 seconds which results in a speedup factor of 3.9. In Table 2, we show the run time of each module per iteration since the algorithm can stop optimizing at different iterations according to the stopping conditions. For the real data set, we implement our FPGA design on Altera Arria 10 10AS048E1F29E1SGX FPGA. The proposed FPGA design of the $L_{1/2}$-NMF algorithm for the real data

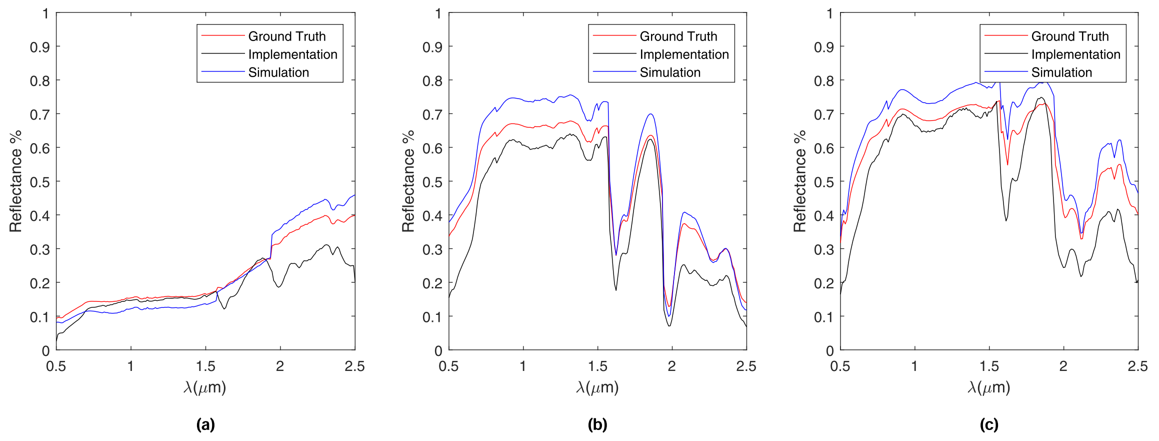


FIGURE 29. Simulation and implementation results comparison of synthetic data set (a)Biotite HS28.3B (b)Carnallite NMNH98011 (c)Ammonioalunite NMNH145596.

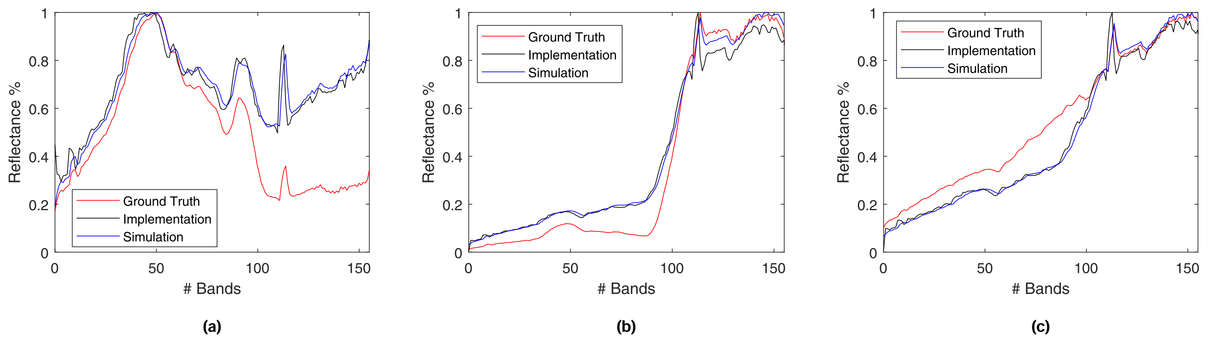


FIGURE 30. Simulation and implementation results comparison of samson hyperspectral data set (a) Water (b) Tree (c) Soil.

set occupies 124474 logic elements, 258479 registers, around 17 Mb of memory and 190 DSP blocks. Our tests show that the simulation run time for the algorithm on MATLAB is 152.89 seconds while the implementation run time on FPGA is 135.46 seconds wich results in a speedup factor of 1.14.

5) RESULTS COMPARISON

The implementation results are compared to MATLAB simulation results and the ground truth signatures [52] using Spectral Angular Distance given by:

$$\mathrm{SAD} = \cos^{-1}\left(\frac{A^{T}\hat{A}}{\|A\|\,\|\hat{A}\|}\right) \tag{9}$$

Figures 29 and 30 show the results comparison between MATLAB simulation, FPGA results and ground truth signatures of the three selected end-members. The simulated and the implemented results are plotted according to the calculated SAD values. We calculate the SAD values for each detected end-member against the ground truth signatures of of all end-members. We plot the signature of the detected end-member with the lowest SAD value against the ground truth signature of the same end-member. In our implementation, we use $\delta = 15$, small value $= 10^{-6}$, max. iterations value $= 3000$ and $\epsilon = 10^{-3}$.

TABLE 3. Synthetic data set SAD results comparison.

End-member	MATLAB SAD	FPGA SAD
Biotite	0.1239	0.2663
Carnallite	0.0299	0.1044
Ammonioalunite	0.0204	0.1604

TABLE 4. Samson data set SAD results comparison.

End-member	MATLAB SAD	FPGA SAD
Water	0.3327	0.3308
Tree	0.1297	0.1044
Soil	0.1415	0.1673

Tables 3 and 4 show the SAD values calculated between MATLAB simulation resulting end-members versus their corresponding ground truth signatures and FPGA implementation resulting end-members versus the corresponding ground truth signatures. From the tables, it can be seen that the implementation results have higher SAD values than simulation results. This is due to truncations and conversions that occur within the implemented architecture of the

$L_{1/2}$-NMF algorithm. This comes from the fact that after each multiplication or division process, the resulting values are truncated to have the same data length in all the stages of the hardware implementation. In addition, the approximation that results from the conversion from fixed point to floating point representations and vice versa affects the accuracy of the resulting values.

VI. CONCLUSION

In this paper, we presented the first FPGA hardware implementation of the $L_{1/2}$-NMF algorithm. Although $L_{1/2}$-NMF algorithm outperforms many implemented algorithms such as NFINDR, PPI and VCA, it has not been yet implemented in the literature. The proposed design is tested using both synthetic and real hyperspectral data sets. In the proposed design, the mathematical operations are broken down to element-by-element operations. Matrix multiplication is implemented as a multiplier and an accumulator. Each stage of the design has its own control blocks. Two types of control blocks are used in the design. The first type is used to generate read enable, write enable, write addresses, multiplier, adders and subtracter enables. The second control block type is focused on generating read addresses synchronized with blocks outputs. The design is implemented using Mentor Graphics FPGAdv 8.1 and synthesized using Altera Quartus II 15.0 web edition for the synthetic data set and Altera Quartus Prime 15.1 standard pro edition for the real data set. The experimental results were performed on Altera Cyclone V 5CGXFC9E7F35C8 FPGA for the synthetic data set and Altera Arria 10 10AS048E1F29E1SG FPGA for the real data set. The hardware results are compared to the MATLAB simulation results using the same data set and ground truth signatures via SAD calculation. The experimental results show the proposed hardware design successfully unmixes the data set with SAD results comparable to the simulation results. The maximum MATLAB SAD value was 0.1239 while the maximum FPGA SAD value was 0.2663 for the synthetic data set. On the other hand, the maximum MATLAB SAD value was 0.3327 while the maximum FPGA SAD value was 0.3308 for the real data set. The proposed design of the $L_{1/2}$-NMF algorithm works with a maximum speed of 52.6 MHz for the synthetic data set and 104.32 for the real data set. When compared to the MATLAB simulation, the FPGA hardware design has higher speed which resulted in speed up factors of 3.9 and 1.14 for the synthetic and real data sets respectively. In the future, we will exploit pipelined and parallel implementation of matrix multiplication to decrease the run time of the implemented algorithm as well as to better utilize the available resources.

REFERENCES

[1] L. Miao and H. Qi, "Endmember extraction from highly mixed data using minimum volume constrained nonnegative matrix factorization," *IEEE Trans. Geosci. Remote Sens.*, vol. 45, no. 3, pp. 765–777, Mar. 2007.

[2] W. Wang and Y. Qian, "Adaptive $L_{1/2}$ sparsity-constrained NMF with half-thresholding algorithm for hyperspectral unmixing," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 8, no. 6, pp. 2618–2631, Jun. 2015.

[3] Y. Qian, S. Jia, J. Zhou, and A. Robles-Kelly, "Hyperspectral unmixing via $L_{1/2}$ sparsity-constrained nonnegative matrix factorization," *IEEE Trans. Geosci. Remote Sens.*, vol. 49, no. 11, pp. 4282–4297, Nov. 2011.

[4] J. M. Bioucas-Dias, A. Plaza, N. Dobigeon, M. Parente, Q. Du, P. Gader, and J. Chanussot, "Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 5, no. 2, pp. 354–379, Apr. 2012.

[5] N. Keshava and J. F. Mustard, "Spectral unmixing," *IEEE Signal Process. Mag.*, vol. 19, no. 1, pp. 44–57, Jan. 2002.

[6] X. Liu, W. Xia, B. Wang, and L. Zhang, "An approach based on constrained nonnegative matrix factorization to unmix hyperspectral data," *IEEE Trans. Geosci. Remote Sens.*, vol. 49, no. 2, pp. 757–772, Feb. 2011.

[7] M. Craig, "Minimum-volume transforms for remotely sensed data," *IEEE Trans. Geosci. Remote Sens.*, vol. 32, no. 3, pp. 542–552, May 1994.

[8] J. Boardman, F. Kruse, and R. Green, "Mapping target signatures via partial unmixing of AVIRIS data: In summaries," in *Proc. 5th JPL Airborne Earth Sci. Workshop*, 1995, pp. 23–26.

[9] J. W. Boardman, "Automating spectral unmixing of aviris data using convex geometry concepts," in *Proc. JPL, Summaries 4th Annu. JPL Airborne Geosci. Workshop. AVIRIS Workshop*, vol. 1, 1993, pp. 11–14.

[10] M. E. Winter, "N-FINDR: An algorithm for fast autonomous spectral end-member determination in hyperspectral data," *Proc. SPIE*, vol. 3753, pp. 266–275, Oct. 1999.

[11] A. Plaza, P. Martinez, R. Perez, and J. Plaza, "Spatial/spectral endmember extraction by multidimensional morphological operations," *IEEE Trans. Geosci. Remote Sens.*, vol. 40, no. 9, pp. 2025–2041, Sep. 2002.

[12] J. Nascimento and J. Dias, "Vertex component analysis: A fast algorithm to unmix hyperspectral data," *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 4, pp. 898–910, Apr. 2005.

[13] X. Tao, B. Wang, and L. Zhang, "Orthogonal bases approach for the decomposition of mixed pixels in hyperspectral imagery," *IEEE Geosci. Remote Sens. Lett.*, vol. 6, no. 2, pp. 219–223, Apr. 2009.

[14] M. Berman, H. Kiiveri, R. Lagerstrom, A. Ernst, R. Dunne, and J. Huntington, "ICE: A statistical approach to identifying endmembers in hyperspectral images," *IEEE Trans. Geosci. Remote Sens.*, vol. 42, no. 10, pp. 2085–2095, Oct. 2004.

[15] Y. Ma, Q. Jin, X. Mei, X. Dai, F. Fan, H. Li, and J. Huang, "Hyperspectral unmixing with Gaussian mixture model and low-rank representation," *Remote Sens.*, vol. 11, no. 8, p. 911, Apr. 2019.

[16] O. Eches, N. Dobigeon, C. Mailhes, and J.-Y. Tourneret, "Bayesian estimation of linear mixtures using the normal compositional model. Application to hyperspectral imagery," *IEEE Trans. Image Process.*, vol. 19, no. 6, pp. 1403–1413, Jun. 2010.

[17] X. Du, A. Zare, P. Gader, and D. Dranishnikov, "Spatial and spectral unmixing using the beta compositional model," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 6, pp. 1994–2003, Jun. 2014.

[18] Y. Zhou, A. Rangarajan, and P. D. Gader, "A Gaussian mixture model representation of endmember variability in hyperspectral unmixing," *IEEE Trans. Image Process.*, vol. 27, no. 5, pp. 2242–2256, May 2018.

[19] T. Nouri, M. M. Oskouei, and H. Zekri, "A comparison study of ORASIS and VCA for mineralogical unmixing of hyperspectral data," *J. Indian Soc. Remote Sens.*, vol. 44, no. 5, pp. 723–733, Oct. 2016.

[20] C.-I. Chang, *Hyperspectral Data Exploitation: Theory and Applications*. Hoboken, NJ, USA: Wiley, 2007.

[21] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, Oct. 1999.

[22] P. Paatero and U. Tapper, "Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values," *Environmetrics*, vol. 5, no. 2, pp. 111–126, Jun. 1994.

[23] D. Donoho and V. Stodden, "When does non-negative matrix factorization give a correct decomposition into parts?" in *Proc. Adv. Neural Inf. Process. Syst.*, 2003.

[24] T. Nouri and M. M. Oskouei, "Processing of Hyperion data set for detection of indicative minerals using a hybrid method in Dost-Bayli, Iran," *Int. J. Remote Sens.*, vol. 37, no. 20, pp. 4923–4947, Oct. 2016.

[25] V. P. Pauca, J. Piper, and R. J. Plemmons, "Nonnegative matrix factorization for spectral data analysis," *Linear Algebra Appl.*, vol. 416, no. 1, pp. 29–47, Jul. 2006.

[26] S. Jia and Y. Qian, "Constrained nonnegative matrix factorization for hyperspectral unmixing," *IEEE Trans. Geosci. Remote Sens.*, vol. 47, no. 1, pp. 161–173, Jan. 2009.

[27] R. Liu, B. Du, and L. Zhang, "Hyperspectral unmixing via double abundance characteristics constraints based NMF," *Remote Sens.*, vol. 8, no. 6, p. 464, May 2016.

[28] X. Lu, H. Wu, Y. Yuan, P. Yan, and X. Li, "Manifold regularized sparse NMF for hyperspectral unmixing," *IEEE Trans. Geosci. Remote Sens.*, vol. 51, no. 5, pp. 2815–2826, May 2013.

[29] P. O. Hoyer, "Non-negative matrix factorization with sparseness constraints," *J. Mach. Learn. Res.*, vol. 5, pp. 1457–1469, Nov. 2004.

[30] Z. Yang, G. Zhou, S. Xie, S. Ding, J.-M. Yang, and J. Zhang, "Blind spectral unmixing based on sparse nonnegative matrix factorization," *IEEE Trans. Image Process.*, vol. 20, no. 4, pp. 1112–1125, Apr. 2011.

[31] R. Huang, X. Li, and L. Zhao, "Nonnegative matrix factorization with data-guided constraints for hyperspectral unmixing," *Remote Sens.*, vol. 9, no. 10, p. 1074, Oct. 2017.

[32] L. Sun, W. Ge, Y. Chen, J. Zhang, and B. Jeon, "Hyperspectral unmixing employing $l_1 - l_2$ sparsity and total variation regularization," *Int. J. Remote Sens.*, vol. 39, no. 19, pp. 6037–6060, 2018.

[33] Z. Zhang, S. Liao, H. Zhang, S. Wang, and Y. Wang, "Bilateral filter regularized l_2 sparse nonnegative matrix factorization for hyperspectral unmixing," *Remote Sens.*, vol. 10, no. 6, p. 816, 2018.

[34] Z. Xu, H. Zhang, Y. Wang, X. Chang, and Y. Liang, "$L_{1/2}$ regularization," *Sci. China Inf. Sci.*, vol. 53, no. 6, pp. 1159–1169, 2010.

[35] D. Fernandez, C. Gonzalez, D. Mozos, and S. Lopez, "FPGA implementation of the principal component analysis algorithm for dimensionality reduction of hyperspectral images," *J. Real-Time Image Process.*, vol. 16, no. 5, pp. 1395–1406, Oct. 2019.

[36] C. Gonzalez, D. Mozos, J. Resano, and A. Plaza, "FPGA implementation of the N-FINDR algorithm for remotely sensed hyperspectral image analysis," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 2, pp. 374–388, Feb. 2012.

[37] A. Barberis, G. Danese, F. Leporati, A. Plaza, and E. Torti, "Real-time implementation of the vertex component analysis algorithm on GPUs," *IEEE Geosci. Remote Sens. Lett.*, vol. 10, no. 2, pp. 251–255, Mar. 2013.

[38] G. M. Callicó, S. Lopez, B. Aguilar, J. F. López, and R. Sarmiento, "Parallel implementation of the modified vertex component analysis algorithm for hyperspectral unmixing using opencl," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 8, pp. 3650–3659, Aug. 2014.

[39] D. Valencia and A. Plaza, "FPGA-based hyperspectral data compression using spectral unmixing and the pixel purity index algorithm," in *Proc. Int. Conf. Comput. Sci.* Berlin, Germany: Springer, 2006, pp. 888–891.

[40] J. M. P. Nascimento, M. Vestias, and G. Martin, "FPGA-based architecture for hyperspectral unmixing," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, Jul. 2015, pp. 1761–1764.

[41] C. Gonzalez, S. Lopez, D. Mozos, and R. Sarmiento, "Fpga implementation of the hysime algorithm for the determination of the number of endmembers in hyperspectral data," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 8, no. 6, pp. 2870–2883, Jun. 2015.

[42] J. Bioucas-Dias and J. Nascimento, "Hyperspectral subspace identification," *IEEE Trans. Geosci. Remote Sens.*, vol. 46, no. 8, pp. 2435–2445, Aug. 2008.

[43] J. Lei, L. Wu, Y. Li, W. Xie, C.-I. Chang, J. Zhang, and B. Huang, "A novel FPGA-based architecture for fast automatic target detection in hyperspectral images," *Remote Sens.*, vol. 11, no. 2, p. 146, Jan. 2019.

[44] C. González, S. Sánchez, A. Paz, J. Resano, D. Mozos, and A. Plaza, "Use of FPGA or GPU-based architectures for remotely sensed hyperspectral image processing," *Integration*, vol. 46, no. 2, pp. 89–103, Mar. 2013.

[45] C. A. Lee, S. D. Gasster, A. Plaza, C.-I. Chang, and B. Huang, "Recent developments in high performance computing for remote sensing: A review," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 508–527, Sep. 2011.

[46] L. Sterpone, M. Porrmann, and J. Hagemeyer, "A novel fault tolerant and runtime reconfigurable platform for satellite payload processing," *IEEE Trans. Comput.*, vol. 62, no. 8, pp. 1508–1525, Aug. 2013.

[47] J. A. Clemente, J. Resano, C. González, and D. Mozos, "A hardware implementation of a run-time scheduler for reconfigurable systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 7, pp. 1263–1276, Jul. 2010.

[48] J. Settle and N. Drake, "Linear mixing and the estimation of ground cover proportions," *Int. J. Remote Sens.*, vol. 14, no. 6, pp. 1159–1177, 1993.

[49] *Cyclone V Device Handbook*, Altera, San Jose, CA, USA, 2019.

[50] *Arria 10 Core Fabric and General Purpose I/Os Handbook A10*, Altera, San Jose, CA, USA, 2019.

[51] F. Zhu, Y. Wang, B. Fan, S. Xiang, G. Meng, and C. Pan, "Spectral unmixing via data-guided sparsity," *IEEE Trans. Image Process.*, vol. 23, no. 12, pp. 5412–5427, Dec. 2014.

[52] R. N. Clark, G. A. Swayze, T. V. King, A. J. Gallagher, and W. M. Calvin, "The U.S. Geological survey, digital spectral library: Version 1 (0.2 to 3.0 μm)," U.S. Geol. Surv., Denver, CO, USA, Open-File Rep. 93-592, 1993.

MOSTAFA GUDA received the B.Sc. degree from the Electronics and Communications Department, College of Engineering, Arab Academy for Science, Technology and Maritime Transport, Egypt, in 2014. He is currently pursuing the M.Sc. degree with the Department of Electronics and Communications, College of Engineering, Arab Academy for Science, Technology and Maritime Transport. His research interests are in the fields of adaptive signal processing, image processing, remote sensing, and field-programmable gate array (FPGA) hardware implementation.

SAFA GASSER received the Ph.D. degree in adaptive signal processing and control from the University of California at Santa Cruz, Santa Cruz, where she delivered the valedictory address at her graduation ceremony. She is currently an Associate Professor of communications with the Department of Electronics and Communications, Arab Academy for Science, Technology and Maritime Transport (AASTMT). Her research interests include adaptive signal processing, in addition to machine learning, remote sensing, and communications. She is a former Councilor of the IEEE AASTMT Chapter.

MOHAMED S. EL-MAHALLAWY (Member, IEEE) received the B.Sc. and M.Sc. degrees from the Electronics and Communications Department, Faculty of Engineering, Arab Academy for Science, Technology and Maritime Transport, Egypt, in 1998 and 2002, respectively, and the Ph.D. degree in image processing and pattern recognition from Cairo University, Egypt, in 2008. He was a Post-Doctoral Fellow with the Universiti Technolgi Malaysia, from 2011 to 2012. He is currently a Professor with the Electronics and Communications Department, Faculty of Engineering, Arab Academy for Science, Technology and Maritime Transport.

KHALED SHEHATA received the B.Sc. degree from the Military Technical College, Cairo, Egypt, in 1981, the M.Sc. degree from Cairo University, Egypt, in 1991, and the Ph.D. degree from the Naval Postgraduate School, Monterey, CA, USA, in 1996. He worked as a Research Assistant. He worked as a Researcher in Egypt and the Director of the VLSI Design Center, AOI, Egypt. He has been a Professor with the College of Engineering, Arab Academy for Science and Technology, since 2000. He is currently the Dean of the College of Engineering and Technology, Arab Academy of Science and Technology and Maritime Transport, Egypt. His research interests include analog and digital VLSI design, and electronics and communications system design. He has more than 100 scientific research articles in these areas.