TARGET DETECTION OF HYPERSPECTRAL IMAGERY ON FPGAS

DETECCIÓN DE OBJETIVOS EN IMÁGENES HIPERESPECTRALES SOBRE FPGAS

Juan Martín Bárez Alonso

GRADO EN INGENIERÍA DE COMPUTADORES
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin de Grado

Septiembre 2020

Directores:

Carlos González Calvo
José Manuel Mendías Cuadros

Resumen en castellano

En los últimos años ha ocurrido un resurgimiento de la carrera espacial motivado especialmente por empresas comerciales. Sus aeronaves son equipadas con una multitud de sensores, siendo uno de ellos las cámaras hiperespectrales. Este tipo de cámaras toma imágenes en cientos de bandas espectrales diferentes, con el objetivo de proporcionar información sobre el terreno.
A causa del gran tamaño de las imágenes hiperespectrales, estas son enviadas a la Tierra para su procesado, con el consecuente coste de transmisión y almacenamiento. Preferentemente estas imágenes deberían procesarse o comprimirse in situ para enviar solo una fracción de los datos obtenidos. Dados el entorno espacial y las características este tipo de algoritmos, las FPGAs o ASICs se postulan como un sistema óptimo para su implementación.
Este trabajo presenta una implementación sobre FPGAs del algoritmo Reed-Xiaoli de detección de anomalías para imágenes hiperespectrales. Para su implementación se ha realizado un análisis de las operaciones del algoritmo, centrada en una versión en punto flotante y otra en aritmética de enteros, y de las repercusiones que tienen ciertas decisiones con la precisión que se alcanza. Demostrando de esta manera cómo algoritmos complejos con operaciones en punto flotante pueden ser ejecutados en FPGAs al transformarlos para utilizar aritmética de enteros.

Palabras clave

Imágenes hiperespectrales, Algoritmo RX, Aritmética de punto flotante, Aritmética de enteros, Hardware reconfigurable, VHDL.

Abstract

In recent years there has been a resurgence in the space race, motivated especially by commercial companies. Their aircrafts are equipped with a multitude of sensors, one of them being hyperspectral cameras. This type of camera takes images in hundreds of different spectral bands, with the aim of providing information of the ground.
Because of the large size of hyperspectral images, they are sent to ground stations for processing, with the consequent cost of transmission and storage. Preferably these images should be processed or compressed on site and only a fraction of the data obtained should be sent. Given the spatial environment and the characteristics of these types of algorithms, FPGAs or ASICs are postulated as an optimal system for their implementation.
This work presents an FPGA implementation of the Reed-Xiaoli algorithm of anomaly detection for hyperspectral images. For its implementation, an analysis of the operations of the algorithm has been made, centered in a floating point version and another one in integer arithmetic, and of the repercussions that certain decisions have with the precision that is reached. Thus, demonstrating how complex algorithms with floating point operations can be executed in FPGAs by transforming them to use integer arithmetic.

Keywords

Hyperspectral images, RX algorithm, floating point arithmetic, integer arithmetic, reconfigurable hardware, VHDL

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and objectives

Space exploration serves many purposes, the most obvious being gathering information about our planet and its surroundings. For this purpose, sensors capable of gathering information are created, such as antennas or telescopes that are used both from the Earth and sent aboard spaceships. One of those are hyperspectral cameras, which take pictures in hundreds of different bands. Their data allows to find objects, detect materials or identify processes. As technology advances, these sensors evolve requiring appropriate processing solutions to interpret the data or compress it and send it to ground.

The objective of this work is the implementation of one of these algorithms in a way that the processing in the aircraft is preferable to the transmission of the raw data.
For this purpose, different algorithms will be evaluated and one will be chosen, more specifically, the Reed Xiaoli algorithm. A first implementation of the floating point algorithm will be done and its transformation to integer arithmetic will be studied. This step is necessary because the major impediment of these algorithms to be implemented in hardware is the high number and complexity of its operations. With the arithmetic well defined, its implementation will be adapted and a comparison of accuracy and performance between the first floating point version and the integer version will be made.

1.2 Related work

In this section we will review the state of the art on the use of FPGAs (field programmable gate arrays) in space applications in general and implementations related to the algorithm developed in this work.

In [15] a study is carried out on the current situation of the use of FPGA on board aircraft for hyperspectral analysis and implementations of two algorithms, ISRA and N-FINDR, are presented. The results show numerous advantages of FPGA over other types of solutions such as GPUs, such as its smaller size and weight and its resistance to radiation. In addition, their reconfiguration capabilities even after the system is launched are emphasized.

In [11] an implementation of a target generation algorithm is presented. It uses an inverter based on the Gauss Jordan elimination method, same as the one that will be used in this work, and the results are presented on the same platform.

In [9] a study of the same algorithm that will be implemented in this work is carried out on an FPGA. The results are positive with a reduction in calculation time compared to software-based solutions. However, this study was performed only on a floating point implementation which limited it to the processing of previously dimensionally reduced hyperspectral images.

In [21] the processing of data on board is proposed with the aim of reducing network usage and accelerating data processing. One of the steps of this processing is also the RX algorithm that will be implemented here.

In general, the previous works present good results on these systems and expectations to an increase of use, motivated by more and more advanced image capture systems that take the hardware to their limits.

1.2.1 Reconfigurable hardware

FPGAs are chips based on an array of configurable blocks called CLBs interconnected by an also configurable network (see Figure 1.1).

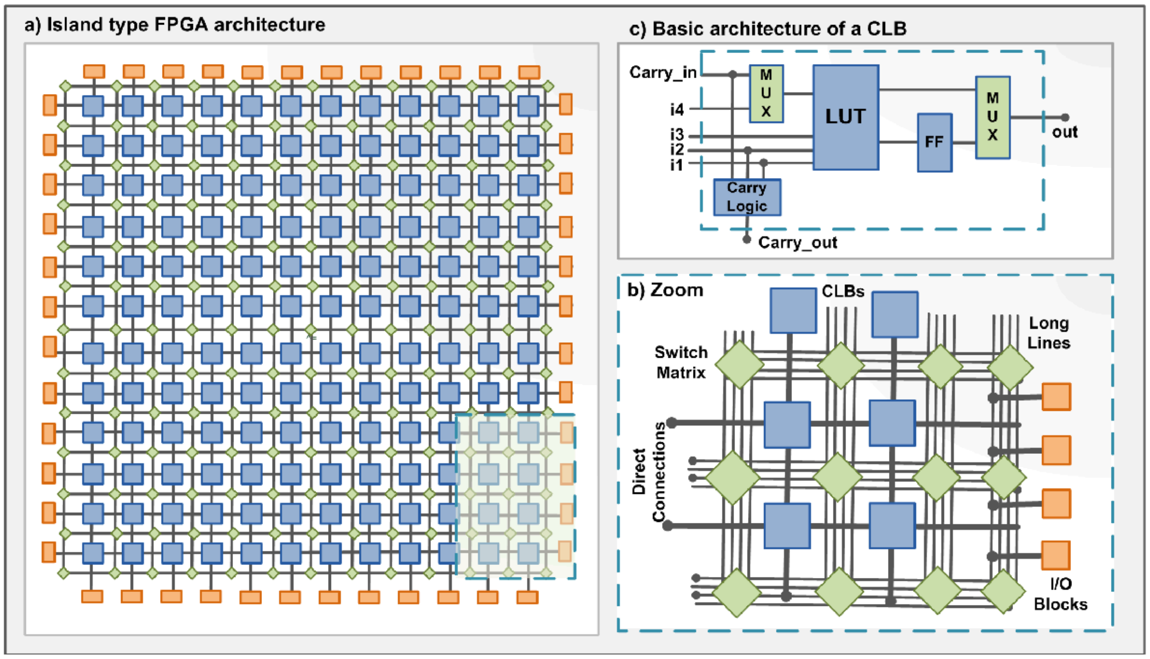


Figure 1.1: *Generic FPGA hardware architecture. Depicted is the basic structure of a CLB and the interconnecting network.*

Unlike general purpose systems such as CPUs or GPUs, this architecture allows the design of algorithms with arbitrary calculation widths, resulting in very good performance when processing images, both in power and time (see Figure 1.2).

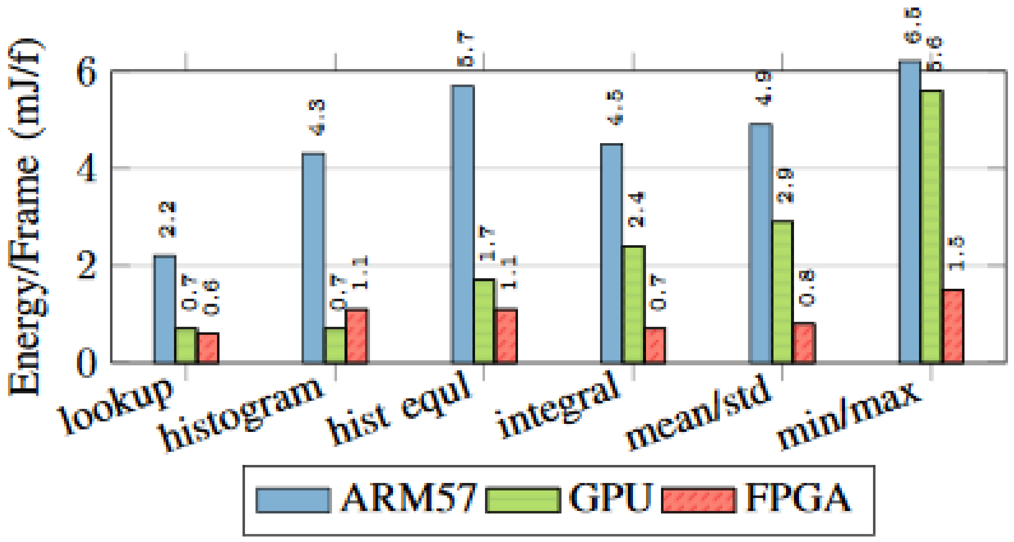


Figure 1.2: *In image analysis kernels such as lookuptable, histogram, and histogram equalization, the energy/frame consumption of the FPGA achieves an average reduction of 1.2× compared to the GPU. For kernels with more branching conditions and complex memory access patterns, such as integral image, mean/std, and min/max locations, the FPGA's implementation achieved an average reduction ratio of 3.5× compared to the GPU[18]*

The space environment not only limits avaiable power, it also presents a challenge in the form of ionizing radiation. As this is one of the target markets for FPGA manufacturers, there are numerous chips with the necessary radiation resistance certifications.
ASICs provide the same design flexibility as FPGAs but their manufacturing rigidity allows them to achieve better performance by only including the specific design logic. However, their cost in small to medium scale projects is prohibitive. In addition, the flexibility of FPGAs allows reconfiguration already in the ship, allowing the use of different algorithms or bug fixes.

In addition, since most algorithms share certain basic operations such as storage or high-precision arithmetic operations, FPGA manufacturers include certain prefabricated blocks in the circuit, which although remove some flexibility, provide better performance than the same logic in CLBs. These blocks are mainly RAM blocks and DSPs that allow a variety of operations, including multiplication or accumulation. This heterogeneous architecture (see Figure 1.3) allows the implementation of high-performance algorithms where it would be impossible using only logic and brings FPGAs a little closer to the scope of ASICs [22].

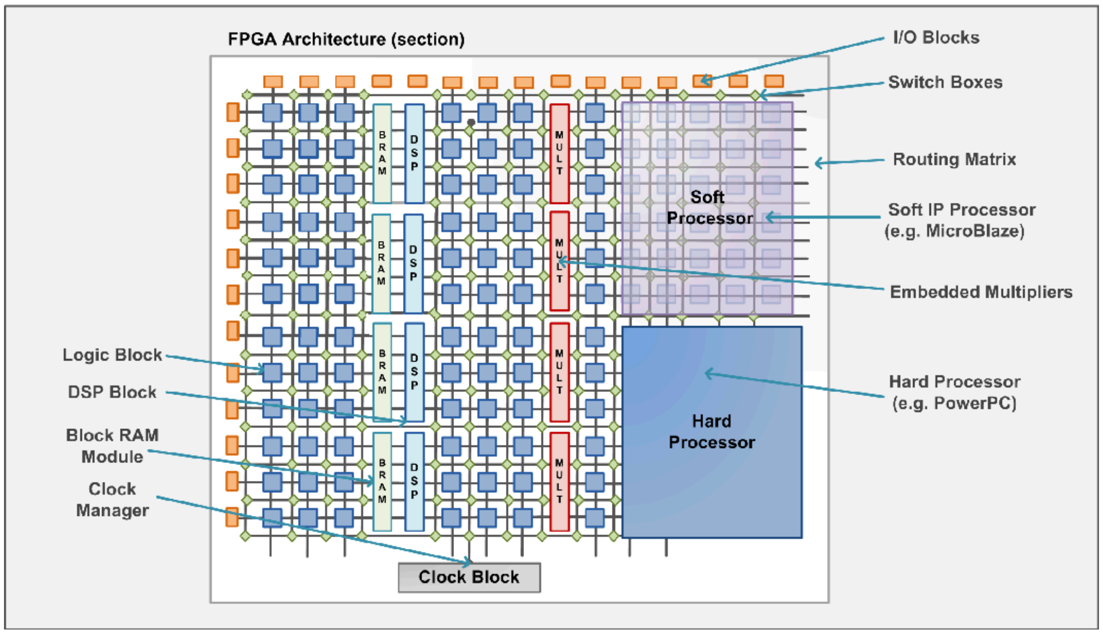


Figure 1.3: *Heterogeneous FPGA, depicting general configurable resources, DSPs, BRAMs and soft (implemented in logic) and hard (prefabricated) CPUs*

1.2.2 Hyperspectral imagery

A standard camera captures images composed of 3 distinct light bands, which we humans perceive as red, green and blue. Their spectra correspond to high wavelengths between 564 and 580nm for red, mediums between 534 and 545nm for green and short ones between 420 and 440nm for blue. The rest of the spectra are invisible to our eyes.
However, hyperspectral cameras capture information in many more spectra, both between the visible bands and outside them, allowing a much wider spectrum to be analyzed. These bands are usually shown as a third dimension, giving the name of hypercube to these images (see Figure 1.4).

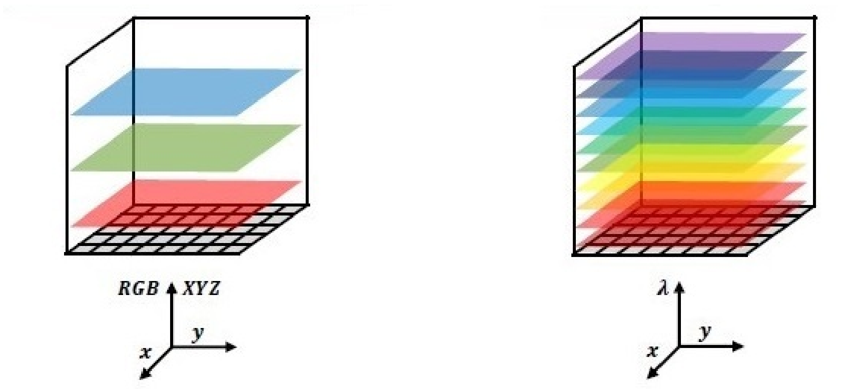


Figure 1.4: *Comparison between the three bands of a standard camera and the multiple bands of an hyperspectral camera*

These spectra or bands form together a hyperspectral signature for each material, which when compared with aerial images allow the recognition of different types of vegetation, mineral deposits or contaminants on the Earth's crust (see Figure 1.5).

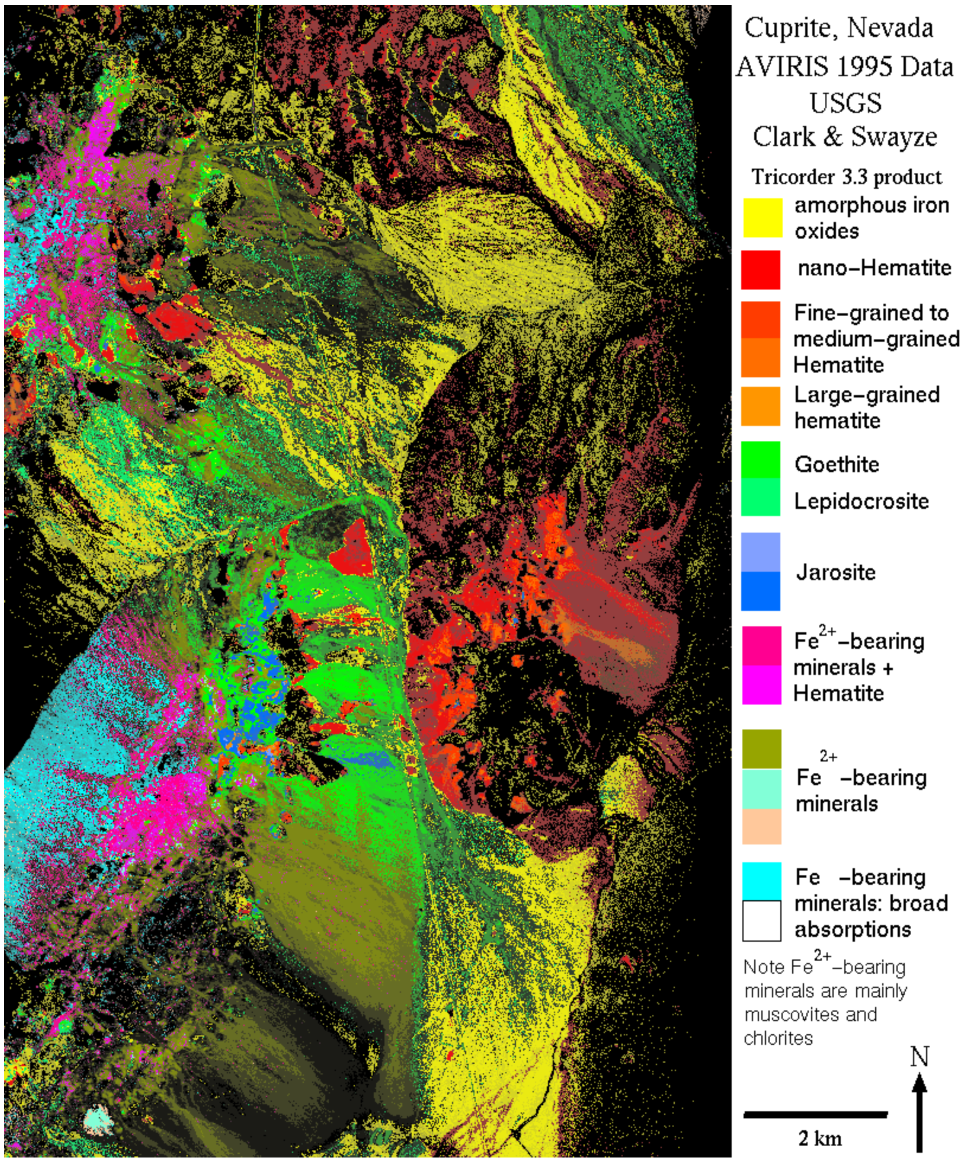


Figure 1.5: *The map shown is a mineral map from an AVIRIS scene flown over Cuprite, Nevada, in 1995 [1]. It shows the mapping of many different minerals*

In this type of images, we can talk about two types of resolutions: spatial and spectral, the latter being unique in this type of cameras. Spatial refers to the number of meters covered by each pixel, so for the same camera you can change from one image to another. The spectral resolution refers to the separation between different wavelengths measured in a given range, that is, the more bands captured in a lower range, the higher the spectral resolution will be [5].

As technology advances, these resolutions continue to increase highlighting the need for commensurate processing systems.

1.2.3 Anomaly detection

In theory, a material should always have the same spectral signature. In practice, the captured signature will never be the same as the one measured in a laboratory because of differences in lighting, atmospheric effects, noise, etc. resulting in spectral variation for similar materials [7].

This has led to the development of algorithms that instead of classifying observations according to their signature, seek to classify the observations into unusual or anomalous materials and background. This assumes that the material -the target- is spectrally distinguishable from the background. The background is derived globally from the image assuming that it follows a normal distribution.

RX algorithm

The Reed-Xiaoli anomaly detector algorithm is a common algorithm that serves as the basis for many others.

The algorithm is defined by the following expression [17]:

$$\delta^{RX}(x) = (x - \mu)^T K^{-1} (x - \mu)$$

Where x is a hyperspectral pixel, a vector of size equal to the number of bands, μ is the mean of each band and K is the covariance matrix. It is important to say that the results generated by the algorithm are grayscale 2D images. The anomalies have a high value, so the first anomaly corresponds to the pixel with the highest value, and so on.

Other methods

Apart from the RX algorithm, there are other methods for detecting anomalies, although a notable number of them are based on RX.

Subspace methods

Subspace methods are global and apply principal component analysis (PCA) or singular value decomposition (SVD) to the hypercube. The first PCA/SVD bands are supposed to be the background and are removed differently by each method. Not removing enough bands means that the anomalies can be lost in the background noise and removing too many would lead the anomalies to disappear. The determination of the correct number of bands to be removed is still under study [6].

Subspace RX In this method, the RX algorithm is applied to a limited number of PCA bands. The first components are discarded.

RX after orthogonal subspace projection [16] In this method, the first PCA/SVD components define the subspace of the background and the data is projected onto an orthogonal space before applying XR.

RX after partialling out the clutter subspace [14] In this method, clutter effect on a pixel is discarded component wise taking each of its spectral components as a linear combination of its high variance principal components. The RX algorithm is applied to the results.

Complimentary subspace detector [8] This algorithm is not based on RX. In CSD, the principal components with higher variance are used to define the subspace of the background and the other PCs to define the subspace of the target. The pixel is then projected onto the two subspaces and the result is the difference between them.

Local methods

In the local methods the background is derived from the neighboring pixels or surroundings of the pixel under test (PUT). Two windows are defined, a guard and an exterior, and the neighbors are the pixels that are between these two. Sometimes, for example in local RX, a third one is used where the covariance of the background is calculated in a window larger than the average of the background (see Figure 1.6).

Quasi-local RX [12] Quasi-local RX is a compromise between global RX and local RX, where the global covariance matrix is decomposed using eigenvector/eigenvalues. The eigenvectors are maintained, but the eigenvalues are replaced by the maximum local variance, resulting in lower detector scores in areas with high variance.

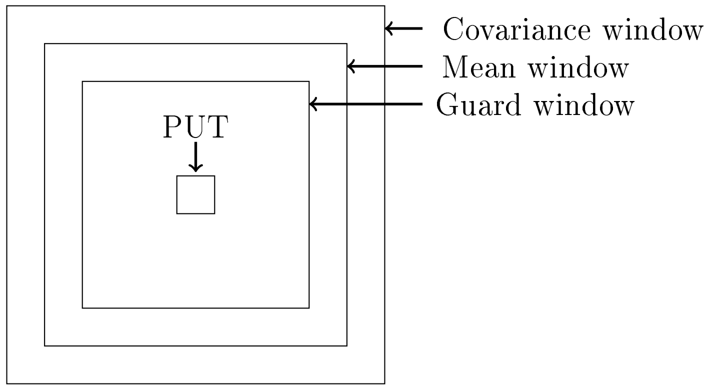


Figure 1.6: *Sliding triple window used in the local AD methods.*

Segmentation based methods

In complex scenes it is difficult to assume that the background will be defined by a normal distribution, so methods have been developed to separate it into different classes.

Class-Conditional RX [13] In this method the image is first segmented and the covariance and mean matrix calculated for each of these classes. Each pixel will correspond to the class in which its RX value is lower.

There are more methods within those based on segmentation, from those using stochastic functions such as Method Based on Multivariate Normal Mixture Models to methods based on Self-organizing maps.

1.2.4 Comparing floating and fixed point

In computer systems, there are two numerical representations for real numbers, fixed and floating point. They have different arithmetic, which gives them different range and resolution with the same number of bits. Therefore, there are certain applications or platforms more akin to one of them. For example, the ability of floating point numbers to contain in the same number of bits both very large and very small numbers and to adjust their resolution accordingly is very attractive from the programmer's point of view but the simplicity of fixed point operations allow their use in small microcontrollers or to save resources in FPGAs (see Figure 1.7).

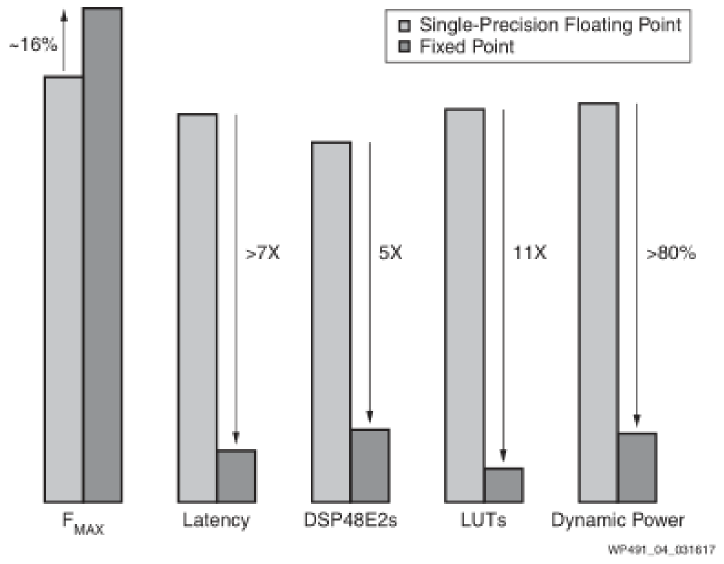


Figure 4: **Fixed Point - Similar Performance with Reduced Latency, Resources, and Power**

Figure 1.7: *A FIR filter, originaly implemented as a single-precision floating-point filter and converted to fixed-point. The fixed-point design shows both resource reduction and latency improvements*

Due to the high complexity of hyperspectral image analysis, current FPGAs have little capacity to implement some of these algorithms. Therefore, one of the objectives of this work is to perform two parallel implementations, one in floating point and another one in fixed point and compare their results.

1.3 Project plan

First, an algorithm and a hardware platform are chosen. Desirably, this algorithm should be known and commonly found in the literature.
This algorithm will be implemented in software and this implementation tested with real images against existing software such as ENVI or Spectral Python.
The different algorithm stages will then be optimized for a hardware architecture and the transformation of the algorithm from floating point to integer or fixed point logic will be studied.
This transformation requires design decisions that sacrifice precision with the goal of saving hardware resources, so the underlying hardware will have to be considered.
Subsequently, a hardware validation of the design will be realized.
Finally, a study will be carried out on the accuracy of the results obtained.

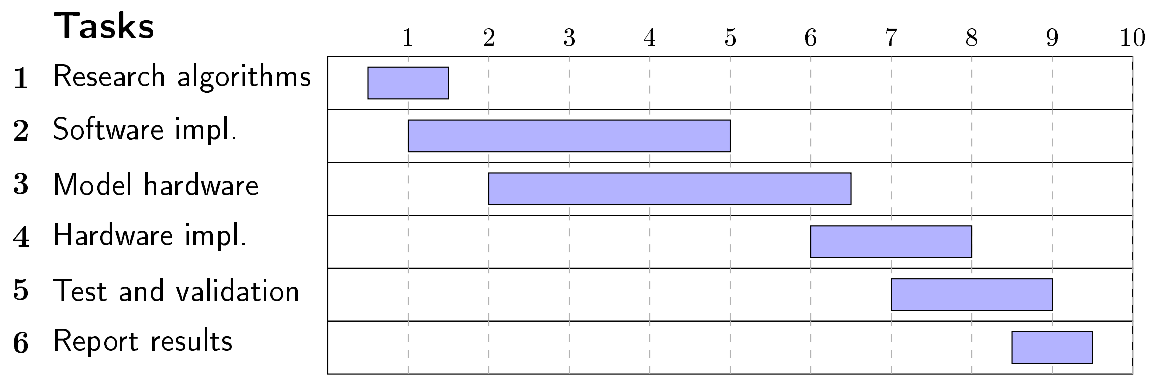


Figure 1.8: *Gantt diagram with an overview of months and tasks*

Chapter 2

Software Model

The RX algorithm has been chosen for this work because it is the benchmark for this kind of algorithms [6], [19], [16] and many existing algorithms basing on RX in some manner.
In order to become familiar with the algorithm and create a platform where tests can be easily performed, a software implementation has been made as a first step.
The first step is to divide the algorithm into simpler operations:

- Calculate the mean, deviation and K covariance matrix of the image
- Calculate K^{-1} that is, the inverse of the covariance matrix
- Calculate δ^{RX} for each pixel in the image
- Sort the results

2.0.1 Mean, deviation and covariancce matrix

One of the bottlenecks in this type of FPGA-based systems is the input and output of data [20]. Since the calculations of mean, variance and covariance matrix need the original matrix -a cube- for this, it has been decided to calculate these in a CPU and transmit the results to the FPGA. In addition, the operations are relatively simple for a CPU. Next, the pseudocode of the three is presented.

Algorithm 1 Pseudocode for the mean, deviation and covariance matrix

```
1:  ▷ $bands$ : number of bands in the image
2:  ▷ $pixels$ : number of pixels in the image
3:  ▷ $A$ : the image in the form of a 2D matrix with $pixels \times bands$
4:  ▷ $^T$ is used to denote the transpose of a matrix

5:  function MEAN($A$)
6:      for $i \leftarrow 0$ to $bands - 1$ do
7:          $sum \leftarrow 0$
8:          for $j \leftarrow 0$ to $pixels - 1$ do
9:              $sum \leftarrow sum + A[i][j]$
10:         $mean[i] \leftarrow sum/pixels$
        return $mean$

11: function DEVIATION($A, mean$)
        return $A^T - mean$                ▷ This operation is a matrix subtraction

12: function COVARIANCE($deviation$)
        return $deviation^T * deviation/(pixels - 1)$
```

The covariance, a $band^2$ matrix is then transmitted to the FPGA to calculate its inverse.

2.0.2 Inverse

Choosing the algorithm

Before starting the implementation, several algorithms to perform the inverse have been studied.

QR Factorization [4]: QR factorization breaks down the A matrix into the product of two matrices $A = QR$, with Q being an orthogonal matrix and R a superior triangular matrix. With this triangular matrix it becomes easy to calculate the inverse. However, although QR factorization can be efficiently performed on a powerful matrix multiplication module or several modules that can be executed simultaneously, such as in a GPU, it does not take advantage of the capabilities provided by our system such as arbitrary width arithmetic units.

Gauss Jordan elimination method [11]: The Gauss Jordan method dictates that if we have a A matrix that can be transformed into the identity matrix through elementary operations, these same operations transform the identity matrix into A^{-1}. Since it is possible to execute these elementary operations in an entire row at once and the operations between rows are independent, this method is easily parallelizable. Therefore, this was the chosen method.

Generally speaking, the execution of the algorithm takes place in such a way that:

1. An identity matrix is generated
2. The same operations are performed on both matrices until the A matrix is transformed into the identity matrix
3. The result is in the matrix generated in the first step

As seen in the following pseudocode, these elementary operations are performed in 3 steps:

Algorithm 2 Pseudocode of the Gauss Jordan method

```
A : a square matrix with the size $n * n$
$A^{-1}$ : an identity matrix with the size $n * n$

function INVERSE($A$)
    for $i \leftarrow 0$ to $n-1$ do    ▷ Forward elimination to build the upper triangular matrix
        ▷ row $i$ acts as the pivot
        if $A[i][i] = 0$ then    ▷ If the later divisor is 0
            for $j \leftarrow i+1$ to $n-1$ do
                if $A[i][j] \neq 0$ then
                    $A[i] \leftarrow A[j],\ A[j] \leftarrow A[i]$
        for $j \leftarrow 0$ to $n-1$ do
            $A^{-1}[j] \leftarrow A^{-1}[j] - A^{-1}[i] * (A[j][i]/A[i][i])$
            $A[j] \leftarrow A[j] - A[i] * (A[j][i]/A[i][i])$    ▷ These two lines run in parallel
        ▷ After a complete iteration of the outer loop, the pivot contains the desired form


    for $i \leftarrow n-1$ to $0$ do    ▷ Backward elimination to build a diagonal matrix
        for $j \leftarrow i-1$ to $0$ do
            $A^{-1}[j] \leftarrow A^{-1}[j] - A^{-1}[i] * (A[j][i]/A[i][i])$
            $A[j] \leftarrow A[j] - A[i] * (A[j][i]/A[i][i])$    ▷ These two lines run in parallel


    for $i \leftarrow 0$ to $n-1$ do    ▷ Last division to build identity matrix
        $A^{-1}[i] \leftarrow A^{-1}[i] * (1/A[i][i])$
        $A[i] \leftarrow A[i] * (1/A[i][i])$
        ▷ There is no need to update the values in the starting matrix
    return $A^{-1}$
```

2.0.3 Matrix multiplication

With the inverse of the convariance matrix calculated in the FPGA, the real RX algorithm can continue to be performed, where a hyperspectral pixel is multiplied with this inverse. This calculation will require the CPU to transmit the original image, the original image next to the average, or the deviation already calculated to the FPGA.
The RX algorithm dictates that:

$$rx(x) = (x-\mu)^T K_{N\times N}^{-1}(x-\mu)$$

$K_{N\times N}^{-1}$	being the inverse matrix with a dimension of $N \times N$,
$(x-\mu)$	being the deviation of a pixel a dimension of $N \times 1$ and
$(x-\mu)^T$	it's transpose, with a dimension of $1 \times N$.

In this step, the inverse matrix gets effectively multiplied by a unique pixel across all bands. Through row reduction, a single value for this pixel is recovered which can then be mapped to a 2d image.

Since the operands are heterogeneous, use can be made of the associative law on matrix products that dictates that:

$$A * (B * C) = (A * B) * C$$

to optimize the operation.

In the implementation section, a study on what alternative to use will be carried out, as this depends heavily on the hardware architecture.

2.1 Adapting the arithmetic

As shown before in Figure 1.7, floating point arithmetic operations are very inefficient compared to fixed point operations. In the next chapter a transformation strategy will be defined and the steps followed to make this transition will be shown.

Taking advantage that the RX algorithm only needs relative and not exact values, ie, the highest value found in the results will be the most anomalous (see 1.2.3) regardless of whether it exceeds a certain threshold or not, the fixed-point representation may be simplified by ignoring the fractional part and keeping only the integer.

2.1.1 DSP Blocks

The DSP blocks are prefabricated circuits found next to the FPGA logic and implement arithmetic operations. These blocks allow to perform operations more efficiently and operate at higher frequencies than equivalent logic in the fabric, besides not occupying space in it. However, as they are prefabricated blocks, they do not offer the same flexibility as the rest of the FPGA logic. This implies that the size of the operands and the cost of operation resources will not always be proportionally related. Therefore, different widths have been tested and their resource usage has been noted.

The results of these tests for multiplication are in the following table:

Width operand A	Width operand B	DSP48 Slices used
25	18	1
35	25	2
52	24	3
42	35	4
64	25	4
52	42	6

Table 2.1: *Depending on the width of the operands, the operation will use a different amount of resources (data obtained for signed multiplication in Vivado 2019.2 on Xilinx Series 7 chips)*

For subtraction operations up to 48bits, only one DSP is used, so they are not expected to require more than one block.
The divisions cannot be implemented by this type of blocks, so they will use standard logic and can be used with an arbitrary width.

With this, data for all the operations that are going to be carried out in the FPGA is recorded.

2.1.2 Determining multiplier size

The operand dimensions for the multiplier still must be determined. For this, the first operation for the inverse was done with all possible shift values, with all previous operations unshifted and all later operations implemented in floating point.

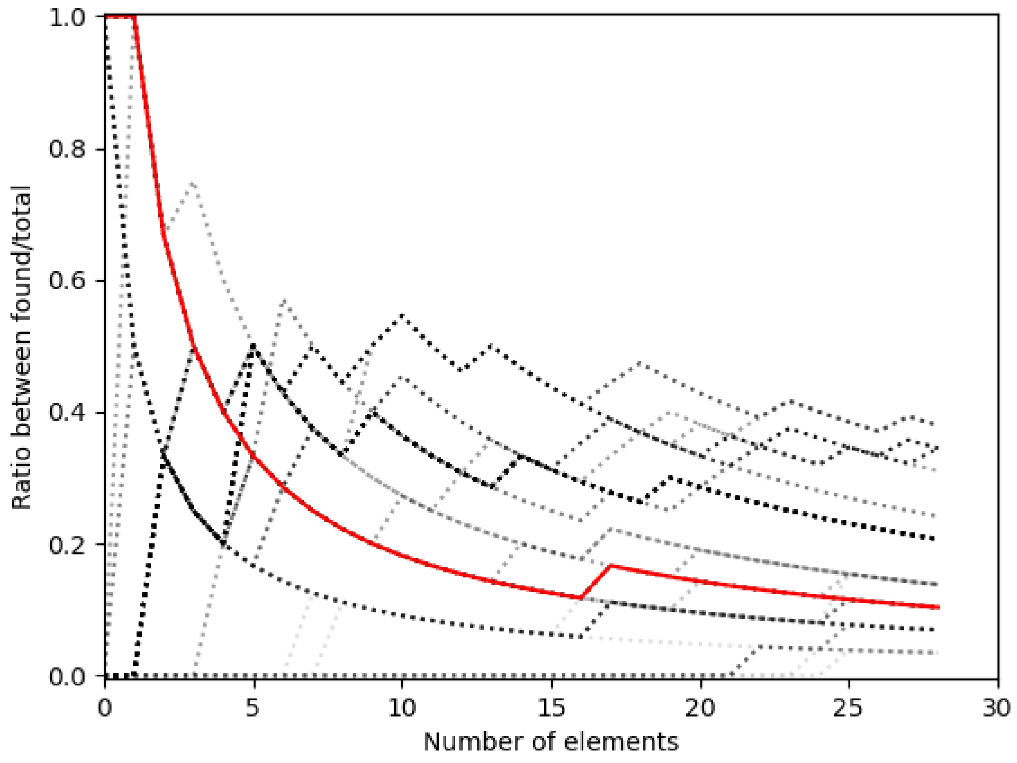


Figure 2.1: *Ratio of anomalies found in the first x using different bit widths for the multiplication. A red line represents the results for a value of 20. Dataset: HYDICE*

The results depict the ratio of anomalies found till that point in x. The line for 20 is depicted red since this was the chosen value, since it provides the highest detection for the first anomalies, those being the highest result numbers, without producing an overflow. The value of the multiplier operands should also approximate $64 - 20 = 44$ for an operand. The widths representing 42 and 35 were then chosen as a starting point as they are the cheapest ones resource swise that fullfill this requirement.

2.1.3 Improving accuracy

It is also necessary to create rules to shift the results to maintain the highest possible accuracy without producing overflow.
For example, the divisions made in the inverse calculation produce very small numbers where the fractional part is relevant to the final result. Since this part is lost in integer arithmetic, it is necessary to multiply the operands to produce results where the point is shifted to the left. These multiplications will always be done in powers of 2, negative if required to decrease the bit width, as they are trivial to implement in an FPGA and do not require resources.

The results indicated that the greatest loss of accuracy occurred in the inverse calculation. Therefore, the model was revised and shifts were included in the initial matrix capture, in the initialization in the case of A^{-1} and in the transfer to the FPGA in the case of A. Different shifts were also included for each phase of the inverse.

Chapter 3

Implementation

3.1 General overview of the system

The camera provides the pixels in the image by bands. The first operations to be performed with this data are to calculate the average, with it the deviation and then the covariance. Because of the relative simplicity of these operations but their high memory requirements, these three operations are performed on a CPU and their results sent to the FPGA. The FPGA will start the calculation of the subsequent operations only when it has the complete results of the covariance.
The data calculated by the CPU is entered into the FPGA through FIFOs.
The FPGA will then calculate the inverse of the matrix. Meanwhile the CPU will have to write the calculated averages and the values it had previously received from the camera, one by one. When the inverse is finished, the FPGA will perform the last two matrix multiplications and save the resulting data. With the last pixel processed, the FPGA will write the anomalies ordered from highest to lowest in another FIFO to be read by the CPU.

3.2 Description by module

3.2.1 Control

This module acts on the lower modules, both to control the data transfer between them and to arbitrate the access to the RAMS and the FIFOs that communicate with the CPU.
It is worth mentioning that it also performs some checks in the covariance transfer to ensure that the first division of the inverse is not performed with a 0, that is, that the position (0, 0) in the covariance matrix is different from 0.

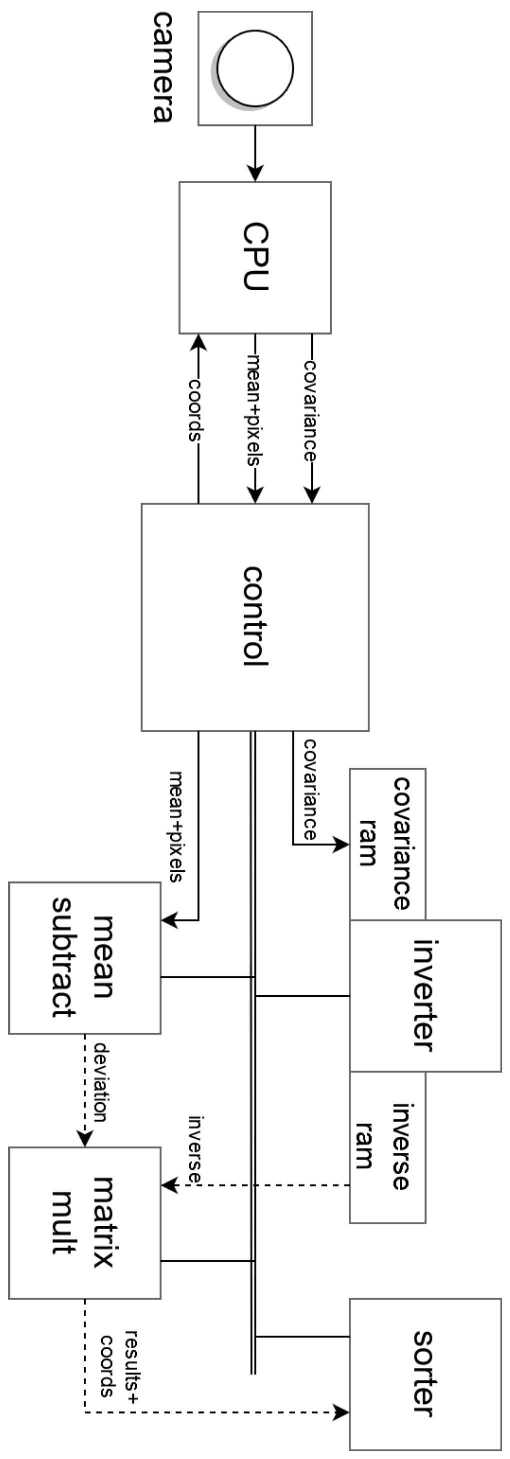


Figure 3.1: *A schematic of the whole system*

3.2.2 Inverter

The inverse is the most complex module and the one that consumes more hardware resources. In addition, the study carried out on software has shown that it is by far the step most susceptible to worsening the accuracy of the final results. For these reasons, special emphasis has been placed on its design.

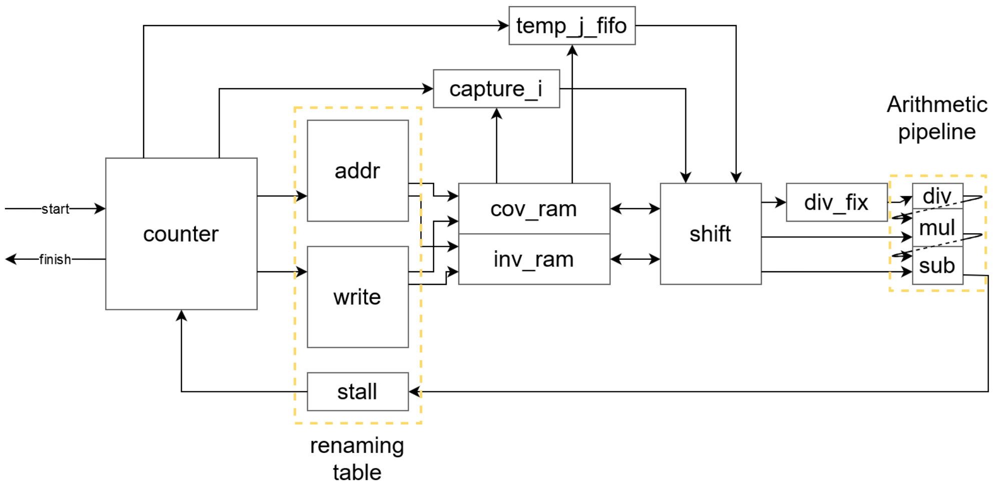


Figure 3.2: *Schematic of the implemented inverter*

As can be seen in Figure 3.2, there are numerous processes and sub-modules within the inverter. The process that acts as an interface to the outside is *counter*, which as its name suggests contains counters to calculate the rows to be read, written, etc. It is the main process of the module. The counters are translated into addresses by another pair of processes, more specifically *addr*, *write* and *stall* and a renaming table to which these three have access. The data that are read from the memories are then saved in temporary storage or directly to the arithmetic units. Before the arithmetic units is another process, *shift*, which shifts the data as modeled in software. These units are used for the three steps of the algorithm, upper triangle, lower triangle and diagonal, and it is the *counter* process that controls their execution order. The rest of the processes will be explained in more detail within the chapter.

Agorithm optimizations for hardware

To improve the performance of the module, operations on the A matrix and the $A-1$ matrix are executed simultaneously. In addition, the divisor and DSP pipelines are used to queue all possible consecutive operations. Pipeline stalling only occurs, if calculations are still being processed with the last pivot row. Table 3.1 shows the arithmetic units inside the module and their latencies. These latencies represent the length of its pipeline.

Arithmetic Unit	Latency	Quantity	Remarks
Division	77	1	Only one division is required each cycle
Multiplication	6	bands*2	To compute a whole row for both A and A^-1
Subtraction	2	bands*2	To compute a whole row for both A and A^-1

Table 3.1: *Latencies and number of the arithmetic units in the inverter module.*

Equations 3.3 indicate which calculations are found within the DSP pipeline at any given time. The results of each operation are sent directly to the calculation of the next operation, while the counters control that the other operands arrive at the right time.

$$\begin{aligned}
A^{-1}[86] &\leftarrow A^{-1}[86] - A[i]^{-1} * A[86][i]/A[i][i], & A[86] &\leftarrow A[86] - A[i] * A[86][i]/A[i][i] \\
&\vdots & &\vdots \\
A^{-1}[10] &\leftarrow A^{-1}[10] - A[i]^{-1} * A[10][i]/A[i][i], & A[10] &\leftarrow A[10] - A[i] * A[10][i]/A[i][i] \\
A^{-1}[9] &\leftarrow A^{-1}[9] - A[i]^{-1} * div\_result, & A[9] &\leftarrow A[9] - A[i] * div\_result \\
&\vdots & &\vdots \\
A^{-1}[3] &\leftarrow A^{-1}[3] - A[i]^{-1} * div\_result, & A[3] &\leftarrow A[3] - A[i] * div\_result \\
A^{-1}[2] &\leftarrow A^{-1}[2] - mul\_result, & A[2] &\leftarrow A[2] - mul\_result \\
A^{-1}[1] &\leftarrow A^{-1}[1] - mul\_result, & A[1] &\leftarrow A[1] - mul\_result \\
A^{-1}[0] &\leftarrow sub\_result, & A[0] &\leftarrow sub\_result
\end{aligned}$$

Figure 3.3: *Operations representing the arithmetic pipeline at any given moment. Colors denote if a operation is currently in progress.*

One of the operands is the row j, which is used at two different times within the algorithm (see Figure 3.4). Since a second read on the BRAM cannot be done since it is busy performing the read of the next operations, this data is stored at the time of reading in a FIFO and will be fetched when needed.

$$A^{-1}[j] \leftarrow A^{-1}[j] - A[i] * A[j][i] / A[i][i]$$

Figure 3.4: *As can be seen in the algorithm, row j is used at two different times.*

Additionaly, it can be observed that in the first step in which the upper triangular matrix is constructed, the algorithm requires a check on the pivot row and a possible exchange of rows. This is necessary because this value is later the dividend, so a 0 would cause a failure in the calculation.
BRAM memory readings have a latency cycle, so reading an inappropriate value two cycles in a row -in the case of reading an inappropriate and then an appropriate value, there would be the possibility of performing an inplace row swap with the pivot and the row just after it- would not only add latency to the calculation but also increase the complexity of the module. Therefore, the dividend checks are done on the writes, recorded in a renaming table that will be checked at the time of reading. This ensures that the reads will always be valid for the calculation. In the case of the very first division, this dividend comes directly from the CPU and the upper module *control* is responsible for reordering this row if necessary.
This renaming table is located in registers, so it is possible to access it without any latency and as it only contains indexes, it does not overload the FPGA resources. In addition, this table is local, so the results have to be reordered in the RAM itself before leaving the inverse module. Since these swaps only occur in the calculation of the upper triangle, the lower triangle can be used to reorder them. The reordering system is very simple, the data enters the pipeline according to the order that exists in the renaming table and is written in its natural order. This implies that the results of this reordering will be correct as long as both rows that have been rotated are at the same time in the processing pipeline, which in the case of fixed point is approximately 90 in size. Experimental results show that it is rarely necessary to rotate rows -although enough to recommend the inclusion of a method to deal with it-, and that these rotations rarely exceed one or two positions in the pipeline.

In the transformation to integer arithmetic it was discovered that the calculation of the inverse is the one introducing more error in the final results of the algorithm, therefore an exhaustive study on how to minimize it has been made. For this it has been necessary to reduce the values in which the limited precision produced overflows and increase small values to give them more weight in the operations. By performing the operations of identity generation, upper triangle, lower triangle and diagonal independently, it has been possible to place different shift values and further refine the resolution of the algorithm. These operations are performed by the *shift* process. It should also be said that there is an error in the generation of Xilinx dividers. When you enter numbers near the precision limit you lose control of the sign. Therefore, *div_fix*, a process that converts all the entered operands into positives and saves their position in a pipeline has been placed before the division. When the results are produced, the tag is checked in the pipeline, the negative is calculated and replaced if necessary.

3.2.3 Mean subtract

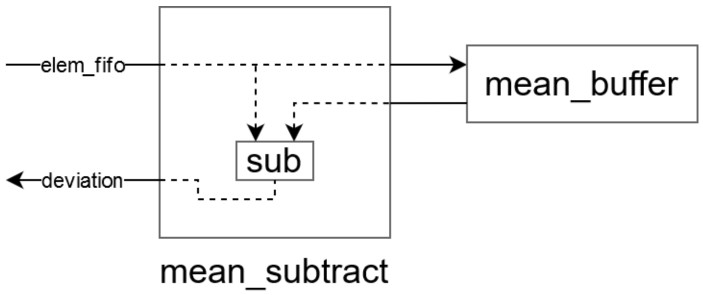


Figure 3.5: *Schematic of mean subtract.*

The *mean subtract* module receives the calculated average and the original pixels of the image and subtracts them. This calculation is the deviation and although it had already been calculated by the CPU, it is possible that the latter discards the data to free up space. The calculation of the average is required because it is assumed that its size being much smaller, the CPU can keep it in memory. In case the deviation can be received directly from the CPU, this module can be simply deleted. The module receives the elements from the upper module that reads them from the same FIFO. The first elements are the average and are stored in a BRAM that is treated as a circular buffer, and the following elements are directly subtracted and returned to the upper module again.

3.2.4 Matrix multiplication

As noted above (see 2.0.3), this calculation can be implemented in two different ways. Following will be a comparison of the first required multiplication in both methods:

> $(x-\mu)^T K^{-1}_{N\times N}$: A row from the inverse and the whole column of the deviation get read, each element multiplied with its correspondent and all products added together. If stalls were to be avoided, this sum would need to be computed every cycle, which can easily be achieved with an adder tree.

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} * \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 1*a+2*b+3*c \\ 1*d+2*e+3*f \\ 1*g+2*h+3*i \end{pmatrix}$$

Figure 3.6: *First proposed method for the computation of a single pixel: red, blue and green represent data processed in the first, second and third cycles respectively. Note that the entire deviation data of that pixel gets used every cycle*

> $K^{-1}_{N\times N}(x-\mu)$: The inverse gets also read row by row, but the deviation matrix only by elements. Each element of the first row of the matrix gets multiplied with the first element of the deviation, the result accumulated, and continued with the next pair row/element. This goes for N cycles, that is, a whole inverse matrix and a whole pixel in the deviation matrix. The result is N accumulated values which get flushed every N cycles, which ends up being the same throughput as the former method.

$$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix} * \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = \begin{pmatrix} \text{1*a} + \text{2*d} + \text{3*g} & \text{1*b} + \text{2*e} + \text{3*h} & \text{1*c} + \text{2*f} + \text{3*i} \end{pmatrix}$$

Figure 3.7: *Second proposed method: red, blue and green represent data processed in the first, second and third cycles respectively. Here only an element of the deviation data gets accessed each cycle.*

While both methods have equivalent cost in time -the former has the added latency of the adder tree, the latter the latency of the accumulators- and also similar cost in DSP usage, data input by row is less taxing on the CPU and its FIFO structure can be reused for the second multiplication. Henceforth, the second approach was chosen.

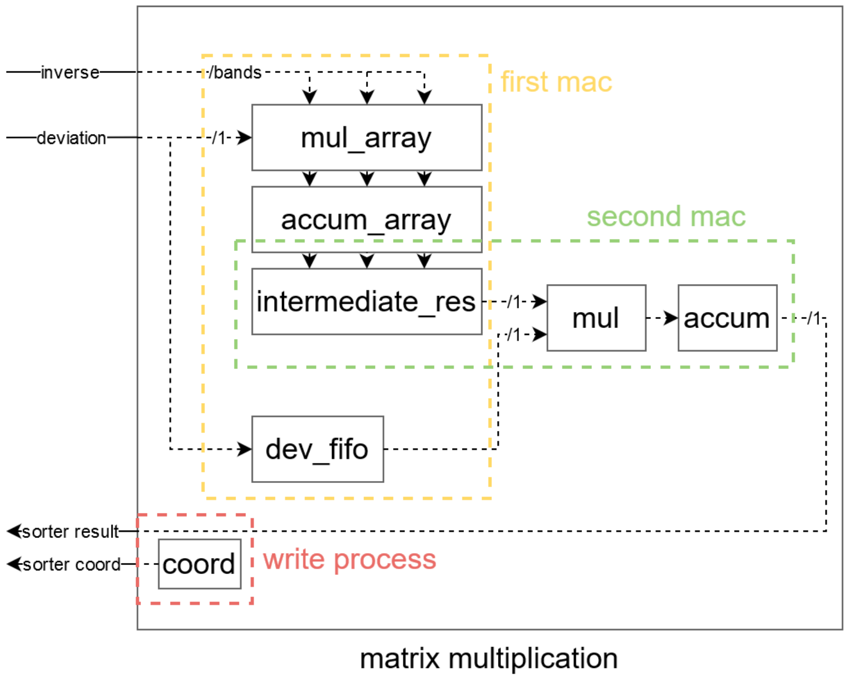


Figure 3.8: *Schematic of the dual matrix multiplier*

The second multiplication is similar in both steps, a $1 \times N$ by $N \times 1$ multiplication. One operand comes every cycle and each N cycles all products get added together. This sum is realized through an accumulator.

The module contains three subprocesses:

- *first_mac* reads the inverse and performs its multiplication with the received deviation. This deviation is also stored in a FIFO. The products are then accumulated till a whole pixel has been computed.
- *second_mac* stores the results of *first_mac* in registers and performs the multiplication with data from the FIFO, with the result being accumulated. Every cycle, the registers are shifted so a new multiplication is done.
- *write_proc* controls the writing of the results from *second_mac* to the sorter and computes the coords.

3.2.5 Coordinate sorter

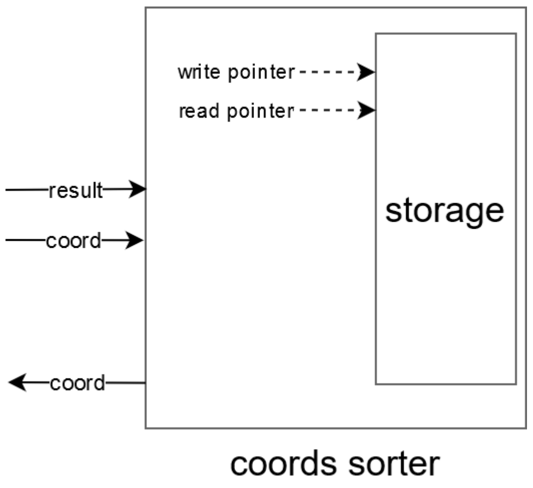


Figure 3.9: *Schematic of the coordinate sorter*

This module receives a value and a pair of coordinates every *bands* cycles. These values are written in a BRAM memory that acts as an ordered list. Each entered value is compared with the head of the list, the highest value is saved and the other is saved in a temporary variable, compared with the second value in the list, and so on. Since a value is received each *bands*, the maximum number of possible values to be stored in this list is also *bands*. The rest of the values are discarded. When the last value has been introduced, the module communicates the highest pixels, that is, the most anomalous ones, to the superior module so that they are communicated to the CPU.

Chapter 4

Results

4.1 Reconfigurable platform

The architecture described in the previous section has been implemented using the VHDL language. To test its correct operation, the Vivado environment and the Virtex 690T FPGA 4.1 have been used. Although it is not a radiation-protected FPGA, the architecture of this algorithm is scalable so that its adaptation to other sensor sizes or FPGAs should not be a problem.

4.2 Hyperpectral image datasets

Two hyperspectral images have been used for the work, one taken by the HYDICE sensor and the other by the AVIRIS sensor. Both images are commonly used as reference in hyperspectral applications.

In the field of the scene captured by HYDICE, 15 panels of different sizes were placed on a field in a 3 x 5 meter configuration. The following images show a false color image with the bands 50, 37, and 17 as red, green and blue respectively [10] and the location of the panels as detected by software.

Part number	Slices	Logic cells	Flip-Flops	BRAM	DSP Slices
XCE7VX690T	108,300	693,120	866,400	1,470Kb	3,600

Table 4.1: *Basic specifications of the Virtex 690T FPGA*

Horizontal res	64
Vertical res	64
Bands	210 (169*)
Spectral res	400 - 2500nm
Spatial res	1,56meter/pixel
Size	1,6 MB

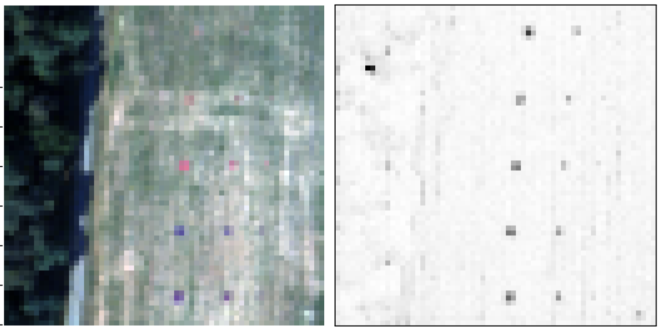


Figure 4.1: *HYDICE sensor information [3], image in false color and anomaly map. *In this work, images and results are reported for 169 bands*

The image taken by the AVIRIS sensor was taken on September 16, 2001, five days after the terrorist attacks that brought down the WTC towers and its surrounding buildings. The spatial resolution of this image is very high because a very low altitude flight was performed. Along with the false color image, an image with the anomalies detected by software is provided. To ease the recognition of these anomalies, a circle has been painted around each group of anomalues for the first 30.

Horizontal res	614
Vertical res	512
Bands	224
Spectral res	360 - 2500nm
Spatial res	1,7meter/pixel
Size	140 MB

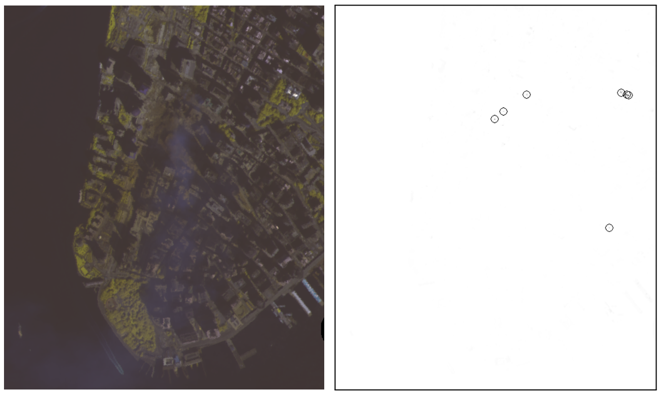


Figure 4.2: *AVIRIS sensor information [2], image in false color and anomaly map*

4.3 Adequacy of approximation

4.3.1 Floating point

The results provided by the floating point version of this system are the same as those provided by the equivalent software version, so it can be considered valid.

4.3.2 Fixed point

The fixed point version of the system is an approach to the floating point system with the intention of maintaining the highest possible accuracy with limited use of resources. Therefore, the results are different and an assessment of their accuracy must be made. For this purpose, three metrics have been used.

In the first and simplest, it has been verified that the number of detected anomalies can be found in the first x positions in both results, regardless of the order.

Extending the previous strategy, it has been checked for the non-coinciding elements, if any neighbor has been detected in the environment. A neighbor is defined as an adjacent pixel, both in a straight line and in diagonal.

Finally, the first strategy has been extended again, this time it has been checked if for the non-coincidences an anomaly with a spectral similarity of less than 5 degrees has been found.

Spectral similarity The Spectral Angle Mapper (SAM) is a physics-based spectral classification that uses an n-D angle to match pixels with reference spectra. The algorithm (see section 4.3.2) determines the spectral similarity between two spectra by calculating the angle between the spectra and treating them as vectors in a space with a dimensionality equal to the number of bands. This technique, when used in calibrated reflectance data, is relatively insensitive to the effects of illumination and albedo.

$$\alpha = \cos^{-1}\left(\frac{\sum_{i=1}^{nb} t_i r_i}{\left(\sum_{i=1}^{nb} t_i^2\right)^{1/2}\left(\sum_{i=1}^{nb} r_i^2\right)^{1/2}}\right)$$

α = spectral angle between vectors
nb = number of spectral bands
t = target pixel
r = reference pixel

Figure 4.3: *Spectral Angle Mapper algorithm*

4.3.3 Results for HYDICE

Every pixel is found in the same order in reference and simulated calculations. Also, since those pixels are found equally, it is obvious that they are also found as neighboring pixels and are spectrally similar. Since it is an image taken to test the sensor, the targets are big and clear, and the image is quite easy to analyze.

4.3.4 Results for AVIRIS

The WTC image is more complex and the detection of exact pixels and neighbors quickly falls. However, the spectral signatures match for the most part, especially considering the number of real hot spots found in the image. Therefore, the results can also be taken as successful.

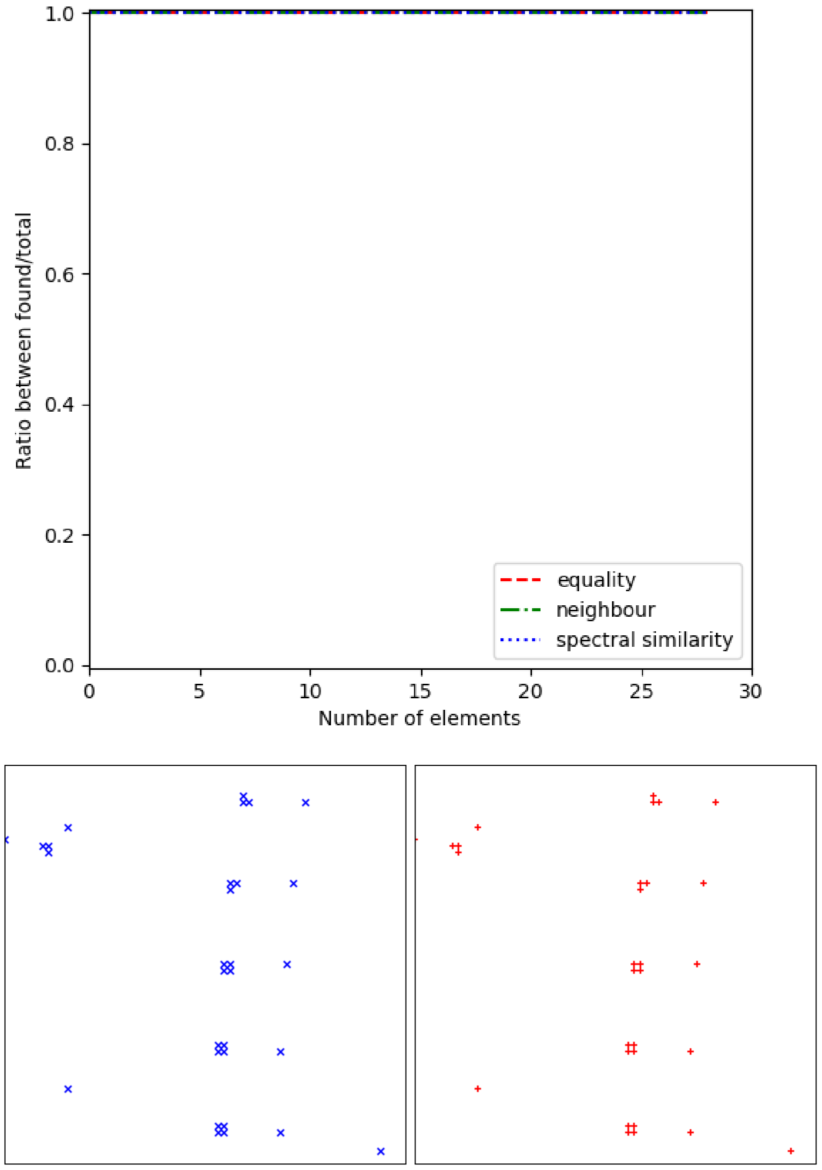
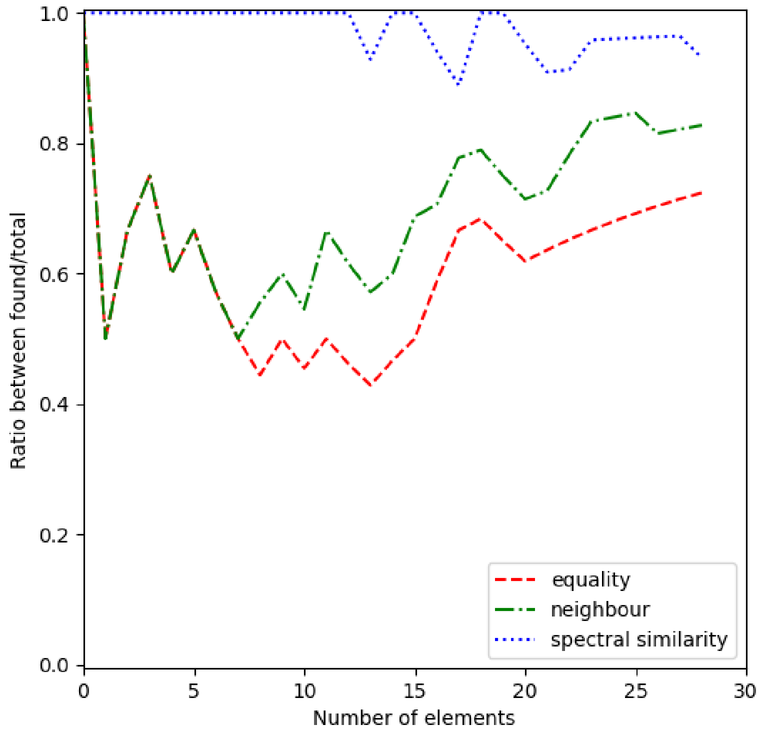


Figure 4.4: *A diagram showing similarities between referenced and achieved results and reference and achieved results mapped as in the original image for the HYDICE dataset*



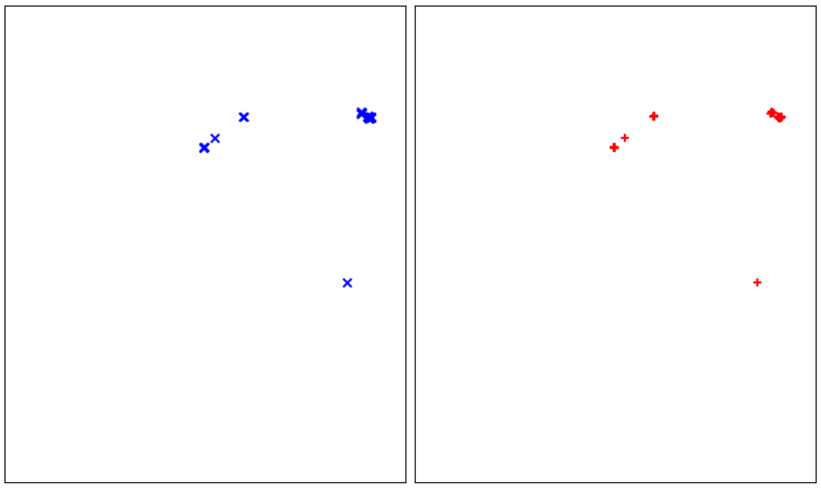


Figure 4.5: *A diagram showing similarities between referenced and achieved results and reference and achieved results mapped as in the original image for the AVIRIS dataset*

It should be noted that the shift values within the calculations have been adapted for each image with the intention of detecting the greatest number of anomalies possible, so that the results obtained in a real system will be slightly lower. Even so, the values between the two images are quite similar, despite the great difference between the data they represent.
In addition, since it is a reconfigurable system, these values can be modified after the system is put into operation, both to accept a wider range of images, and to obtain more accurate results.

4.4 Computational efficiency

When evaluating performance, both resources used (section 4.4) and processing time (Table 4.4) must be assessed. These data are given both in those obtained for the two sensors and according to their characteristics, more specifically number of bands and pixels.
It should be noted that performance data has been obtained by ignoring the input and output of data, which is often the bottleneck in this type of system.

Module	LUT	Register	BRAM	DSP
Inverse	$410 * bands + 3484$	$316 * bands + 10427$	$< bands$	$10 * bands$
Mean subtraction	$bands + 41$	136	0	1
Matrix multiplication	$80 * bands + 75$	$91 * bands + 223$	$pixels/64$	$4 * bands$
Sort results	186	176	0	0

Table 4.2: *FPGA resources used in relation to number of bands and number of pixels*

Stage	Latency
Inverse	$\mathcal{O}(bands + bands^2) = \mathcal{O}(bands^2)$
Mean subtraction	$\mathcal{O}(1)$
Matrix multiplication	$\mathcal{O}(bands * pixels + bands)$
Sort results	$\mathcal{O}(2 * bands) = \mathcal{O}(bands)$

Table 4.3: *Latency between end of previous module and end of current one*

Finally, the total resource utilization for both datasets and processing time is also given in Table 4.4.

	Hydice	Aviris
LUT	224.363	296.155
Register	98.470	126.274
BRAM	629	133
DSP	2.371	3.141
Frecuency	5,5 ns	5,5 ns
Cycles	1.443.418	140.927.349
Full computation	7,938799 milliseconds	775,10042 milliseconds

Table 4.4: *Resource and processing time for both datasets*

Chapter 5

Conclusions

Hyperspectral image processing is a very powerful tool that facilitates mining operations, target tracking or contamination analysis. The technological advances in these cameras accentuates the need to perform this type of analysis on board and with it the use of specific platforms such as FPGAs.

The complete analysis of this type of images is not always feasible and target detection algorithms such as the one presented here allow not only a significant reduction in bandwidth requirements but also allow the use of this type of sensors in real-time applications.

The results obtained are positive since they show a reduction in the use of resources and compared to other previous implementations thanks to its approach through the use of fixed point logic. The use of a reconfigurable platform also allows the precision of this system to be adjusted even after it has been put into operation.

Bibliography

[1] AVIRIS scene flown over Cuprite, Nevada.

[2] AVIRIS Sensor information. *https://www.indexdatabase.de/db/s-single.php?id=28.*

[3] HYDICE Sensor information. *https://www.indexdatabase.de/db/s-single.php?id=84.*

[4] Carlos Alberto Oliveira de Souza Junior, João Bispo, João M. P. Cardoso, Pedro C. Diniz, and Eduardo Marques. Exploration of FPGA-Based Hardware Designs for QR Decomposition for Solving Stiff ODE Numerical Methods Using the HARP Hybrid Architecture. *Electronics*, 9(5):843, May 2020.

[5] José Manuel Amigo. Chapter 1.1 - Hyperspectral and multispectral imaging: setting the scene. In José Manuel Amigo, editor, *Data Handling in Science and Technology*, volume 32 of *Hyperspectral Imaging*, pages 3–16. Elsevier, January 2020.

[6] Dirk Borghys, Ingebjørg Kåsen, Véronique Achard, and Christiaan Perneel. Hyperspectral Anomaly Detection: Comparative Evaluation in Scenes with Diverse Complexity, November 2012.

[7] Ricardo Augusto Borsoi, Tales Imbiriba, José Carlos Moreira Bermudez, Cédric Richard, Jocelyn Chanussot, Lucas Drumetz, Jean-Yves Tourneret, Alina Zare, and Christian Jutten. Spectral Variability in Hyperspectral Data Unmixing: A Comprehensive Review. *arXiv:2001.07307 [eess]*, January 2020. arXiv: 2001.07307.

[8] Chein-I Chang and Shao-Shan Chiang. Anomaly detection and classification for hyperspectral imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 40(6):1314–1325, June 2002.

[9] Carlos Colomé García, Gonzalo Pericacho Sánchez, Carlos Colomé García, and Gonzalo Pericacho Sánchez. Implementación del algoritmo RX para la detección de anomalias en imágenes hiperespectrales de la superficie terrestre mediante hardware reconfigurable, 2013.

[10] Xiaoxiao Feng, Luxiao He, Qimin Cheng, Xiaoyi Long, and Yuxin Yuan. Hyperspectral and Multispectral Remote Sensing Image Fusion Based on Endmember Spatial Information. *Remote Sensing*, 12(6):1009, January 2020.

[11] Carlos González, Sergio Bernabé, Daniel Mozos, and Antonio Plaza. FPGA Implementation of an Algorithm for Automatically Detecting Targets in Remotely Sensed Hyperspectral Images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9(9):4334–4343, September 2016.

[12] Nir Gorelik, Dan Blumberg, Stanley R. Rotman, and Dirk Borghys. Target detection using nonsingular approximations for a singular covariance matrix, January 2012.

[13] Edisanter Lo and John Ingram. Hyperspectral anomaly detection based on minimum generalized variance method. In *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XIV*, volume 6966, page 696603. International Society for Optics and Photonics, April 2008.

[14] Edisanter Lo and Alan Schaum. A hyperspectral anomaly detector based on partialing out a clutter subspace. In *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XV*, volume 7334, page 733404. International Society for Optics and Photonics, April 2009.

[15] Sebastian Lopez, Tanya Vladimirova, Carlos Gonzalez, Javier Resano, Daniel Mozos, and Antonio Plaza. The Promise of Reconfigurable Computing for Hyperspectral Imaging Onboard Systems: A Review and Trends. *Proceedings of the IEEE*, 101(3):698–722, March 2013.

[16] Stefania Matteoli, Marco Diani, and Giovanni Corsini. A tutorial overview of anomaly detection in hyperspectral images. *IEEE Aerospace and Electronic Systems Magazine*, 25(7):5–28, July 2010.

[17] J. M. Molero, A. Paz, E. M. Garzón, J. A. Martínez, A. Plaza, and I. García. Fast anomaly detection in hyperspectral images with RX method on heterogeneous clusters. *The Journal of Supercomputing*, 58(3):411–419, December 2011.

[18] Gilberto Ochoa-Ruiz, Ouassila Labbani, El-Bay Bourennane, Philippe Soulard, and Sana Cherif. A high-level methodology for automatically generating dynamic partially reconfigurable systems using IP-XACT and the UML MARTE profile. *Design Automation for Embedded Systems*, 16(3):93–128, September 2012.

[19] Dalton Rosario. A semiparametric model for hyperspectral anomaly detection, January 2012.

[20] Qingshan Tang, Habib Mehrez, and Matthieu Tuna. Multi-FPGA prototyping board issue: the FPGA I/O bottleneck. In *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, pages 207–214, July 2014.

[21] James Theiler, Bernard Foy, Claira Safi, and Steven P. Love. Onboard CubeSat data processing for hyperspectral detection of chemical plumes. In David W. Messinger and Miguel Velez-Reyes, editors, *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XXIV*, page 5, Orlando, United States, May 2018. SPIE.

[22] Xin Zhou, Norihiro Tomagou, Yasuaki Ito, and Koji Nakano. Efficient Hough Transform on the FPGA using DSP Slices and Block RAMs. In *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, pages 771–778, May 2013.