FPGA-based Hyperspectral Covariance Coprocessor for Size, Weight, and Power Constrained Platforms

David A. Kusinsky
MIT Lincoln Laboratory
Lexington, MA
Email: kusinsky@ll.mit.edu

Miriam E. Leeser
Northeastern University
Boston, MA
Email: mel@coe.neu.edu

***Abstract*—Size, weight, and power (SWaP) are important factors in the design of any remote sensing platform. These remote sensing platforms, such as Unmanned Air Vehicles (UAVs) and microsatellites, are becoming increasingly small. This creates a need for remote sensing and image processing hardware that consumes less area, weight, and power, while delivering processing performance. It is also advantageous to utilize the same hardware for multiple platform tasks. The purpose of this research is to design and characterize an FPGA-based hardware coprocessor that parallelizes the calculation of covariance, a time-consuming step common in hyperspectral image processing. Our design is compared to a CPU-based implementation and shown to have an overall SWaP advantage. We evaluate our coprocessor using a metric that is useful in the consideration of future SWaP-constrained remote sensing platforms: floating point operations per Watt-kg (FLOPs/W-kg). Additional hardware capacity exists in our design to implement other remote sensing platform tasks.**

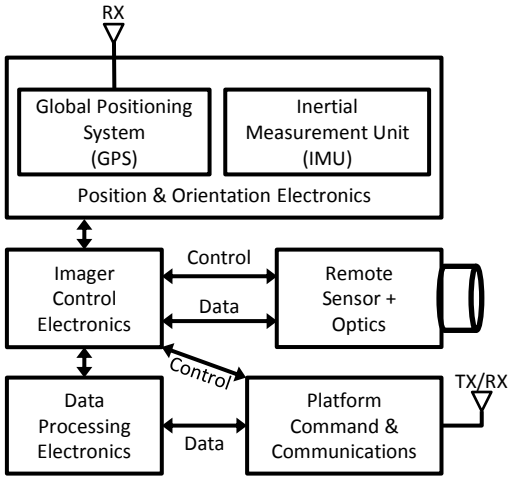

Fig. 1. Example Remote Sensing Platform

I. Introduction

Remote sensing can be defined as the overhead observation of the earth. This observation is performed by sensors that reside on board remote sensing platforms. Airborne and spaceborne remote sensing platforms are often constrained by size, weight, and power. This limits the amount of processing that can be accomplished by the platform, pushing these tasks to ground-based processing systems.

An example remote sensing platform is shown in Figure 1. This example platform contains a single remote sensor, a hyperspectral imager; in practice a platform may have several sensors. In addition, a remote sensing platform may have up to four types of electronics to perform its tasks: imager control, data processing, position and orientation, and command/communications.

From this description, it is clear that improving the size, weight, or power of any of these distinct blocks is advantageous to the overall platform. Additionally, if hardware tasks can be consolidated into a fewer number of hardware modules, the overall platform can benefit. Our research is focused on the tasks of the data processing electronics, with the eventual goal of implementing the imager control tasks as well.

This work is sponsored by the Department of the Air Force under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government. Approved for public release; distribution is unlimited.

One type of sensor that can be employed in remote sensing is a hyperspectral imager (HSI). HSI sensors collect a series of two-dimensional images. Each image collected corresponds to one wavelength band in the electromagnetic spectrum [1]. Depending on the sensor used, the number of spectral bands imaged can be over two hundred. This generates a large amount of image data that, if not compressed or processed, can exceed a communication system's maximum data rate [2]. Onboard processing of HSI data, within the SWaP constraints of the platform, can help alleviate this problem and enable real-time processing performance.

One common operation performed during the processing of HSI data is the computation of covariance. In this paper we present the design of an FPGA-based covariance coprocessor, suitable for SWaP-constrained HSI platforms. The coprocessor was targeted for the Xilinx ML605 evaluation board, which contains a Xilinx Virtex-6 FPGA and DDR3 memory.

This work provides several contributions. To our knowledge, this is the first FPGA-based HSI data processing platform to perform the covariance calculation on HSI data having a large number of spectral bands, 50 in our case. Second, this is the first time that a user-generated processing algorithm has been successfully integrated inside the OpenCPI framework on the ML605. Third, we demonstrate that HSI covariance calculation in FPGAs is an attractive option when compared to CPU-based approaches. Last, we show that our coprocessor

978-1-4799-1365-7/13/$31.00 ©2013 IEEE

has substantial unused resources that can be used to perform additional HSI data processing operations in future research. These unused resources can also be used to perform additional platform tasks, such as directly interfacing with HSI cameras and aiding in platform communication and navigation.

The rest of the paper is organized as follows. First, a background on hyperspectral imaging, remote sensing platforms, and covariance is provided. Section III discusses the architecture of the coprocessor and the details of the covariance computation, which is performed using single-precision arithmetic. We then cover the implementation of the coprocessor through the use of the Xilinx ISE design suite and OpenCPI middleware [3]. The performance of our coprocessor is compared to a CPU-based processing platform. This comparison includes processing throughput, power dissipation, and energy expended during processing. For more details see [4].

II. Background

A. Hyperspectral Imaging

HSI is typically set apart from simple three-color and multispectral imagery in that the spectral bands being imaged are very closely-spaced, very high in number (several hundred), or both. Because processed HSI sensor data typically contains one spectral dimension and two spatial dimensions, this data is typically referred to as a hyperspectral image cube. Because of the large number of closely spaced bands, HSI is particularly useful in remote sensing applications that will exploit this high spectral resolution and wide spectral range. Shaw and Burke define three major applications for HSI: anomaly detection, target recognition, and background characterization [1].

One example of an HSI platform is NASA's Airborne Visible/Infrared Imaging Spectrometer (AVIRIS). AVIRIS is an airborne scientific instrument that images the ground in the visible, near-infrared, and short-wave infrared. AVIRIS produces image cubes having 224 spectral bands and 512 x 614 spatial pixels [5]. AVIRIS data is commonly used in remote sensing research [6], [7], [8], has a large number of bands, and much of the data collected by AVIRIS is available to the academic community. For these reasons, a 512 x 614 x N_b image cube of AVIRIS data was used in our research, where $N_b = 50$ bands. This image cube is shown in Figure 2.

B. HSI Platform Size, Weight, and Power

Minimizing size, weight, and power (SWaP) is a common design goal for airborne, spaceborne, and even ground-based HSI platforms. Research has been done on band selection techniques [9] and HSI data dimensionality reduction [2]. The former has the potential to reduce HSI sensor size and weight, and the latter can lead to reductions in data storage/downlink and processing requirements.

However, some of these techniques push additional processing requirements closer to the HSI sensor itself, wherever it may be operating. With the shrinking size of highly capable remote sensing platforms (UAVs, microsatellites, etc.) and the equally-shrinking power and sensor weight capacity, the processing performance per Watt of power must improve to compensate, or the capabilities of the remote sensing platform may be diminished.

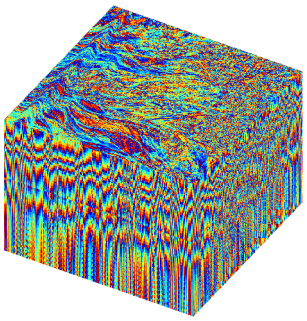

Fig. 2. 50-Band Hyperspectral Image Cube

C. Covariance

The calculation of covariance is a necessary step in many HSI data processing and target detection algorithms, such as matched filtering and anomaly detection [6]. In some applications, several covariance calculations may be performed per frame of imagery and can be a limit to performance [10].

Because we are using pixel data from an image cube to compute covariance, we are calculating the *sample covariance matrix*, defined as:

$$\Sigma_{j,k} = \frac{1}{N_{pix} - 1} \sum_{i=1}^{N_{pix}} (x_{ij} - \hat{x}_j)(x_{ik} - \hat{x}_k), \qquad (1)$$

Here, i represents the pixel number and N_{pix} the number of pixels. j and k are the band numbers. $\hat{x}_j$ and $\hat{x}_k$ represent the means of bands j and k, respectively, over all image pixels.

D. Related Work

There is related work that presents the computation of covariance in several types of hardware, including FPGAs. Martelli et al. [11] recognized that covariance computation can be a bottleneck for some systems. They implemented a covariance computation on an FPGA to compute a 6x6 covariance matrix. This was used in a linear SVM classifier for a pedestrian detection application. Their research utilized fixed-point data representations and six features during the covariance computation. The computation described here is significantly larger.

In [12], four DSPs were chosen to perform the covariance calculations for 80 channels of data in a multi-beam echo sounding application. FPGAs were also used in their processing hardware to perform digital down-conversion and other tasks, but were not used for covariance computations.

There has also been research into understanding the number of hyperspectral bands needed for a given detection application to achieve good detection algorithm performance. Costa [9] demonstrated that, depending on the detection algorithm and band selection technique, the number of bands used for detection could be reduced as much at 50% with only a 12% drop in the probability of detection. This research is relevant to the covariance coprocessor presented here, as the number of AVIRIS bands used (50) is a subset of the available bands.

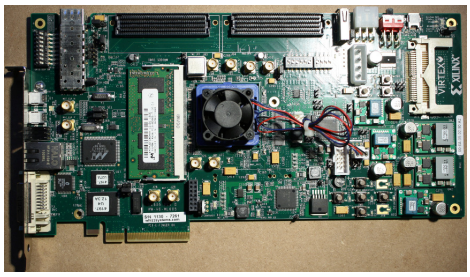

Fig. 3. Xilinx ML605 Evaluation Board

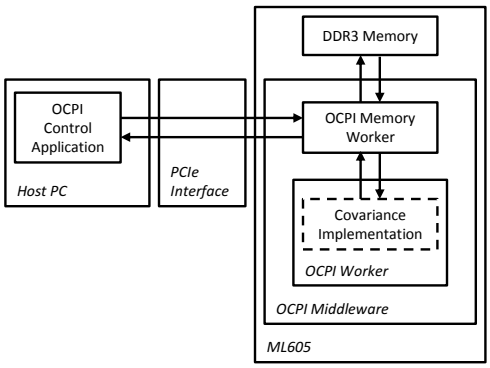

Fig. 4. Design High Level Block Diagram

III. Design

A. Covariance Algorithm

The computation of sample covariance requires many multiply accumulate operations (MACs). These operations map well to FPGAs. Recall the sample covariance calculation in Eq. (1). Every ith pixel intensity in spectral band j, x_{ij}, must be multiplied against its intensity in band k, x_{ik}. This occurs after the respective means of bands j and k across all pixels are subtracted from these two intensity values. The resulting product is then accumulated over all the pixels in the image cube.

In the covariance calculation for our AVIRIS image cube, there are two areas where parallelism can be exploited. First, some or all of the band-to-band intensity products mentioned above for a single pixel can be computed in parallel. Second, these band-to-band products can be computed for several pixels in parallel. Both types of parallelism were exploited in our design.

B. Hardware Platform

The hardware platform used in this research is the Xilinx ML605 evaluation board, shown in Figure 3, which contains a Xilinx Virtex-6 LX240T FPGA. This FPGA features 768 DSP48E1 slices, which are used by the adders and multipliers in our design, as well as 14,976 Kb of internal block RAM used for data storage.

Additional hardware available on the ML605 used in this research is 32 MB of BPI flash memory, 512 MB of DDR3 memory, and a PCI Express connector located on the board's edge. Also of interest for future work is the ML605's support for two FPGA Mezzanine Cards (FMCs), which can be used for external hardware interfacing. The ML605 is well-suited for this research because of its highly capable FPGA, onboard DDR3 memory for image storage, and PCI Express interface.

1) Hardware Interfaces: The PCI Express interface is used as the means to transfer data and control signals between the coprocessor and a host PC, as described later in this section. To facilitate testing of our covariance implementation, the open source framework OpenCPI is utilized. OpenCPI [3] is an open-source middleware solution that is geared towards heterogeneous processing applications. OpenCPI (OCPI) enables ML605 communications with a host PC over PCI Express, as well as reading and writing to the DDR3 memory on the ML605.

2) OCPI ML605 Middleware: The covariance implementation in this research interfaces with and executes within the OCPI middleware in the ML605's FPGA. A custom Verilog wrapper was created to perform this interfacing. Figure 4 shows a block diagram of the high-level design and relevant interfaces. The OCPI middleware, OCPI worker, and OCPI control application were also modified as needed to communicate with our implementation.

3) OCPI Software: The OCPI control application on the host PC is used for several tasks: writing the image cube to the DDR3 memory on the ML605, starting/stopping the coprocessor, monitoring status, and retrieving the covariance results. The host PC and control application were used as a control and diagnostics tool and are not actively needed by the coprocessor during processing.

It is important to note that the image cube transferred to the ML605 memory from the host PC is already mean-subtracted. This section shows that significant FPGA resources are available after implementation of our coprocessor. These resources can be used to implement the subtraction of means in the future.

The OCPI ML605 middleware enables the testing of our hardware implementation, at the cost of additional FPGA resource utilization. Direct interfacing with an HSI sensor can remove the need for OCPI and its associated overhead, freeing additional resources for other tasks.

C. Hardware Covariance Implementation

Figure 5 shows the datapath of the design. N input pixels are passed through a FIFO constructed from FPGA block memory. The output of the FIFO is used by parallel multipliers, 50 per pixel, during which the next N pixels are enqueued into the FIFO. This buffering keeps the OCPI DDR3 memory interface occupied with read requests, which maximizes its performance.

The multipliers consist of single-precision Xilinx multiplier IP cores. These multipliers are provided new data every clock cycle to compute one row to sum into a 50x50 accumulator matrix. This is performed in parallel for N pixels. Therefore, there are N accumulator matrices and $50N$ multipliers. The first clock cycle provides data for the $b_{1,1}, b_{1,2}, \cdots, b_{1,50}$ band-band products for each pixel. The second clock cycle provides

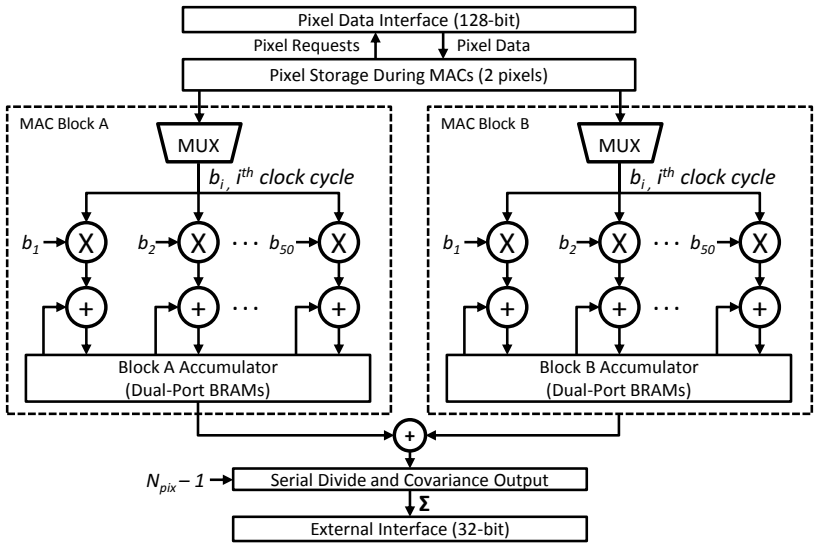

Fig. 5. Detailed Datapath of Covariance Implementation

the $b_{2,1}, b_{2,2}, \cdots, b_{2,50}$ products. This proceeds for 50 clock cycles to compute all 50 rows of the accumulator matrix.

In our research, $N = 2$ was chosen: computing the 50x50 accumulator matrix for two pixels in parallel. Figure 5 shows the datapath with $N = 2$. One parallel pixel is processed by MAC block A, and the other by MAC block B. More pixels could be processed in parallel consuming more FPGA resources, if available.

FPGA block RAMs are used to implement storage of the accumulator matrices, and Xilinx single-precision adder cores are used for the additions. As with the parallel multipliers, $50N$ adders are used. The block RAMs are configured as simple dual-port RAMs. The read address of the block RAMs are set to follow the current row of the band-to-band products being output by the multipliers. The write address is set to follow the row corresponding to the sum being output by the adders. Therefore, the write address lags behind the read address by the adder latency, and read-write collisions are avoided.

Once all pixels in the image have been processed through the datapath, the N accumulator matrices must be added together and divided by $N_{pix} - 1$ to obtain the final covariance matrix. Because the number of pixels is large ($N_{pix} = 512 * 614 = 314,368$), a serial sum of $N_b^2 = 2500$ accumulator matrix values takes a small amount of time compared to the MACs. Therefore, this sum is performed serially and only requires a single adder. The results of this serial sum are provided directly to a divider that divides by $N_{pix} - 1$ to produce one entry of the 50x50 covariance matrix each clock. This data is provided as an output of our datapath, along with an enqueue signal that is used by the OCPI worker shown in Figure 4 to schedule writes to the ML605 DDR3 memory.

D. FPGA Resource Utilization

The FPGA resource utilization of our design is reported by the Xilinx design tools. Two cases were considered. The first is the FPGA resource utilization for just the covariance calculation. The second is the resource utilization for the full coprocessor, after its integration with the OCPI FPGA middleware. Table I provides the major FPGA resources for both cases.

TABLE I. FPGA RESOURCE UTILIZATION

FPGA Resource	# Available	Covariance Calculation	Full Coprocessor
Slice Registers	301,440	51,082 (16%)	83,559 (27%)
Slice LUTs	150,720	38,940 (25%)	80,829 (53%)
RAMB36E1s	416	0 (0%)	39 (9%)
RAMB18E1s	832	308 (37%)	311 (37%)
DSP48E1s	768	502 (65%)	502 (65%)

The utilization of the FPGA registers and lookup tables (LUTs) for the covariance calculation is very low (25% or less). It can be seen that OCPI adds 11% and 28% overhead, respectively, to these resources to obtain a testable design. Some of these resources could be freed by interfacing directly to an HSI sensor, thereby allowing the removal of the OCPI middleware. Regardless, the full coprocessor design utilizes less than 55% of the total FPGA resources in each category, with the exception of the DSP48E1s (65% utilization).

IV. RESULTS

A. Design Validation

Before measuring performance, our design was validated for proper operation through a combination of hardware simulations and analysis of coprocessor output. For hardware simulations of our VHDL code, the ModelSim simulation environment was used to validate proper operation of our FPGA covariance implementation.

Our hardware implementation was tested with the same pixel data used in our simulation testbench. The testbench results were found to exactly match those of our hardware coprocessor. Both results were found to match a covariance calculation performed using the software package MATLAB.

TABLE II. ML605 AND FPGA POWER SUMMARY

Supply Currents	Idle	Running
FPGA VCCAUX	0.80A	0.81A
FPGA VCC1V5	1.0A	0.95A
FPGA VCCINT	4.03A	6.0A
FPGA VCC2V5	0.04A	0.04A
Total ML605 Input	1.86A	2.13A
FPGA Power (W)	7.63	**9.55**
Total ML605 Power (W)	22.6	**25.9**
Processing Energy (J)	N/A	**2.1**

TABLE III. SBC POWER AND ENERGY VS. ML605 COPROCESSOR

	SBC Idle	SBC Four Threads	SBC Eight Threads	ML605 Coprocessor
Voltage (V)	12	12	12	12.14
Current (A)	1.07	4.6	5.2	2.13
Power (W)	12.8	**55.2**	**62.4**	25.9
Processing Energy (J)	N/A	**3.8**	**4.3**	2.1

B. Coprocessor Performance

Performance measurements were made to determine the runtime of our coprocessor for the 512 x 614 x 50 image cube. Registers in the hardware design were used to record the number of clock cycles that the covariance module takes to perform the computation for each loop of execution.

On average, the coprocessor took 10,286,763 clock cycles at 125MHz to process a full image cube, or 82.3 ms. This corresponds to almost 12.2 image cubes per second processing throughput. The image cube requires approximately 1.57 GFLOP to compute, and therefore our coprocessor currently achieves 19.1 GFLOPS during the covariance computation.

Power measurements of the coprocessor were performed to determine the power dissipation and total energy used to perform the covariance computation. This was accomplished by using the built-in capability of the ML605 evaluation board to monitor supply currents on major voltage rails. This includes the main input supply for the entire ML605, as well as the individual FPGA supply voltages.

This allows a determination of the FPGA's contribution to the ML605 total power, and the increase in FPGA power during active processing. A summary of all voltage rails measured and power consumption of each is provided in Table II. Total FPGA and ML605 power is also provided in this table. Note that the ML605 power increased by only 3.3W (15%) between idle and active processing states.

C. Comparison Platform Performance

A Single Board Computer (SBC) was used as a performance and power comparison platform. This platform consists of a Congatec conga-BM67 CPU board mounted to an ACTIS Computer KCAC-0320 carrier board that provides access to external peripherals and power. The BM67 contains a 2.1 GHz Intel Corei7 2710QE processor, 512MB of DDR3 memory, and runs CentOS Linux from flash memory. This reference SBC was chosen specifically to model processing hardware that is typical of SWaP-constrained remote sensing platforms.

Performance measurements were made to determine the runtime of the SBC on the 512x614x50 image cube. A multithreaded C version of the covariance algorithm was written to perform calculations in parallel, much like the HW coprocessor. Each thread processed an equal share of the total number of pixels. Because the SBC platform's quad-core processor supports eight logical cores, both four- and eight-thread versions of the code were executed. The code was compiled using gcc 4.7.2 with the *-lpthread -lrt -m32 -march=corei7 -mfpmath=sse -Ofast -flto* compiler flags. As with our FPGA-based platform described in Section III, the input pixel data is mean-subtracted per (1).

On average, the SBC took 79ms to process a single image cube using 4 threads, and 69ms using 8 threads. This corresponds to about 12.7 and 14.5 image cubes per second processing throughput, respectively. These measurements place the comparison platform at 22.8 GFLOPS during the covariance calculation.

Power measurements were performed with a digital multimeter wired in series with the +12V DC power input to the SCB to measure average current. Measurements were taken during an idle state and during the execution of the four- and eight-thread code. The power and energy measurements of the SBC are shown in bold in Table III. The power and energy measurements of the FPGA-based coprocessor are included in the rightmost column for comparison.

From these measurements, it can be seen that the SBC platform requires less power and energy when idle than the ML605. This is due to frequency scaling of the Corei7 CPU on the SBC, which lowers the clock frequency during idle states, reducing power consumption. However, the SBC platform dissipates more than twice as much power as the ML605 during active processing. The energy required for processing on the SBC is about twice that of the ML605 coprocessor.

D. SWaP Implications

The performance, power, and FLOPS measurements are further utilized to investigate the attractiveness of both the covariance coprocessor and SBC from the SWaP perspective. One such metric to consider is the number of FLOPS per Watt of power consumed (FLOPS/W). This is a popular metric in the field, but it omits the weight portion of the system design, which is also of high importance for current and future airborne and spaceflight processing systems. Therefore, we also consider another metric, FLOPS/(W·kg) during the evaluation of these types of systems.

The ML605 and SBC were weighed to support this SWaP analysis, and the results are provided in Table IV. For this analysis, the average power seen during processing was used. These results indicate that the coprocessor performance in FLOPS/W is approximately twice that of the SBC, and around 2.75 times that of the SBC in FLOPS/(W·kg).

V. CONCLUSION

Our research investigated and implemented a parallelized sample covariance calculation, targeted to an FPGA-based platform. This platform, the Xilinx ML605 was chosen because of its PCI Express interface, onboard DDR3 memory, and

TABLE IV. ML605 COPROCESSOR AND SBC SWAP COMPARISON

	ML605 Coprocessor	SBC
FLOPS	19.1G	22.8G
Power (W)	25.9	62.4
Mass (kg)	0.336	0.476
FLOPS/W	**737M**	**365M**
FLOPS/(W·kg)	**2.2G**	**0.8G**

potential for lower size, weight, and power. An open-source middleware framework, OpenCPI, was utilized to extend the covariance calculation into a complete coprocessor.

We described the major aspects of the datapath and control of the covariance implementation, and the validation of proper hardware operation through simulation.

Our coprocessor was found to provide around 12 sample covariance calculations per second on 512 x 614 x 50 AVIRIS hyperspectral image cubes. Performance was evaluated and found to be promising for use on SWaP-constrained HSI platforms, providing a greater than 2X improvement in power and energy dissipation over a CPU-based platform.

Contributions of our research include being the first coprocessor to perform FPGA-based covariance calculations on HSI data having this many spectral bands. Additionally, this is the first time that a VHDL/Verilog user application has been successfully integrated inside the OpenCPI ML605 middleware. We have shown that FPGA-based processing platforms are an attractive option for the processing of HSI data, and are competitive from a size, weight, and power perspective. More details can be found in [4].

Future work includes adding mean-subtraction to our hardware implementation, interfacing the coprocessor directly to a hyperspectral camera, and the addition of HSI processing steps after the sample covariance calculation. This additional processing will make the FPGA even more attractive for remote sensing platforms since a larger amount of the image processing can be done in a small form factor with high energy efficiency.

ACKNOWLEDGMENT

The authors would like to thank Xilinx for providing the ML605 board used in this research, and Bluespec, Inc. for providing their BSV compiler. We would also like to thank Shep Siegel of Atomic Rules for supporting our work with OpenCPI, and Marc Burke of Lincoln Laboratory for providing the SBC used in our research.

REFERENCES

[1] G. A. Shaw and H. K. Burke, "Spectral imaging for remote sensing," *Lincoln Laboratory Journal*, vol. 14, no. 1, pp. 3–28, 2003.

[2] S. Cook and J. Harsanyi, "Onboard processor for compressing HSI data," in *Proceedings of the 31st Applied Imagery Pattern Recognition Workshop (AIPR02)*, 2002.

[3] Mercury Federal Systems, Inc. OpenCPI - open component portability infrastructure. http://opencpi.org. [Online; accessed 21-Feb-2013].

[4] D. A. Kusinsky, "Fpga-based hyperspectral covariance coprocessor for size, weight, and power constrained platforms," Master's thesis, Dept. Elect. and Comput. Eng., Northeastern Univ., Boston, MA, 2013.

[5] C. I. o. T. Jet Propulsion Laboratory. AVIRIS - airborne visible / infrared imaging spectrometer. http://aviris.jpl.nasa.gov/index.html. [Online; accessed 21-Feb-2013].

[6] D. Manolakis, D. Marden, and G. A. Shaw, "Hyperspectral image processing for automatic target detection applications," *Lincoln Laboratory Journal*, vol. 14, no. 1, pp. 79–116, 2003.

[7] A. Paz and A. Plaza, "Clusters versus GPUs for parallel target and anomaly detection in hyperspectral images," *EURASIP Journal on Advances in Signal Processing*, vol. 2010.

[8] S. Bernabé, S. López, A. Plaza, and R. Sarmiento, "GPU implementation of an automatic target detection and classification algorithm for hyperspectral image analysis," *IEEE Geoscience and Remote Sensing Lett.*, vol. 10, no. 2, Mar. 2013.

[9] J. S. Costa, "Band selection techniques for hyperspectral chemical agent detection," Master's thesis, Dept. Elect. and Comput. Eng., Northeastern Univ., Boston, MA, 2011.

[10] M. L. Pieper, D. Manolakis, R. Lockwood, T. Cooley, P. Armstrong, and J. Jacobson, "Hyperspectral detection and discrimination using the ACE algorithm," in *Imaging Spectrometry XVI*, 2011.

[11] S. Martelli, D. Tosato, M. Cristani, and V. Murino, "Fast FPGA-based architecture for pedestrian detection based on covariance matrices," in *2011 18th IEEE International Conference on Image Processing*, 2011, pp. 389–392.

[12] B. Yao, H. Li, T. Zhou, B. Chen, and H. Yu, "Real-time implementation of multiple sub-array beam-space MUSIC based on FPGA and DSP array," in *Fifth IEEE International Symposium on Embedded Computing*, 2008, pp. 186–191.