León, G.; González, C.; Mayo Gual, R.; Mozos, D.; Quintana-Ortí, ES. (2019). Noise estimation for hyperspectral subspace identification on FPGAs. The Journal of Supercomputing. 75(3):1323-1335. https://doi.org/10.1007/s11227-018-2425-3

Noname manuscript No.
(will be inserted by the editor)

Noise Estimation for Hyperspectral Subspace Identification on FPGAs

Germán León · Carlos González · Rafael Mayo · Daniel Mozos · Enrique S. Quintana-Ortí ·

Received: date / Accepted: date

Abstract We present a reliable and efficient FPGA implementation of a procedure for the computation of the noise estimation matrix, a key stage for subspace identification of hyperspectral images. Our hardware realization is based on numerically stable orthogonal transformations, avoids the numerical difficulties of the normal equations methods for the solution of linear least squares problems (LLS), and exploits the special relations between coupled LLS problems arising in the hyperspectral image. Our modular implementation decomposes the QR factorization that comprises a significant part of the cost into a sequence of suboperations, which can be efficiently computed on an FPGA.

Keywords Hyperspectral images, subspace identification, noise estimation, least squares problems, FPGAs, high performance, energy consumption

1 Introduction

Onboard hyperspectral sensors gather large amounts of data with a high sampling rate, producing for example more than 18 Mbytes/s for AVIRIS[1] and about 1.33 Mbytes/s for HYPERION[2] [6]. Under a conventional linear mixing model [3], the spectral vectors of these data are a linear combination of a few *endmembers*, often much smaller than the number of spectral bands. A crucial

This work was supported by MINECO projects TIN2014-53495-R and TIN2013-40968-P.

G. León, R. Mayo, E.S. Quintana-Ortí
Depto. de Ingeniería y Ciencia de Computadores, Univ. Jaume I, Castellón, Spain
E-mail: {leon,mayo,quintana}@uji.es

C. González, D. Mozos
Depto. de Arquitectura de Computadores y Automática, Univ. Complutense Madrid, Spain
E-mail: {carlosgo,mozos}@ucm.es

[1] http://aviris.jpl.nasa.gov
[2] http://eo1.usgs.gov

initial step for hyperspectral applications, known as subspace identification, consists in determining the minimum dimension of the subspace that enables an accurate and economic representation of the spectral vectors, and reduces the cost and storage requirements for subsequent hyperspectral operations.

HySime [2] is an effective yet costly hyperspectral dimensionality reduction algorithm for subspace identification. Exploiting the hardware concurrency of high performance architectures is therefore crucial to deliver both (near) real-time response and moderate energy consumption for subspace identification in a number of scenarios, including biological threat detection, monitoring of chemical contamination, wildfire tracking, etc.

The key stage in HySime corresponds to the initial estimation of the noise present in the original hyperspectral image via the *noise correlation matrix*. In this paper we target the efficient implementation and execution of this initial stage of HySime on field programmable gate array (FPGAs). Our hardware formulation relies on a recent numerically reliable and *structure-aware* algorithm for noise estimation [1], based on the QR factorization, this algorithm avoids the numerical pitfalls of the normal equations method [4]. In more detail, in this paper we present a reliable and efficient FPGA implementation of the modular procedure for the computation of the noise estimation matrix. Our implementation builds upon the FPGA realization in [7] for the basic dense QR factorization, specializing that procedure to exploit the structure of the matrix operands appearing in subspace identification.

The paper is organized as follows. In Section 2 we provide a short overview of the practical method introduced in [1] to obtain the noise estimation matrix necessary in hyperspectral subspace identification. There, we also describe how to compute this matrix via a reliable and fast (structure-aware) algorithm based on the QR factorization. In Section 3 we elaborate in detail the scalar procedure for the QR factorization that underlies the structure-aware algorithm and its FPGA implementation. Finally, in Section 4 we evaluate the performance and energy consumption of our FPGA realization, and in Section 5 we close the paper with a few concluding remarks.

All experiments employ single–precision floating-point (IEEE 754) arithmetic. Although fixed point can be employed in some of the components in hyperspectral image processing, the type of numerical problems being solved in the two stages of HYSIME (linear least squares systems and eigenvalue problems) recommend the use of floating-point arithmetic, mainly because of limited representation rage of fixed precision.

We will employ the following notation in the remainder of the paper. For a $p \times q$ matrix A, partitioned column-wise as $A = (a_1, a_2, \ldots, a_q)$, we define $A_{\partial_j} = (a_1, \ldots, a_{j-1}, a_{j+1}, \ldots, a_q)$, $1 \leq j \leq q$. Furthermore, $A_{\partial_j, \partial_j}$ denotes the matrix that results from deleting the j-th column and row vectors of A, and $A_{\partial_j, k}$ is the k-th column vector of A_{∂_j}. Given a vector v, $\|v\|_2$ denotes the vector 2-norm [4].

2 Noise Estimation via the Solution of Coupled LLS problems

2.1 Computing the noise correlation matrix

Let us consider that matrix $Z \in \mathbb{R}^{n \times l}$, partitioned column-wise as $Z = (z_1, z_2, \ldots, z_l)$ contains the n spectral vectors (or image pixels), each consisting of l bands. The success of the HySime algorithm for subspace identification is based on the correlation between neighboring spectral bands of hyperspectral images [2].

The HySime algorithm for noise estimation assumes that z_j is a linear combination of the remaining $l-1$ bands; that is,

$$z_j = Z_{\partial_j} \beta_j + \xi_j, \tag{1}$$

where $\beta_j \in \mathbb{R}^{l-1}$ is the *regression vector* and $\xi_j \in \mathbb{R}^n$ is the *modeling error vector*. Least squares estimators for β_j and ξ_j are then given, respectively, by the solution of the linear least squares (LLS) problem:

$$\tilde{\beta}_j = \min_{x \in \mathbb{R}^{l-1}} \|Z_{\partial_j} x - z_j\|_2, \tag{2}$$

and the residual

$$\tilde{\xi}_j = z_j - Z_{\partial_j} \tilde{\beta}_j. \tag{3}$$

Upon solving the l "coupled" LLS problems in (2), and computing the ξ_j parameters for $j = 1, 2, \ldots, l$, the *noise correlation matrix* can be thus approximated as

$$\tilde{R}_n = \frac{1}{n} \left(\tilde{\xi}_1, \tilde{\xi}_2, \ldots, \tilde{\xi}_l\right)^T \left(\tilde{\xi}_1, \tilde{\xi}_2, \ldots, \tilde{\xi}_l\right) \in \mathbb{R}^{l \times l}. \tag{4}$$

2.2 Solution of coupled LLS problems

The previous elaboration in this section exposes that the key to obtain the noise correlation matrix lies in the reliable and efficient solution of the coupled LLS problems in (2). In numerical analysis, it is known that the numerical difficulties associated with the utilization of the normal equations method for the solution of LLS problems can be avoided by relying on the *QR factorization* [4]. In the particular case of noise estimation, in [1] we presented the following specialized structure-aware algorithm that exploits the relations between the coupled LLS problems using the reliable QR factorization.

The structure-aware algorithm in [1] commences with the computation of the "initial" QR factorization

$$Z_{\partial_1} = Q_1 R_1 = Q_1 (r_2, r_3, \ldots, r_l); \tag{5}$$

where $(r_2, r_3, \ldots, r_l)$ stands for a column-wise partitioning of R_1;

$$d_1 = Q_1^T z_1 = \begin{pmatrix} d_1^t \\ d_1^b \end{pmatrix} \begin{matrix} ; \\ \end{matrix} \begin{matrix} \} \, l-1 \\ \} \, n-l+1 \end{matrix} \tag{6}$$

and the model error vector

$$\xi_1 = Q_1 \begin{pmatrix} 0 \\ d_1^b \end{pmatrix}. \tag{7}$$

The remaining $l-1$ model error vectors, $\tilde{\xi}_j,\ j = 2, 3, \ldots, l$, can be next inexpensively obtained as follows.

Let us define:

$$Y_j = Q_1^T (Z_{\partial_j} \Pi), \tag{8}$$

where $\Pi \in \mathbb{R}^{(l-1)\times(l-1)}$ simply permutes the first column vector of Z_{∂_j} to the last column vector. Then,

$$\begin{aligned} Y_j &= Q_1^T (Z_{\partial_j} \Pi) && (9)\\ &= Q_1^T (z_2, \ldots, z_{j-1}, z_{j+1}, \ldots, z_l, z_1) && (10)\\ &= (r_2, \ldots, r_{j-1}, r_{j+1}, \ldots, r_l, d_1) && (11)\\ &= \begin{pmatrix} U_{11} & U_{12} & u_{13} \\ 0 & H_{22} & y_{23} \\ \underbrace{0}_{j-2} & \underbrace{0}_{l-j} & \underbrace{y_{33}}_{1} \end{pmatrix} \begin{matrix} \} \, j-2 \\ \} \, l-j+1 \\ \} \, n-l+1 \end{matrix} && (12) \end{aligned}$$

with U_{11} upper triangular and H_{22} upper Hessenberg (i.e., all its entries below the first subdiagonal equal zero). Therefore, the QR factorization of Y_j can be obtained via, e.g., a sequence of $l-j$ Givens rotations [4], $G_1, G_2, \ldots, G_{l-j}$, that annihilate the entries in the first subdiagonal of H_{22}; i.e.,

$$G_{l-j} \cdots G_2 G_1 Y_j = \begin{pmatrix} U_{11} & U_{12} & u_{13} \\ 0 & U_{22} & h_{23} \\ 0 & 0 & h_{33} \end{pmatrix}, \tag{13}$$

where U_{22} is upper triangular and $h_{23} = G_{l-j} \cdots G_2 G_1 y_{23}$. This can then be followed by a single Householder reflector [4], say H_j, which annihilates the entries of h_{33}, so that

$$\begin{aligned} H_j G_{l-j} \cdots G_2 G_1 Y_j &= \bar{Q}_j^T Y_j && (14)\\ &= \begin{pmatrix} U_{11} & U_{12} & u_{13} \\ 0 & U_{22} & u_{23} \\ 0 & 0 & 0 \end{pmatrix} = \bar{R}_j && (15) \end{aligned}$$

Input: $Z \in \mathbb{R}^{n \times l}$ containing the spectral vectors
Output: $\tilde{\xi}_j \in \mathbb{R}^n,\ j = 1, 2, \ldots, l$, noise estimations
Cost: $2nl^2 - l^3/6$ flops

Factorize $Z_{\partial_1} = Q_1 R_1$

$$d_1 = Q_1^T z_1 = \begin{pmatrix} d_1^t \\ d_1^b \end{pmatrix},\ \bar{\xi}_1 = \begin{pmatrix} 0 \\ d_1^b \end{pmatrix},$$

with $d_1^t \in \mathbb{R}^{l-1}, d_1^b \in \mathbb{R}^{n-l+1}$

for $j = 2, 3, \ldots, l$

Assemble $Y_j = (r_2, \ldots, r_{j-1}, r_{j+1}, \ldots, r_l, d_1)$

Factorize $Y_j = \bar{Q}_j \bar{R}_j$, with $\bar{R}_j$ upper triangular and $\bar{Q}_j^T = H_j G_{l-j} \cdots G_2 G_1$

$$d_j = \bar{Q}_j^T z_j = \begin{pmatrix} d_j^t \\ d_j^b \end{pmatrix},\ \bar{\xi}_j = \bar{Q}_j \begin{pmatrix} 0 \\ d_j^b \end{pmatrix},$$

with $d_j^t \in \mathbb{R}^{l-1}, d_j^b \in \mathbb{R}^{n-l+1}$

end

$$\left(\tilde{\xi}_1, \tilde{\xi}_2, \ldots, \tilde{\xi}_l\right) = Q_1 \left(\bar{\xi}_1, \bar{\xi}_2, \ldots, \bar{\xi}_l\right)$$

Figure 1 HySimeSA: Noise estimation based on the structure-aware algorithm.

is upper triangular. In other words, $Y_j = \bar{Q}_j \bar{R}_j$ is a QR factorization of Y_j, and therefore,

$$d_j = \bar{Q}_j^T Q_1^T z_j = \bar{Q}_j^T r_j, \tag{16}$$

with the proper partitioning, yields the required approximation of the noise vector

$$\tilde{\xi}_j = Q_1 \bar{Q}_j \begin{pmatrix} 0 \\ d_j^b \end{pmatrix}. \tag{17}$$

This "structure-aware" algorithm is formally stated in Figure 1. The initial QR factorization requires $2l^2(n - l/3)$ floating-point arithmetic operations (flops), and the computation of d_1, $\tilde{\xi}_1$ contributes a negligible cost to that figure. The triangular matrices $\bar{R}_j$ are not explicitly constructed, and the orthogononal transformations corresponding to $\bar{Q}_j$ can be computed and applied with a cost of only $l^2/2$ flops per band/factorization. The subsequent computations for d_j in (16) and $\tilde{\xi}_j$ in (17) contribute an insignificant cost. In total, this structure-aware algorithm performs to $2nl^2 - l^3/6$ flops. At this point we note that, as in practical hyperspectral imaging $n \gg l$, the most significant part of this computational cost comes from the initial QR factorization.

3 FPGA Realization of the QR Factorization

3.1 Blocked algorithm

In subspace identification problems for hyperspectral images, the QR factorization in (5) involves a matrix operand $\hat{Z}_{\partial_1}$ comprising up to several hundreds of thousands of rows (image pixels) but a few hundreds of columns only (bands). Our formulation of the QR factorization takes this into account to decompose it into a sequence of suboperations that partition the kernel by blocks and allow an efficient implementation on an FPGA. Concretely, consider $\hat{Z} = Z_{\partial_1} \in \mathbb{R}^{n \times \hat{l}}$, with $\hat{l} = l - 1$, partitioned by blocks of rows as

$$\hat{Z} = \begin{pmatrix} \hat{Z}_1 \\ \hat{Z}_2 \\ \vdots \\ \hat{Z}_{n_t} \end{pmatrix}, \tag{18}$$

where, for simplicity, we assume that $n = n_t \cdot \hat{l}$, so that each block $\hat{Z}_j$ contains $\hat{l}$ rows. (In case n is not an integer multiple of $\hat{l}$, we simply need to fill $\hat{Z}$ with the appropriate number of rows, with all their entries set to zeros. Given that $n \gg \hat{l}$, this produces a minor increase in the cost of the following algorithm.) The blocked implementation then begin by computing the QR factorization of the last block:

$$\hat{Z}_{n_t} = \hat{Q}_{n_t} \hat{R}_{n_t}, \tag{19}$$

to then proceed upwards, successively computing the QR factorizations

$$\begin{pmatrix} \hat{Z}_j \\ \hat{R}_{j+1} \end{pmatrix} = \hat{Q}_j \hat{R}_j \tag{20}$$

for $j = n_t - 1, n_t - 2, \ldots, 1$. Here we note that $\hat{R}_{j+1}$ is already an upper triangular matrix, obtained from the QR factorization in the previous step. By taking into account the triangular structure of this operand, the blocked algorithm for the QR factorization of $\hat{Z}$ then exhibits the same computational cost as that of a conventional QR factorization.

3.2 Scalar procedure

Given a nonzero vector $x \in \mathbb{R}^n$, the Householder reflector

$$H = \mathsf{House}(x) = I_n - \tau v v^T \tag{21}$$

, where

$$v = x + \alpha e_1 \tag{22}$$

$$\tau = 2/(v^T v) \tag{23}$$

$$\alpha = \pm |||x||_2 \tag{24}$$

, I_n denotes the square identity matrix of order n and e_1 is the first column of I_n, satisfies $y = Hx = \mp||x||_2 e_1$ [4]; that is, all entries of x are annihilated by the application (from the left) of the Householder reflector H, except the first entry of x which, after the application, becomes $\mp||x||_2$.

Figure 2 illustrates the calculation of the QR factorization, via Householder reflectors, of a matrix of the form $\begin{pmatrix} \bar{Z} \\ \bar{R} \end{pmatrix}$ where $\bar{Z}, \bar{R} \in \mathbb{R}^{\hat{l} \times \hat{l}}$, and $\bar{R}$ is upper triangular; see (20). The scalar procedure uses the FLAME notation [5], and mimics routine GEQR2 from the *Linear Algebra Package* (LAPACK) [8]. Internally, the scalar procedure relies on routine LARFG to generate a Householder reflector for the vector $\left(\omega_{11},\ z_{21},\ r_{01}^T,\ \rho_{11}\right)^T$. Routine LARF then applies this reflector (from the left) to the trailing submatrix composed of $z_{12}^T,\ Z_{22},\ R_{02}$ and r_{12}^T. The Householder reflector H is not explicitly built, but applied implicitly using the parameters v, τ. In particular, note that the application of the Householder reflector to a matrix A can be performed as $HA = (I - \tau v v^T)A = A - \tau v (v^T A)$, which boils down to a matrix-vector product, $w = v^T A$, followed by a (scaled) rank-1 update, $A = A - \tau v w^T$.

In practice, the upper triangular factor overwrites the corresponding entries of $\bar{Z}$ and $\bar{R}$. Furthermore, the parameters v_j and τ_j that define the Householder reflector H_j (which annihilates the subdiagonal entries in the j-th column of the matrix) are respectively stored using the annihilated entries of the column plus the j-th entry of an additional vector of size $\hat{l}$. (Here, the first entry of v equals 1 and does not need to be explicitly stored.)

3.3 FPGA implementation

Figure 3 displays the hardware architecture that implements the scalar procedure for the QR factorization of $W = \begin{pmatrix} \bar{Z} \\ \bar{R} \end{pmatrix}$. Module Memory provides the input data matrices $\bar{Z}$, $\bar{R}$; module House calculates the Householder reflector; and module RowHouse applies the Householder reflector to the appropriate blocks of the data matrix.

As described in the previous two subsections, matrix $\hat{Z} = Z_{\partial_j} \in \mathbb{R}^{n \times \hat{l}}$ is partitioned, passed to and processed by the FPGA in blocks of dimension $\hat{l} \times \hat{l}$, starting from the bottom and proceeding upwards. While processing two of these blocks, say $\hat{Z}_j$ and $\hat{Z}_{j+1}$, the block immediately above them (i.e., $\hat{Z}_{j-1}$) is transferred to the FPGA in order to overlap communication with computation and avoid idle periods. Module Memory is composed of three banks: input, upper and lower. The input bank stores the block in transference while the remaining two banks provide the information to the rest of the processing

Algorithm: $\left[\bar{Z}, \bar{R}\right] := \text{QR\_2x1}(\bar{Z}, \bar{R})$

Partition $\bar{Z} \rightarrow \left(\begin{array}{c|c} Z_{TL} & Z_{TR} \\ \hline Z_{BL} & Z_{BR} \end{array}\right)$, $\bar{R} \rightarrow \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array}\right)$

where Z_{TL}, R_{TL} are both 0×0

while $n(Z_{TL}) < n(\bar{Z})$ **do**

Repartition

$$\left(\begin{array}{c|c} Z_{TL} & Z_{TR} \\ \hline Z_{BL} & Z_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} Z_{00} & z_{01} & Z_{02} \\ \hline z_{10}^T & \omega_{11} & z_{12}^T \\ \hline Z_{20} & z_{21} & Z_{22} \end{array}\right), \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} R_{00} & r_{01} & R_{02} \\ \hline r_{10}^T & \rho_{11} & r_{12}^T \\ \hline R_{20} & r_{21} & R_{22} \end{array}\right)$$

where ω_{11}, ρ_{11} are both 1×1

$H := \text{House}\left(\begin{array}{c} \omega_{11} \\ \hline z_{21} \\ \hline r_{01} \\ \hline \rho_{11} \end{array}\right)$	LARFG returns τ, v s.t. $H = I - \tau v v^T$
$\left(\begin{array}{c} z_{12}^T \\ \hline Z_{22} \\ \hline R_{02} \\ \hline r_{12}^T \end{array}\right) := H \left(\begin{array}{c} z_{12}^T \\ \hline Z_{22} \\ \hline R_{02} \\ \hline r_{12}^T \end{array}\right)$	LARF applies H via τ, v

Continue with

$$\left(\begin{array}{c|c} Z_{TL} & Z_{TR} \\ \hline Z_{BL} & Z_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} Z_{00} & z_{01} & Z_{02} \\ \hline z_{10}^T & \omega_{11} & z_{12}^T \\ \hline Z_{20} & z_{21} & Z_{22} \end{array}\right), \left(\begin{array}{c|c} R_{TL} & R_{TR} \\ \hline R_{BL} & R_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} R_{00} & r_{01} & R_{02} \\ \hline r_{10}^T & \rho_{11} & r_{12}^T \\ \hline R_{20} & r_{21} & R_{22} \end{array}\right)$$

endwhile

Figure 2 Unblocked algorithm for the QR factorization using Householder reflectors. $n(\cdot)$ is a function that returns the number of columns of its input argument.

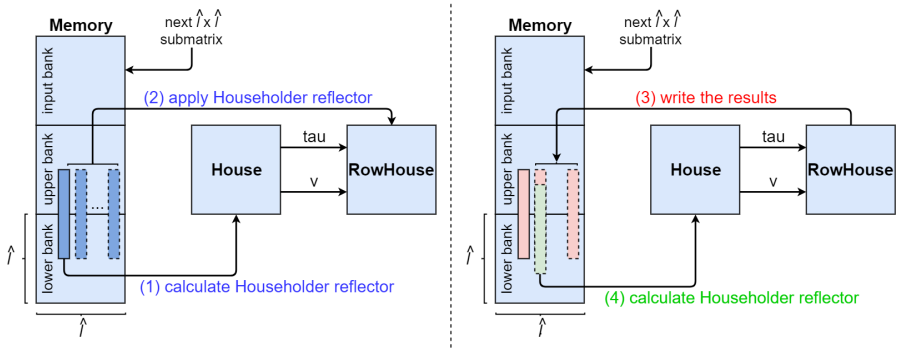


Figure 3 Hardware architecture used to implement QR factorization and steps involved in an intermediate iteration.

system. When the factorization of a submatrix is completed, the roles of the banks are rotated so that the upper bank becomes the lower bank, the input bank becomes the upper bank, and the lower bank becomes the input one.

The calculation modules (House and RowHouse) work in parallel. As soon as the first column is updated, the Householder reflector for the next iteration can be calculated. Thus, steps (3) and (4) in Figure 3 are executed simultaneously. The generation of the Householder reflector exhibits a reduced degree

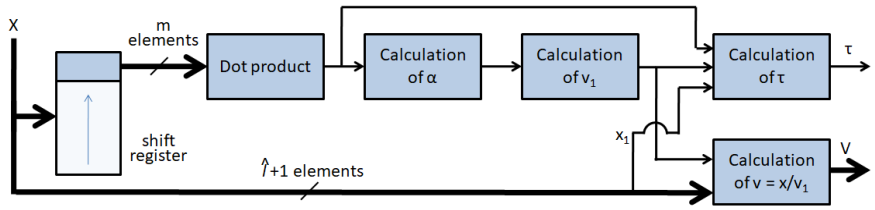


Figure 4 Hardware architecture used to implement House module.

of parallelism and does not have a contiguous source of data. Therefore, the design of module House aims to offer low latency and consume a small number of digital signal processing (DSP) resources. The majority of DSPs are dedicated to the implementation of the RowHouse module to take full advantage of the parallelism of this operation.

Figure 4 shows the design of the House module to implement the calculation of v (22) and τ (23).

To perform these calculations, we must first obtain α (24), which implies the realization of the dot–product of x. Following the design policy we have just described, the inner dot–product of the reflector calculation has a reduced size ($m << \hat{l} + 1$). To obtain the final result of the dot–product, we use an accumulator that adds the sequence of partial dot–products obtained. The accumulator consists of a low latency adder and a complex automaton that matches the partial sums, in order to reduce the number of sums until reaching the total sum. Therefore, a circuit is required which takes the input of $\hat{l} + 1$ elements and sequentially supplies the input of m elements to the dot–product operator. For this circuit, shift registers are used in order to reduce routing resources. Simulations for $\hat{l} = 256$ determines that the lowest latency is obtained with a size of 16 elements in each partial dot–product.

When the dot–product is obtained, then we calculate α and the first element of v, following Equations (24) and (22) respectively. At this point, we have the dot–product of x, α and the first element of v, so we just can calculate τ and v. To avoid the need of the final vector v to calculate τ, we rewrite Equation (23) in the following sense:

$$\tau = 2/((x^T x) - x_1^2 + v_1^2)/v_1^2) \tag{25}$$

At the end of the reflector calculation, a division is made to all elements of the column. This calculation is performed in parallel to the last calculation of τ, which has a latency of 16 cycles. Thus, with 16 dividers in parallel we can make the division of a column of 256 elements without increasing the overall latency of the operation.

Once we have calculated the Householder reflector we must apply it to the rest of the matrix. In Subsection 3.2 we explain that the application of the Householder reflector H to a matrix A can be performed as $HA = A - \tau v(v^T A)$. Figure 5 shows the design of the RowHouse module following this Equation.

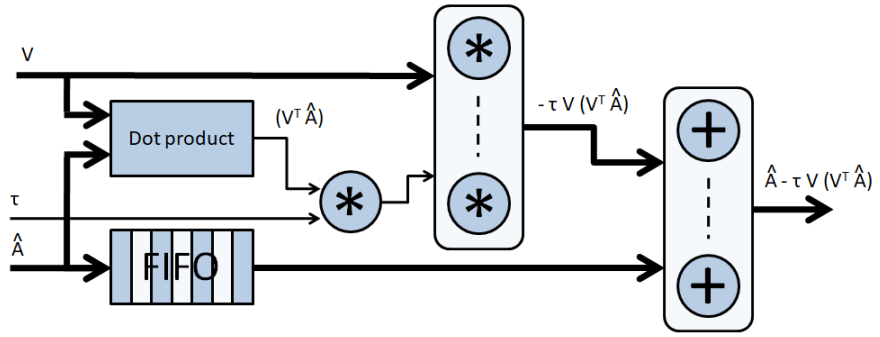


Figure 5 Hardware architecture used to implement RowHouse module.

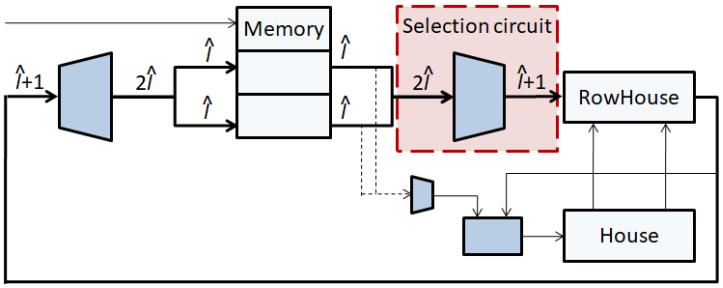


Figure 6 General scheme of the QR factorization with the selection circuit.

To apply the Householder reflector calculated in the previous stage we will proceed as follows. First, one column of matrix A is introduced and the dot–product with v is calculated. Second, the result of the dot–product is multiplied by τ (changing its sign). Third, we multiply the scalar result calculated in the previous step by the vector v. Finally, the column of A initially introduced is multiplied by the vector calculated in the previous step.

There are two versions of module RowHouse. The first one utilizes the maximum number of floating–point units (following a non-blocking pipeline architecture). The second version adapts the design depending on the number of resources, implementing operators of smaller size and multiplexing the data entry (following a blocking pipeline architecture) in time. Following the design policy we have described previously, in this paper we use the first one to take full advantage of the parallelism of this operation, so we can introduce the different columns of A one by one in each clock cycle.

Each iteration of a submatrix calculation needs $\hat{l}+1$ elements from the $2\hat{l}$ elements column provided by the lower and upper banks. Specifically, the i-th iteration will work with the sub–column M_i=$\{m_i ... m_{n+1+i}\}$ where $\forall m_i \in M$. Figure 6 shows a general scheme of the QR factorization with the selection circuit (for simplicity in Figure 3 it did not appear). The design of a selection circuit suitable for FPGA architecture is very important to avoid congestion

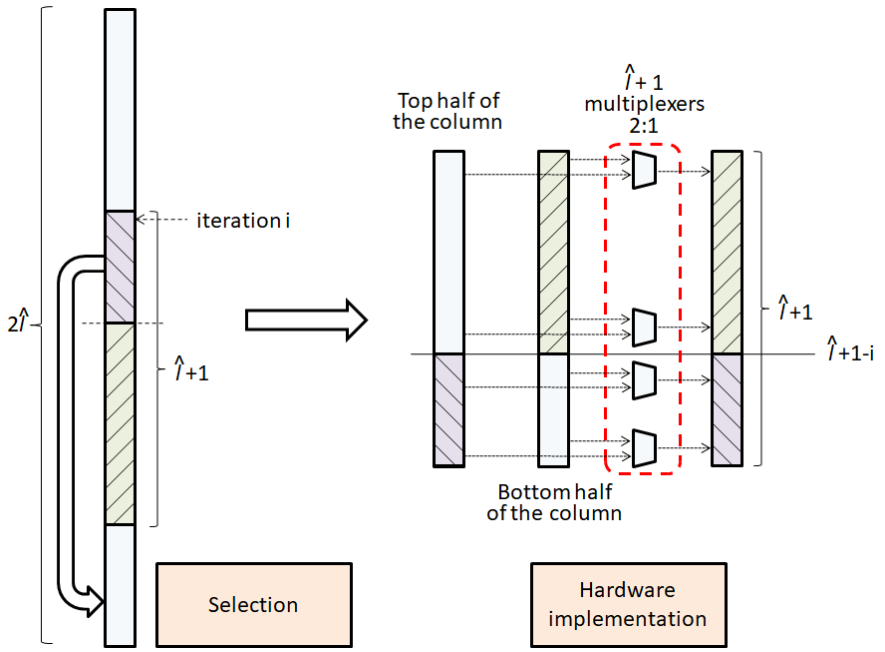


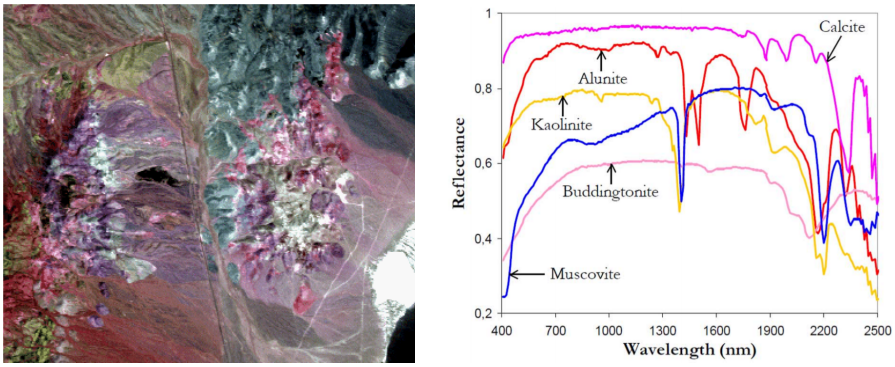
Figure 7 Hardware architecture used to implement the selection circuit.

problems when performing the routing of the circuit, therefore it deserves a detailed explanation.

To simplify the selection circuit of the sub–column, M_i will be exchanged, so for the i-th iteration the permuted column Mp_i will be $\{m_{n+1}...m_{n+1+i},m_i...m_n\}$ [see Figure 7]. Thus, it is achieved that the k component can only be m_k if $k > i$, or m_{n+1+k} otherwise, so the hardware cost will be $\hat{l}+1$ 2:1 multiplexers. If we not permute the column, the hardware cost is corresponding to a barrel shifter with hardware cost of $\hat{l}+1$ $2\hat{l}$:1 multiplexers. The biggest problem of a barrel shifter of the size required for this problem, is the problem of congestion in routing resources that prevents its implementation on current FPGAs.

4 Experimental Results

This Section is organized as follows. In Subsection 4.1 we describe the FPGA board used in our experiments. Subsection 4.2 describes the hyperspectral data sets that will be used for demonstration purposes. Finally, Subsection 4.3 shows the resources used for our hardware implementation, the processing time for the data sets and the power consumption.



(a) (b)

Figure 8 (a) False color composition of the AVIRIS hyperspectral over the Cuprite mining district in Nevada. (b) U.S. Geological Survey mineral spectral signatures of the exposed minerals of interest.

4.1 FPGA Architecture

The hardware architecture described in Subsection 3.3 has been implemented directly coded using VHDL language for the specification of the QR factorization. Moreover, we have used the Vivado 2016 environment and the Embedded Development Kit (EDK) environment to specify the complete system. The full system has been implemented on a VC709 board, a reconfigurable board with a single Virtex–7 XC7VX690T, two DDR3 SDRAM DIMM slots which holds up to 4 GB each one, an RS232 port, and some additional components not used by our implementation.

The Xilinx Virtex–7 XC7VX690T FPGA has a total of 866,400 slice registers, 433,200 slice look-up tables (LUTs) and 134,381 LUT-FF pairs. In addition, the FPGA includes some heterogeneous resources such as 3,600 DSP48E1s and 1,470 distributed block RAMs. In our implementation, we took advantage of these resources to optimize the design. Block RAMs are used to implement the FIFOs and the memories, so the vast majority of the slices are used for the implementation of the QR factorization together with the DSP48E1s.

4.2 Hyperspectral Image Data Sets

The hyperspectral datasets used in these experiments are the well-known AVIRIS Cuprite scene, available online in reflectance units[3], the AVIRIS World Trade Center (WTC) scene and the HYDICE Washington DC Mall scene[4]. These scenes have been widely used as standard hyperspectral benchmarks.

[3] http://aviris.jpl.nasa.gov/freedata
[4] https://engineering.purdue.edu/ biehl/MultiSpec/hyperspectral.html

(a) (b)

Figure 9 (a) False color composition of an AVIRIS hyperspectral image collected by NASAs Jet Propulsion Laboratory over lower Manhattan on September 16, 2001. (b) Location of thermal hot spots in the fires observed in World Trade Center area, available [online]: http://pubs.usgs.gov/of/2001/ofr-01-0429/hotspot.key.tgif.gif.

The AVIRIS Cuprite scene [see Figure 8(a)] comprises a relatively large area (350 lines by 350 samples and 20-m pixels) and 224 spectral bands between 0.4 and 2.5 μm, with nominal spectral resolution of 10 nm. Bands 1-3, 105-115 and 150-170 were removed prior to the analysis due to water absorption and low SNR in those bands. The site is well understood mineralogically, and has several exposed minerals of interest including *alunite*, *buddingtonite*, *calcite*, *kaolinite* and *muscovite*. Reference ground signatures of the above minerals, available in the form of a U.S. Geological Survey library (USGS)[5], are shown in Figure 8(b)].

For the second image, the instrument was flown by NASA's Jet Propulsion Laboratory over the World Trade Center (WTC) area in New York City on September 16, 2001, just five days after the terrorist attacks that collapsed the two main towers and other buildings in the WTC complex [see Figure 9]. The data set consists of 614×512 pixels, 224 spectral bands and a total size of (approximately) 140 MB. The leftmost part of Figure 9 shows a false color composite of the data set selected for experiments using the 1682, 1107 and 655 nm channels, displayed as red, green and blue, respectively. The rightmost part of Figure 9 shows a thermal map centered at the region where the towers collapsed.

Figure 10 shows a simulated color IR view of an airborne hyperspectral data flightline over the Washington DC Mall provided with the permission of Spectral Information Technology Application Center of Virginia who was responsible for its collection. The sensor system used in this case measured pixel response in 210 bands in the 0.4 to 2.4 m region of the visible and infrared spectrum. Bands in the 0.9 and 1.4 m region where the atmosphere is opaque

[5] http://speclab.cr.usgs.gov/spectral-lib.html

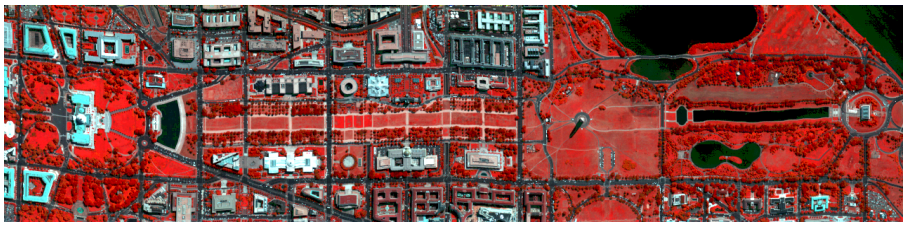


Figure 10 IR view of an airborne hyperspectral data flightline of HYDICE sensor over the Washington DC Mall, available [online]: https://engineering.purdue.edu/ biehl/MultiSpec/hyperspectral.html.

have been omitted from the data set, leaving 191 bands. The data set contains 1208 scan lines with 307 pixels in each scan line. It totals approximately 145 Megabytes. The image at left was made using bands 60, 27, and 17 for the red, green, and blue colors respectively.

4.3 Performance Evaluation

Table 1 Summary of resource utilization on the Virtex–7 XC7VX690T.

Resource	LUT	LUTRAM	BRAM	DSP
Available	433,200	174,200	1,470	3,600
Memory module	67,147	0	768	0
House module	36,797	886	0	62
RowHouse module	148,518	9,242	129	2,056
QR Factorization	278,357	10,125	1,013	2,118

Table 2 Execution time (in seconds) for our FPGA implementation of the QR factorization.

	AVIRIS Cuprite	AVIRIS WTC	HYDICE Washington DC Mall
	44MB	140MB	145MB
	188 bands	224 bands	191 bands
Execution time	1.31	3.36	3.84

We study three metrics of our FPGA implementation: Board usage, computational performance and power consumption.

Table 1 shows the resources necessary for our hardware implementation for a QR factorization of size $\hat{l} = 256$ for the different modules and for the complete system. Although we can think due the total resources utilization is around the 65% we can introduce improvements that requires additional resources or even jointly implements other algorithms, based on our experience, we can say that congestion problems will occur, making circuit routing impossible.

In reference to performance, in order to factorize a block of dimension 256×256, the implementation requires a total of 109,669 clock cycles and proceeds at 40 MHz. Table 2 shows the execution time for the initial QR factorization using the AVIRIS Cuprite and WTC scenes, and the HYDICE Washington DC Mall scene. If we compare the execution times of the AVIRIS Cuprite and AVIRIS WTC images we can note that it not scale with the image sizes (the processing time of the AVIRIS WTC image should be $140MB/40MB = 3.5$ times the processing time of the AVIRIS Cuprite image, instead of 2.57). This is due to the fact that as the number of bands increases, fewer penalties occur between the House and RowHouse modules due to the wait of this last module to finish the calculation of the Householder reflector. The same idea applies to the HYDICE Washington DC Mall image, which despite having a very similar size to the AVIRIS WTC image, has a longer processing time because it has fewer bands.

Table 3 Execution time (in seconds) for the HySimeSA algorithm using single–precision on an Intel Xeon E5645 processor [1].

	AVIRIS Cuprite	AVIRIS WTC
1 core	2.28	8.24
2 cores	1.44	4.93
4 cores	1.05	3.54
6 cores	0.99	3.43

It is interesting to compare our implementation with existing parallel implementations in other platforms like the one we can found in [1], where a multicore implementation on an Intel Xeon E5645 processor is presented. In that work, authors show the execution time of the entire HySimeSA algorithm, but given that the initial QR factorization requires $2l^2(n - l/3)$ flops and the rest of operations requires $l^3/2$ flops, we can consider that the execution time of the initial QR factorization is practically the execution time of the entire HySimeSA algorithm. Table 3 resumes the execution time using single–precision floating-point arithmetic with different number of cores for the AVIRIS Cuprite and WTC scenes. Comparing Tables 2 and 3, we can see that our implementation in the Xilinx Virtex–7 XC7VX690T FPGA obtain a similar result than using 6 cores of the Intel Xeon E5645 processor for larges images.

Looking at power consumption, the static power consumption is 0.58 Watts while the dynamic power consumption is 4.03 Watts, for a total of 4.61 Watts. This power consumption is significantly lower when compared to the parallel multicore implementation for an Intel Xeon E5645 [1], whose Thermal Design Power (TDP) is 80 watts (for calculations like the QR factorization, it is normal to dissipate a power that will be close to that TDP).

5 Concluding Remarks

We have described an FPGA implementation of the HySime algorithm for subspace identification that builds upon the numerically-reliable QR factorization for solving the LLS problems in this method. Furthermore, our implementation exploits the tall-and-skinny structure of hyperspectral images, with many more rows than columns (i.e., pixels than spectral bands), to compute this factorization via a specialized, structure-aware algorithm that proceeds by blocks, from bottom upwards.

Our experimental results using an actual implementation on a Xilinx Virtex-7 XC7VX690T shows the amount of resources employed by each model of the FPGA and the global usage using three well-known benchmarks for hyperspectral imaging. In addition, these results indicate a fair rate of floating-point arithmetic operations per second (given the moderate frequency of the FPGA) and a very appealing low power dissipation, of around 4.61 Watts only, compared with the several dozens of Watts that are usually required by a conventional multicore architecture.

References

1. Benner, P., Novaković, V., Plaza, A., Quintana-Ortí, E.S., Remón, A.: Fast and reliable noise estimation for Hyperspectral subspace identification. IEEE Geoscience and Remote Sensing Letters **12**(6), 1199–1203 (2015)
2. Bioucas-Dias, J., Nascimento, J.: Hyperspectral subspace identification. IEEE Trans. Geo. Rem. Sens. **46**, 2435–2445 (2008)
3. Bioucas-Dias, J., Plaza, A., Dobigeon, N., Parente, M., Du, Q., Gader, P., Chanussot, J.: Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches. IEEE JSTARS **5**(2), 354–379 (2012)
4. Björck, A.: Numerical Methods for Least Squares Problems. Society for Industrial and Applied Mathematics (SIAM) (1996)
5. Gunnels, J.A., Gustavson, F.G., Henry, G.M., van de Geijn, R.A.: FLAME: Formal linear algebra methods environment. ACM Trans. Math. Software **27**(4), 422–455 (2001). URL http://doi.acm.org/10.1145/504210.504213
6. Kerekes, J., Baum, J.: Spectral imaging system analytical model for subpixel object detection. IEEE Trans. Geo. Remote Sens. **40**(5), 1088–1101 (2002)
7. León, G., González, C., Mayo, R., , Quintana-Ortí, E.S., Mozos, D.: Energy-efficient QR factorization on FPGAs. In: Proc. 17th Int. Conf. Computational and Mathematical Methods in Science and Engineering (CMMSE 2017). Cádiz, Spain (2017)
8. Anderson et al, E.: LAPACK Users' guide, 3rd edn. SIAM (1999)