2011 IEEE 17th International Conference on Parallel and Distributed Systems

FPGA Design of an Automatic Target Generation Process for Hyperspectral Image Analysis

Sergio Bernabé*, Sebastián López†, Antonio Plaza*, Roberto Sarmiento† and Pablo García Rodríguez‡
*Hyperspectral Computing Laboratory
Department of Technology of Computers and Communications
University of Extremadura, Avda. de la Universidad s/n, E-10003 Cáceres, Spain
E-mail: {sergiobernabe,aplaza}@unex.es
†Institute for Applied Microelectronics (IUMA)
University of Las Palmas de Gran Canaria, E-35017 Tafira Baja, Spain
Email: seblopez@iuma.ulpgc.es
‡Grupo de Ingeniería de Medios (GIM)
University of Extremadura, Avda. de la Universidad s/n, E-10003 Cáceres, Spain
Email: pablogr@unex.es

***Abstract*—Onboard processing of remotely sensed hyperspectral data is a highly desirable goal in many applications. For this purpose, compact reconfigurable hardware modules such as field programmable gate arrays (FPGAs) are widely used. In this paper, we develop a new implementation of an automatic target generation process (ATGP) for hyperspectral images. Our implementation is based on a design methodology that starts from a high-level description in Matlab (or alternative C/C++) and obtains a register transfer level (RTL) description that can be ported to FPGAs. In order to validate our new implementation, we develop a quantitative and comparative study using two different FPGA architectures: Xilinx Virtex-5 and Altera Stratix-III Altera. Experimental results have been obtained in the context of a real application focused on the detection of mineral components over the Cuprite mining district (Nevada), using hyperspectral data collected by NASA's Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS). Our experimental results indicate that the proposed implementation can achieve peak frequency designs above 200MHz in the considered FPGAs, in addition to satisfactory results in terms of target detection accuracy and parallel performance. This represents a step forward towards the design of real-time onboard implementations of hyperspectral image analysis algorithms.**

***Keywords*-Hyperspectral image analysis, target detection, field programmable gate arrays (FPGAs).**

I. Introduction

Hyperspectral imaging [1] is concerned with the measurement, analysis, and interpretation of spectra acquired from a given scene (or specific object) at a short, medium or long distance by an airbone or satellite sensor. Hyperspectral imaging instruments such as the NASA Jet Propulsion Laboratory's Airbone Visible Infra-Red Imaging Spectrometer (AVIRIS) [2] are now able to record the visible and near-infrared spectrum (wavelength region from 0.4 to 2.5 micrometers) of the reflected light of an area 2 to 12 kilometers wide and several kilometers long using 224 spectral bands [3]. The resulting "image cube" (see Fig. 1) is a stack of images in wich each pixel (vector) has an associated spectral signature or fingerprint that uniquely characterizes the underlying objects. The resulting data volume typically comprises several GBs per flight [4].

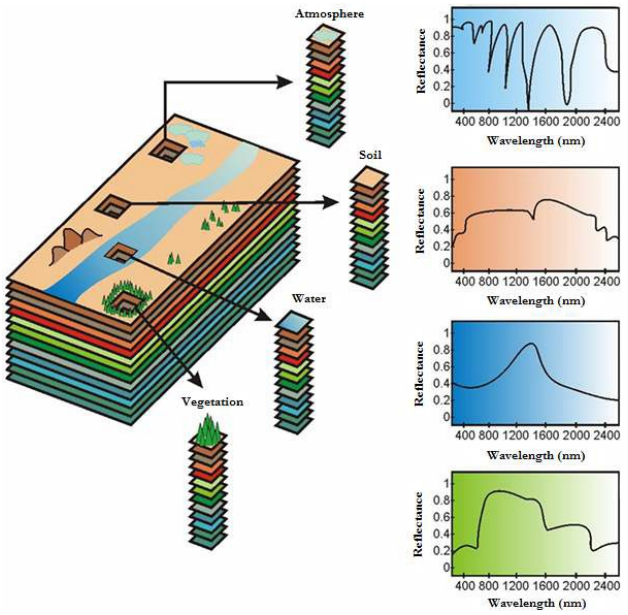

Figure 1. Concept of hyperspectral imaging.

One of the most important challenges in hyperspectral image analysis is computational complexity, which results from the need to process enormous data volumes [5]. With recent advances in reconfigurable computing, especially using field programmable gate arrays (FPGAs) [6], [7], [8],

1521-9097/11 $26.00 © 2011 IEEE
DOI 10.1109/ICPADS.2011.64

hyperspectral imaging algorithms can now be accelerated for onboard exploitation using high-performance FPGAs. One of the most critical issues when designing hardware-based (in general) and FPGA-based (in particular) implementations of hyperspectral imaging algorithms is the complexity of the design, which introduces a learning curve for algorithm developers [9], [10], [11]. Although the hardware design is usually carried out at the register-transfer level (RTL), the design can be achieved through a flow starting from a high-level Matlab or C/C++ code, thus allowing for the reutilization of available high-level versions of hyperspectral imaging algorithms. This simplifies the design flow and the adaptation of available algorithms, which is a very important aspect as we anticipate that the success of hyperspectral imaging algorithms will be soon given by their adaptivity to high performance computing platforms able to operate onboard the sensor. For instance, the Embedded Matlab module[1] allows generating efficient code in C and C++ from a high-level Matlab description. Then, tools such as Mentor Graphics' Catapult C[2] can bridge the gap between the high-level implementation and a lower-level synthesis for FPGA hardware design. These tools offer the potential to significantly help the programmer in the development of low-level hardware code with a more productive level of abstraction.

In this paper, we explore the utility of these tools to develop several FPGA-based synthesis version of a popular technique for automatic target detection in hyperspectral images. The technique, called automatic target generation process (ATGP) [12], automatically detects a set of spectrally distinct *targets* in a hyperspectral image, with the ultimate goal of characterizing the different spectral constituents that compose the scene. It is widely used in applications involving the detection and monitoring of fires, oil spills, and other types of chemical and biological agents [13]. The remainder of the paper is organized as follows. Section II describes the considered implementation of ATGP algorithm, which represents a modification over the original method described in [12] which avoids a matrix inverse operation that cannot be efficiently implemented in hardware. Section III describes the flow of high-level design from a Matlab code. Section IV describes several hardware versions of the proposed implementation of ATGP. Section V provides a experimental assessment of the developed parallel versions on two different FPGA architectures: Xilinx Virtex-5 and Altera Stratix-III, using a well-known hyperspectral data set collected by AVIRIS in order to evaluate the target detection accuracy of the proposed versions. Finally, section VI concludes with some remarks and hints at plausible future research lines.

[1]http://www.mathworks.es/help/toolbox/eml/index.html
[2]http://www.mentor.com/esl/catapult/overview

II. ATGP IMPLEMENTATION

In this section we describe our implementation of the ATGP algorithm. The original algorithm is based in an iterative process in which orthogonal projections are applied to find a set of spectrally distinct pixels [12]. Let $\mathbf{x}_0$ be an initial target signature (i.e., the pixel vector with maximum length in the original n-dimensional hyperspectral image $\mathbf{F} \in \mathbf{R}^n$). This algorithm uses an orthogonal projection operator which is given by the following expression:

$$P_{\mathbf{U}}^{\perp} = \mathbf{I} - \mathbf{U}(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T, \quad (1)$$

where $\mathbf{U}$ is a matrix of spectral signatures, $\mathbf{U}^T$ is the transpose of this matrix, and $\mathbf{I}$ is the identity matrix. This orthogonal projection operator is applied to all image pixels, with $\mathbf{U} = [\mathbf{x}_0]$. It then finds a target signature, denoted by $\mathbf{x}_1$, with the maximum projection in $< \mathbf{x}_0 >^{\perp}$, which is the orthogonal complement space linearly spanned by $\mathbf{x}_0$. A second target signature $\mathbf{x}_2$ can then be found by applying another orthogonal subspace projector $P_{\mathbf{U}}^{\perp}$ with $\mathbf{U} = [\mathbf{x}_0, \mathbf{x}_1]$ to the original image, where the target signature that has the maximum orthogonal projection in $< \mathbf{x}_0, \mathbf{x}_1 >^{\perp}$ is selected as $\mathbf{x}_2$. The above procedure is repeated until a set of target pixels $\{\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_t\}$ is extracted, where t is an input parameter to the algorithm.

Our modification of the ATGP consists of using the Gram-Schmidt method for calculating the orthogonal projection instead of the classic orthogonal projection, which involves an inverse calculation that is expensive to compute in hardware. The ultimate goal is to orthogonalize a set of vectors in an inner product space, usually the space $\mathbf{R}^n$ in which the original hyperspectral image $\mathbf{F}$ is defined. The Gram-Schmidt process is a finite set of linearly independent vectors $\mathbf{S} = \{\mathbf{v}_1, \cdots, \mathbf{v}_k\}$ for $k \leq n$, and generates an orthogonal set of vectors $\mathbf{S}' = \{\mathbf{u}_1, \cdots, \mathbf{u}_k\}$ which extends the same k-dimensional subspace of $\mathbf{R}^n$ as $\mathbf{S}$. Under these assumptions, we can now define the projection operator using the following expression:

$$proj_{\mathbf{u}}(\mathbf{v}) = \frac{< \mathbf{v}, \mathbf{u} >}{< \mathbf{u}, \mathbf{u} >}\mathbf{u}, \quad (2)$$

where $< \mathbf{v}, \mathbf{u} >$ denotes the inner product of vectors $\mathbf{v}$ and $\mathbf{u}$. The Gram-Schmidt process now continues as follows:
where the sequence $\mathbf{u}_1, \cdots, \mathbf{u}_k$ is the system of orthogonal vectors required, and the normalized vectors $\mathbf{e}_1, \cdots, \mathbf{e}_k$ form an orthonormal set. The calculation of the sequence $\mathbf{u}_1, \cdots, \mathbf{u}_k$ is called Gram-Schmidt orthogonalization, while the calculation of the sequence $\mathbf{e}_1, \cdots, \mathbf{e}_k$ is referred to as Gram-Schmidt orthonormalization (i.e. the vectors are normalized). To verify that the above equations produce an orthogonal sequence, we calculate first $< \mathbf{u}_1, \mathbf{u}_2 >$ substituting the above equation for $\mathbf{u}_2$, we would obtain a result equal to 0. Same for calculating $< \mathbf{u}_1, \mathbf{u}_3 >$ again replacing the equation by $\mathbf{u}_3$, we obtain a result equal to

$$
\begin{aligned}
\mathbf{u}_1 &= \mathbf{v}_1, & \mathbf{e}_1 &= \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|} \\
\mathbf{u}_2 &= \mathbf{v}_2 - \mathrm{proj}_{\mathbf{u}_1}(\mathbf{v}_2), & \mathbf{e}_2 &= \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|} \\
\mathbf{u}_3 &= \mathbf{v}_3 - \mathrm{proj}_{\mathbf{u}_1}(\mathbf{v}_3) - \mathrm{proj}_{\mathbf{u}_2}(\mathbf{v}_3), & \mathbf{e}_3 &= \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|} \\
\mathbf{u}_4 &= \mathbf{v}_4 - \mathrm{proj}_{\mathbf{u}_1}(\mathbf{v}_4) - \mathrm{proj}_{\mathbf{u}_2}(\mathbf{v}_4) - \mathrm{proj}_{\mathbf{u}_3}(\mathbf{v}_4), & \mathbf{e}_4 &= \frac{\mathbf{u}_4}{\|\mathbf{u}_4\|} \\
&\vdots & &\vdots \\
\mathbf{u}_k &= \mathbf{v}_k - \sum_{j=1}^{k-1} \mathrm{proj}_{\mathbf{u}_j}(\mathbf{v}_k), & \mathbf{e}_k &= \frac{\mathbf{u}_k}{\|\mathbf{u}_k\|}
\end{aligned}
$$

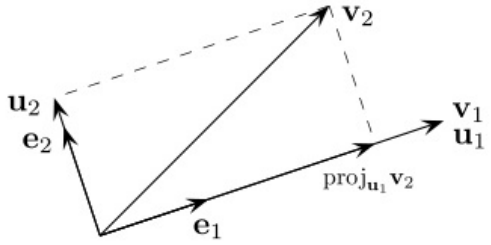

Figure 2. Geometric representation of the Gram-Schmidt process.

0. The general test is obtained by mathematical induction. In geometrical terms (see Fig. 2), this method proceeds as follows: to calculate $\mathbf{u}_i$, $\mathbf{v}_i$ is projected orthogonally to the subspace $\mathbf{U}$ generated by $\mathbf{u}_1, \cdots, \mathbf{u}_{i-1}$, which is the same as the subspace spanned by $\mathbf{v}_1, \cdots, \mathbf{v}_{i-1}$. The vector $\mathbf{u}_i$ is defined as the difference between $\mathbf{v}_i$ and the projection of this, ensuring that it is orthogonal to all vectors in the subspace $\mathbf{U}$.

III. Implementation flow: from high-level to low-level design

This section presents the implementation flow which starts from a high-level sequential version developed in Matlab or C/C++ code to the low-level implementation. The process is summarized in Fig.3 and consists of the following steps:

1) We use the Embedded MATLAB module to generate efficient code in C/C++, so that we can create prototypes and develop embedded systems to accelerate fixed-point algorithms.
2) We use Catapult C for high-level synthesis of FPGA hardware design. From a given code in C, Catapult C produces an output for generating RTL code.
3) We use a design tool for FPGAs (such as Xilinx ISE Design suite 13.1[3] or Altera Quartus II[4]) to compile the design and obtain the frequency as well as the amount of FPGA resources requested that can be achieved with the hardware implementation. The software package Precision RTL Synthesis is used to optimize the design and introduce advanced optimizations, obtaining a more accurate RTL description based on the FPGA device used.

[3]http://www.xilinx.com/support/documentation/dt_ise13-1.htm

[4]http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html

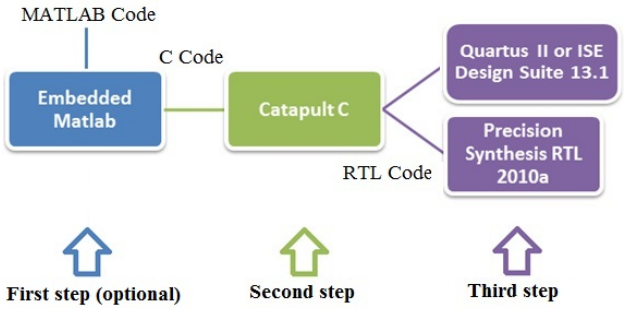

Figure 3. Methodology for high-level design.

The methodology described in Fig. 3 represents the design flow from a very general perspective. The flow may be more specific if the original source code is available in Matlab. For illustrative purposes, Fig. 4 shows a more specific description of the flow in the case that the original code is developed in Matlab.

IV. FPGA implementations

We have developed three FPGA implementations of the ATGP algorithm, two of them following the flow of high-level design from Matlab and another version from C. Next, we explain the different versions implemented in Matlab and C code, describing first those using operations in floating point and later those using operations in fixed point.

A. Implementations using operations in floating point

For the floating point implementations, we have implemented a Matlab version and another version in C. As mentioned above, the main innovation of these versions is the replacement of the inverse function by the Gram-Schmidt orthogonalization method. The goal of these versions is to reduce the computational cost without affecting the output of the algorithm. For illustrative purposes, Fig. 5 provides a general scheme of the modified ATGP algorithm. Fig. 6) describes the Matlab implementation, where lines 20-29 describe the process for Gram-Schmidt orthogonalization that constitutes the main modification performed in the high-level description of the algorithm. The remaining parts of the algorithm do not change and can be efficiently implemented in C code.

B. Implementations using operations in fixed point

In order to develop this version we used the Matlab toolbox for fixed-point operations. The size of the fixed-point variables must be defined in advance. Also we need to define in advance the sum and product formats. For

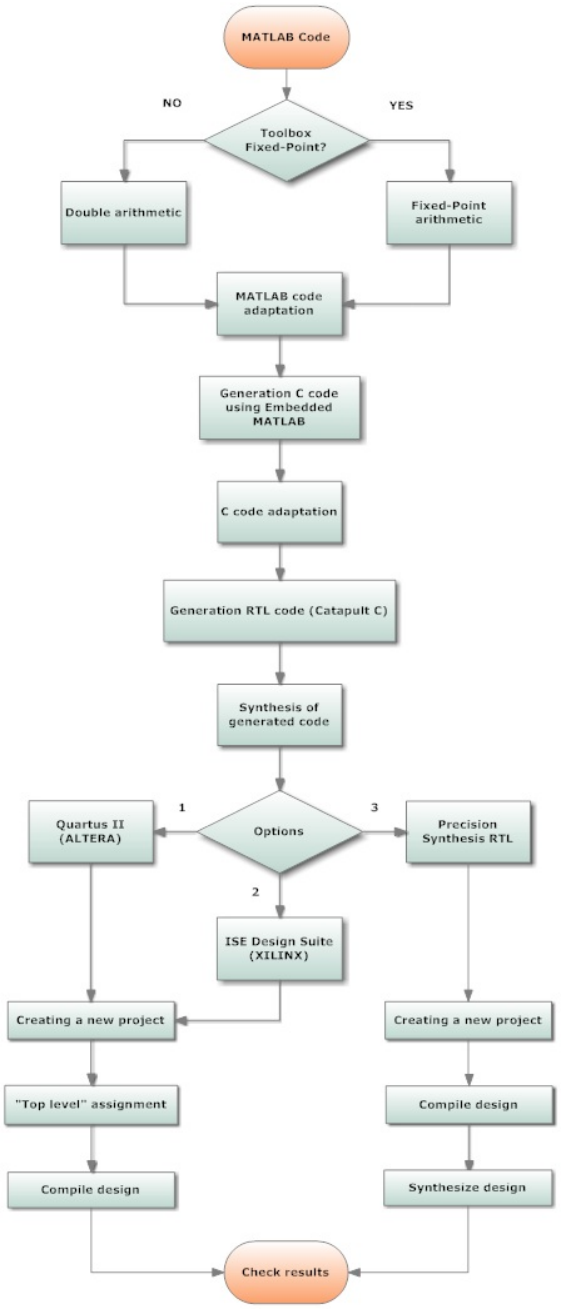

Figure 4. Methodology for high-level design using Matlab code as input.

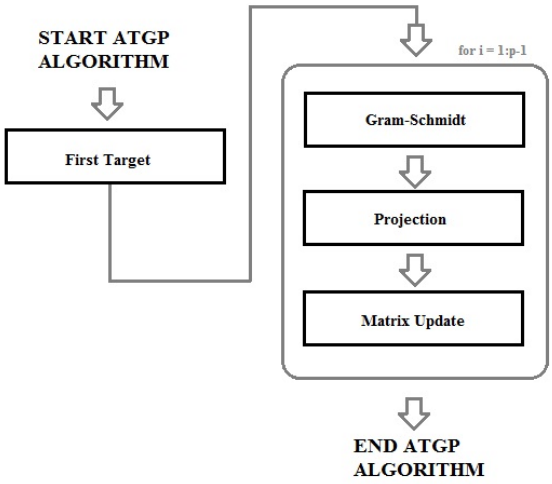

Figure 5. General scheme of the modified ATGP algorithm.

```
function [U,P] = ATGP(HIM,p)
% An automatic target detection and classification algorithm
% ------------------------------------------------------------------------
% Input:   HIM : hyperspectral image cube [nrows x ncols x nchannels]
%          p   : desired number of endmembers to be extracted
%
% Output:  U   : set of extracted targets [nchannels x p]
%          P   : spatial coordinates of the extracted targets (positions
%                rows x cols)

disp(' === Start ATGP run ===')

% The part of code that goes here does not vary substantially

% ATGP algorithm
for i = 1:p-1
    UC = U(:,1:i);
    % Calculate GRAM-SCHMIDT method
    w=ones(nb,1);
    if(i==1)
        f(:,i)=[ones(nb-1,1);-(sum(UC(1:nb-1,1))/UC(nb,1))];
        u(:,1)=UC(:,1);
        c2(1)=(w'*u(:,1))/(u(:,1)'*u(:,1));
    else
        c1 = (u(:,1:i-1)'*UC(:,i))'./(sum(u(:,1:i-1).*u(:,1:i-1)));
        u(:,i) = UC(:,i) - sum(c1(ones(1,nb),:).*u(:,1:i-1),2);
        c2(i) = (w'*u(:,i))/(u(:,i)'*u(:,i));
        f(:,i) = w -  sum(c2(ones(1,nb),:).*u(:,1:i),2);
    end

    % The part of code that goes here does not vary substantially
end
disp(' === Eng ATGP ===');
```

Figure 6. Modification of the ATGP algorithm in Matlab code.

illustrative purposes, Fig. 7 shows the initialization of a variable-Point fixed generic Matlab and initialization of the same variable with support for Embedded Matlab. It should be noted that the use of this toolbox allows for the design of a more optimal hardware configuration, as will be shown in experiments in the following section.

V. Experimental Results

A. Data description

The scene used for experiments in this work was collected by the AVIRIS instrument, which was flown by NASAs Jet Propulsion Laboratory over the well-known AVIRIS Cuprite data set, available online in reflectance units after atmospheric correction. This hyperspectral image (see Fig. 8(a)) was collected in the summer of 1997 and is available online in reflectance units after atmospheric correction. The portion used in experiments corresponds to a 350×350-pixel subset of the sector labeled as f970619t01p02_r02_sc03.a.rfl

```
Pi_ = fi(3.1416);
Pi_ = fi(3.1416,1,64,32'SumMode','KeepLSB','ProductMode','KeepLSB', …
'ProductWordLength',128,'ProductFractionLength',64,'SumWordLength', …
128,'SumFractionLength',64);
The second initialization parameters have the following meanings:
•   3.1416: Is the numerical value that initializes the Fixed-Point.
•   1: Indicates that the variable is unsigned.
•   64: Indicates the total size of the variable bits.
•   32: Indicates the size corresponding to the decimal part.
•   'SumMode','KeepLSB': Mode sum equal to KeepLSB.
•   'ProductMode','KeepLSB': Mode multiplication equal to KeepLSB.
•   'ProductWordLength',128: Product size equal to 128.
•   'ProductFractionLength',64: Size decimal part of the product equal to 64.
•   'SumWordLength', 128: Size sum equal to 128.
•   'SumFractionLength',64: Size of the decimal part of the sum equal to 64.
```

Figure 7. Initialization of a variable-Point Fixed.

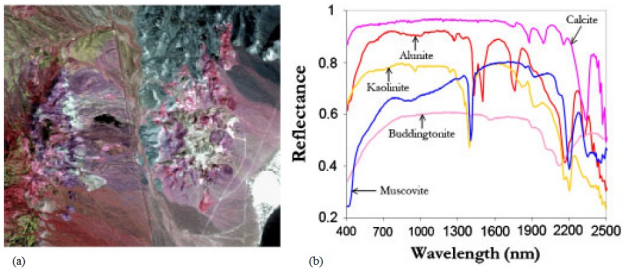

Figure 8. (a) False color composition of the AVIRIS hyperspectral over the Cuprite mining district in Nevada and (b) U.S. Geological Survey mineral spectral signatures used for validation purposes.

in the online data, which comprises 188 spectral bands in the range from 400 to 2500 nm and a total size of around 50 MB. Water absorption and low SNR bands were removed prior to the analysis. The site is well understood mineralogically, and has several exposed minerals of interest, including alunite, buddingtonite, calcite, kaolinite, and muscovite. Reference ground signatures of the above minerals (see Fig. 8(b)), available in the form of a USGS library will be used to assess ATGP algorithm in this work.

B. FPGA hardware

The selected FPGA devices are manufactured by two of the most powerful industries in this field: Xilinx and Altera. The family of Virtex-5[5] FPGAs are the first to own a 65 nm technology. These devices, manufactured in 1.0v, triple-process technology oxide, provide up to 330,000 logic cells, 1200 I/O pins, 48 low-power transistors, and PowerPC 440 processor, and Ethernet MAC blocks PCIe endpoint, depending on the selected model. We have used the 5VLX155FF1760 Xilinx Virtex-5 in our experiments. On the other hand, the family of Stratix III FPGAs arise from the evolution of Stratix II family, incorporating a 65 nm technology implementation capacity DDR3 533 MHz and offer yields up to 1.6 Gbps LVDS. This family of FPGAs combines high performance with the lowest power possible. In our experiments, we have used the EP3SL200F1152C Stratix-III FPGA.

[5]http://www.xilinx.com/support/documentation/

C. Implementation results

The hardware implementation results are determined from the synthesis of Verilog code[6] obtained for the considered Virtex-5 and Stratix-III FPGAs. Both codes makes use of RTL Synthesis Precision Tool. Specifically, we have conducted a comparison between the three modified versions of the ATGP algorithm. The first implementation is based on the use of floating point operations, starting from C code. The second implementation is also based on floating point operations but starting from a high-level design in Matlab. Finally, the third implementation uses fixed-point. The results in terms of hardware utilization on each of the FPGAs used will be presented, describing the number of registers, look-up tables (LUTs) and multiplication blocks (DSPs). On the other hand, we report the value of the maximum frequency that can be achieved by the proposed implementations.

For this purpose, we have used a small portion of the AVIRIS Cuprite image. The considered image portion comprises 36×36 pixels and 188 spectral bands. The number of targets to be detected is $t = 5$. The results in terms of hardware utilization for this experiment are summarized in Tables I, II and III. These results suggest that the use of hardware resources is higher in the third implementation, mainly due to the fact that the fixed-point toolbox of Matlab requires more logic. The best performance in our experiments is obtained by the first implementation, which achieves a maximum frequency of execution of nearly 200 MHz, which is faster than that reported for the other methods. In addition, it is remarkable the good performance achieved by the Stratix III FPGA from Altera, which achieves the best overall performance results across the three tested implementations. Finally, it is worth noting that the first and third implementations are good candidates for conducting a high-level design from Matlab or from C/C++. Both implementations could be satisfactorily ported to Virtex-5 and Stratix III FPGAs, which is mainly due to the optimizations introduced by tools such as Catapult C or Precision RTL Synthesis, obtaining results with high frequencies and low latencies. These results indicate that the development of hardware-based implementations of hyperspectral imaging algorithms starting from high-level implementations (Matlab or C/C++) is a feasible goal that can be achieved through an adequate flow of design. Further experiments with additional hyperspectral scenes and hyperspectral imaging algorithms are highly desired in order to extrapolate these observations to other techniques and analysis scenarios.

VI. Conclusions and Future Research

This paper described several FPGA versions of an automatic target generation process (ATGP) algorithm for hyperspectral image analysis. In our implementations, we

[6]http://www.verilog.com

Table I
RESOURCES USED IN THE VIRTEX-5 FOR THE SYNTHESIS OF THE MODIFIED ATGP ALGORITHM.

Resources	Implementation 1		Implementation 2		Implementation 3	
	Units	Percentage	Units	Percentage	Units	Percentage
LUTs (out of 97280)	16925	17.40%	19938	20.50%	31138	32.01%
CLB Slices (out of 24320)	4232	17.40%	4985	20.50%	7785	32.01%
Latches (out of 97280)	6583	6.77%	12258	12.60%	20391	20.96%
DSP48Es (out of 128)	12	9.38%	9	7.03%	7	5.47%

Table II
RESOURCES USED IN THE STRATIX-III FOR THE SYNTHESIS OF THE MODIFIED ATGP ALGORITHM.

Resources	Implementation 1		Implementation 2		Implementation 3	
	Units	Percentage	Units	Percentage	Units	Percentage
LUTs (out of 159120)	16338	10.27%	21712	13.65%	30773	19.34%
Registers (out of 159120)	6224	3.91%	13160	8.27%	21077	13.25%
# DSP blocks	12	2.08%	7	1.22%	10	1.74%

Table III
RESTRICTIONS ON DIFFERENT MODIFICATIONS OF ATGP ALGORITHM.

Restriction	Implementation 1		Implementation 2		Implementation 3	
	Virtex-5	Stratix-III	Virtex-5	Stratix-III	Virtex-5	Stratix-III
Max.Freq.(MHz)	192.123	195.503	148.192	213.493	185.736	200.924
Min.Period(ns)	5.205	5.115	6.748	4.684	5.384	4.977
Latency(cycles)	517	342	4916945	310855	875	626
Latency(time(ms))	0.00269	0.00175	33.17956	14.56186	0.00471	0.00312

have investigated the impact of including the Gram-Schmidt orthogonalization method for calculating the orthogonal projections calculated by this algorithm instead of using an orthogonal subspace projector that includes a very expensive matrix inverse operation. For this purpose, several versions were obtained through an RTL design flow from high-level Matlab or C code. The use of this flow has allowed us to develop efficient hardware implementations from high-level algorithmic descriptions, which is an important contribution in the hyperspectral imaging community as most of the available algorithms are designed using a high-level perspective and not all algorithms map well into hardware, hence our proposed approach allows us to preliminarily test the suitability of a certain algorithm for hardware implementation. In our experiments we have reported frequencies above 200MHz without compromising target detection accuracy, which represents an innovative contribution in this field. In future work, we will develop real hardware implementations (instead of synthesis results) and experiment with additional algorithms and FPGA platforms to test the validity of our proposed design flow strategies in different analysis scenarios.

ACKNOWLEDGMENT

This work has been supported by the European Communitys Marie Curie Research Training Networks Programme under reference MRTN-CT-2006-035927 (HYPER-I-NET), DR.SIMON project: Dynamic Reconfigurability for Scalability In Multimedia Oriented Networks, TEC2008-06846-C02-02 and Cooperative Research Thematic Network Image processing and multidimensional signal (PRISMA) - TEC2005-24739-E. Funding from the Spanish Ministry of Science and Innovation (HYPERCOMP/EODIX project, reference AYA2008-05965-C04-02) is also gratefully acknowledged.

REFERENCES

[1] A. F. H. Goetz, G. Vane, J. E. Solomon, and B. N. Rock, "Imaging spectrometry for Earth remote sensing," *Science*, vol. 228, pp. 1147–1153, 1985.

[2] R. O. Green, M. L. Eastwood, C. M. Sarture, T. G. Chrien, M. Aronsson, B. J. Chippendale, J. A. Faust, B. E. Pavri, C. J. Chovit, M. Solis *et al.*, "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)," *Remote Sensing of Environment*, vol. 65, no. 3, pp. 227–248, 1998.

[3] C.-I. Chang, *Hyperspectral Imaging: Techniques for Spectral Detection and Classification*. Kluwer Academic/Plenum Publishers: New York, 2003.

[4] A. Plaza and C.-I. Chang, *High Performance Computing in Remote Sensing*. Taylor & Francis: Boca Raton, FL, 2007.

[5] A. Plaza, J. Plaza, A. Paz, and S. Sanchez, "Parallel hyperspectral image and signal processing," *IEEE Signal Processing Magazine*, vol. 28, no. 3, pp. 119–126, 2011.

[6] P. Lysaght, B. Blodget, J. Masona, J. Young, and B. Bridgford, "Enhanced architectures, design methodologies and cad tools for dynamic reconfiguration of xilinx fpgas," *Proceedings of the International Conference on Field Programmable Logic and Applications*, pp. 1–6, 2006.

[7] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Computing Surveys*, vol. 34, pp. 171–210, 2002.

[8] R. Tessier and W. Burleson, "Reconfigurable computing for digital signal processing: a survey," *Journal of VLSI Signal Procesing Systems*, vol. 28, pp. 7–27, 2001.

[9] A. Plaza and C.-I. Chang, "Clusters versus FPGA for parallel processing of hyperspectral imagery," *International Journal of High Performance Computing Applications*, vol. 22, no. 4, pp. 366–385, 2008.

[10] M. Hsueh and C.-I. Chang, "Field programmable gate arrays (FPGA) for pixel purity index using blocks of skewers for endmember extraction in hyperspectral imagery," *International Journal of High Performance Computing Applications*, vol. 22, pp. 408–423, 2008.

[11] C. Gonzalez, J. Resano, D. Mozos, A. Plaza, and D. Valencia, "FPGA implementation of the pixel purity index algorithm for remotely sensed hyperspectral image analysis," *EURASIP Journal on Advances in Signal Processing*, vol. 969806, pp. 1–13, 2010.

[12] H. Ren and C.-I. Chang, "Automatic spectral target recognition in hyperspectral imagery," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 39, pp. 1232–1249, 2003.

[13] A. Paz and A. Plaza, "Clusters versus GPUs for parallel automatic target detection in remotely sensed hyperspectral images," *EURASIP Journal on Advances in Signal Processing*, vol. 915639, pp. 1–18, 2010.