AN FPGA IMPLEMENTATION OF THE IMAGE SPACE RECONSTRUCTION ALGORITHM FOR HYPERSPECTRAL IMAGING ANALYSIS

By

Javier Morales Morales

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

ELECTRICAL ENGINEERING

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS

January, 2007

Approved by:

Gladys O. Ducoudray, Ph.D Member, Graduate Committee	Date
Miguel Velez Reyes, Ph.D Member, Graduate Committee	Date
Nayda G. Santiago, Ph.D President, Graduate Committee	Date
Mercedes Ferrer, Ph.D Representative of Graduate Studies	Date
Isidoro Couvertier, Ph.D Chairperson of the Department	Date

Abstract of Thesis Presented to the Graduate School
of the University of Puerto Rico in Partial Fulfillment of the
Requirements for the Degree of Master of Science

AN FPGA IMPLEMENTATION OF THE IMAGE SPACE RECONSTRUCTION ALGORITHM FOR HYPERSPECTRAL IMAGING ANALYSIS

By

Javier Morales Morales

January 2007

Chair: Nayda G. Santiago
Major Department: Electrical and Computer Engineering

The Image Space Reconstruction Algorithm (ISRA) has been used in hyperspectral imaging applications to monitor changes in the environment and specifically, changes in coral reef, mangrove, and sand in coastal areas. This algorithm is one of a set of iterative methods used in the hyperspectral imaging area to estimate abundance. However, ISRA is highly computational, making it difficult to obtain results in a timely manner. We present the use of specialized hardware in the implementation of this algorithm, specifically the use of VHDL and FPGAs in order to reduce the execution time. The implementation of ISRA algorithm has been divided into hardware and software units. The hardware units were implemented on a Xilinx Virtex II Pro XC2VP30 FPGA and the software was implemented on the Xilinx Microblaze soft processor. This case study illustrates the feasibility of this alternative design for iterative hyperspectral imaging algorithms. The main bottleneck found in this implementations was data transfer. In order to reduce or eliminate this bottleneck we introduced the use of block-rams (BRAMS) to buffer data and have

data readily available to the ISRA algorithm. The memory combination of DDR and BRAMS improved the speed of the implementation.

Results demonstrate that the C language implementation is better than both FPGA's implementations. Nevertheless, taking a detailed look at the improvements in the results, FPGA results are similar to results obtained in the C language implementation and could further be improved by adding memory capabilities to the FPGA board. Results obtained with these two implementations do not have significant differences in terms of execution time.

Resumen de Tesis Presentado a Escuela Graduada
de la Universidad de Puerto Rico como requisito parcial de los
Requerimientos para el grado de Maestría en Ciencias

IMPLEMENTACIÓN UTILIZANDO FPGA DEL ALGORITMO DE RECONSTRUCCIÓN DEL ESPACIO DE LA IMAGEN PARA EL ANÁLISIS DE IMAGENES HYPERESPECTRALES

Por

Javier Morales Morales

Enero 2007

Consejero: Nayda G. Santiago
Departamento: Ingeniería Eléctrica y Computadoras

El algoritmo de Reconstrucción del Espacio de la Imagen con sus siglas en inglés (ISRA) es utilizado en aplicaciones para monitoriar cambios en el medio ambiente, especificamente cambios en coral, mangle y arena en áreas cercanas a la costa. Este algoritmo iterativo es uno de los más utilizados para estimar abundancia en el estudio de imagenes hiperespectrales. Sin embargo este algoritmo es altamente computacional haciendo difícil obtener resultados rápidamente. En este trabajo se presenta el uso de herramientas especializadas (especificamente VHDL y FPGA) para implementar este algoritmo. La implementación de ISRA se divide en dos áreas, una de "hardware" y la otra de "software". La parte de "hardware" se implementó utilizando un Xilinx Virtex II Pro XC2VP30 FPGA y la parte de "software" utilizando el procesador Xilinx Microblaze.

Este estudio demuestra la viabilidad de utilizar FPGA para implementar algoritmos que se utilizan para el estudio de imagenes hiperespectrales. Uno de los

problemas encontrados en este tipo de aplicación fué la dificultad de mover la información de una manera eficiente. Este problema se resolvió utilizando una combinación de dos tipos de memorias, DDR y BRAMs. La importancia de la memoria BRAM es la capacidad de acceso rápido de data, utilizado en las computaciones matemáticas requeridas. Los resultados demuestran que la implementación creada en C es superior que las creadas en el FPGA. Sin embargo las mejoras obtenidas en la implementación apuntan a la posibilidad de obtener mejorías mayores al incrementar la cantidad de memoria en las tarjetas con las cuales se trabaje el problema.

ACKNOWLEDGMENTS

I would like to extend my gratitude and appreciation to my advisor Professor Nayda Santiago. Her guidance and instruction has played an invaluable part in my graduate studies. An special thanks to Professor's Miriam Lesser (NEU), Gladys Ducoudray (UPRM), Manuel Toledo (UPRM), and Miguel Vélez (UPRM), for all their support during this work.

It has been a pleasure to work in the Integrated Circuit Design Laboratory (ICDL) at University of Puerto Rico, Mayagüez (UPRM) and in the Rapid Prototyping Laboratory at Northeastern University (NEU). Specially thanks to the students Marcos Mejías, Julio Sosa, Nelson Medero, Aixa Santos, Michael Ortíz, Alejandro Fernández, Albert Conti (NEU), and Sherman Braganza (NEU). They have provided a friendly, encouraging, and supportive environment for me to work.

I would also like to extend my appreciation to The Bernard M. Gordon Center for Subsurface Sensing and Imaging Systems sponsored by the Engineering Research Centers Program of the US National Science Foundation under grant EEC- 9986821, for funding my research.

Finally I would like to recognize the best family anyone could ever ask for, especially my grandparents Saturnino and Sylvia, my wife Daniela, my mother Lillian, my uncle Jorge, and my aunt Sandra. I could not have done this without their unconditional love, support, and understanding.

TABLE OF CONTENTS

LIST OF TABLES

Table page

LIST OF FIGURES

LIST OF ABBREVIATIONS

FPGA	Field Programmable Gate Array.
DSP	Digital Signal Processor.
ASIC	Application Specific Integrated Circuit.
ISRA	Image Space Reconstruction Algorithm.
MB	Microblaze Soft Core Microcontroller.
OPB	On Chip Peripheral Bus.
HSI	Hyperspectral Images.
RAM	Random Access Memory.
BRAM	Random Access Memory Block.
DDR	Double Data Rate Memory.

CHAPTER 1
INTRODUCTION

Hyperspectral sensors produce images with hundreds of channels of spectral data and million of pixels. Most image processing algorithms involve large amounts of data and most of these algorithms present large degrees of parallelism. Most of this parallelism can not be exploited on a sequential microprocessor and the large amount of data cause the memory bandwidth to be the bottleneck. An FPGA architecture for abundance estimation problem is presented that takes advantage of the parallelism in the algorithm.

In recent years, iterative algorithms used for abundance estimation have played an important role in the study of hyperspectral imaging [4–6]. Numerous algorithms exist for abundance estimation, some of which are: Image Space Reconstruction Algorithm (ISRA), Expectation Maximization Maximum Likelihood (EMML), Non Negative Sum To One (NNSTO), and Non Negative Least Square (NNLS). Unfortunately, the performance of these algorithms is relatively slow and place a heavy burden on computing systems. Studies of software implementations of ISRA show that the number of iterations is the main reason behind the long execution times [4]. Hardware implementations of algorithms related to hyperspectral image studies have shown considerable speedup, for example 146 percent, when implemented on FPGAs [7–10].

The Image Space Reconstruction Algorithm (ISRA) has been used in hyperspectral imaging applications to monitor changes in the environment and specifically, changes in coral reef, mangrove, and sand in coastal areas. This algorithm is one of

a set of iterative methods used in the hyperspectral imaging area to estimate abundance. However, ISRA due to the amount of data is highly computational, making it difficult to obtain results in a timely manner. Specialized hardware for the implementation of iterative algorithms, specially the use of FPGAs, may be exploited in order to reduce execution time. Implementation platforms, which reduce execution time of these algorithms, allows a scalable analysis of hyperspectral images. This work illustrates the use of FPGAs as a possible target architecture for hyperspectral imaging applications and the analysis of pros and cons of this implementation.

There are some important aspects of having hardware implementation of hyperspectral imaging algorithms. Hyperspectral images due to their inherent characteristic of large amount of data are difficult to process in a timely manner. Hardware units such as FPGAs provide a valuable architecture to overcome problems with intensive computations and data movements. For faster convergence hardware implementations can perform real time unmixing. If real time algorithms were feasible for hyperspectral imaging, sensors with real time estimations would be possible. This work shows the use of FPGAs to speed-up iterative hyperspectral imaging algorithms.

The abundance estimation algorithm implemented demonstrates the potential use of FPGA's in the hyperspectral imaging analysis, specially with iterative or intensive computational algorithms. We present two differents hardware implementations of the ISRA algorithm. These implementations are based on data transfer, arithmetics computations improvements and the introduction of BRAMS memory. The combination of DDR and BRAM memory on the hardware implementation provides a considerable improvement on speed.

The use of Xilinx Microblaze Soft Core Microcontroller allows the easy access to the target peripherals, like Double Data Rate (DDR) Memory, Block-RAM Memory and the serial port (RS232). The implementation process consisted on first mapping

the ISRA equation into hardware implementations. Second, the communication plan between hardware units was formulated and implemented. Finally, the memory arrangement was made that best fitted the hardware and communication pattern. The tradeoffs between speed, area, and memory are presented in this thesis.

1.1 Overview

Chapter 2 contains a literature review, explaining hyperspectral imaging, spectral unmixing, FPGAs, ISRA, and floating point computations. Chapter three, presents the different hardware and software implementations of the ISRA algorithm and the design methodology. Results of the different implementations are presented in chapter four. The variable of interest in these results is the execution time. Conclusions, lessons learned, and future directions of our research are shown in chapter five.

CHAPTER 2
LITERATURE REVIEW

In this chapter we discuss background material on hyperspectral imaging, Image Space Reconstruction Algorithm (ISRA), and Field Programmable Gate Arrays (FPGAs).

2.1 Hyperspectral Imaging

Hyperspectral Imaging (HSI) is used for environmental applications such as mineral detection, vegetation monitoring, etc. In HSI, hundreds of images are taken at narrow and contiguous spectral bands providing us with high spectral resolution data that can be used to discriminate between objects based on their spectral signature [11, 12]. HSI sensors on environmental applications have high spectral resolution and low spatial resolution, so that, the measured spectral signature is a mixture of the spectral signatures of the objects in the field of view of the sensor [12].

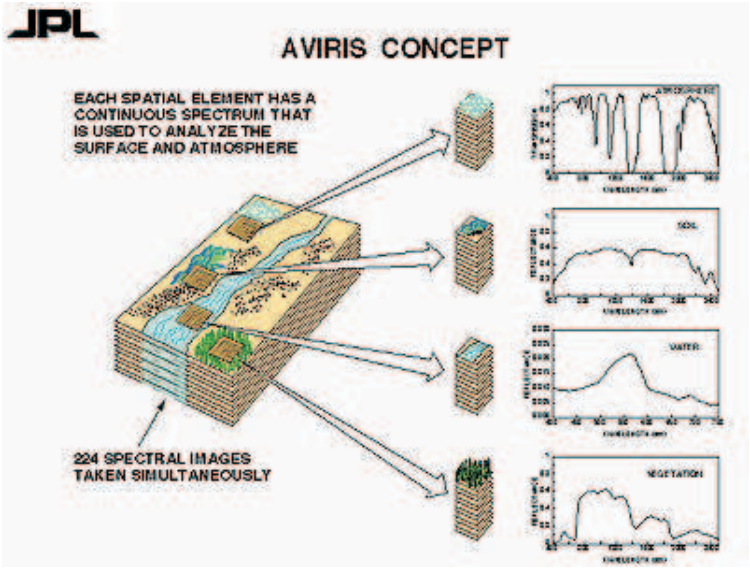

Figure 2–1: Hyperspectral Imaging. (Figure taken from NASA JPL)

Figure 2–1 shows how HSI sensor scans an area and the fashion in which the acquired data can be represented as a three dimensional cube. This data cube has spatial dimensions and a spectral dimension. Some examples of HSI sensors are the Airborne Visible/Infared Imaging Spectrometer (AVIRIS) and Hyperion. Hyperion has a spatial resolution of 30 meters, spectral resolution of 10nm ranging from $0.4 - 2.5\mu$m and 220 bands. Another example is AVIRIS, it has a spatial resolution that ranges from $4 - 20m$ meters depending on the airplane altitude, spectral resolution of 10nm ranging from $0.4 - 2.45\mu$m and 224 bands.

Hyperspectral sensors provide high spectral resolution but relative low spatial resolution. Mixed pixels are consequence of low spatial resolution of HSI sensor or could be as a results of different materials combined in a homogeneous mixture [13]. The measure spectral signature is a mixture of the signatures of the objects of the field of view of the sensor [11]. Spectral unmixing is the procedure of decompose the measure spectrum of mixed pixels into a set of spectra, endmember, and a set of corresponding abundance fractions [13], [12]. When any knowledge of the endmembers and the abundances is not known, the unmixing process is referred as Full Unmixing Problem (FUP). When a priory information of the endmembers is known, the process is referred as Abundance Estimation Problem (AEP). In the literature, different approaches to solve the unmixing problem are presented but most of them are based on the Linear Mixing Model (LMM) [12], [13], [14], given by:

$$\mathrm{b} = \sum_{i=1}^{\mathrm{n}} \mathrm{x_i a_i} + w = \mathbf{A}\mathrm{x} + w \tag{2.1}$$

where $\mathbf{A}$ is the matrix of the endmember and $\mathbf{A} \epsilon \Re_{+}^{m \times n1}$ [1] is the matrix of the endmember. The spectral signature of the i-th endmember is denoted as a_{ij} and $\mathbf{x} \epsilon \Re_{+}^{n}$ is the vector of the abundances. The measured pixel spectrum $\mathbf{b} \epsilon \Re_{+}^{m}$ and the noise vector is denoted as w. The number of endmember is n and m is the number of spectral channel of the sensor [12], [13]. Variables $\mathbf{A}$ and $\mathbf{b}$ are constrained to be positive in order to have physical meaning; in addition, the abundance vector needs to satisfy $x \geqslant 0$ and $\sum^{\mathrm{n}} \mathrm{x_i} \leq 1$.

The abundance estimation problem (AEP) can be viewed as a constrained Distance Minimization Problem (DMP) given by:

$$\hat{x} = \arg \min_{x} D\left(\mathbf{b},\mathbf{Ax}\right); \text{If } \mathbf{x} \geq 0. \tag{2.2}$$

where $\mathbf{D(b;Ax)}$ is a "distance" function, $\mathbf{A}$ is the endmember matrix, $\mathbf{b}$ is the pixel observed and $\mathbf{x}$ is the abundance vector.

2.2 Spectral Unmixing

An important challenge in HSI processing is to decompose the mixed pixels into the materials that contribute to the pixel (*endmember*) and a set of corresponding fractions of the spectral signature in the pixel (*abundances*). This problem is known as the *unmixing problem* [12, 13]. Mixed pixels are caused by a low spatial resolution of HSI sensor or as a result of different materials combined in a homogeneous mixture [13]

Pixel unmixing has important applications such as object quantification, mineral identification, plants health, automatic materials detection, and others [13, 15]. In addition, it can be used to generate a training set for image classification. Pixel

[1] $\mathbf{A} \epsilon \Re_{+}^{m \times n}$: $\mathbf{A}$ is and m x n matrix with positive real numbers.

unmixing algorithms can be partitioning in two main components: *endmember determination* and *abundance estimation* algorithms. Endmember determination is the process of determining signatures which can be considered pure. Endmember determination methods require a trained analyst or a-priori information to interact with the algorithms [13]. Abundance estimation methods are highly automated; little human interaction is required for the algorithms to execute. This is illustrated in Fig. 2–2.

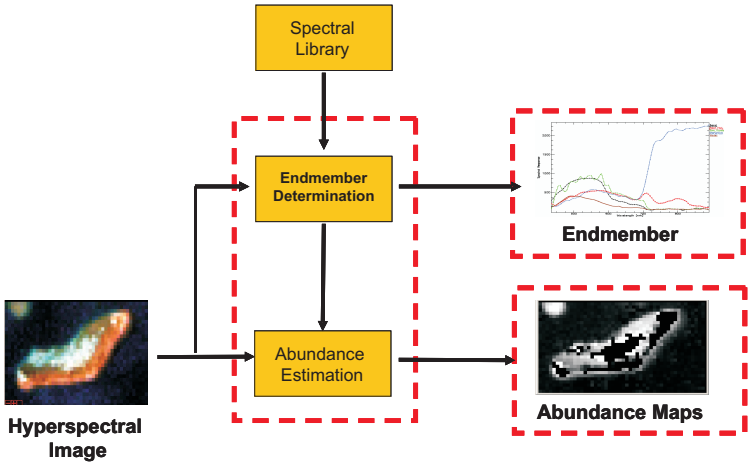

Figure 2–2: Two-Stages Unmixing Process Diagram

The most common type of iterative abundance estimation algorithms assume the endmembers are known and only estimate the abundances, this is called supervised classification [13]. This type of classification is when a-priori information is known.

2.3 Image Space Reconstruction Algorithm (ISRA)

The Image Space Reconstruction Algorithm (ISRA) is used to estimate the proportion of each endmember ($\mathbf{A}$) present in a pixel ($\mathbf{b}$) of a hyperspectral image. The abundance problem can be seen from the perspective of a distance minimization problem where the distance between the measured pixel or spectra and the estimate is the smallest. ISRA it is an iterative algorithm and an example of a Positive Constraint Only Algorithms. This means that only non negative constrains are

consider. This algorithm guarantees positive values in the result of the abundance and the convergence of the algorithm.

$$LS\left(\mathbf{Ax},\mathbf{b}\right) = \|\mathbf{Ax} - \mathbf{b}\|_2^2 \tag{2.3}$$

ISRA is used in many applications such as image reconstruction in emission tomography [16] and HSI data unmixing [12]. ISRA is a supervised classification method. These means a priory information of the endmembers is known. Equation 2.4 describes the base algorithm:

$$\hat{x}_j^{k+1} = \hat{x}_j^k \frac{\sum_{i=1}^{m} b_i a_{ij}}{\sum_{i=1}^{m} a_{ij} a_i^T \hat{\mathbf{x}}^k} \tag{2.4}$$

The number of bands and the number of endmembers are represented by m and n, respectively. Matrix $\mathbf{A}\ \epsilon\ \Re_+^{m\times n}$ is the endmembers matrix $(m \times n)$, where a_{ij} is an element of $\mathbf{A}$ and $\mathbf{a}_i$ is a vector of the spectral response of an endmember in all bands i. The term $\mathbf{b}\ \epsilon\ \Re_+^m$ is the pixel in observation (m), and $\hat{\mathbf{x}}$ is the abundance vector. Figure 2–3 shows a pseudo-code for the ISRA algorithm.

```
// For all pixels
for (t = 0; t < P; t++) {
  //Calculate abundances using ISRA for ONE pixel
  for (k = 1; k <= MAX_ITER; k++) {
    // For all Endmembers
    for (j = 0; j < N; j++) {
    // For all bands
      for (i = 0; i < M; i++) {
        numerator = numerator + A[i][j]*B[i][t];   // Aij*Bi
        // Calculate Dot Product -> transpose(Ai)*X
        //(X from previous iteration)
        for (s = 0; s < N; s++)
           dot += A[i][s]*X[s][t];
           denominator += dot * A[i][j];  // Aij*transpose(Ai)*X
            //(X from previous iteration)
           dot = 0;
      }
        // Calculate new X
        tempX = X[j][t]*(numerator/denominator);
        X[j][t] = tempX;
        numerator = 0;
        denominator = 0;
    }
  }
}
```

Figure 2–3: ISRA Pseudo-Code.

In this pseudo-code the variable P and MAX_ITER means the pixel quantity and the maximum iterations per pixel in the analysis, respectively. The ISRA equation is separated into the numerator computation and denominator computation. When these are obtained, they are divided and multiplied by the previous abundance X.

2.4 Processing Elements

Modern day designers have several devices to choose from as the implementation fabric for their application. These devices can be classified as either general purpose, application specific hardware, or reconfigurable hardware.

General-purpose hardware is a term used to describe devices that are capable of understanding instructions issued by a programmer. A general-purpose processor (GPP) is a microprocessor that has been optimized to offer moderate performance in a wide range of application domains. A programmer can issue a command to tell the device to perform any one of its pre-determined instructions at any given time.

General-purpose hardware is suitable for a variety of applications but they may fail to provide an implementation platform that is capable of meeting the system requirements for higher performance applications. In those cases, designers use application specific hardware. Application specific hardware usually takes the form of an application specific integrated circuit (ASIC). ASICs are optimized to perform only the specific function they were designed for. Reconfigurable hardware attempts to couple the performance of ASICs with the flexibility of general-purpose hardware. The most common type of reconfigurable hardware uses an array of field programmable gates (FPGAs). These gates can be configured to perform specific boolean operations. The gates are interconnected through the devices reprogrammable interconnect fabric.

2.4.1 Field Programmable Gate Arrays (FPGAs)

FPGAs are chips that can be electrically programmed and reprogrammed to implement complex functions in digital logic [17]. An ASIC implementation is often more generic to justify its high development cost, so it may be less efficient than specialized one [18]. The high set-up costs of ASICs make them unattractive in low volumes. However, the lowered costs and the rapid development of applications in FPGAs offer an alternative for implementing DSP and re-programmable solutions [18]. FPGAs demonstrate that they are powerful custom hardware for applications that require intensive computations [8, 9]. They have the programmability of software and the functional efficiency of hardware. They can be customized by the end user for a specific application. FPGAs is basically composed of an array of input-output ports, programmable routing resources, and configurable logic block's (CLB). Figure 2–4 presents a general structure of an FPGA and Figure 2–5 shows an architecture overview of Xilinx's Virtex II Pro.

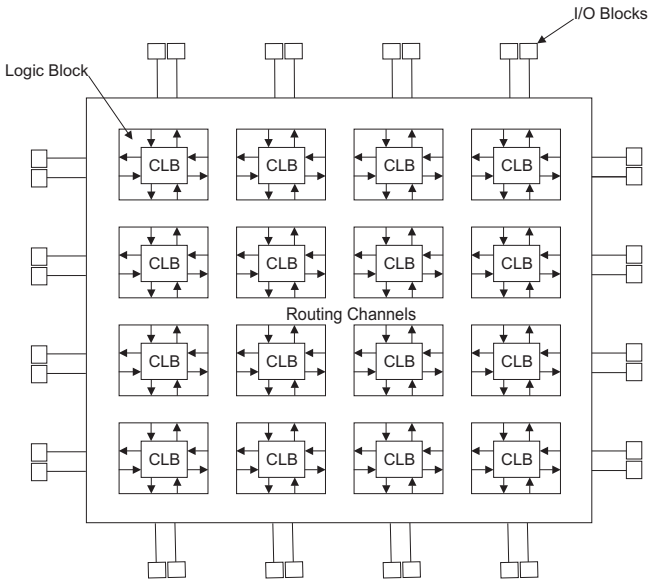

Figure 2–4: General Structure of an FPGA.

Configurable logic blocks are the fundamental part present on FPGAs. Figure 2–6 present 2-slide Virtex CLB. The functionality of CLBs can be changed by reconfiguring the contents by itself. Each CLB contains 4 slices and 2 tri-state

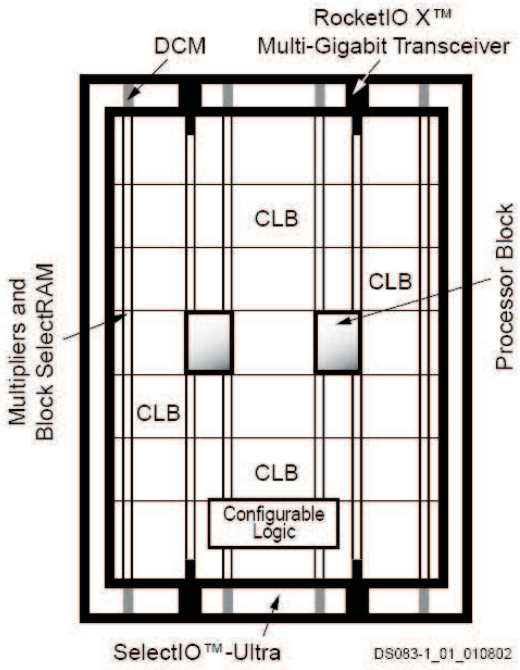

Figure 2–5: Xilinx's Virtex II Pro Architecture Overview. (Figure taken from Xillinx [1].)

buffers. Each slice is identical to the others contained within the CLB. A single slice, seen in Figure 2–7, provides two function generators, two storage elements, arithmetic logic gates, multiplexers, and fast carry logic [19]. The function generators may be configured as 4-input look-up tables (LUTs), as 16-bit shift registers, or as 16-bit distributed SelectRAM+ memory. In addition, either storage element may be configured as an edge-triggered D flip-flop or a level sensitive latch. Each CLB has its own internal interconnect, as well as access to general routing resources.

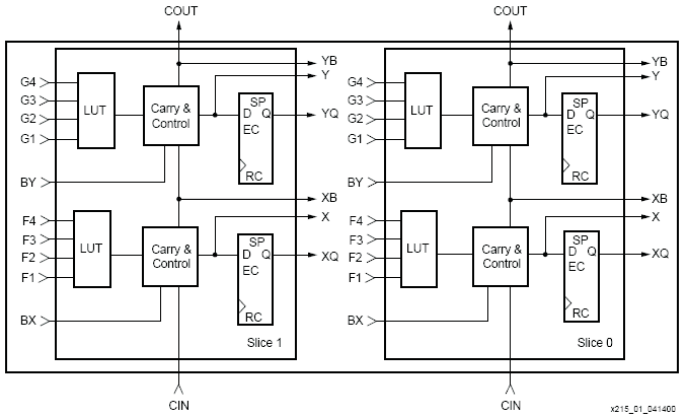

Figure 2–6: 2 Slice Virtex CLB. (Figure taken from Xilinx [2].)

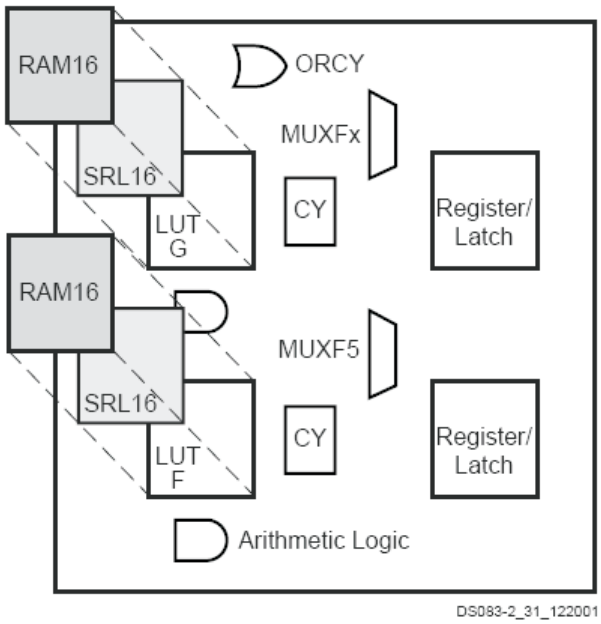

Figure 2–7: Virtex-II Pro Xilinx Slice Configuration. (Figure taken from Xilinx [2].)

Although CLBs remain the fundamental building block of an FPGA, increasing device densities have allowed additional resources into their FPGA architectures. Modern FPGA devices, such as the Virtex-II Pro, contain other reconfigurable elements such as BlockRAMs, multipliers, and general-purpose processors. Each BlockRAM provides an 18Kb dual-ported memory structure with two independently clocked and independently controlled synchronous ports that access a common storage area. Each multiplier element provides an 18-bit by 18-bit signed multiplier. They are optimized for high speed operations and have low power consumption compared to an equivalent multiplier implementation using CLBs. Finally soft cores processor, such as Microblaze, provide a multi-stage instruction pipeline capable of executing stored instructions. Soft processor is an intellectual property (IP) core implemented using the logic primitives of the FPGA.

Figure 2–8 illustrates a routing architecture model which describes or represents several commercial FPGA routing architectures. Important parts present in this figure are:

- A **wire segment** is a wire unbroken by programmable switches. One or more switches may attach to the wire segment. Each end of a wire segment typically has a switch attached.
- A **track** is a sequence of one or more wire segments in a line.
- **Routing channel** is a group of parallel tracks.
- **Connection block** provides connectivity from the inputs and outputs of a logic block to the wire segments in the channels. There can be connection blocks in the vertical direction and in the horizontal direction.
- The **switch block**, which provides connectivity between the horizontal and the vertical wire segments. In Figure 2–8, the switch block provides connectivity among the wire segments incident to its four sides.

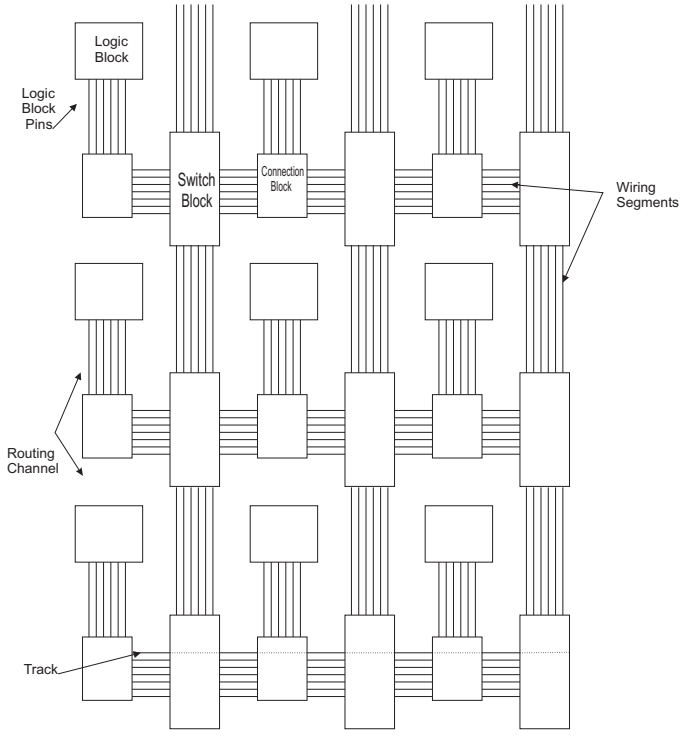

Figure 2–8: General FPGA Routing Architecture

2.5 System-on-Chip

Advances in semi-conductor industry, specifically on silicon devices, allow to designers the integration of all components present in a computer or other electronic

system into a single chip, this concept that has been termed System-on-Chip (SoC). The most common application for SoC is in the embedded systems area.

SoC may contain digital, analog, mixed-signal, and often radio-frequency functions, all on one chip. A typical SoC consists of: one or more microcontroller, microprocessor or DSP core, memory blocks (ROM, RAM, EEPROM and Flash), ADCs and DACs, timing sources (oscillators and phase-locked loops), peripherals (counter-timers, real-time timers and power-on reset generators), external interfaces (USB, FireWire, Ethernet, UART, etc), voltage regulators and power management circuits.

This integration has the potential to offer increased reliability, increased performance, lower resource utilization, and lower cost. However, such a high level of transistor density makes design and verification of these systems difficult [20]. To facilitate the design of SoC systems, the systems are build using existing components that have well-defined contents and interfaces. This use of existing components decrease development costs and time to market.

2.6 Platform FPGAs

FPGA manufactures, such as Xilinx, have begun introducing FPGAs with architectures capable of providing complete on-chip solutions. These platform FPGA architectures contain clock managers, arithmetic units, embedded memories, processors, high speed I/O etc. The Virtex-II Pro FPGA in addition to containing the traditional elements that are characteristic of previous Platform FPGA generations, contains the MicroBlaze Soft Core Processor Block.

2.6.1 MicroBlaze Architecture

The Microblaze (MB) is a soft processor system designed by Xilinx. This processor is an intellectual property (IP) core that is implemented using the logic primitives of the FPGA. The architecture is shown in Figure 2–9.

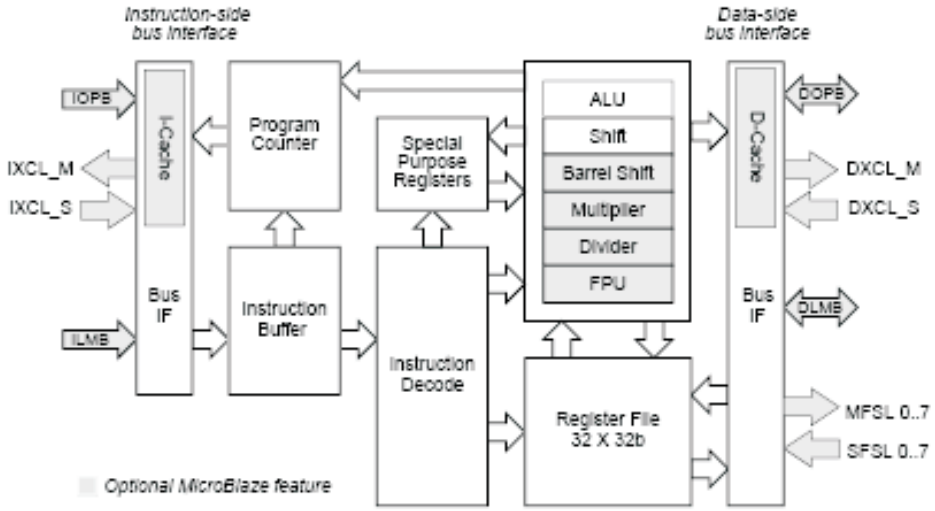

Figure 2–9: MicroBlaze Core Block Diagram. (Figure taken from Xilinx [3].)

The Microblaze is a processor system that introduces an integrated single precision, IEEE-754 compatible Floating Point Unit (FPU). Its core is a 3-stage pipeline, 32-bit RISC Harvard architecture soft processor core with 32 general purpose registers, Arithmetic Logic Unit (ALU), and an instruction set optimized for embedded applications. It supports both on-chip block RAM and external memory. The MicroBlaze has the On-Chip Peripheral Bus (OPB) to interface all the different devices and custom logic. Depending on the configured options, the MicroBlaze will use between 900 and 2600 Look-Up Tables (LUTs) and the number of processors on a single FPGA is only limited by the size of the FPGA.

2.6.2 Memory Architecture

MicroBlaze has a Harvard memory architecture. The term of Harvard Memory architecture is used to describe machines with separate memories or machines that have one main memory but when separate caches for instructions and data [21]. Microblaze, in this case the instruction and data accesses are done in separate address spaces. Each address space has a 32-bit range and handles up to 4 GByte of instructions and data memory. The instruction and data memory ranges can be made to overlap by mapping them to the same physical memory. Both instruction and data

interfaces of MicroBlaze are 32 bit wide and use big endian, bit reversed format. The MicroBlaze supports word, halfword, and byte accesses to data memory [3].

2.6.3 Floating Point Unit (FPU)

The Microblaze introduces an integrated single precision, IEEE-754 compatible Floating Point Unit (FPU). Applications created using floating-point operations in software have a higher execution time. FPU has a low latency resulting very useful in these cases. Key features for FPU are: 6 cycles for addition, subtraction, and multiplication, 30 cycles for division, 3 cycles for comparison operations, status and exceptions support (illegal operation, divide by zero, overflow, underflow, and denormalized). Other features of the FPU is the IEEE-754 format 32-bit float including infinity, not-a-number, and signed zero [22].

2.7 Related Work

Some algorithms used in image classification and dimensionality reduction have been implemented on FPGAs. Hongtao and Hairong implement a version of Independent Component Analisys (ICA) called Parallel Independent Component Analisis (pICA) [8]. The implementation of ICA algorithm (used for hyperspectral images reduction) in hardware provides an optimal parallelism environment and potential faster real time solution. The pICA algorithm was synthesized on a Xilinx VIRTEX V1000E. The FPGA maximum frequency is 20.161 MHz. It uses a pilchard board (max freq. 133 MHz) to transfer data to the CPU on a 64 bit memory bus. The pICA uses 92 percent of slices of the Xilinx V1000E. The pICA algorithm was divided into three temporally independent functional modules. The three modules are: unit estimation, internal/external decorrelation, and selection. Nordin *et al.* [7] proposed a pipeline structure for ICA implementation on FPGAs. This implementation can provide improvements of speed up to 146 percent.

Wei and Charoensak in [10], use a non iterative ICA version to determine or detect difference in sequence of images. In this work, FPGA does not offer an

optimized hardware implementation when compared to Application Specific Integrated Circuit (ASIC). Wei and Charoensak work in [10], allows short development time and enables verification of algorithms in hardware at a low cost. ICA has been successfully applied in various signal processing applications such as audio signal processing, EEG, ECG, watermarking and financial signal analysis. Due to intensive computation, ICA has not been applied very successfully in real time applications.

Hyperspectral images have a considerable quantity of information but this data must be reduced to identify the useful information. Leeser *et al.* [9] implemented on FPGA the k-means algorithm. This algorithm clusters pixels into classes, based on the spectral similarity of each pixel to other members of the class. FPGA can provide a considerably speedup and provides ease of testing of variant and trade offs. Clustering the data provides an organization very useful for other processing downstream. Each k-means iterations require a distance computation between every data point and the clusters, it uses an FPGA to accelerate this computation. FPGAs are very useful for this application for the amount of parallelism and processing bit widths that can adapted to the task.

An important field that incorporates the uses of FPGAs is the compression of hyperspectral image. Miguel *et al.* in [23], proposed a reduced-complexity coding for the Set Partitioning in Hierarchical Trees (SPIHT) algorithm. SPIHT is a progressive image coder, which approximates an image with a few bits of data, and then improves the quality of approximation as more information is encoded. The authors explain that with the use of more FPGAs they achieve a very good compression ratio.

In all these implementations the authors demonstrates the importance of using FPGAs on hyperspectral and image processing analysis.

2.8 Floating Point Arithmetic

In this section we discuss the different floating point operations required to implement the Image Space Reconstruction Algorithm. Once the algorithm was analyzed, the following primitive operations were identified as consuming most of execution time: multiplication and addition. In the following sections we present each operation in details.

2.8.1 Floating Point Adder

The floating point adder uses the IEEE Standard 754 [24]. The process to add or subtract two floating point operands are as follows:

- Decode or unmix, each operand is separated in three parts sign, exponent and mantissa.
- Add an implicit 1 to the MSB of each mantissa operand.
- Align both mantissa using Exponent Rule ($e_1 - e_2$).
- Compare both operands.
- Compute signs, add or subtract the mantissa.
- Normalize the mantissa, this process eliminate the implicit 1 added in the second step.
- Compute the exponent.
- Encode (mix) the three components of the number.

The complete process is shown in Fig. 2–10.

2.8.2 Floating Point Multiplier

A Floating Point Multiplier (FPM) is required to implement ISRA. The floating point libraries developed by Miriam Leeser and Pavle Belanovic [25] were modified to match our needs. These libraries are available to be distributed under General Public License (GPL). The FPM behavior is as follow:

- Decode or unmix, each operand is separated in three parts sign, exponent and mantissa.

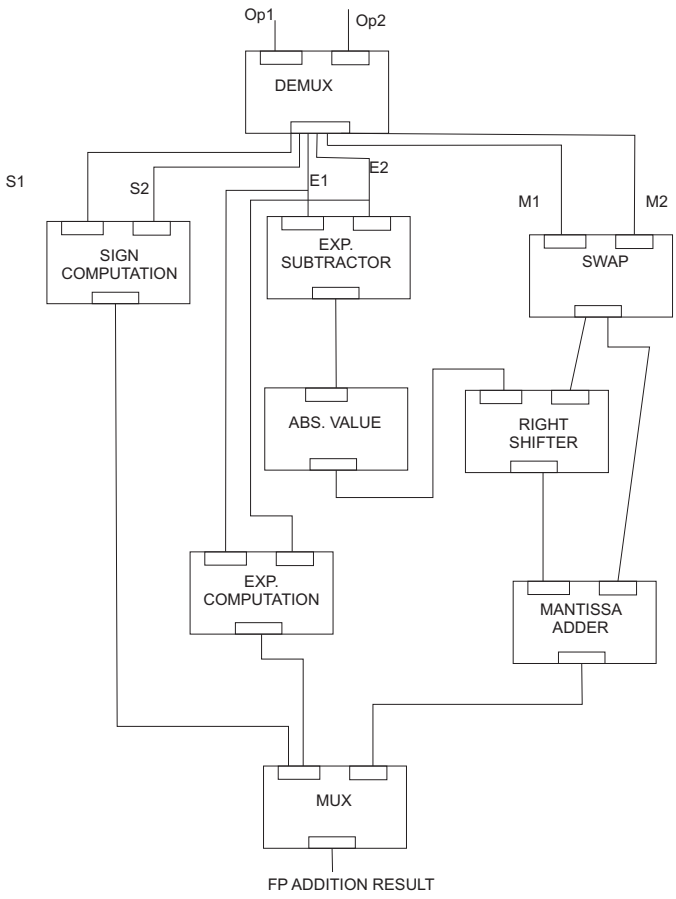

Figure 2–10: Floating Point Adder Process

- Add an implicit 1 to the MSB of each mantissa operand.
- Mantissas of the two operands are multiplied using a fixed point multiplier.
- Exponents are added.
- A bias (127) is removed from the value of the exponent addition.
- Sign bit is calculated using a *XOR*.
- Normalize the mantissa, this process eliminate the implicit 1 added in the second step.
- Encode (mix) the three components of the number.

A complete representation of floating point multiplication is given in Fig. 2–11.

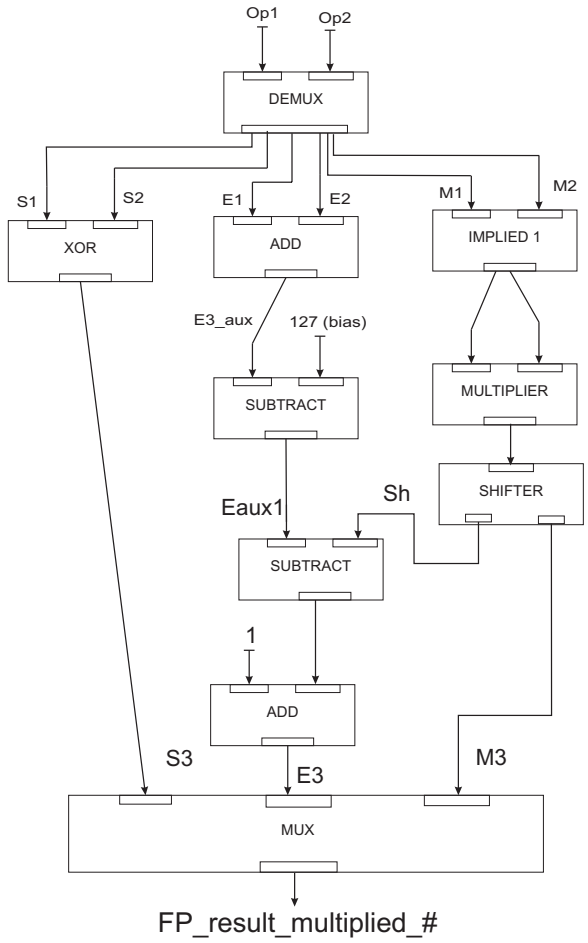

Figure 2–11: Floating Point Multiplier Process

2.9 Summary

This chapter begins with an introduction to hyperspectral imaging, FPGAs and how they differ from other modern processing elements. It then introduces System-on-Chip and FPGA's platforms. Next, a survey of related work is presented and finally, an introduction of floating point arithmetics is presented.

CHAPTER 3
OBJECTIVES AND DESIGN METHODOLOGY

In this chapter, the objectives and development tools used in this thesis is presented. A detailed description of the different implementations of ISRA algorithm is included.

3.1 Objectives

The main objective on this research is to reduce the execution time of iterative abundance estimation algorithm. Different design methods in the process of mapping the application are presented. To achieve this goal the following objectives must be accomplished:

- Develop a process or methodology to implement iterative abundance estimator algorithms on an FPGA hardware platform.
- Implement Image Space Reconstruction Algorithm (ISRA) that incorporates the proposed methodology or process.
- Present different design methods in order to reduce the execution time of this algorithm.
- Evaluate, verify, and assess the proposed methodology or process.
- Devise a set of experiments to evaluate the execution time of iterative abundance estimators algorithms.

3.2 Design Methodology

In the following sections different methodologies to accomplish the objectives of the proposed research work are presented. The tools used during research were

MATLAB 7.1, Microsoft Visual C++ 6.0, Virtex II Pro FPGA, ModelSim SE 6.2b, Xilinx ISE 7.1i and Xilinx XPS 7.1i. All this equipments and tools are located at the Integrated Circuit Design Laboratory (ICDL) of the University of Puerto Rico at Mayaüez.

3.2.1 Development Tools

The implementation of the ISRA algorithm was subdivided into two separate problems:

- The implementation of the ISRA equation using VHDL.
- The creation of the Xilinx Microblaze soft core microcontroller in order to work with the data transfer to and from the ISRA equation.

Each implementation utilizes a separate design flow. In the final stages of the design process, the implementation of the ISRA equation and Microblaze were merged together to form a complete application.

The implementation of the Microblaze required the use of Xilinx's Embedded Development Kit (EDK). The EDK is a development environment that provides application designers with the tools necessary to build embedded soft cores processor systems. The steps within the EDK that are necessary to build the embedded processor system include: hardware platform creation, software platform creation, and software application creation. The hardware platform is defined by the Microprocessor Hardware Specification (MHS) file. The MHS file defines our system architecture, memory modules, and embedded processors. It also defines the systems connectivity as well as the configurable options and the address map for each memory module in our system.

The Platform Generator (platgen) parses the MHS file and generates the appropriate netlists and HDL wrappers. These files are then imported into Xilinxs ISE Project Navigator, where they are instantiated in the application.

The software platform is defined by the Microprocessor Software Specification (MSS) file. The MSS file defines driver and library customization parameters for peripherals, standard I/O devices, interrupt handler routines and other software features. The Library Generator (libgen) tool parses the MSS file and configures the libraries and drivers that are required for the application.

Software application creation involves the creation of the Data transfer to and from the ISRA equation that executes on the embedded processor. The code is written in C. Once the source files are created, they are compiled and linked to generate executables in the Executable and Link (ELF) Format.

The ISRA equation was developed using a different design flow. First, ModelSim is used to develop a VHDL description and test the ISRA equation. The appropriate project files that enable the design to be imported into the Xilinx's ISE Project Navigator were created.

In the final stages of the design flow the ISRA equation and the Microblaze are imported into Xilinxs ISE Project Navigator. Implementation specific interfaces are then instantiated to connect the ISRA equation and the Microblaze. Finally the design is synthesized, placed and routed.

3.2.2 ISRA Implementation Using Double Data Rate (DDR) Memory Interface

This section describes a first approach to implement the ISRA algorithm. ISRA algorithm is divided in two main blocks, numerator and denominator. In this section we present details about implementation of these main blocks. In order to create the numerator and denominator blocks the floating point libraries presented in [25], were modified and adapted to the specific application. Created peripherals was necessary to build numerator and denominator parts and then used to adapt these parts to the entire system.

The numerator of the ISRA algorithm consists of an array of adders and multipliers arranged to accomplish a dot product operation. Figure 3–1 shows this arrangement. This figure shows the use of a register. This register accumulates the partial results of the previous multiplication.

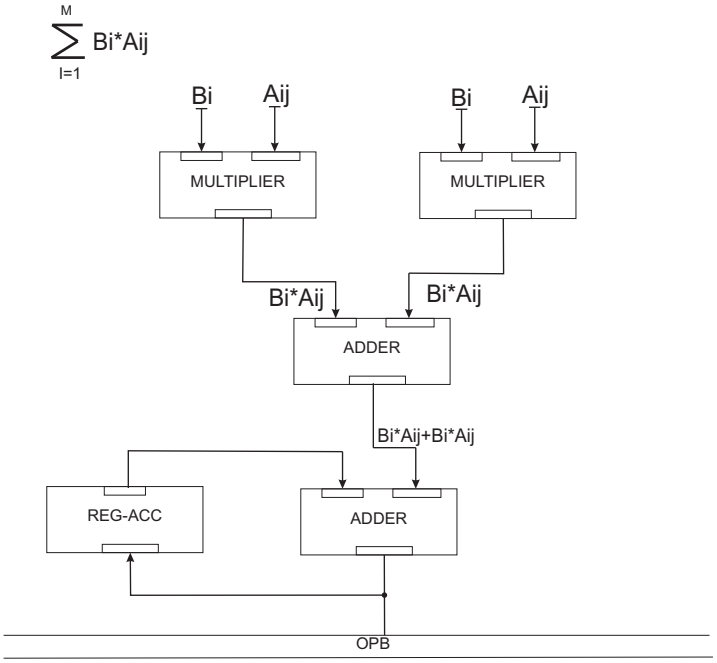

Figure 3–1: Numerator's Block Diagram

The second main block is the denominator. The denominator is composed of two multipliers, an adder and, an accumulator register. We can create with these arrays of adders and multipliers two dot product computations. The first dot product computation is shown by $A^T\hat{X}^k$ and the other dot product is the result of $A^T\hat{X}^k$ with A_{ij}. As in the numerator, a register accumulates partial results of each multiplication. A C-program controls the register, so that when the final result is obtained, the accumulation stops and the register is reset. Figure 3–2 present denominator arrangement.

A global view of the complete hardware implementation of ISRA algorithm on this first implementation is shown on Figure 3–3. This figure shows the two main blocks, numerator and denominator presented previously. Two important

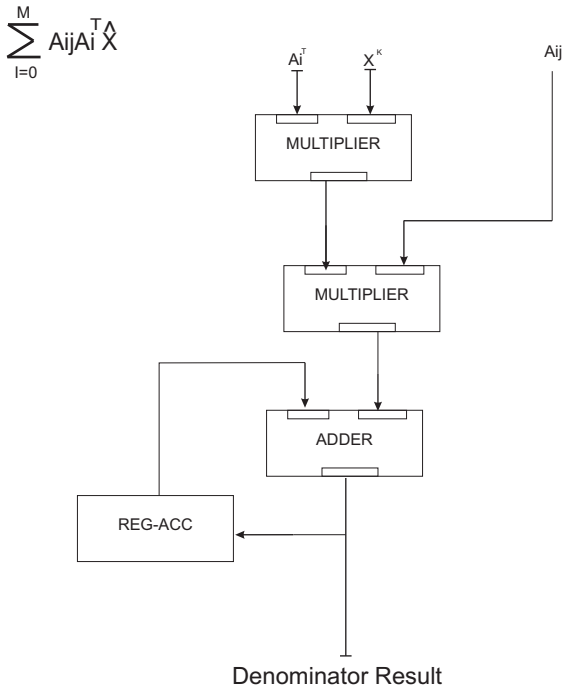

Figure 3–2: Denominator's Block Diagram

units present in this figure are a multiplier and the Xilinx Microblaze Soft Core Microcontroller. Microblaze has an important role on this implementation. It is used to send and receive data to and from the different I/O pins, and to compute the floating point division required by the ISRA equation. The Microblaze waits for the numerator and denominator computation to finish in order to compute a floating point division, and then sends the corresponding result using the OPB bus to the multiplier block show on Figure 3–3 in order to complete the computation.

3.2.3 ISRA Implementation Using Double Data Rate (DDR) and Block RAM (BRAM) data Interface

A second approach to implement ISRA was studied. Important modifications were made to this implementation to reduce execution time. The most important implementations is as follows:

- Floating point libraries developed by Miriam Leeser and Pavle Belanovic in [25] was used, to implement arithmetic units required by ISRA.

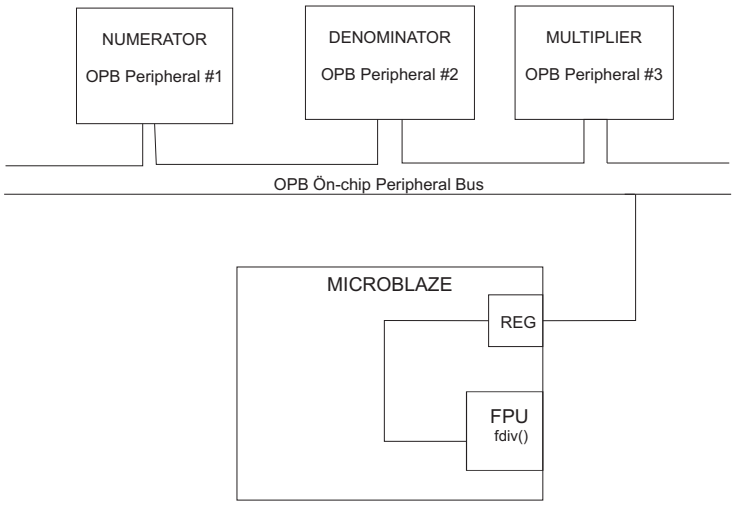

Figure 3–3: Complete Hardware Implementation Diagram

- A second modification was the use of BRAMs in order to reduce the data transfer bottleneck created by the Double Data Rate DDR memory.

Similar to the previous implementation of ISRA, the algorithm is divided in two main blocks, numerator and denominator. In this section we present details about implementation of this main blocks.

As in the previous section, the numerator part consist of a dot product operation. To develop a dot product computation using Leeser and Belanovic floating point library [25] we need denorms, floating point multiplier, round norm and, accumulator block as shown on Fig. 3–4.

The ISRA denominator consist of two dot products computations. The first dot product was the calculation of $a_i^T \hat{X}^k$ and the second was the result of $a_i^T \hat{X}^k$ times a_{ij}. The denominator part is shown on Fig. 3–5.

In this figure the numerator block represent the dot product computation. A register is used in order to synchronize the data movement for the calculation of the second dot product. After due $a_i^T \hat{X}^k$ computation is completed the register sends the a_{ij} value to calculate the second dot product.

A complete block diagram of the Image Space Reconstruction algorithm is shown in Fig. 3–6. In this figure we use the numerator and denominator blocks

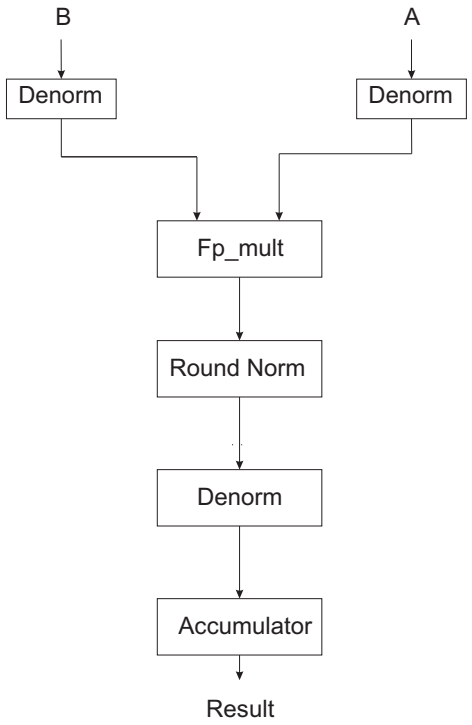

Figure 3–4: Numerator's Block Diagram

discussed previously in this section and incorporate to the design a new floating point divider and multiplier. We used a floating point divider created by the Xilinx CORE Generator. This divider was a single precision floating point divider optimized for space.

On Figure 3–7, present the different components required to accomplish the architecture organization of ISRA for this second implementation. This architecture is composed of DDR memory, Xilinx Microblaze soft core microcontroller, OPB Bus, BRAM controllers, BRAMs, and the ISRA logic.

The purpose of the DDR memory is to store the hyperspectral image, the endmembers, and the abundances. To begin the first ISRA computation the system have to wait until microblaze feeds the different BRAMs with the endmembers, first abundances and, the first two pixels in observation in order to begin the computation. An important fact to create this implementation is to fit ISRA algorithm onto the FPGA. For FPGA and memory capacity problems only stores two pixels in a BRAM. When ISRA finish with the analisys of the first pixel, has the second

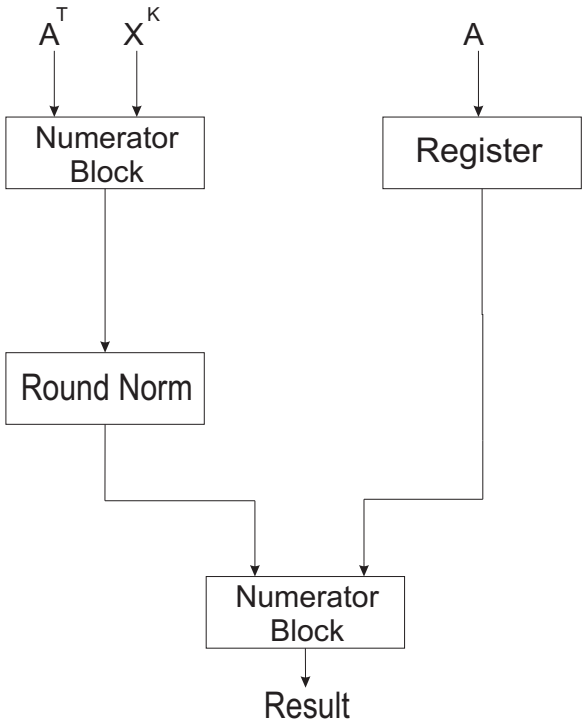

Figure 3–5: Denominator's Block Diagram

pixel available to continue with the second pixel analisys. When ISRA begins with the second pixel analysis, the microblaze has the task of feed the BRAM with a new pixel in the position of the analyzed pixel without interrupting the pixel analisys process. This process reduces the data transfer bottleneck created by the DDR memory.

3.2.4 Software Implementation

The Microblaze is a processor system designed by Xilinx that introduces an integrated single precision, IEEE-754 compatible Floating Point Unit (FPU). The MicroBlaze core is a 3-stage pipeline, 32-bit RISC Harvard architecture soft processor core with 32 general purpose registers, Arithmetic Logic Unit (ALU), and an instruction set optimized for embedded applications. It supports both on-chip block RAM and external memory.

A C program was created to send and receive data to and from the different units created in hardware. To develop the first methodology we use the floating point unit on Microblaze to make the floating point division required by ISRA algorithm.

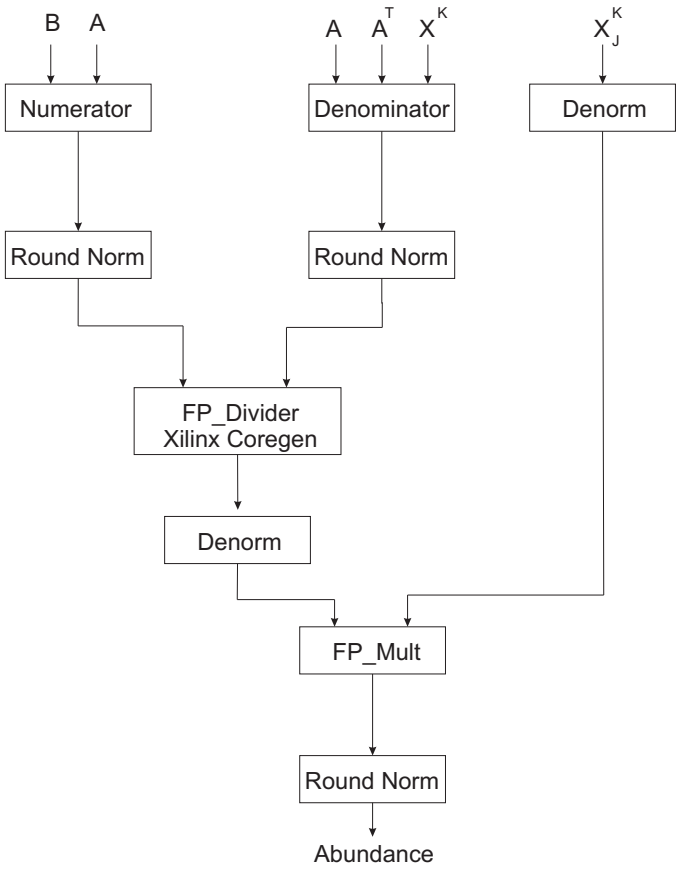

Figure 3–6: ISRA Block Diagram

The C program and all VHDL codes were compiled by Xilinx compiling tools and downloaded into the FPGA.

3.3 Summary

This chapter begins with an explanation of the differents designs methodologies and implementation to accomplish the objectives. Finally, we provide an explanation on the software used for this purpose.

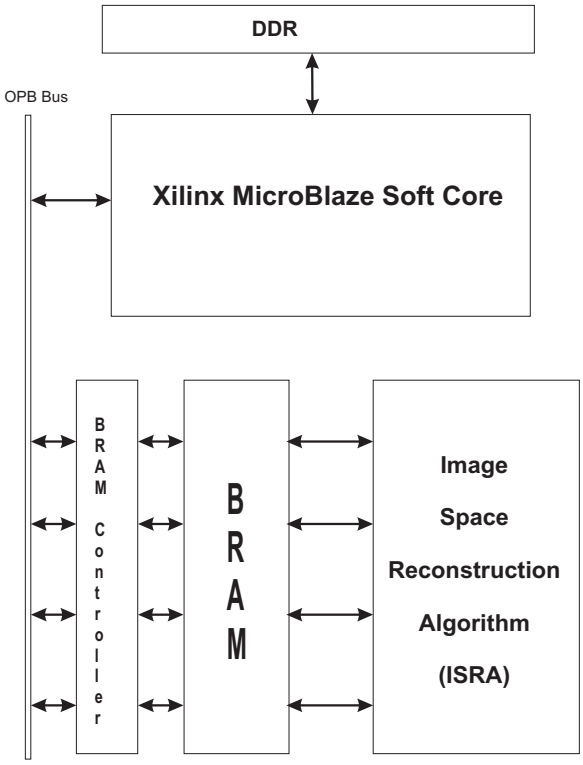

Figure 3–7: ISRA Architecture Organization

CHAPTER 4
EXPERIMENTAL RESULTS

This chapter begins by describing the technique that was used to measure the performance of the implementations presented and the HSI data use to test and compare the different implementations of ISRA algorithm. The performance results were presented for each implementation that was discussed in Chapter 3. Finally, the similarities and differences between the performance results for each implementation is analyzed.

4.1 Algorithm Validation

In order to verify that the correct results were obtained from the FPGA implementation, the following steps were taken. First, results obtained from a Matlab implementation of the algorithm were compared with those published in [4]. Second, the results obtained using the FPGA were compared with those obtained in Matlab. Once we verified that we were obtaining the correct results, then we proceeded to study the algorithm scalability and obtaining timing results.

4.2 Technique for Performance Measurement

The main variable of interest when measuring performance is execution time which is directly proportional to the number of iterations. A counter was used to measure the execution time of ISRA algorithm. In software implementations a timer function available in the system was used. In C-language and Matlab, the library time.h and the functions tic and toc were used. For the hardware implementations, a counter was used to obtain the execution time. A 32-bit register counter was incremented synchronously with the Microblaze's clock. This counter was initialized

at zero then, the value of the counter was read before starting and after ending the computations. The execution time is the number of clock cycles elapsed multiplied by the clock period (inverse of clock frequency).

4.3 Implementation Results

Real HSI data was used to test and compare the differents implementations of Image Space Reconstruction Algorithm. The HSI data used for the validation of the algorithms was taken by Hyperion HSI sensor in La Parguera, Puerto Rico. The Matlab implementation of the algorithm was done using Matlab 7.0. Both the Matlab and the C implementations were done on a Pentium 4 3.06 GHz computer with 1G RAM running the Windows XP operating system.

The HSI date used to validate the different implementations of the ISRA algorithm consists of real data taken with the Hyperion sensor over the Cayo Enrique Reef in La Parguera at Lajas, Puerto Rico. The following figures 4–1 and 4–2, shows a segment of the Hyperion Image, along with an Ikonos Image for better identification of the endmembers. The data used in this work consisted of 1632 pixels of the image and 4 endmembers spectra with 102 bands each. The endmembers were assumed to be in the image were sea grass, coral reef, sand, and sea water.

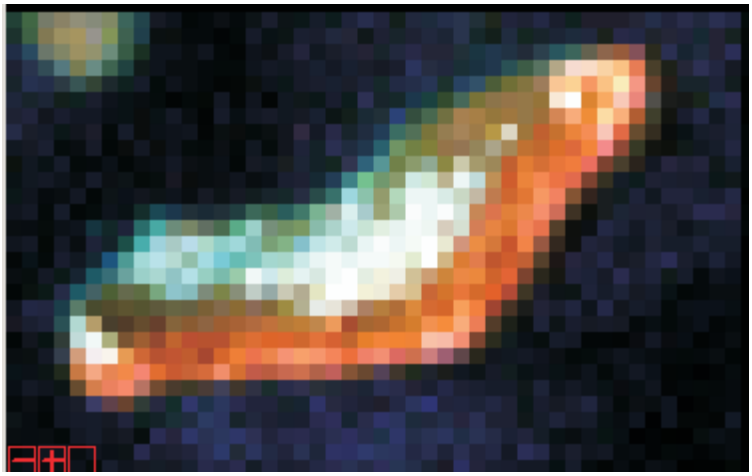

Figure 4–1: Hyperspectral Image from Hyperion Sensor

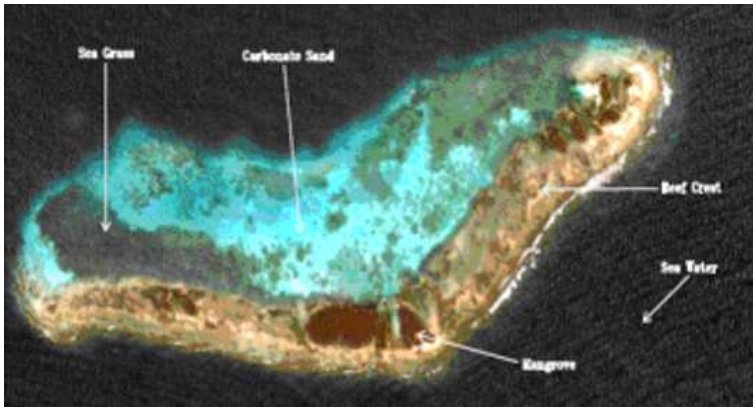

Figure 4–2: Hyperspectral Image from Ikonos Sensor

To measure the performance of the software's and hardware's implementations for both one pixel and complete HSI analysis, the following cases were chosen : 50, 150, 250, and 350 iterations. Once the clock cycle consumption of each implementation is measured, they were compared to one another. Tables and figues 4–1, 4–3, 4–2 and 4–4 presents the execution time of the abundance estimation analysis for the complete hyperspectral image and for only one pixel. Table and figure 4–3, 4–5, show the results that were obtained from the simulation of a complete HSI and for one pixel analysis. ModelSim 6.0 was used for the simulation and verification process. These tables show the performance of the different implementations presented in this work.

Table 4–1: Timing Results in Minutes of Differents Iterations for the entire Hyperspectral Image.

	Iterations			
Implementations	**50**	**150**	**250**	**350**
Matlab	7.984	23.013	38.099	54.141
C	0.067	0.171	0.267	0.367
FPGA with DDR	61.200	197.200	394.400	537.200
FPGA with DDR + BRAMS	1.563406	1.563423	1.563431	1.563439

4.4 Analysis of Results

Results show that the best execution times were obtained using C in both cases. However, the FPGA implementation using DDR and BRAMs present competitive

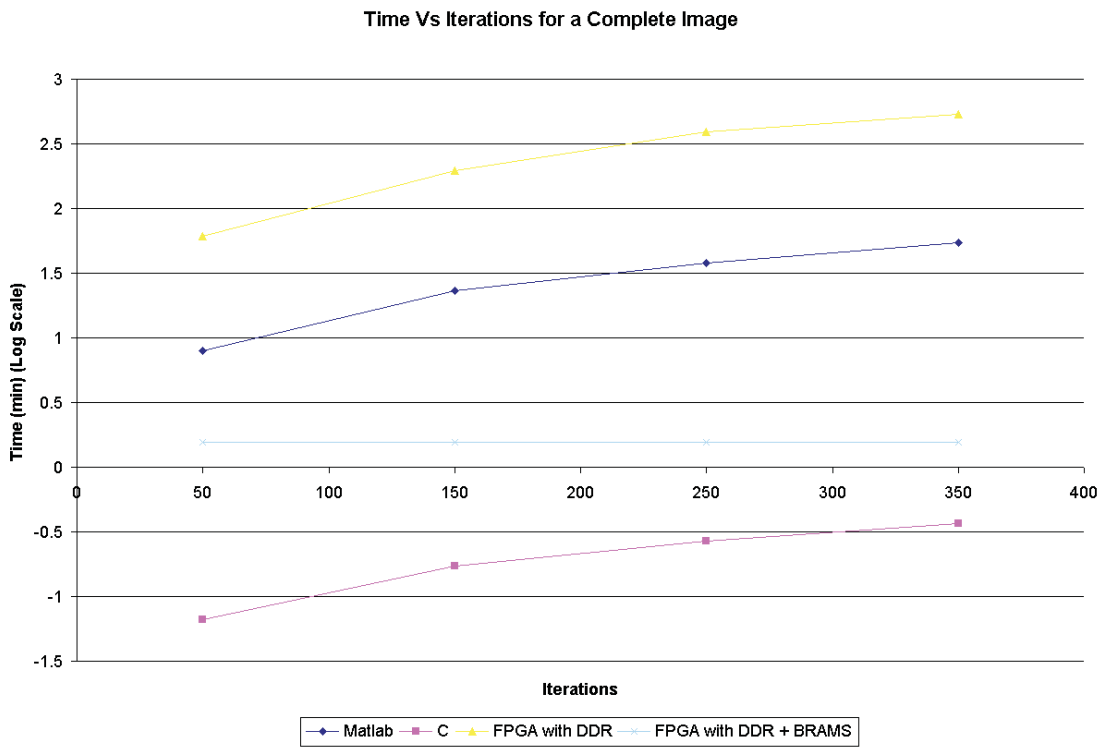

Figure 4–3: CTime results for complete image processing.

Table 4–2: Timing Results in Seconds of Differents Iterations for One Pixel ISRA Computation of the Hyperspectral Image.

	Iterations			
Implementations	**50**	**150**	**250**	**350**
Matlab	0.0002014	0.0155	0.03225	0.0595
C	0.0025	0.00675	0.01075	0.01525
FPGA with DDR	2.25	7.25	14.5	19.75
FPGA with DDR + BRAMS	0.057478	0.0574788	0.0574791	0.0574793

results. Table 4–4, resumes a comparison of the second FPGA implementation (FPGA with a combination of DDR and BRAM memory) with others implementations (hardware and software) presented in this work.

Results shown in Table 4–4, present the improvements in how many times the second FPGA implementation is faster in comparison with the others implementations presented in this work. The negative values means that no improvements are present with this comparison. In other words, the hardware implementations doesn't present an improvement. If we compare both FPGA's implementations, specially in

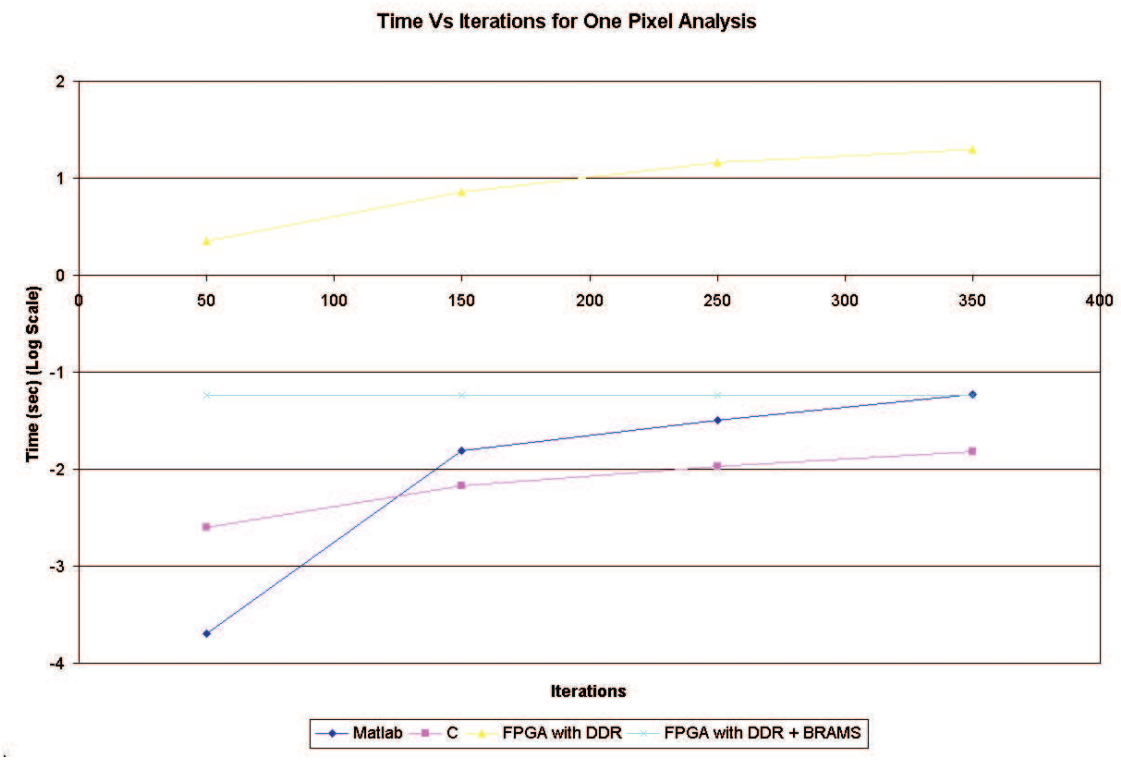

Figure 4–4: One Pixel Timing Results Plot

Table 4–3: Timing results for simulation using ModelSim.

	Iterations			
Implementations	**50**	**150**	**250**	**350**
One Pixel	2.50E-07	7.50E-07	1.25E-06	1.75E-06
Complete Hyperspectral Image	4.08E-04	1.22E-03	2.04E-03	2.86E-03

the worst case (350 iterations) we see an 343 times improvement in the execution time when we use a combination of BRAM and DDR instead if we only use DDR memory.

The FPGA implementation using DDR memory only has a larger execution time when comparing with other implementations. We have to take important details in consideration in order to analyze this results. One of the important factors to consider is the floating point single precision divider required to complete each interaction done in software. The software divider takes more than 35 clock cycles to get a valid result. This process introduce a considerable delay in the analysis. Another important detail is the bottleneck in the data transfer formed by the OPB

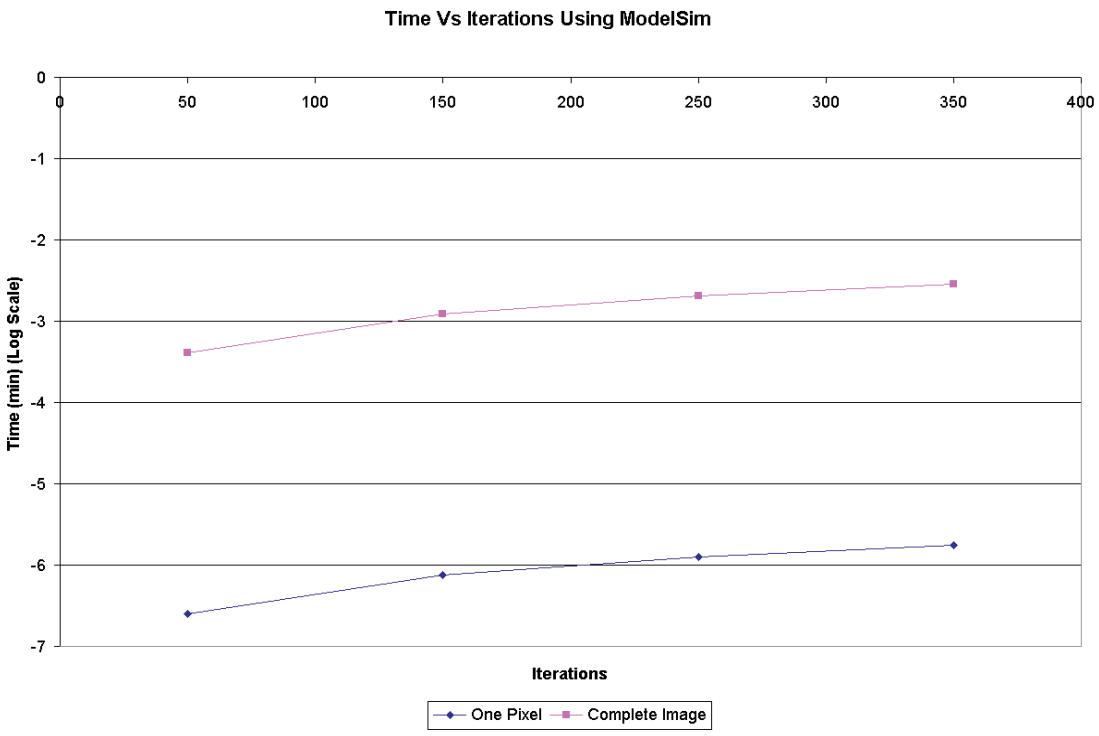

Figure 4–5: ModelSim Results Plot

Table 4–4: Execution times when comparing the FPGA with DDR and BRAM memory implementation with other alternatives.

	Iterations			
Comparison	**50**	**150**	**250**	**350**
Matlab	5.11	14.71	24.37	34.63
C	-0.43	-0.11	-0.17	-0.23
FPGA with DDR	39.15	126.13	252.27	343.60

bus. The On Chip Peripheral Bus (OPB) is the principal bus of the Microblaze (MB). All peripheral presents in the MB are attached to this bus, creating a bottleneck in the data transfer. Considering the delay formed by the software divider, data transfer bottleneck created in the OPB bus, and the delay formed by the data transfer of the DDR memory to the microblaze explains some of the large delays in the implementation.

The second hardware implementation has a large improvement in execution time. Most of the problems occurring in the last implementation were solved. In this later hardware implementation the floating point libraries presented in [25] and

the floating point divider created by the Xilinx Core Generator were used. The large delay caused by the software floating point divider was eliminated.

In this variation of the solution, the data is readily available in the BRAMs for the computation of the ISRA equation. The most important tasks of the MB is maintaining the BRAMs with the required data to make the mathematical computations. When these results are studied, can deduce that taking into consideration the architectural features of the FPGAs, timing improvements can be made, making this hardware feasible for the algorithm implementation. Taking advantage of this access, the data transfer bottleneck was eliminated using a combination of DDR and BRAM memory.

4.5 Summary

This chapter present the performance results for all of the implementations. It can be seen that the C language implementation provide the fastest results. The FPGA's implementation provides comparable performance to the C language implementation when the DDR and BRAM memory is used. Analyzing the results, it can be seen the importance of the use of appropriate memory schemes in the implementation. Finally, the simulated results for the complete HSI and for one pixel analysis were presented.

CHAPTER 5
CONCLUSION AND FUTURE WORK

5.1 Conclusion

This thesis presents the comparison between two hardware implementations of ISRA algorithm on FPGAs. These implementations demonstrate the feasibility of the use of FPGA's in the hyperspectral imaging analysis specially with iterative or intensive computational algorithms. The hardware implementations were used for abundance estimation analysis of HSI data. We have noticed that each algorithm should be mapped to the hardware that it runs on in order to obtain appropriate timing results.

Results show that better results are obtained when implementing the ISRA algorithm in C. However, even though the C implementation is better, when analyzing results we can observe that if we had a board with larger memory available, we could possibly obtain better results, making the FPGA an adequate alternative for algorithm acceleration in hyperspectral imaging. The particular details on the FPGA board will influence the results obtained in the algorithm implementation. The results of this work demonstrate that the main bottleneck that limits improving execution time is the board's memory capacity and data transfer.

ASIC implementations are another alternative to accelerate HSI algorithms. However, since the cost of producing an ASIC is large in terms of price and time, this alternative is not appropriate for algorithm testing and development. FPGAs are simple to be reprogrammed, their configuration can be easily changed, and they are cost effective. One of the disadvantages of FPGA implementations is that it is

highly dependent on memory capacity. If the FPGA does not have enough available memory, its important to find alternatives to fit the algorithm to the available memory on the board. FPGA's boards that have large memory spaces are expensive. This thesis was developed with the Xilinx XUP FPGA Board that contains one Virtex-II Pro XC2VP30 FPGA.

The board we used has 306 kB of BRAM and 13,696 slices. The first implementation used 89 percent of slice and the second implementation used 98 percent of BRAM and 93 percent of slice. Memory and slices available in the FPGA is an important constraint since they limit the design process. For that reason we have to limit the size of HSI and the endmembers to a maximum of 700 kB and the parallelism to two ISRA computations at the same time. The use of DDR memory to store the complete image and the endmembers and send only two pixels for the BRAMS at the computation time eliminates the data transfer bottleneck of the system. This was possible because ISRA computes the analysis pixel by pixel. This alternative may bye used when the FPGA does not have enough memory available and acceptable performance is needed in terms of time.

Our main contributions are:

- We demonstrated the feasibility of the use of FPGA's in the hyperspectral imaging analysis specially with iterative or intensive computational algorithms.
- We demonstrated that the main bottleneck that limits improving execution time is the board's memory capacity and data transfer.
- Finally, we have shown that if an FPGA does not have enough available memory, its important to find alternatives to fit the algorithm to the available memory on the board and minimize communication bottlenecks.

5.2 Future Work

In this research, a hardware implementation of Image Space Reconstruction Algorithm was addressed, so this allows for future work in the following topics:

- The use of larger and fasters FPGA chips, such as the Wildstar II Pro can significantly improve the results that may be obtained.
- Use FPGA's with large memory spaces available in order to explore this type of analysis using biomedical imaging. High memory FPGA's can be used to analyze larger images.
- The implementation of this algorithm using the PowerPC soft core instead of the Microblaze.
- Eliminate the complete use of microcontroller's in the design and test its effect on the design.
- Implement additional hyperspectral algorithms on FPGA based on the conclusions of this thesis.

REFERENCE LIST

[1] Xilinx. *Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet*, 2005. http://direct.xilinx.com/bvdocs/publications/ds083.pdf.

[2] The Xilinx Corporation. *Design Tips for HDL Implementation of Arithmetic Functions.*, June 2000. Accessed September 2006. http://www.xilinx.com/bvdocs/appnotes/xapp215.pdf.

[3] The Xilinx Corporation. *MicroBlaze Processor Reference Guide*, October 2005. Accessed September 2006. http://www.xilinx.com/ise/embedded/mb_ref_guide.pdf.

[4] Samuel Torres Rosario. Iterative algortitms for abundance estimation on unmixing of hyperspectral imagery. *Master Thesis, University of Puerto Rico*, 2004.

[5] W. Worstell H. Kudrolli and V. Zavarzin. SS3D - fast fully 3d PET iterative reconstruction using stochastic sampling. *Nuclear Science, IEEE Transactions*, 49(1):124 –130, Feb 2002.

[6] A. R. De Pierro. On the relation between ISRA and the EM algorithm for positron emission tomography. *IEEE Transactions on Medical Imaging*, 12(2), June 1993.

[7] Hsu C. Nordin A. and Szu H. Design of fpga ica for hyperspectral imaging processing. *Proceedings of the SPIE. The International Society for Optical Engineering*, 4391:444 – 454, 2001.

[8] Hongtao Du and Hairong Qi. An FPGA implementation of parallel ICA for dimensionality reduction in hyperspectral images. *Proceedings of IEEE International, Geoscience and Remote Sensing Symposium. IGARSS 04*, 5:3257 –

3260, Sept. 2004.

[9] M. Estlick M. Leeser, J. Theiler and J. J. Szymanski. Design tradeoffs in a hardware implementation of the K-Means clustering algorithm. *Proceedings of the IEEE. Sensor Array and Multichannel Signal Processing Workshop.*, pages 520 – 524, March 2000.

[10] Yu Wei and C. Charoensak. FPGA implementation of non-iterative ICA for detecting motion in image sequences. In *Control, Automation, Robotics and Vision, 2002. ICARCV 2002. 7th International Conference on*, volume 3, pages 1332–1336, December 2002.

[11] Miguel Velez-Reyes, Luis O. Jimenez-Rodriguez, Daphnia M. Linares, and Hector T. Velazquez. Comparison of matrix factorization algorithms for band selection in hyperspectral imagery. *Algorithms for Multispectral, Hyperspectral, and Ultraspectral Imagery VI*, 4049(1):288–297, 2000.

[12] Miguel Velez-Reyes, Angela Puetz, Michael P. Hoke, Ronald B. Lockwood, and Samuel Rosario. Iterative algorithms for unmixing of hyperspectral imagery. *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery IX*, 5093(1):418–429, 2003.

[13] Keshava N. and Mustard J. F. Spectral unmixing. *IEEE Signal Processing Magazine*, 19:44 – 57, January 2002.

[14] Plaza A.; Martinez P.; et.al. Spatial-spectral endmember extraction by multidimensional morphological operations. *IEEE Transactions on Geoscience and Remote Sensing.*, 9:2025 – 2041, 2002.

[15] Kruse F.A. Visible-infrared sensor and case studies. *Remote Sensing for the Earth Sciences: Manual of Remote Sensing*, page 3, 1999.

[16] Daube-Witherspoon M.E. and Muehllehner G. An Iterative Image Space Reconstruction Algorithm Suitable for Volume ECT. *In IEEE Transactions on Medical Imaging*, (2), June 1986.

[17] S. Hauck. The roles of FPGAs in reprogrammable systems. *Proceedings of the IEEE*, 86(4):615–638, 1998.

[18] Smith R.W. Walke R.L. and Lightbody G. 20 gflops qr processor on a xilinx virtex-e fpga. *Proceedings of the SPIE. The International Society for Optical Engineering*, 4116:300 – 310, 2000.

[19] H. Verma. Field programmable gate arrays. *IEEE Potentials*, 18(4):34–36, 1999.

[20] Joshua Noseworthy. Enabling communications between an fpga's embedded processor and its reconfigurable resources. *Master Thesis, Northeastern University*, 2005.

[21] Hennessy J.L and Patterson D.A. *Computer Organization and Design, The Hardware/Software Interface*. Morgan Kaufmann Publishers, Inc, second edition, 1998.

[22] The Xilinx Corporation. *MicroBlaze Frequently Asked Questions*. Accessed September 2006. http://www.xilinx.com/ipcenter/processor_central/microblaze/doc/mb_faq.pdf.

[23] Chang A. Hauck S. Ladner R.E. Miguel A.C, Askew A.R. and Riskin E.A. Reduced complexity wavelet-based predictive coding of hyperspectral images for fpga implementation. *Proceedings DCC. Data Compression Conference*, pages 469 – 478, 2004.

[24] IEEE Standards Board and ANSI. IEEE standard for binary floating-point arithmetic. *IEEE STD 754-1985*, 1985.

[25] P. Belanovic and M. Leeser. A library of parameterized floating-point modules and their use. *Proceedings of the Reconfigurable Computing, 12th International Conference on Field-Programable Logic and Applications*, 2438:657, 2002.

BIOGRAPHICAL SKETCH

Javier Morales was born in August 9, 1981 in Humacao, Puerto Rico. Javier is son of Saturnino and Sylvia Morales. In December of 2004 he obtained his bachelor's degree in Electrical Engineering with specialization in Electronics on University of Puerto Rico, Mayagüez Campus. In January 2005, he started his graduate education. He worked under the supervision of Dr. Nayda G. Santiago. Javier spent two years doing research in the area of remote sensing and FPGA's. His areas of interest are field programmable gate array (FPGA's), design, simulation, debugging, and analysis of digital circuits, algorithms acceleration, remote sensing, and embedded systems.