FPGA Implementation of a Hardware Optimized Automatic Target Detection and Classification Algorithm for Hyperspectral Image Analysis

Rubén Macias, Sergio Bernabé, Daniel Báscones, and Carlos González

***Abstract*— In hyperspectral image (HSI) analysis, one of the most important tasks is target detection, requiring the execution of algorithms with high computational complexity. Recently, research efforts have focused on on-board real-time target detection to provide timely responses for swift decisions. Therefore, it is necessary to use a technology that provides the performance needed for real-time target detection, and at the same time meets the satellite payload requirements. Field-programmable gate arrays (FPGAs) have very interesting properties in terms of performance, size, and power consumption, which have become the standard option for on-board processing. In this letter, we present a hardware optimized implementation for FPGAs of the automatic target detection and classification algorithm (ATDCA) using the Gram–Schmidt (GS) method for orthogonalization purposes. The ATDCA-GS algorithm is directly coded using VHDL and verified on a Virtex-7 XC7VX690T FPGA using real hyperspectral data [collected by Hyperspectral Digital Imagery Collection Experiment (HYDICE) sensor and by NASA's Airborne Visible/Infrared Imaging Spectrometer (AVIRIS)] and a synthetic image. Experimental results demonstrate that our hardware version of the ATDCA-GS algorithm outperforms previous implementations (multicore processors, GPUs, and accelerators) in both computation time (obtaining real-time performance) and power consumption, demonstrating the suitability of FPGAs for this purpose.**

***Index Terms*— Automatic target detection and classification algorithm (ATDCA), field-programmable gate arrays (FPGAs), Gram–Schmidt (GS) orthogonalization, hyperspectral imaging, reconfigurable hardware.**

I. Introduction

HYPERSPECTRAL imaging improves upon traditional imaging by capturing hundreds of wavelengths per image pixel. This gives hyperspectral data a high information density per unit of area, which has expanded the domains of many analysis techniques in the remote sensing field. Algorithms for target detection (static, moving, or even changing over time like fires) require fast response times for decision-making at ground stations.

Target detection from satellites with real-time performance has always been a challenging task. On one hand, to achieve high detection accuracy, most state-of-the-art detection algorithms are computationally complex with excessive processing times, which makes them not suitable for application on resource-constrained satellites. On the other hand, on-ground high-performance computing platforms such as multicore processors and GPUs are generally not suitable for satellite applications. In contrast, field-programmable gate arrays (FPGAs) offer excellent ionizing radiation tolerance and a high degree of flexibility. It is undoubtedly a good alternative to adopt in the rough environment of outer space, although computational power is limited.

In the past, several implementations of targets detection algorithms in FPGA architectures have been developed [1]. In [2], we can also find a discussion on the implementation of algorithms for target detection and classification in real-time. Through software optimizations, other implementations for target detection algorithms have also been proposed in [3]. One of the target detection algorithms most widely studied and used in hyperspectral imaging is the automatic target detection and classification algorithm (ATDCA) [4]; however, a full FPGA implementation of this algorithm with real-time performance is not yet available (to the best of our knowledge) to the community [5].

In this letter, an optimized FPGA-based hardware version of ATDCA is developed using the Gram–Schmidt (GS) method to obtain an orthogonal projector [7], which allows us to detect the necessary number of targets in a hyperspectral image (HSI). Using a Xilinx Virtex-7 XC7VX690T FPGA device, the experimental results demonstrate that the proposed architecture obtains real-time performance, outperforming (in terms of computation time and power consumption) previous implementations existing in the literature in multiple processing devices (accelerators, multicore processor, and GPUs). In more detail, the main contributions of this letter are summarized as follows.

1) A hardware optimized version of ATDCA, one of the most popular algorithms for target detection, using the GS method for orthogonalization purposes.
2) An effective and novel pipelined architecture is presented to accelerate our algorithm on FPGA using VHDL. It can be easily synthesized for different numbers of spectral bands, being suitable for onboard real-time processing.
3) The first (to the best of our knowledge) full FPGA implementation of ATDCA with real-time performance.

The organization of the rest of this letter is as follows. Section II briefly presents the description of our hardware optimized version of the ATDCA-GS algorithm. Section III describes the entire system and the core ATDCA-GS for

Manuscript received 24 February 2022; revised 13 June 2022; accepted 29 June 2022. Date of publication 7 July 2022; date of current version 18 July 2022. This work was supported by the Spanish Ministry of Science and Innovation under Grant PID2020-112916GB-I00 and Grant RTI2018-093684-B-I00. *(Corresponding author: Carlos González.)*

The authors are with the Computer Arquitecture and Automation, Complutense University of Madrid, 28040 Madrid, Spain (e-mail: rumacias@ucm.es; sebernab@ucm.es; danibasc@ucm.es; carlosgo@ucm.es).

Digital Object Identifier 10.1109/LGRS.2022.3189109

Algorithm 1 Pseudocode Proposed for Hardware Implementation of ATDCA-GS

1: **INPUTS:** $\mathbf{F} \in \mathbf{R}^n$ and t;
2: $\mathbf{U} = [\mathbf{x}_0 \mid 0 \mid, \cdots, \mid 0]$;
3: $\mathbf{B} = [0 \mid 0 \mid, \cdots, \mid 0]$;
4: $\mathbf{w} = [\mathbf{1}, \ldots, \mathbf{1}]$;
5: $P_{\mathbf{U}}^{\perp} = [\mathbf{1}, \ldots, \mathbf{1}]$;
6: **for** $i = 1$ to $t - 1$ **do**
7: $\quad \mathbf{B}[:, i] = \mathbf{U}[:, i]$;
8: $\quad$ **for** $j = 2$ to i **do**
9: $\quad\quad proj_{\mathbf{B}[:, j-1]}(\mathbf{U}[:, i]) = \frac{\mathbf{U}[:,i]^T \mathbf{B}[:, j-1]}{den[j-1]} \mathbf{B}[:, j-1]$;
10: $\quad\quad \mathbf{B}[:, i] = \mathbf{B}[:, i] - proj_{\mathbf{B}[:, j-1]}(\mathbf{U}[:, i])$;
11: $\quad$ **end for** j
12: $\quad proj_{\mathbf{B}[:,i]}(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{B}[:,i]}{\mathbf{B}[:,i]^T \mathbf{B}[:,i]} \mathbf{B}[:, i]$;
13: $\quad den[i] = \mathbf{B}[:, i]^T \mathbf{B}[:, i]$
14: $\quad P_{\mathbf{U}}^{\perp} = P_{\mathbf{U}}^{\perp} - proj_{\mathbf{B}[:,i]}(\mathbf{w})$;
15: $\quad \mathbf{v} = P_{\mathbf{U}}^{\perp} \mathbf{F}$;
16: $\quad i = \text{argmax}_{\{1,\ldots,r\}} \mathbf{v}[:, i]$;
17: $\quad \mathbf{x}_i \equiv \mathbf{U}[:, i+1] = \mathbf{F}[:, i]$;
18: **end for**
19: **OUTPUT:** $\mathbf{U} = [\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_{t-1}]$;

Virtex-7 FPGA as the target platform. Section IV provides an experimental assessment of both processing performance and target detection accuracy using representative and well-known HSIs of different sizes. Finally, in Section V we can find concluding remarks.

II. Hardware Optimized ATDCA-GS

Our algorithm is a novel version of the ATDCA-GS algorithm [7], which has been optimized for FPGAs. For a number t of targets to be detected, an array of t elements $\mathbf{U}$ is created, the first being set to the pixel vector of maximum length $\mathbf{x}_0$ in the input image $\mathbf{F}$. Subsequent targets will be detected by generating a projector $P_{\mathbf{U}}^{\perp}$ orthogonal to the already found targets. It is then used to project the full image $\mathbf{F}$, and the pixel with maximum projection is added to the target set. The process is repeated until t targets are found.

Orthogonality of $P_{\mathbf{U}}^{\perp}$ is ensured with an orthogonal base $\mathbf{B}$ which is updated on each iteration with the new target in $\mathbf{U}$ using the GS method. Traditionally, a random vector $\mathbf{w}$ is generated on each iteration to update $P_{\mathbf{U}}^{\perp}$. Our implementation fixes it to $[\mathbf{1}, \ldots, \mathbf{1}]^{\mathbf{T}}$, avoiding the generation of random vectors and reducing the computational cost of step 12, while making the algorithm deterministic.

More advantages over [7] include the reuse of $P_{\mathbf{U}}^{\perp}$ over different iterations of the main loop, approximately halving the complexity by eliminating the need of recalculating it on each iteration. The calculation of $\mathbf{B}[:, i]^T \mathbf{B}[:, i]$ is also stored between iterations saving time when updating $\mathbf{B}$.

Looking at hardware implementation specifics, the denominator from step 12 is saved in step 13, and then reused in step 9 in further iterations. Hardware resources from steps 9 to 10 are reused in steps 12–14 since they share the operation structure, saving additional resources. Finally, both the internal loop and the projection $P_{\mathbf{U}}^{\perp} \mathbf{F}$ are performed in a pipelined fashion to increase clock speed.

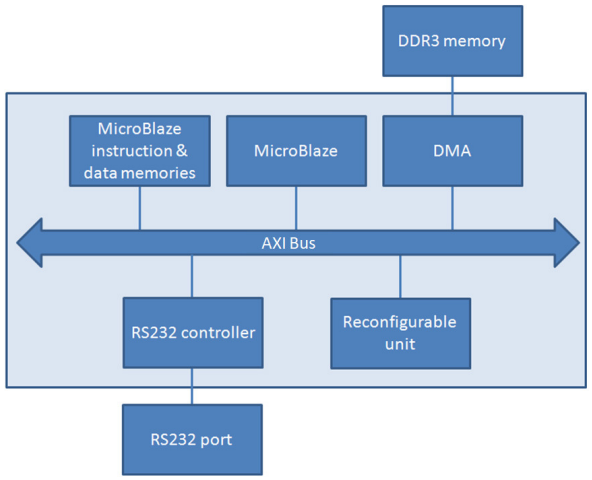

Fig. 1. Diagram of cores and connections of the complete system.

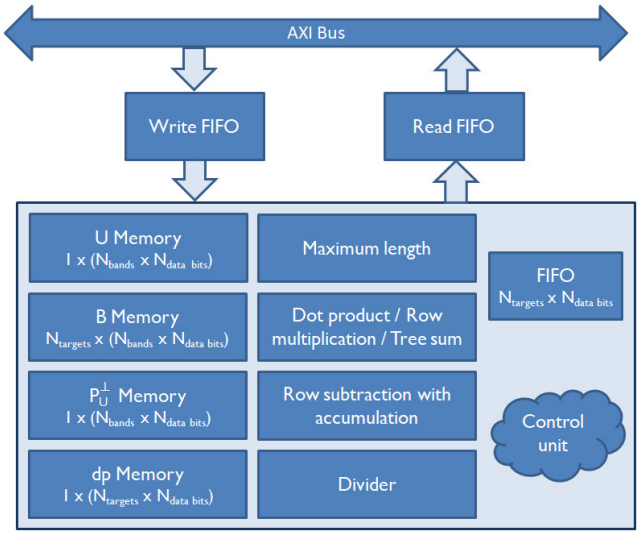

Fig. 2. Modules and communications of the ATDCA-GS core.

III. FPGA Implementation of the ATDCA-GS

In this section, we present the proposed FPGA implementation in detail. Fig. 1 shows the diagram of cores and connections of a generic system used to create an execution platform for any algorithm we place within the reconfigurable unit and also allows us to perform hardware/software codesign. In our case, within the reconfigurable unit we place our hardware optimized version of the ATDCA-GS algorithm. The HSI is located in the DDR3 memory, so we use the MicroBlaze and DMA, along with a prefetch policy, for data input. Finally, the positions within the HSI of the targets detected will be send through an RS232 port using a controller.

Fig. 2 shows the division into modules of the ATCDA-GS core and the I/O communications through the AXI bus. Thus, our algorithm can be integrated into any other system that implements the AXI bus. Bellow, we explain in detail the hardware architecture of each of the modules, and at the end of this section, a step-by-step description of how the complete system carries out target detection from an HSI is provided.

The module used to perform the dot product of two vectors is shown in Fig. 3. Its implementation is based on first use

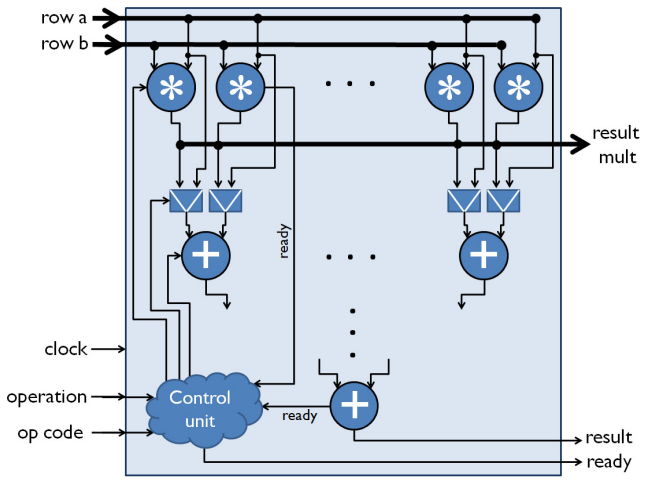

Fig. 3. Hardware architecture used to implement dot product, row multiplication, and tree adder.

an array of multipliers and then a tree of adders. Given two vectors $A = [a_1, a_2, \ldots, a_n]$ and $B = [b_1, b_2, \ldots, b_n]$, we first calculate element-wise products obtaining a new vector $[a_1 \times b_1, \ldots, a_n \times b_n]$. Then, each of these multiplications is added together in a binary adder tree. As an example, $[a_1 \times b_1 + a_2 \times b_2, \ldots, a_{n-1} \times b_{n-1} + a_n \times b_n]$ is obtained at the first level. The binary reduction is repeated until we obtain the dot product. This module is also used to implement the element-wise multiplication of two given vectors $[a_1 \times b_1, \ldots, a_n \times b_n]$ or the sum of the elements of a vector $[a_1 + a_2 + \cdots + a_n]$ by bypassing either the binary adder or element-wise product units. A pipelined architecture enables continuous operation at a rate of one output per clock cycle.

This module is used in step 9 of Algorithm 1 for the dot product calculation of $\mathbf{U}[:, i]^T \mathbf{B}[:, j-1]$, and subsequently for the multiplication of the divisions results by the elements of $\mathbf{B}[:, j-1]$. Since we cannot perform the first of the multiplications before the calculation of the last dot product has started, we use an FIFO to temporarily save the results of the divisions. The module is then used in step 12 to calculate first the dot product $\mathbf{B}[:, i]^T \mathbf{B}[:, i]$, then the sum of the elements of $\mathbf{B}[:, i]$ (since the vector $\mathbf{w}$ is fixed to $[\mathbf{1}, \ldots, \mathbf{1}]^{\mathbf{T}}$), and finally for the multiplication of the division result by the elements of $\mathbf{B}[:, i]$. Steps 2 and 15 also benefit from this module to calculate, respectively, the dot product of a pixel with itself (to get the vector lengths and then the maximum) and the dot product of a pixel with the projection.

Fig. 4 shows the architecture used to perform accumulative vector subtractions element by element on a preloaded initial vector. This module is used in steps 10 and 14 to calculate, respectively, the values of the current orthogonal vector $\mathbf{B}[:, i]$ and the current projection $P_{\mathbf{U}}^{\perp}$.

Fig. 5 shows the *maximum length* module, used to find the maximum of a sequence of values provided. Initially, this module is used to find the pixel with maximum length in the original HSI (the initial target signature) and then (in each iteration) it is used to find the pixel vector with maximum length after applying the orthogonal projection operator. The values with which the pixels are compared are obtained using the dot product module on $\sum_{k=1}^{N} \mathbf{f}_i(k) \times \mathbf{f}_i(k)$ (step 2) and $\sum_{k=1}^{N} P_{\mathbf{U}}^{\perp}(k) \times \mathbf{f}_i(k)$ (step 15). Each new value is then compared with the previous maximum and, if greater, replaces it along with its index (position) within the HSI. These operations are performed in steps 2 and 16.

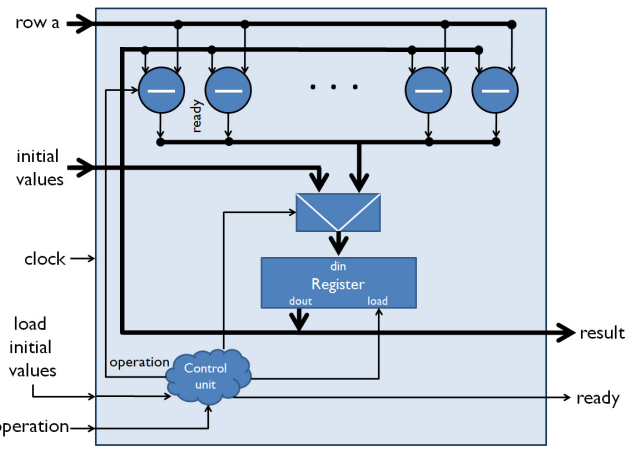

Fig. 4. Hardware architecture for accumulative vector subtractions.

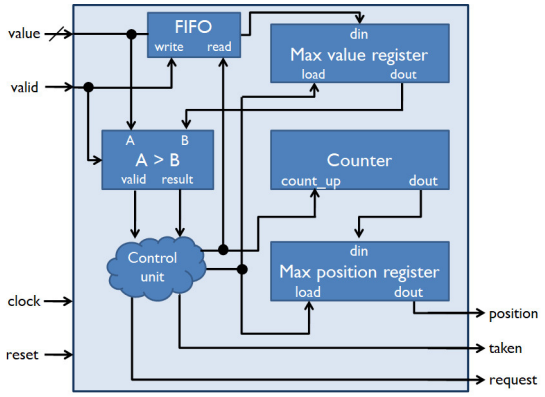

Fig. 5. Hardware architecture for find the maximum length.

Finally, we make a step-by-step description of how the target detection from an HSI is carried out.

1) All pixel data from the *write FIFO* are read, and the pixel with maximum length in the HSI is calculated and selected.
2) The index of the pixel with maximum length is written in the *read FIFO*.
3) Outside of the ATDCA-GS core, Microblaze reads this index and writes the pixel data corresponding to it in the *write FIFO*. The ATDCA-GS core then reads and stores the pixel data in both the **U** and **B** memories. Henceforth, Microblaze uses a prefetching approach to store all the pixel data in the *write FIFO* to be used in 6).
4) Orthogonal vector calculation (inner loop). In a pipelined way, we first calculate the dot products of the target found in the previous iteration with all the preexisting orthogonal vectors. Second, we divide the results by the denominators calculated in the previous iterations of the second inner loop, and then, the results are multiplied by the corresponding orthogonal vector. Finally, by means of accumulative subtractions, the current orthogonal vector is updated.

TABLE I
HSI DATASET DIMENSIONS AND SIZE CHARACTERISTICS

	HYDICE	**AVIRIS Cuprite**	**AVIRIS** WTC	**Synthetic**
No. of pixels	64 × 64	350 × 350	614 × 512	750 × 650
No. of bands	169	188	224	224
Total size	5.28 MB	46 MB	140 MB	437 MB

5) Projection calculation. In a pipelined way, we first calculate the dot product of the orthogonal vector computed in the previous step by itself. Second, we calculate the sum of the elements of the orthogonal vector and divide the result of the dot product by the sum of the elements, which is then multiplied by the current orthogonal vector. Finally, the accumulated projection is updated by subtracting the previous result.
6) All the pixels are read from the *write FIFO*, and their projections and projection lengths are calculated using the dot product module. The new target is the pixel with maximum projection length. If there are still targets to be detected, we return to 2).

IV. EXPERIMENTAL RESULTS

A. HSI Datasets

The HSI dataset used in these experiments is composed of one HSI obtained by Hyperspectral Digital Imagery Collection Experiment (HYDICE) sensor, two HSIs obtained by the NASA's Airborne Visible/Infrared Imaging Spectrometer (AVIRIS)m and one synthetic HSI. Table I summarizes their dimensions and size characteristics, and these HSIs are described below.

1) *HYDICE:* A subset of the well-known forest radiance dataset, with 15 panels of five types of targets distributed on each row with three different sizes (the scene is extensively described in [8]).
2) *AVIRIS Cuprite:* Obtained in Nevada over the Cuprite mining district, the site is well understood mineralogically and has several exposed minerals of interest including alunite, buddingtonite, calcite, kaolinite, and muscovite. It corresponds to the sector labeled as f970619t01p02_r02_sc03.a.rfl in the online data.
3) *AVIRIS WTC:* Obtained over the World Trade Center (WTC), New York, just five days after the terrorist attacks of September 11, 2001. In the area where the towers collapsed, thermal hot points are labeled from "A" to "H" in [5].
4) *Synthetic:* We have also considered a bigger synthetic dataset to evaluate the scalability of our implementation. The data have been constructed using a set of 30 signatures from the United States Geological Survey (USGS) library, and the procedure is described in [6] to simulate natural spatial patterns.

B. Target Detection Accuracy Evaluation

In this section, we evaluate the similarity of the targets detected by our implementation and the well-known targets for the AVIRIS WTC and AVIRIS Cuprite scenes. In the first scene, the targets correspond to the pixels labeled from "A" to "H". In the second scene, with the known positions of the minerals. To do that, we use a widely used metric for this purpose, the spectral angular distance (SAD) [8].

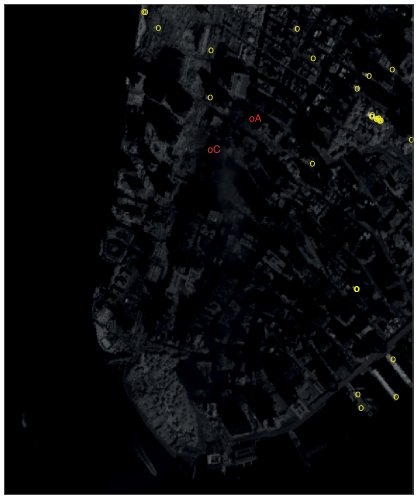

Fig. 6. Positions of the targets detected in the AVIRIS WTC scene.

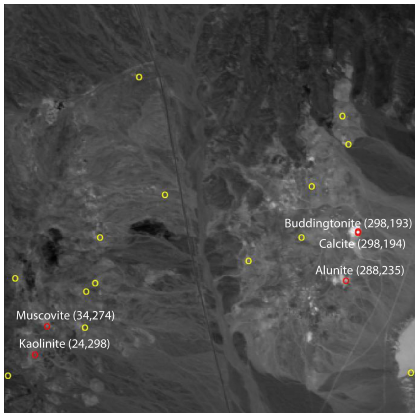

Fig. 7. Positions of the targets detected in the AVIRIS Cuprite scene.

TABLE II
SPECTRAL SIMILARITY BETWEEN KNOWN GROUND TARGETS AND TARGET PIXELS DETECTED BY OUR ATDCA-GS IMPLEMENTATION

(a)

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**
0.0^o	27.2^o	0.0^o	15.6^o	27.8^o	4.0^o	2.7^o	24.3^o

(b)

Alunite	**Buddingtonite**	**Calcite**	**Kaolinite**	**Muscovite**
5.48^o	4.08^o	5.87^o	11.14^o	5.68^o

According to the literature [7], the number of targets to be detected in the AVIRIS WTC scene was set to 30 and for the AVIRIS Cuprite scene was set to 19. Figs. 6 and 7 show the positions where the targets have been detected in the scenes. Table II reports the SAD values (in degrees) between the most similar target pixels detected and the known targets. For the AVIRIS WTC scene, we perfectly detect the targets labeled as "A" and "C," and there are more difficulties in detecting very small targets. For the AVIRIS Cuprite scene, SAD values are low, so again the extracted targets were spectrally similar to the known ground-truth targets. Finally, we emphasize that these

TABLE III

FPGA RESOURCE UTILIZATION OF THE ATDCA-GS ALGORITHM TO PROCESS 169, 188, OR 224 BANDS, AND UP TO 32 TARGETS

	BRAMs	DSPs	slice LUTs	LUT-FF pairs	slice registers	Frequency
ATDCA-GS (169 bands)	391 (26%)	1286 (36%)	169104 (39%)	14320 (6%)	22950 (2%)	119 MHz
ATDCA-GS (188 bands)	436 (29%)	1318 (36%)	188115 (43%)	14326 (6%)	22974 (2%)	118 MHz
ATDCA-GS (224 bands)	517 (35%)	1390 (38%)	208114 (48%)	14344 (6%)	22938 (2%)	116 MHz

TABLE IV

PROCESSING TIMES FOR DIFFERENT IMPLEMENTATIONS OF THE ATDCA-GS ALGORITHM EXISTING IN THE LITERATURE IN MULTIPLE PROCESSING DEVICES (MULTICORE PROCESSOR, GPUS, AND ACCELERATORS) AND OUR FPGA IMPLEMENTATION

	HYDICE (7 targets)	**AVIRIS Cuprite (19 targets)**	**AVIRIS WTC (30 targets)**	**Synthetic (30 targets)**
NVIDIA Tesla C1060 GPU [7]	-	0.0560 sec	0.1663 sec	-
NVIDIA GeForce GTX 580 GPU [7]	-	0.0456 sec	0.1554 sec	-
Intel Xeon CPU [9]	0.0077 sec	0.0592 sec	-	0.3056 sec
NVIDIA GeForce GTX 1080 GPU [9]	0.0033 sec	0.0339 sec	-	0.1536 sec
Intel Xeon Phi Accelerator [9]	0.0170 sec	0.0738 sec	-	0.5855 sec
Odroid [9]	0.0300 sec	0.9642 sec	-	-
Xilinx Virtex-7 XC7VX690T FPGA [this letter]	0.0003 sec	0.0233 sec	0.0944 sec	0.1463 sec

similarity results are consistent with those in the literature with an equivalent software implementation [7].

C. Performance Evaluation

Table III shows FPGA resource utilization for our proposed hardware implementation of the ATDCA-GS algorithm for different numbers of bands and up to 32 targets. We use the VC709 board containing a Xilinx Virtex-7 XC7VX690T FPGA [with a total of 866 400 slice registers, 433 200 slice look-up tables (LUTs), 134 381 LUT-FF pairs, and hetereogeneous resources such as 3600 DSPs and 1470 distributed BRAMs], an RS232 port, two DDR3 SDRAM DIMM slots, and some additional components not used.

Table IV reports the processing times for different implementations of the ATDCA-GS algorithm existing in the literature in multiple processing devices (multicore processor, GPUs, and accelerators) and for our hardware implementation of the ATDCA-GS algorithm on the considered FPGA architecture. Our FPGA implementation obtains better results in all the scenes (speedups from 1.05 to 100) and offer similar performance than the GTX 1080 GPU but consuming 4.17 W (in average for the considered scenes) instead of the 148.09 W [9] (also in average) for the mentioned GPU.

It is important to mention that our ATDCA-GS core scales well when varying the number of targets, bands, or pixels, but the complete system does not. The VC709 board has limited bandwidth between the memory and the FPGA. Therefore, as the image size increases, greater I/O penalty is produced. If I/O speed was not a bottleneck, we would have the same performance as in the HYDICE image which was placed entirely inside the FPGA.

Finally, we emphasize that our reported FPGA processing meets real-time processing performance. For instance, the cross-track line scan time in HYDICE and AVIRIS, push-broom instruments, is quite fast (8.3 ms to collect 512 full pixel vectors). This introduces the need to process the scenes in less than 0.066 s (HYDICE), 5.09 s (AVIRIS WTC), 1.986 s (AVIRIS Cuprite), and 7.903 s (Synthetic).

V. CONCLUSION

In this letter, we have developed an optimized hardware version for FPGA implementation of the ATDCA-GS, a widely used algorithm to detect targets in remotely sensed HSIs. The ATDCA-GS algorithm is directly coded using VHDL and verified on a Virtex-7 XC7VX690T FPGA using real HSIs and a large synthetic image. Our experimental results demonstrate that the proposed hardware implementation can successfully meet strict real-time target detection requirements, outperforming (in terms of computation time and power consumption) previous implementations existing in the literature in multiple processing devices (accelerators, multicore processors, and GPUs).

REFERENCES

[1] A. Plaza and C.-I. Chang, *High Performance Computing in Remote Sensing*. Boca Raton, FL, USA: CRC Press, 2007.

[2] Q. Du and R. Nekovei, "Fast real-time onboard processing of hyperspectral imagery for detection and classification," *J. Real-Time Image Process.*, vol. 4, no. 3, pp. 273–286, Aug. 2009.

[3] J. M. Molero, E. M. Garzón, I. García, and A. Plaza, "Analysis and optimizations of global and local versions of the RX algorithm for anomaly detection in hyperspectral data," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 6, no. 2, pp. 801–814, Apr. 2013.

[4] H. Ren and C.-I. Chang, "Automatic spectral target recognition in hyperspectral imagery," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 39, no. 4, pp. 1232–1249, Oct. 2003.

[5] C. Gonzalez, S. Bernabe, D. Mozos, and A. Plaza, "FPGA implementation of an algorithm for automatically detecting targets in remotely sensed hyperspectral images," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 9, no. 9, pp. 4334–4343, Sep. 2016.

[6] G. S. Miller, "The definition and rendering of terrain maps," *ACM SIGGRAPH Comput. Graph.*, vol. 20, no. 4, pp. 39–48, Aug. 1986.

[7] S. Bernabe, S. Lopez, A. Plaza, and R. Sarmiento, "GPU implementation of an automatic target detection and classification algorithm for hyperspectral image analysis," *IEEE Geosci. Remote Sens. Lett.*, vol. 10, no. 2, pp. 221–225, Mar. 2013.

[8] C.-I. Chang, *Hyperspectral Imaging: Techniques for Spectral Detection and Classification*. New York, NY, USA: Kluwer, 2003.

[9] S. Bernabe, C. Garcia, F. D. Igual, G. Botella, M. Prieto-Matias, and A. Plaza, "Portability study of an OpenCL algorithm for automatic target detection in hyperspectral images," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 11, pp. 9499–9511, Nov. 2019.