Received 29 September 2017; revised 27 March 2018; accepted 21 June 2018.
Date of publication 12 July 2018; date of current version 3 March 2021.

Digital Object Identifier 10.1109/TETC.2018.2854412

A 3.3 Gbps CCSDS 123.0-B-1 Multispectral & Hyperspectral Image Compression Hardware Accelerator on a Space-Grade SRAM FPGA

ANTONIS TSIGKANOS, (Member, IEEE), NEKTARIOS KRANITIS, (Member, IEEE), GEORGE THEODOROU, (Member, IEEE), AND ANTONIS PASCHALIS, (Member, IEEE)

The authors are with the Digital Systems & Computer Architecture Laboratory (DSCAL), Department of Informatics & Telecommunications, National and Kapodistrian University of Athens, Panepistimiopolis, Athens 15784, Greece

CORRESPONDING AUTHOR: A. TSIGKANOS (antts@di.uoa.gr)

ABSTRACT The explosive growth of data volume from next generation high-resolution and high-speed hyperspectral remote sensing systems will compete with the limited on-board storage resources and bandwidth available for the transmission of data to ground stations making hyperspectral image compression a mission critical and challenging on-board payload data processing task. The Consultative Committee for Space Data Systems (CCSDS) has issued recommended standard CCSDS-123.0-B-1 for lossless multispectral and hyperspectral image compression. In this paper, a very high data-rate performance hardware accelerator is presented implementing the CCSDS-123.0-B-1 algorithm as an IP core targeting a space-grade FPGA. For the first time, the introduced architecture based on the principles of C-slow retiming, exploits the inherent task-level parallelism of the algorithm under BIP ordering and implements a reconfigurable fine-grained pipeline in critical feedback loops, achieving high throughput performance. The CCSDS-123.0-B-1 IP core achieves beyond the current state-of-the-art data-rate performance with a maximum throughput of 213 MSamples/s (3.3 Gbps @ 16-bits) using 11 percent of LUTs and 27 percent of BRAMs of the Virtex-5QV FPGA resources for a typical hyperspectral image, leveraging the full throughput of a single SpaceFibre lane. To the best of our knowledge, it is the fastest implementation of CCSDS-123.0-B-1 targeting a space-grade FPGA to date.

INDEX TERMS Field programmable gate array (FPGA), hardware accelerator, hyperspectral imaging, image compression, on-board payload data processing, space data systems

I. INTRODUCTION

Multispectral and hyperspectral imaging is recognized as a key enabling technology for several remote sensing applications such as precision agriculture, surveillance, security, military intelligence, environmental monitoring, mineralogy, etc. Multispectral and hyperspectral sensors capture the spectral fingerprint of an object, thus provide a unique spectral signature enabling accurate object detection and classification. The spectral resolution, from a few tens to several hundreds of bands, results in multispectral and hyperspectral images, respectively. The huge volume of data from high-resolution and high-speed multispectral and hyperspectral imagers along with the limited spacecraft storage resources and downlink bandwidth, make image compression very challenging and a mission critical on-board payload data processing task [1]. Moreover, next generation high-speed airborne and spaceborne imagers are expected to have a 10 cm ground resolution, resulting in an explosive growth in data volume and overall instrument data rate in the GigaPixels/s range.

Image compression algorithms are roughly classified into lossless and lossy. Lossless compression leverages image data redundancy to guarantee perfect reconstruction of the original image data without incurring any distortion at the cost of a rather limited compression ratio. Lossy compression allows small amounts of distortion between the reconstructed and the original image, as a tradeoff to achieve higher compression ratios. The amount of distortion tolerated, depends on the remote sensing application. Lossy compression should allow effective application of post compression image analysis (e.g., anomaly detection, classification), always guaranteed by lossless compression.

Either lossless or lossy, the image compression algorithms used on-board must be designed with respect to certain performance-complexity tradeoffs inherent in space data systems design. The Consultative Committee for Space Data Systems (CCSDS), an organization founded by the major space agencies of the world, has developed image compression algorithms specifically for on-board use addressing memory and computational resources challenges. The CCSDS recommended standards for data and image compression [2]–[4] provide an excellent trade-off between compression effectiveness and hardware complexity. In 2012, CCSDS 123.0-B-1 was issued as the recommended standard for Lossless Multispectral and Hyperspectral image compression. The recommended standard [4], [5] is based on the NASA Fast Lossless (FL) [6] algorithm, an excellent predictive technique which uses an adaptive filtering method, achieving a combination of low complexity and compression effectiveness, being able to provide state-of-the-art coding performance for a large collection of remote sensing sensors [7].

Apart from the constraints on compression algorithm design, space data systems design imposes strict requirements on the on-board data processing hardware platform, which must be able to provide a) radiation hardness, b) high data-rate performance for Gigabit-rate applications and c) dynamic adaptability. The current state-of-the-art SRAM-based FPGA technology offers radiation hardening by design (RHBD), high density and dynamic partial reconfiguration for in-flight adaptability and Time-Space Partitioning (TSP) of on-board data processing tasks. An excellent example of such technology is the RHBD Xilinx Virtex-5QV FPGA which provides exceptional hardness to Single-Event-Upset (SEU), total immunity ($> 125\text{MeV.cm}^2/\text{mg}$) to Single-Event Latchup (SEL), data path protection from Single-Event Transients (SET) and extremely high tolerance (1000 Krad) to Total Ionizing Dose (TID) [8], [9]. Such FPGA technology offers unique advantages over both one-time programmable FPGAs and ASICs and can be considered as an excellent platform for implementation of on-board payload data processing due to its ability to support upgrades after launch, greatly enhancing mission profile and extending valuable mission life time. An adaptable instrument based on an adaptable payload data processing platform could be adjusted to unforeseen events such as changes or extensions in the scientific objectives, resulting in a superior scientific data yield and a reduced risk of a total instrument loss. Moreover, dynamic adaptability offers the possibility to timeshare resources in a TSP manner, when dedicated functions are not necessary concurrently, resulting in significant savings in mass, power, hardware resources and design complexity.

The CCSDS 123.0-B-1 algorithm (or the NASA FL algorithm), has been recently implemented in hardware, targeting space-qualified FPGAs. Existing implementations of CCSDS 123.0-B-1 [10]–[12] or NASA FL [13] target space grade FPGAs. In our preliminary work [12], the current state-of-the-art in data-rate performance of 120 MSamples/sec was achieved.

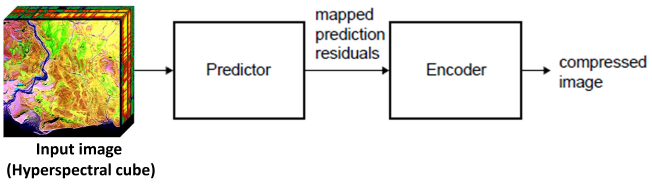

FIGURE 1. CCSDS 123.0-B-1 block diagram.

In this paper, we propose a novel, high data-rate performance hardware accelerator, implementing the CCSDS 123.0-B-1 algorithm as an IP core targeting space-grade FPGAs. For the first time, the introduced architecture based on the principles of C-slow retiming [14], exploits the inherent task-level parallelism of the CCSDS 123.0-B-1 algorithm under BIP ordering and implements a reconfigurable fine-grained pipeline in critical feedback loops, achieving high throughput performance. Moreover, it is a single FPGA solution without external DDR memory requirements that results in significant savings in SWaP-C (Size, Weight, Power and Cost). The CCSDS 123.0-B-1 IP core sets the new state-of-the-art data-rate performance with a maximum data-rate at 213 MSamples/s (3.3 Gbps @ 16-bits per sample) when targeting the Virtex-5QV FPGA using 11 percent of LUTs and 27 percent of BRAMs of the Virtex-5QV FPGA resources for a typical hyperspectral image configuration. To the best of our knowledge, it is the fastest implementation of CCSDS 123.0-B-1 targeting a space-grade reconfigurable SRAM FPGA to date.

The rest of this paper is organized as follows: After a brief overview of the CCSDS 123.0-B-1 standard in Section II and a brief discussion of the main considerations in hardware implementations in Section III, the architecture of the CCSDS 123.0-B-1 hardware accelerator is presented in Section IV. The CCSDS 123.0-B-1 IP core verification and validation frameworks are presented in Section V. Implementation results including FPGA resource usage statistics, operating frequency and data-rate performance along with comparisons with existing state-of-the-art FPGA and CPU/GPU implementations are provided in Section VI before concluding the paper in Section VII.

II. CCSDS 123.0-B-1 ALGORITHM OVERVIEW

The CCSDS 123.0-B-1 Recommended Standard [4], [5] is a formalization of the NASA Fast Lossless (FL) compressor [6], an adaptive predictive technique for lossless compression of multispectral and hyperspectral imagery. The FL compressor achieves a combination of low complexity and compression effectiveness, far exceeding the best results from the literature [6], [7]. The CCSDS 123.0-B-1 is a prediction based algorithm, structured in two functional parts, a predictor and an encoder, illustrated in Figure 1. The predictor estimates the predicted sample value based on the values of nearby samples in a small three-dimensional neighborhood. The prediction residual (i.e., the difference between the predicted and actual sample values) is then mapped to an unsigned integer that can be represented using the same

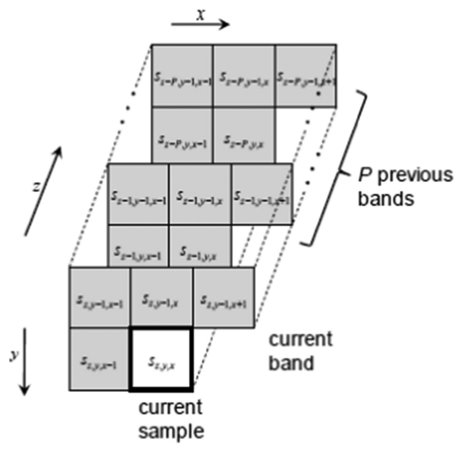

FIGURE 2. **Three dimensional prediction neighborhood.**

number of bits as the input data sample. These mapped prediction residuals make up the predictor output. The predictor adaptively adjusts prediction weights for each spectral band using adaptive linear prediction. The encoder losslessly encodes the mapped prediction residuals using the sample adaptive encoder of NASA FL or as an alternative option, the block-adaptive encoder as specified in the CCSDS 121.0-B-2 standard for data compression intended for monoband image lossless compression [2]. Although block-adaptive encoding is more computationally intensive than sample-adaptive encoding (it requires the evaluation of multiple coding options) and is not as efficient in terms of compression performance as sample-adaptive encoding, it is included to the recommended standard in order to leverage existing implementations of CCSDS 121.0-B-2. The final compressed stream consists of a header that encodes image and compression parameters followed by a body of codewords, produced by the entropy coder.

The predicted sample value $\hat{s}_{z,y,x}$, and the mapped prediction residual $\delta_{z,y,x}$ depend on the values of nearby samples in the current and *P* preceding spectral bands, where *P* is a user-specified parameter (ranging from 0 to 15) which has a high impact on computational complexity. The typical neighborhood of samples used for prediction is illustrated in Figure 2.

The CCSDS 123.0-B-1 algorithm is illustrated in the flowchart of Figure 3 and described in detail below.

First, within each spectral band, the predictor computes a *local sum* $\sigma_{z,y,x}$, which is a weighted sum of the neighboring samples of $s_{z,y,x}$. The user may choose to perform prediction using *neighbor-oriented* or *column-oriented* local sums. When neighbor-oriented local sums are used, the local sum is equal to the sum of four neighboring sample values in the spectral band. When column oriented local sums are used, the local sum is equal to four times the neighboring sample value in the previous row. Then, the local sums are used to calculate *local difference* values. In each spectral band, the *central local difference* $d_{z,y,x}$ is equal to the difference between the local sum $\sigma_{z,y,x}$ and four times the sample value $s_{z,y,x}$.

The three *directional local differences* $d^N_{z,y,x}$, $d^W_{z,y,x}$, $d^{NW}_{z,y,x}$, are each equal to the difference between the local sum $\sigma_{z,y,x}$

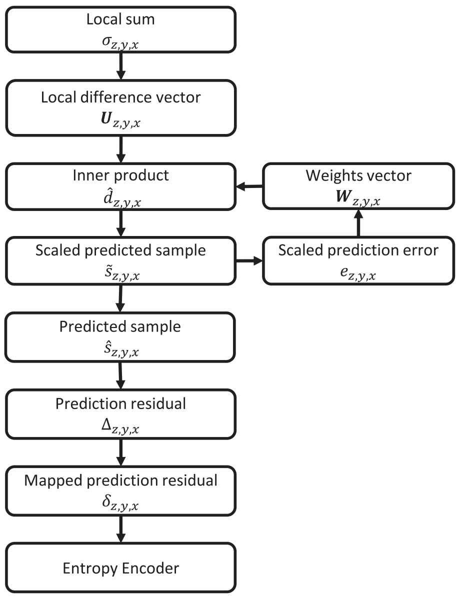
Local sum $\sigma_{z,y,x}$
Local difference vector $\boldsymbol{U}_{z,y,x}$
Inner product $\hat{d}_{z,y,x}$
Weights vector $\boldsymbol{W}_{z,y,x}$
Scaled predicted sample $\tilde{s}_{z,y,x}$
Scaled prediction error $e_{z,y,x}$
Predicted sample $\hat{s}_{z,y,x}$
Prediction residual $\Delta_{z,y,x}$
Mapped prediction residual $\delta_{z,y,x}$
Entropy Encoder

FIGURE 3. **CCSDS 123.0-B-1 algorithm flowchart block diagram.**

and four times a sample value labeled as "N" (North), "W" (West), or "NW" (North West). The user may choose to perform prediction for an image in full or reduced prediction mode. Under reduced mode, prediction depends on a weighted sum of the central local differences computed in *P* preceding bands; the directional local differences are not used, and thus need not be calculated, under reduced mode. Under full prediction mode, prediction depends on a weighted sum of the central local differences computed in *P* preceding bands and the three directional local differences computed in the current band. The use of reduced mode in combination with column oriented local sums tends to yield smaller compressed image data volumes for raw (uncalibrated) input images from push-broom imagers that exhibit significant along-track streaking artifacts. The use of full mode in combination with neighbor-oriented local sums tends to yield smaller compressed image data volumes for whiskbroom imagers, frame imagers, and calibrated imagery.

The next step of the algorithm involves the calculation of the local difference vector $\boldsymbol{U}_{z,y,x}$ whose elements are defined for full or reduced prediction mode according to equations (22) and (23) of [4], respectively. Then, the predicted central local difference $\hat{d}_{z,y,x}$ is calculated as the inner product (eq. (30) of [4]) of the weight vector $\boldsymbol{W}_{z,y,x}$ (eq. (25) in [4]) and the local difference vector $\boldsymbol{U}_{z,y,x}$. The scaled predicted sample $\tilde{s}_{z,y,x}$ is calculated by adjusting the preliminary estimate (local sum $\sigma_{z,y,x}$) by the predicted local difference $\hat{d}_{z,y,x}$ according to eq. (29) of [4]. The final predicted sample value $\hat{s}_{z,y,x}$ is effectively half the scaled predicted sample value $\tilde{s}_{z,y,x}$. Following the calculation of each predicted sample value $\hat{s}_{z,y,x}$, the scaled *prediction error* $e_{z,y,x}$ and a weight update *scaling exponent* $\rho_{z,y,x}$ (eq. (33) in [4]) are used to adaptively update the weight vector $\boldsymbol{W}_{z,y,x}$ using the sign algorithm (also known as the sign-error algorithm) which is similar to the Least Mean Square (LSM) algorithm. It should be noted that a separate weight vector is maintained for each band. To summarize, the principle of the prediction algorithm is to adaptively adjust prediction weights to predict the amount ($\hat{d}_{z,y,x}$) by which the sample value $s_{y,z,x}$ differs from the preliminary estimate $\sigma_{z,y,x}$.

After calculation of the predicted sample $\hat{s}_{y,z,x}$, the prediction residuals $\Delta_{y,z,x}$ are calculated as the difference between the predicted $\hat{s}_{y,z,x}$, and actual $s_{y,z,x}$ sample values and mapped to unsigned integer values $\delta_{y,z,x}$ according to eq. (35) of [4].

The sample-adaptive entropy encoder maps the prediction residuals using length-limited Golomb-Power-Of-2 (GPO2) codes, adaptively selected based on statistics that are updated after each sample is encoded. Since separate statistics are maintained for each spectral band, the compressed size is independent of the sample ordering. As an alternative option, the encoder can use the block-adaptive entropy coding approach recommended in [4]. In this case, depending on the encoding order, the mapped prediction residuals in a block may be from the same or different spectral bands, and thus the compressed image size depends on the encoding order.

III. MAIN CONSIDERATIONS FOR HARDWARE IMPLEMENTATIONS

The CCSDS 123.0-B-1 algorithm was designed specifically for on-board use addressing memory and computational resources challenges. The algorithm uses only integer arithmetic and all mathematical operations can be implemented with fixed shifters, barrel shifters, multipliers, and adders. No divisions are necessary. Although the Recommended Standard facilitates relatively low complexity hardware implementations on FPGAs or ASICs, a few remaining constraints in terms of storage requirements and throughput limitations caused by data dependencies have to be taken into account when developing a hardware implementation [5].

A. LOCAL SUM AND LOCAL DIFFERENCES CALCULATION

The predictor requires only four neighboring samples from the same band to calculate the local sum and local difference. In order to avoid any latencies related to fetching the necessary neighboring samples, they should be buffered in on-chip or external storages resources (e.g., embedded RAM or FIFOs, external DRAM).

B. PREDICTED CENTRAL LOCAL DIFFERENCE CALCULATION

The inner product $\mathbf{W}_{z,y,x} \cdot \mathbf{U}_{z,y,x}$ required for the calculation of the predicted central local difference $\hat{d}_{z,y,x}$ is of major importance in terms of hardware cost and data-rate performance. It not only affects the number of arithmetic operations (multiplications and additions) that have to be performed, but more importantly, it directly affects the volume of data that has to be available for its calculation. The number of elements in the weight update vector $\mathbf{W}_{z,y,x}$ and local difference vector $\mathbf{U}_{z,y,x}$ increases with the number of preceding spectral bands used for prediction P, a user-specified parameter. According to [13] which provides a thorough analysis of the influence of different parameters on compression performance, $P > 3$ does not yield a significant improvement in the compression ratio. Hence, $P = 3$ only is used throughout this paper.

Furthermore, the order in which image samples are processed by the predictor directly affects the amount of storage requirements necessary for the local difference and weight vectors, as well as, the data dependencies that arise which affect pipelining and therefore data-rate performance.

For example, under a BSQ processing order, it is necessary to store the local difference vectors for samples in a given band so that they can be accessed for prediction in the next band. Under both full and reduced modes, prediction in spectral band z makes use of central local differences from the preceding $P_z^* = \min\{z, P\}$ spectral bands. Moreover, under full prediction mode, prediction in spectral band z additionally makes use of three directional local differences computed in the current spectral band z. If the number of local difference values used for prediction at each sample in band z is denoted as C_z (equal to $P_z^* + 3$ for the full prediction mode and P_z^* for the reduced mode), BSQ ordering requires storing a total of $N_x \times N_y \times C_z$ local difference values. However, this limitation can be avoided if the compressor calculates all the elements of the local difference vector for the prediction of each sample [10]. In terms of data dependencies, it is not possible to schedule the prediction of the next sample $s_z(t+1)$ in BSQ order in parallel with the weight update operation of the current sample $s_z(t)$ which directly limits the achievable data-rate performance.

In contrast, under BIP ordering it is only necessary to store a local difference vector with C_z elements, since after prediction of a sample $s_z(t)$ only a single element of the local difference vector needs to be updated for the prediction of $s_{z+1}(t)$, the next sample in BIP order. Of course this implies that N_z weight vectors are stored; one for each band. In terms of data dependencies, it is not necessary to complete prediction of a sample $s_z(t)$ before starting prediction of the next sample $s_{z+1}(t)$ in BIP order, thus making possible to schedule the prediction of sample $s_{z+1}(t)$ in parallel with the weight update operation of sample $s_z(t)$. This is a clear advantage of BIP ordering which makes it a key feature to achieve very high data-rate performance.

C. ENTROPY CODING

Under the sample-adaptive coding, an accumulator (used to maintain a running sum of mapped prediction residuals in the spectral band) and a counter, estimate the mean mapped prediction residual value that selects the code parameter k. Counter and accumulator values are each periodically halved, so that more recent sample values have more impact on the estimated mean value. Under BSQ ordering encoding requires: one previously processed mapped prediction residual, one accumulator and one counter. Under BIL ordering, the same elements and resources are required for each spectral band, which is N_z mapped residuals, accumulators and counter values. Under BIP ordering N_z accumulator values and a single counter value is required. In contrast to the

TABLE 1. **CCSDS 123.0-B-1 IP core features summary.**

Algorithm features	CCSDS 123.0-B-1 IP Core
Pixel dynamic range (D)	$2 \leq D \leq 16$
Image Dimensions	$0 < N_X, N_Y < 2^{16}$ $3 < N_Z < 2^{16}$
Encoding order	Band-Interleaved-by-Pixel
Number of Prediction bands (P)	3
Prediction mode	Full/Reduced
Local sum calculation	Neighbor/Column oriented
Entropy encoder	Sample/Block adaptive *

**Block-adaptive encoding is supported using an external CCSDS 121.0-B-2 IP Core [15].*

block-adaptive optional coding, the length limit makes hardware implementation simpler and reduces the cost of encoding occasional outlier samples. It should be noted that when sample-adaptive coding is used, compression effectiveness does not depend on sample encoding order, thus hardware implementations need not be concerned with compression effectiveness considerations in selecting the sample encoding order. Moreover, the sample-adaptive coding approach does not require the evaluation of multiple coding options.

The block-adaptive optional coding formalized in CCSDS 121.0-B-2 [2] requires the evaluation of the different encoding options for a complete block of J samples. These evaluations require accumulator and comparison operations. The amount of required storage and hardware complexity does not depend on the compression order, but does depend on the number of samples in a block and the number of encoding options to be evaluated. However, the user can leverage existing implementations (e.g., the IP core presented in [15], used in ESA PROBA-3 mission).

IV. CCSDS 123.0-B-1 IP CORE ARCHITECTURE

The CCSDS 123.0-B-1 IP core was developed in compliance with the European Cooperation for Space Standardization standards (ECSS-Q-ST-60-02C) using VHDL. It can be configured at run-time using memory-mapped configuration registers and at compile-time using generic constants that determine the boundaries of run-time configuration. It supports the following features (see Table 1): i) pixel depth up to 16 bits, ii) image dimensions up to 2^{16} pixels for each dimension $(N_x \times N_y \times N_z)$ iii) BIP encoding order, iv) fixed number of prediction bands (P $= 3$ since a higher number of prediction bands does not yield significant improvement in compression effectiveness [16]), v) full and reduced prediction modes, vi) neighbor and column oriented local sum calculation, vii) sample-adaptive encoding.

A. TOP LEVEL ARCHITECTURE

The top level architecture of the CCSDS 123.0-B-1 IP core (depicted in Figure 4) consists mainly of three units: the Predictor and the Encoder comprising the compression engine

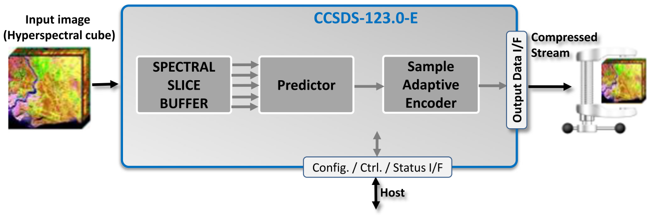

FIGURE 4. **Top level architecture.**

along with an on-chip Spectral Slice Buffer that can accommodate a spectral slice of typical hyperspectral images.

In this paper, we consider a single FPGA chip solution without external DDR memory requirement that results in significant savings in SWaP-C (Size, Weight, Power and Cost). Moreover, it does not require a technology dependent DDR memory controller. The IP core provides simple, FIFO-based streaming interface for streaming data I/O (D-bit for current pixel data and 64-bit for the encoded output). The IP core configuration registers are memory-mapped with a generic interface providing flexibility in integration. If the CCSDS 123.0-B-1 IP core is integrated in an FPGA SoC, the configuration registers can be accessed by standard SoC interfaces (e.g. AMBA slave APB, AXI4 Lite) using an trivial bridge. At system level, the configuration registers can be accessed by standard serial link interfaces such as SpaceWire-RMAP or SPI.

B. SPECTRAL SLICE BUFFER

The compressor engine, assumes a data interface consisting of the 2D-neighbourhood samples; i.e. the current, W, N, NE and NW samples and an end-of-stream strobe asserted with the last sample input. This interface is provided by a preceding component, the Spectral Slice Buffer which consists of a 4-deep cascade of FIFOs, accompanied by 4 stages of elastic buffers which pack neighbourhood samples emitted by intermediate stages of the FIFO cascade. In between the stages, counters may inhibit reading or writing to account for the initial and final spatial regions in the spectral cube. An abstract representation of the neighbourhood dataflow is shown bottom right in Figure 5: N_z samples after the current sample is the W neighbour, $(N_x - 2)^* N_z$ samples later the NE

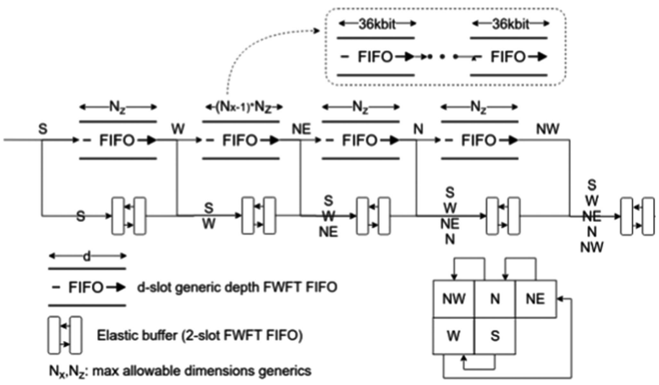
36kbit
36kbit
FIFO
FIFO
N_z
$(N_x-1)^*N_z$
S
W
NE
N
NW
FIFO
S
W
NE
N
NW
d
FIFO d-slot generic depth FWFT FIFO
Elastic buffer (2-slot FWFT FIFO)
N_x, N_z: max allowable dimensions generics

FIGURE 5. **Spectral slice buffer architecture.**

neighbour. After N_z samples the N and yet N_z samples later the NW neighbour. A side-channel cascade of elastic buffers collects the neighbours along the cascade and packs them into an ever-wider word, until the entire neighbourhood is output. The elastic buffer based architecture has an important advantage; each stage is insensitive to latency of previous stages, sharing no control signals across stages and connects only through the ready-valid handshake. This latency insensitivity between stages becomes valuable for the following timing optimization.

Although this architecture feels intuitive, its implementation using on-chip BRAM is challenging; the second stage FIFO requires a large number of Xilinx BRAMs for most realistic image sizes (e.g. AVIRIS, MODIS, CRISM) causing the synthesizer to infer a BRAM cascade using special FPGA column routing. In order to improve the static timing delay we implemented the following optimization: the second stage FIFO is implemented as a cascade of back-to-back FWFT FIFOs, each sized precisely to avoid the BRAM cascade (shown top of in Figure 5). The FIFO output registers (due to FWFT) break the path resulting in very low delay. Therefore, we tradeoff a small amount of extra initial latency (1 cycle per stage of the deep FIFO) and a small amount of extra BRAMs (due to rounding of the FIFO sizes and allowable BRAM data widths), for a significant reduction in the worst path delay. This optimization removes the BRAM cascade from the critical path in images with a large spectral slice. The number of FIFOs in the second stage is calculated with a pure function taking the maximum dimension generics as input, which generates the 2nd stage FIFO cascade with a "for generate" expression.

C. PREDICTOR

The Predictor exploits an adaptive linear prediction method to predict the value of each image sample based on the values of nearby samples in a small 3D neighborhood. Prediction is performed sequentially in a single pass. The prediction residual, i.e. the difference between the predicted and actual sample values, is then mapped to an unsigned integer that is represented using the same number of bits as the input data sample. These mapped prediction residuals make up the preprocessor output.

The Predictor consists of seven logic units pipelined using elastic buffers: 1) the *Local Sum* unit, 2) the *Local Differences* unit 3) the *Inner Product* unit, 4) the *Scaling Exponent Update* unit, 5) the *Predictor* unit 6) the *Weight Update* unit and 7) the *Mapper* unit and a memory to store the weight vectors as depicted in Figure 6. The *Local Sum* unit receives four inputs, the neighboring samples of s_z $(s_n, s_{ne}, s_{nw}, s_w)$, to compute a weighted sum of named *local sum* σ_z. The *Local Sum* unit supports both column oriented and neighbor oriented local sum calculation. The *Local differences* unit receives the four neighboring samples of s_z $(s_N, s_{NE}, s_{NW}, s_W)$, the sample s_z and the extracted *local sum* σ_z from the *Local Sum* unit and calculates four differences: the *central local difference* d_z and the three directional local differences d_z^N, d_z^W, d_z^{NW}. These

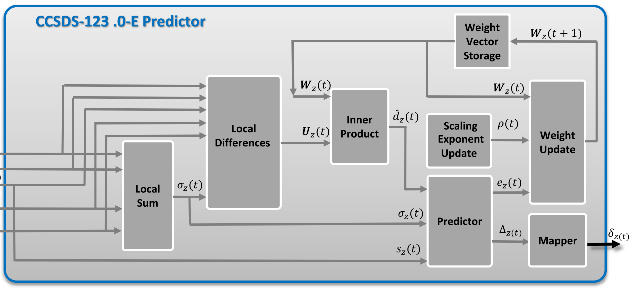

FIGURE 6. **Predictor.**

local differences are used to form and output the local difference vector U_z to be used by the *Inner Product* unit. Since $P = 3$ in the proposed CCSDS 123.0-B-1 IP core implementation, the vector U_z consists either of six local differences in full prediction mode or of three local differences in reduced prediction mode. The *Inner Product* unit implements the inner product calculation (either 6x6 in full prediction mode or 3x3 in reduced prediction mode) between the U_z vector that has been extracted from the *Local Differences* unit and a weight vector W_z from the *Weight Vector Storage* memory. To boost this unit's performance, the *Inner Product* unit exploits pipelined multipliers along with pipelined and optimized adder trees with compressors, to perform high speed inner product computation to produce the predicted central local difference $\hat{d}_z$. The *Scaling Exponent Update* unit receives the predictor parameters to calculate the weight update *scaling exponent* ρ_z (eq. (33) in [4]) that will be used in the *Weight Update* unit to adaptively update each weight vector W_z. The *Weight Update* unit implements the eq. (34) in [4] which adaptively updates every weight vector W_z after calculating each predicted central local difference $\hat{d}_z$. Since a separate weight vector is maintained for each band the updated weight should be stored in the Weight Vector Storage memory to be utilized for the following sample of the same band. The *Predictor* unit is the main unit that implements the prediction calculation to output the scaled predicted sample $\tilde{s}_z$. The unit receives two inputs, the predicted local difference $\hat{d}_z$ and the local sum σ_z and implements the eq. (29) of [4]. Finally, the *Mapper* unit calculates the prediction residuals Δ_z as the difference between the predicted $\hat{s}_z$, and the actual s_z sample values and maps them to positive integer values δ_z according to eq. (35) of [4]. The mapped prediction residual δ_z are stored in a FIFO, pipelining prediction with sample adaptive encoding.

The CCSDS 123.0-B-1 algorithm has a native high complexity in the weight update feedback loop (consisting of the inner product, prediction and weight update computation) that severely degrades the data-rate performance of all CCSDS 123.0-B-1 hardware implementations if this feedback loop should be implemented in one clock cycle in a sample per cycle implementation. The proposed CCSDS 123.0-B-1 IP core architecture exploits BIP ordering which minimizes data dependencies, enabling pipelining. Furthermore, the proposed architecture introduces for the first time, an excellent trade-off, by exploiting the inherent task-level

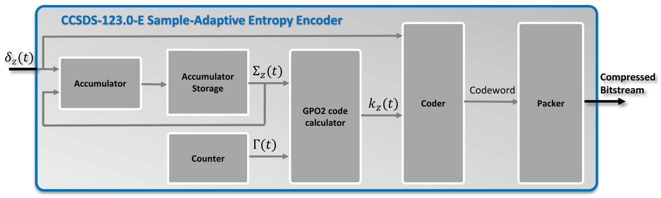

FIGURE 7. **Sample-adaptive entropy encoder.**

parallelism of CCSDS 123.0-B-1 algorithm under BIP ordering to implement C-slow retiming. C-slow retiming [14] is a process of increasing the throughput of a design by enabling fine-grained pipelining of computations within feedback loops. In C-slow retiming, each register is first replaced with C registers before retiming so that C separate streams of computation are distributed through the pipeline, greatly increasing the aggregate throughput. The effect of C-slow retiming is to enable pipelining of the critical path, even in the presence of feedback loops. However, to take advantage of this increased throughput, there needs to be sufficient task-level parallelism. The proposed architecture, exploits the BIP ordering of input samples that provides with sufficient task-level parallelism to implement C-slow retiming which enables a deep pipeline in the critical feedback loop (i.e. in weight update computation).

The inherent task-level parallelism of CCSDS 123.0-B-1 algorithm under BIP ordering enables splitting the weight update feedback loop computation into at most N_z clock cycles, since each band has a unique weight vector W_z. After a sample s_z enters the loop, the corresponding spectral band weight vector W_z is used to calculate the inner product and then moves on to update before being stored in the weight vector storage. The same weight vector W_z will only be required for computation again after N_z cycles, implying that the weight update feedback loop can utilize a deep pipeline. In our implementation the weight update feedback loop is deeply pipelined to achieve very high data-rate performance. The number of pipeline stages in the critical feedback loops such as the weight update computation of the predictor and the accumulation loop in the encoder is a parameter, configurable at compile-time, derived from the minimum N_z of the configuration. We should note, that in the majority of modern cases, the number of spectral bands N_z of multispectral and hyperspectral sensors is large enough (e.g., the AVIRIS optical sensor extracts 224 spectral bands), to provide enough task-level parallelism to support deep pipelining.

D. ENCODER

The Encoder, losslessly encodes the mapped prediction residuals using variable, length-limited Golomb-power-of-2 (GPO2) codes, adjusted based on statistics for each spectral band, according to the sample-adaptive encoder of FL. As an option, the encoder can leverage an existing very high data-rate performance implementation of the block-adaptive Rice encoder as a separate IP core implementing the CCSDS 121.0-B-2 Lossless Data Compression algorithm [15], although block-adaptive encoding does not provide any significant improvement in compression effectiveness under BIP ordering. The Encoder consists of five custom logic units: 1) the *Accumulator* unit, 2) the *Counter* unit 3) the *GPO2 code calculator* unit, 4) the *Coder* unit and 5) the *Packer* unit and one storage element, the *Accumulator Storage* as depicted in Figure 7.

The *Accumulator* unit is responsible to update the value of each spectral band z accumulator according to the eq. (42) of [4] that stores adaptive code selection statistics and is fetched from the *Accumulator Storage* memory. Moreover, the *Accumulator* unit initializes the accumulator value for each spectral band z according to the eq. (41) of [4]. The *Accumulator Storage* (which can be implemented as BRAM or distributed memory depending on the number of spectral bands N_z) stores the code statistics (the accumulator values) for all z spectral bands.

The *Counter* unit updates the value of the counter that is used for the code selection statistics according to the eq. (43) of [4]. The proposed Encoder takes advantage of BIP encoding order, thus a unique counter for all spectral bands can be utilized. The *GPO2 code calculator* unit is responsible for calculating the parameter k_t by evaluating the code statistics (the accumulator's value and the counter's value) according to the eq. (44) of [4]. The calculation of parameter k_t is complex. Therefore, this component is pipelined with intermediate operations distributed among registers to balance the logic levels required for each calculation stage. The *Coder* unit is similarly pipelined and performs the actual encoding by composing the codewords (and their length) for every mapped prediction residual based on the extracted code selection k_z from the *GPO2 code calculator* unit. Finally, the *Packer* unit packs the extracted codewords to double word (64 bits) packets.

E. RECONFIGURABLE FINE-GRAINED PIPELINING

Pipelining critical computation paths creates a tradeoff between maximum attainable frequency and register utilization. Moreover when the path in question is part of a feedback loop the maximum pipeline depth is limited by the algorithmic loop-bound or the maximum task level parallelism within the loop. Therefore the decision to pipeline or not a critical path, should be informed by the desirable resource-performance tradeoff. In case the critical path is not a feed-forward pipeline but part of a feedback loop a more complicated tradeoff is presented: if the available parallelism to exploit in the loop, is dependent on input/configuration parameters, the decision becomes whether to fully exploit the parallelism and limit the functionality of the IP Core, or leave parallelism unexploited to allow corner cases of input configuration. Especially for FPGAs, an IP Core should offer different choices to the end-user in the throughput-resource utilization spectrum allowing to fully exploit their reconfigurability.

Supporting multiple pipeline depths of a path (via an RTL generic/parameter in VHDL/Verilog), especially within a

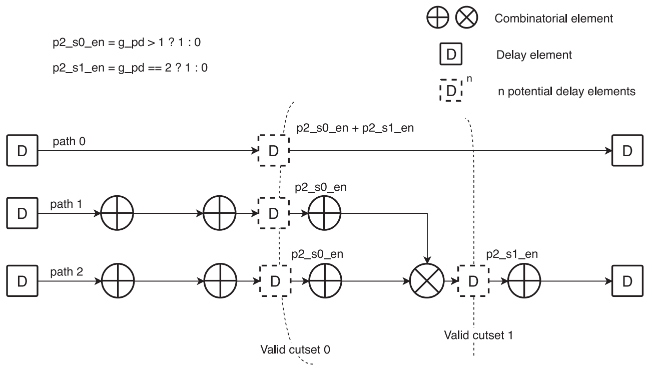

FIGURE 8. Reconfigurable pipeline example.

loop significantly complicates the RTL in most common design styles. Consider an FSM driving datapath control signals where adding a new pipeline stage requires re-organizing the timing of control signals. For this reason the pipeline depth in loops is commonly part of the initial requirements or a result of design-space exploration and is fixed in the RTL. A well understood technique to pipeline loop paths is C-slow retiming [14] commonly employed by the CAD tool, wherein C pipeline registers are added to the loop path causing the loop to compute a C tasks at a time. The registers shorten the combinatorial paths, increasing achievable Fmax. In this work, based on C-slow retiming technique, we have developed an RTL design pattern, to allow reconfiguration by generic (or Verilog parameter) and precisely place registers on the best path sites of a loop. A pipeline register component is central in this design pattern; all potential pipeline registers use it rather than standard registers in the common coding style. The register component accepts a generic depth to indicate the number of delay stages which may be zero. We instantiate this component, in between combinatorial processes implementing (potential) pipeline stages.

To demonstrate the technique consider the simple path in Figure 8 as an example, which may be part of a loop which has a loop-bound dependent on a generic parameter. Path 2 is critical and we wish to pipeline it depending on a “g_pd” generic pipeline depth derived from the loop-bound. After analysis of the datapath we identify “cutset 0”, a valid cutset which splits path 2 in two halves. According to the cutset we place three register components in path 0, path 1 and path 2, using a new integer constant “p2_s0_en”: path 2 site 0 enable. Similarly for “cutset 1” we place 2 registers guarded by path 2 site 1 enable “p2_s1_en”. Notice in path 0, which participates in both cutsets, the pipeline register is instantiated with stages equal to the sum of the two enables “p2_s1_en + p2_s1_en”. To derive the two enables depending on the value of the pipeline budget “g_pd”, in this simple case we can use “if” switches, in more complicated components a static lookup table may be used. If g_pd is greater than 0 enable the pipeline site 0, if equal to 2 also enable pipeline site 1. Notice though that this implies a priority of site 0, if the pipeline budget equals 1, we prefer to place the register in site 0. The priority of pipelining sites given a budget must be discovered by experiments as performed in the next section for this IP Core.

To extend this example into a loop, we can consider path 0 to be a token tag. Upon entering the loop, each token is marked by an incrementing tag. The token caries the tag along, through the loop pipeline until the loop join. In case the loop includes memory the tag is a memory address to write to. By using this tag pattern, the only difference between a feed-forward pipeline and a pipeline which is a loop branch; is the fact that the loop branch pipeline must be shorter than the loop bound. In the predictor loop implemented in this paper, the loop must complete within N_z cycles, to write the updated weight in the weight storage, before it needs to be read.

Most synthesis tools can perform retiming/register balancing optimization, to move registers backwards or forwards in a path. Therefore precisely placing pipeline registers, should ideally not be required. Ideally it should suffice to place “g_pd” registers in the end of a path which the tool may move backwards to optimally distribute across the path. In practice however, the effect of automated retiming was found to be limited as shown by the experiments in the following section, making precise placement of registers beneficial. Similar limitations of automated retiming have been reported in the literature especially for DSPs [23].

F. APPLYING RECONFIGURABLE PIPELINING IN THE PREDICTOR LOOP

In the proposed architecture which uses BIP encoding order, the available parallelism in the predictor and encoder loops, is directly related to the minimum number of spectral bands the IP Core can support. To allow supporting small spectral dimension images, but also high throughput performance we use the pipeline reconfiguration design pattern, which allows the loop path pipelines to have configurable depth. This allows the core to be configured by a single generic at compile time, to support spectral dimensions down to a minimum N_z which internally activates or deactivates pipeline registers in the critical loop paths. In every case to respect the worst loop-bound, the loops are pipelined to *minimum* $N_z - 1$. Because the predictor loop path is the design critical path (all other paths are pipelined enough to have lower delay) this directly changes the maximum attainable frequency. In the following, we consider the predictor loop as an example, the encoder loop is similarly implemented but less critical to timing.

Although the pipeline reconfiguration design pattern is applied at the lowest level of the structural hierarchy (i.e. where the actual operations are performed), a complication arises. There are multiple paths and multiple possible sites which may be available for the instantiation of a pipeline register for a given pipeline budget, at a given level of hierarchy. In each level it is not immediately obvious by theoretical analysis, which distribution of the pipeline budget to the immediately lower level will yield the highest Fmax. To

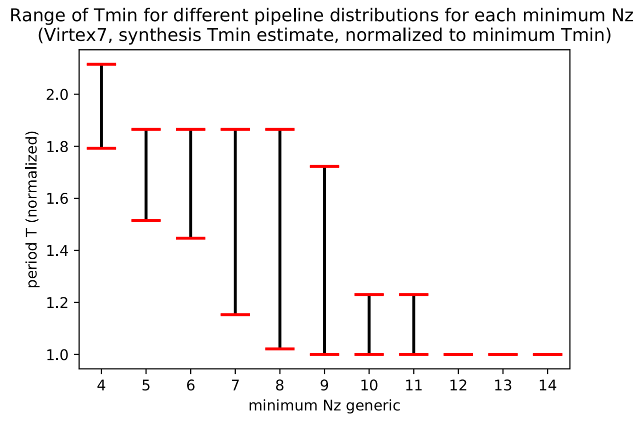

FIGURE 9. **Predictor synthesis best achievable period for different pipeline distributions for each minimum N_Z.**

resolve this we perform design space exploration, guided by synthesis results, to investigate all possible pipeline budget distributions for each level of hierarchy, starting from the lowest, toward the highest levels.

To discover the best possible distribution of the pipeline budget at a given level of hierarchy for each minimum Nz we perform a set of experiments as follows:

- We set as top level the component immediately higher in the hierarchy, to those components whose pipeline distribution is under investigation.
- We expose the generic pipeline depths of lower level components to the synthesized top level generic port.
- We script the Vivado synthesis flow targeting a Virtex-7 device and synthesize for all minimum N_z and all valid pipeline budget permutations.

Using a Virtex-7 device for the experiments allows us to use the Vivado flow, rather than the ISE flow for Virtex-5 which is more cumbersome to automate. Nevertheless, results port well between the two flows regarding the distribution of the pipeline budget and minimum N_z scaling, according to our experiments. We only use the Vivado flow for this experiment due to the need for automation; all other experiments use Xilinx ISE. Notice the span of results in the range plot for each minimum N_z in Figure 9. This implies a

Minimum Nz	*Inner product*	*Prediction calc.*	*Weight update*
4	1	1	1
5	2	0	2
6	2	0	3
7	2	1	3
8	3	0	4
9	4	0	4
10	5	0	4
11	4	0	6
12	4	1	6
13	6	1	5
14	6	1	6

FIGURE 10. **Best pipeline budget distribution in the predictor loop.**

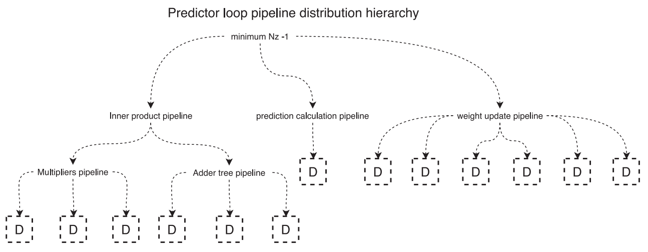

FIGURE 11. **Pipeline budget inheritance graph.**

total pipeline budget in the loop, the synthesis estimated best achievable period is shown (targeting a Virtex-7 part with retiming optimization enabled).

For each minimum N_z the lowest point in the range is the best distribution of pipeline registers, the highest is the worst. This plot showcases the effectiveness of this method and why it is indeed required. For a single minimum N_z e.g. min. $N_z = 7$ depending on the distribution of the pipeline budget in the path we might incur an overhead up to 85 percent for a non-optimal distribution of pipeline registers.

It should be noted that not all permutations of pipeline depth budgets need to be benchmarked by experiments because they are not all valid at each level of hierarchy. Valid distributions are guarded by compile time assertions and ranged integer "pipeline_depth" generic datatypes for each generic port. This greatly limits the number of required experiments. Finally in the design process the results of the best distribution for each applicable pipeline budget are encoded into a lookup table. In Figure 10 the lookup table for the top level component of the predictor loop is shown. Each component has a single generic "pipeline_depth" and internally a lookup table to distribute this pipeline budget to the next hierarchical level. In RTL this is implemented using a combination of constants, pure functions and generics. Recursively each component uses its pipeline budget and its hard-coded lookup table to instantiate the immediately hierarchically lower components, with their pipeline budget as a generic instantiation. In the predictor loop, the recursion starts with a budget of $N_z - 1$ which is distributed to three components. Figure 11 shows how the pipeline budget is inherited downwards in the instantiation graph of the predictor loop.

G. SCHEDULING CCSDS 123.0-B-1 PROCESSING TASKS

In order to describe the exploitation of task-level parallelism of the CCSDS 123.0-B-1 algorithm under BIP ordering, the Gantt chart in Figure 12 illustrates the scheduling of processing tasks in the overall pipeline. It should be noted that an initialization phase of N_z cycles required to initialize the Accumulator storage for each spectral band according to the eq. (41) of [4] is not shown in Figure 12. This initialization is the only component of the total latency, dependent on input parameters, all other initialization latencies are independent of the image and due to either pipelining or operations such as the bitstream header output.

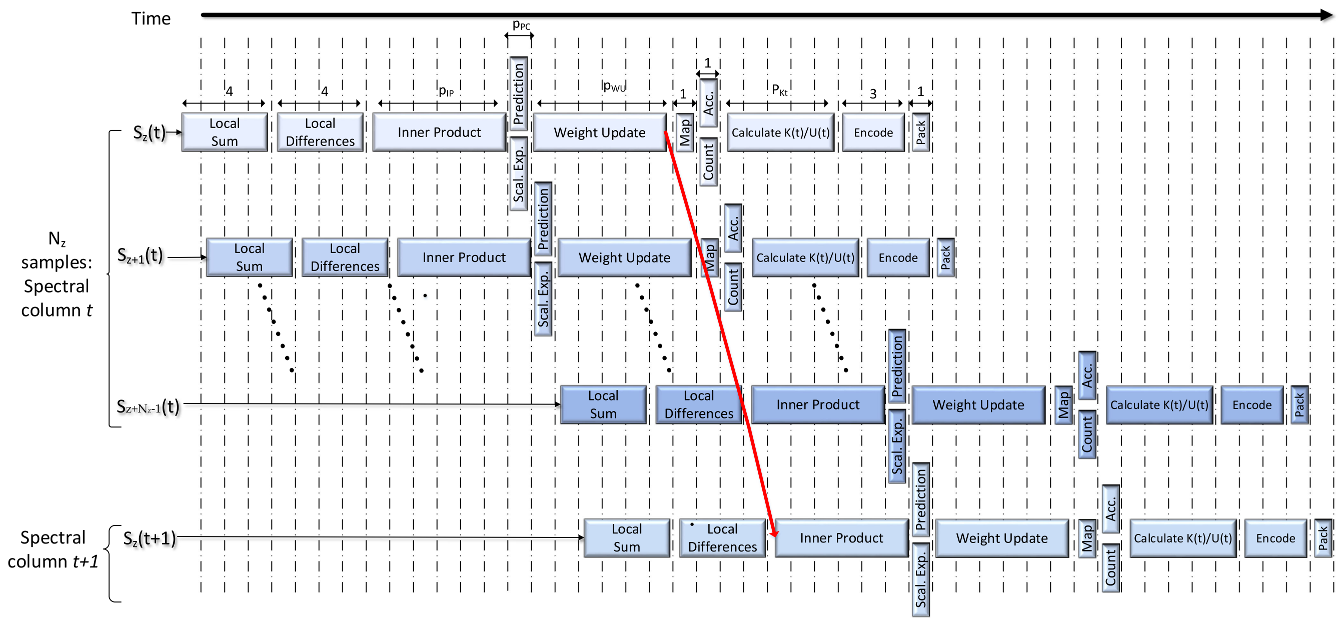

FIGURE 12. Scheduling of CCSDS 123.0-B-1 processing tasks when pipeline is configured as (pIP, pPC, pWU, pKt) $= (6, 1, 6, 5)$.

The data dependency in red shown in Figure 12, illustrates that the computation of the Inner Product of a sample $S_z(t+1)$ cannot be started until the weight vector of the previous sample of the same spectral band $S_z(t)$ has been updated. The proposed architecture enables a configurable fine-grained pipeline of computations in the critical feedback loop according to a single compile-time configuration parameter $(minimum\, N_z)$. This allows deep pipelines to be configured for images with enough spectral bands but also allows graceful degradation of performance for multispectral images with very few spectral bands. It should be noted, however, in most cases -even for multispectral sensors- the number of spectral bands N_z is large enough to achieve very high throughput.

The weight update feedback loop consists of the inner product calculation, scaled predicted sample calculation, prediction calculation and weight update calculation. These components are pipelined with a compile-time configurable number of stages. The sum of the weight update feedback loop components' latencies $p_{IP} + p_{PC} + p_{WU}$ should be less than N_z, otherwise the required weight vector for the sample $S_z(t+1)$ will still be in flight in the loop when it is needed in the input of the Inner Product.

V. CCSDS 123.0-B-1 IP CORE VERIFICATION/ VALIDATION

The CCSDS 123.0-B-1 IP core has been extensively verified using RTL simulation using Mentor Graphics Questa and FPGA-in-the-loop based verification using a significant amount of test images from the corpus of Hyperspectral and Multispectral test images (annex A of [5], available in [17]), as well as, the test pattern image [17] which was explicitly designed for testing CCSDS 123.0-B-1 implementations and a series of random images. The ESA software implementation in C was used as a golden reference model of CCSDS 123.0-B-1 [18]. The IP core was validated on-chip using a Xilinx development board as a hardware demonstrator with a Virtex-5 FPGA interfacing over PCIe.

A. VERIFICATION STRATEGY

Functional verification has been tightly integrated into the development flow. Three verification phases were performed at various levels of detail and thoroughness: unit, regression and integration testing. The overall goal was to enable an agile development flow by minimizing friction and time from the inception of a new feature or optimization to its adoption into the stable version. Version control using a stable and feature branches was used where new features were developed in a feature branch and integrated to stable after passing the integration tests suite.

Unit testing was performed for debugging new features. To aid debugging, these include inline error checks within the waveforms to spot the differences with the cycle accurate golden trace at the earliest moment of a bug's occurrence. Unit tests are small and adapted to excite the bug under examination. After a bug's resolution, the previously failing test case configuration is formalized and inserted into the regression tests suite.

The regression tests represent past bugs or configurations which are suspect to show bugs in complex areas of the design. They are usually ran before committing a new version into version control and consist of mainly the test pattern [17], with various configurations and random tests at "corner case" compile time configurations (very large/small sizes compared to the generic configurations, compression parameters likely to cause overflows etc.).

Integration tests are composed of a set of images from the corpus of tests [17] which are too large to be run within

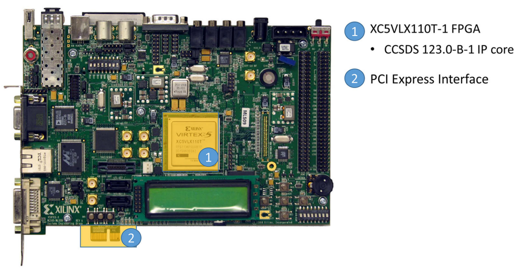

FIGURE 13. **Development board used for Hardware Demonstrator.**

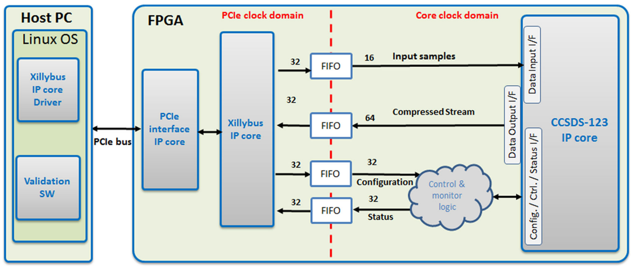

FIGURE 14. **Validation/verification platform block diagram.**

the normal development flow. They are automatically ran by the Continuous integration framework, Gitlab-CI [17] for each RTL version (git commit) on a remote server.

The testing framework is based on VUnit and a set of python scripts, as well as the C golden model of the compressor [18]. Each test suite is described as a CSV file with a test on each line and all the compile time (generics) and run-time (compression) parameters. Python scripts read the tests file and interpret the parameters to invoke the golden compressor binary to produce the verification data. Then a testbench instrumented with VUnit is invoked with the Questa simulator comparing with golden data. The same flow is ran either on a designer's workstation for unit and regression tests or on the remote server running Gitlab-CI, on each git commit, for the integration tests.

B. HARDWARE DEMONSTRATOR PLATFORM

A hardware functional verification platform was developed using the Xilinx XUPV5 development board (Figure 13) which hosts a XC5VLX110T-1. The hardware platform (Figure 14) leverages the Xillibus system [20] which is an end-to-end solution for data transfer between an FPGA and a host. This allows to quickly (compared to simulation) compress a significant number of hyperspectral test images and verify that the compressed bitstream matches the golden reference model and decompresses to the original image using the reference software. The Xillibus IP core interfaces with four asymmetric FIFOs (sample, configuration, compressed data, and status) to download hyperspectral data to the FPGA, upload compressed data to the host, configure and monitor the IP core. Dual clock FIFOs are used to negotiate the clock domain crossing between the PCIe endpoint and IP Core domains. The input and output FIFOs directly connect the IP Core and Xillybus controller. The control and status FIFOs are connected to logic which drives the IP core configuration interface and maintains performance and status metrics.

Host side software is written in Python and runs on Linux, interfacing with the character devices that the Xillybus driver presents. Hyperspectral cube data are down-streamed, concurrently with up-streaming compressed samples and occasionally polling the status registers, in independent threads, which start after the IP core is configured. Finally, the output data is decompressed using the reference software to generate a test pass/fail result.

VI. IMPLEMENTATION RESULTS AND COMPARISONS

The CCSDS 123.0-B-1 IP core was implemented targeting the Xilinx Virtex-5 FX130T (speed grade -1) FPGA which is the commercial equivalent of the space-grade Virtex-5QV. Implementation statistics using representative configurations are shown in Table 2, including the AVIRIS hyperspectral instrument (680x512x224, 16bbp), which enables a direct comparison with other implementations. The on-chip storage requirements (Distributed RAM and BRAM) depend on the number of bands: $3 * N_z$ and $(N_x - 2) * N_z$, samples for the neighbouring sample storage is required for the spectral slice buffer, as well as, N_z weight vectors and N_z accumulator values in the predictor and encoder, respectively.

The proposed implementation of the CCSDS 123.0-B-1 algorithm achieves more than 3.3 Gbps data rate performance when configured for the AVIRIS hyperspectral instrument. Such a data-rate performance enables a seamless integration along with SpaceFibre, the next-generation very high-speed serial data-line (ECSS-E-ST-50-11C) which supports lane rates up to 3.125 Gbps when targeting the same space-grade FPGA technology [21]. The implementation for min. $N_z = 14$ can fully utilize the available throughput of a

TABLE 2. **FPGA implementation statistics (X5VFX130T-1).**

Image	AVIRIS (680x512x224, 16bbp)	MODIS (2030x1354x17, 12bbp)	CRISM (640x420x545, 12bbp)
Device utilization	9462 (11%) LUTs 83 (27%) BRAMs 6 (1%) DSP48E 9990 (12%) FFs	9641 (11%) LUTs 23 (7%) BRAMs 6 (1%) DSP48E 10554 (12%) FFs	17070 (20%) LUTs 179 (60%) BRAMs 6 (1%) DSP48E 23377 (28%) FFs
MS/sec	213	213	213
Gbps	3.3	2.5	2.5

* *Min $N_z = 14$.*

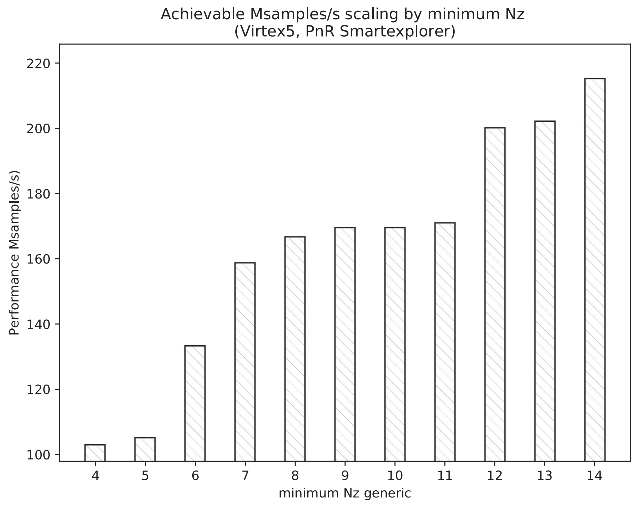

FIGURE 15. Post PnR performance depending on min. N_z generic.

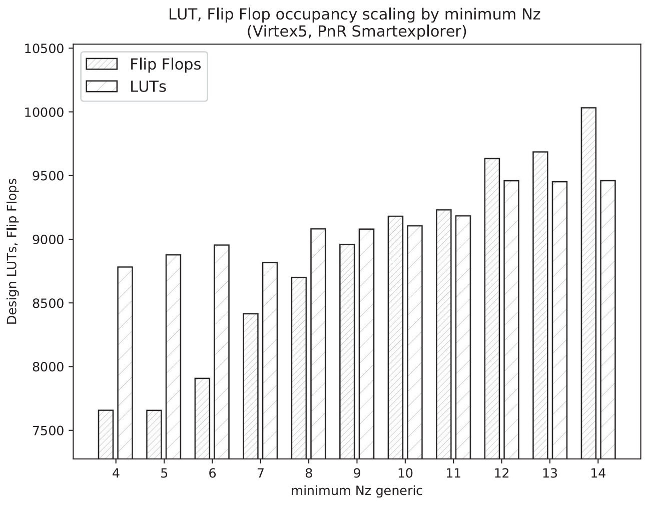

FIGURE 16. Post PnR resource utlization depending on min. N_z generic.

SpaceFibre lane, with a comfortable margin for possible variations in PnR quality of results.

Figure 15 shows the scaling of post-PnR throughput performance of the implementation, depending on the min. N_z generic with all other generics configured for the AVIRIS image. This validates the proposed reconfigurable pipeline architecture; performance increases with the increase in pipeline depth afforded by larger min. N_z and the allocated best pipeline budget distribution. It should be noted that in post-PnR STA, the performance of deep pipelines increases significantly with values of min. $N_z > 11$, a fact that is not apparent in synthesis results (Figure 9) that provide only with rough estimations on FPGA routing delays.

Similarly in Figure 16 considering the effect of the min. N_z generic on resource utilization, the results are as expected. LUT utilization is not significantly affected (small variations are due to different implementation strategies) while flip-flop utilization is increased due to pipelining.

Other FPGA implementations of CCSDS 123.0-B-1 have been recently presented in the literature. In [13], NASA FL was implemented targeting Virtex-5 (SX50T) and Virtex-6 (LX240T) FPGAs. It supports only sample-adaptive encoding, 13-bit maximum bit depth, BIP ordering and $P = 3$. When targeting the Virtex-5 technology, it requires 12697 slice LUTs. The maximum frequency is 40 MHz, compressing one sample every clock cycle, which yields a throughput of 40 MSamples/sec. The SHyLoC IP core [11] is fully configurable and supports different compression orders (BSQ, BIP, BIL). According to synthesis results using a configuration with BIP architecture that enables compressing one sample every clock cycle and does not require external memory, number of prediction bands $P = 3$, sample-adaptive encoding and image dimensions targeting the AVIRIS instrument when targeting the same Virtex-5 FX130T FPGA technology, it requires 4645 LUTs (5.67 percent of total), 11 DSP48E (3.44 percent of total) and 74 BRAMs (24.8 percent of total). The maximum frequency is 85 MHz, which yields a throughput of 85 MSamples/sec.

Detailed statistics comparing the CCSDS 123.0-B-1 FPGA implementation in this paper with existing implementations are provided in Table 3. All implementations require very low FPGA resources for the typical image dimensions and only the BIP implementations can achieve very high data-

TABLE 3. Comparison with other FPGA implementations.

	Other FPGA Implementations		This paper
	NASA FL [13]	SHyLoC [11]	
Sample ordering	BIP	BIP	BIP
Number of prediction bands (P)	3	3	3
Image dimensions (N_x, N_y, N_z)	640, 32, 427	680, 512, 224	680, 512, 224
Max bit depth (D)	13	16	16
Target device	Xilinx Virtex-5 (SX50T)	Xilinx Virtex-5 (FX130T)	Xilinx Virtex-5 (FX130T)
Device utilization (Virtex-5 resources)			
LUTs	12697	4645	9462
BRAMs	8	74*	83*
DSP48E	3	11	6
Max frequency (MHz)	40	85	213
Max throughput (MSamples/s)	40	85	213

Does not require external memory.

TABLE 4. Performance comparison with CPU/GPU implementations (with segmentation).

Platform	Time (ms)	MSamples/s	Watts	MSamples/s/W
CPU Intel i7 2760QM 2.4 GHz OpenMP (4core) [24]	569	127.89	$<45^*$	2.84
GPU NVIDIA GeForce GTX 560M CUDA [24]	226	321.91	$<75^*$	4.29
GPU NVIDIA GeForce GTX 560M CUDA(2x) [24]	204	356.63	$<150^*$	2.37
Virtex-5 Space Grade FPGA (this paper)	366	213.00	4.72^{**}	45.13

**max TDP,* ** *Using Xilinx XPE (Tj = 85°C).*

rate performance. When compared with other implementations targeting the same FPGA reference technology, the proposed hardware accelerator sets the new state-of-the-art in data-rate performance at 213 MSamples/s (3.3 Gbps). Notice that, the initial latency (sum of the number of bands and the total pipeline stages) is negligible for typical hyperspectral image cubes (291 cycles for the AVIRIS image) and is already included in the FPGA performance figures.

Although the power consumption of current GPU technology is prohibitive for on-board deployment, Table 4 presents a comparison of a space grade FPGA implementation with CPU and GPU implementations to set a baseline. It should be noted that the CPU and GPU implementations of [24] leverage image segmentation and segment level parallelism to boost throughput performance at the cost of decreased compression effectiveness [5]. Comparing on the basis of performance per watt, the FPGA implementation is superior by an order of magnitude. However, exploitation of segment level parallelism is also possible in a space-grade FPGA implementation using multiple compression engines that would achieve a higher performance than GPU implementations while maintaining the performance per watt advantage.

The power consumption for the Xilinx FPGA was estimated by the Xilinx Power Estimation tool (XPE). For the power estimation, the clock frequency was set to the maximum achievable (213 MHz) while the used resources were imported from Xilinx ISE. The junction temperature (Tj) was set to 85°C.

VII. CONCLUSION

In this paper, we proposed a novel, high data-rate performance hardware accelerator, implementing the CCSDS 123.0-B-1 algorithm as an IP core targeting space-grade FPGAs. For the first time, the introduced architecture based on the principles of C-slow retiming, exploits the inherent task-level parallelism of the CCSDS 123.0-B-1 algorithm under BIP ordering and implements a reconfigurable fine-grained pipeline in critical feedback loops, achieving high throughput performance. Moreover, it is a single FPGA solution without external DDR memory requirement that results in significant savings in SWaP-C (Size, Weight, Power and Cost).

The proposed CCSDS 123.0-B-1 IP core achieves far beyond the state-of-the-art data-rate performance with a maximum throughput at 213 MSamples/s (3.3 Gbps @ 16-bits) using 11 percent of LUTs and 27 percent of BRAMs of the Virtex-5QV FPGA for a typical hyperspectral image configuration. Such a high data-rate performance can leverage the full throughput of a SpaceFibre lane. To the best of our knowledge, it is the fastest implementation of CCSDS 123.0-B-1 targeting a space-grade reconfigurable SRAM FPGA to date.

REFERENCES

[1] I. Blanes, E. Magli, and J. Serra-Sagrista, “A tutorial on image compression for optical space imaging systems,” *IEEE Geosci. Remote Sens. Mag.*, vol. 2, no. 3, pp. 8–26, Sep. 2014.
[2] The Consultative Committee for Space Data Systems, Lossless Data Compression Recommended Standard CCSDS 121.0-B-2, 2012.
[3] The Consultative Committee for Space Data Systems, Image Data Compression Recommended Standard, CCSDS 122.0-B-1, 2005.
[4] The Consultative Committee for Space Data Systems, Lossless Multispectral & Hyperspectral Image Compression Reccomended Standard, CCSDS 123.0-B-1, 2012.
[5] The Consultative Committee for Space Data Systems, Lossless Multispectral & Hyperspectral Image Compression Informational Report, CCSDS 120.2-G-1, Dec. 2015.
[6] M. Klimesh, “Low-complexity lossless compression of hyperspectral imagery via adaptive filtering,” *IPN Progress Report*, vol. 42-163, pp. 1–10, Nov. 2005.
[7] J. E. Sanchez, E. Auge, J. Santalo, I. Blanes, J. Serra-Sagrista, and A. Kiely, “Review and implementation of the emerging CCSDS recommended standard for multispectral and hyperspectral lossless image coding,” in *Proc. 1st Int. Conf. Data Compression Commun. Processing*, Jun. 2011, pp. 222–228.
[8] Radiation-Hardened, Space-Grade Virtex-5QV Family Data Sheet DS192, Xilinx, Inc., Jan. 2018.
[9] G. Swift, C. Carmichael, G. Allen, G. Madias, E. Miller, and R. Monreal, “Compendium of XRTC radiation results on all single-event effects observed in the Virtex-5QV,” in *Proc. Conf. ReSpace/MAPLD*, Aug. 2011.
[10] L. Santos, L. Berrojo, J. Moreno, J. F. Lopez, and R. Sarmiento, “Multispectral and hyperspectral lossless compressor for space applications (HyLoC): A low-complexity FPGA implementation of the CCSDS 123 standard,” *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 9, no. 2, pp. 757–770, Feb. 2016.
[11] L. Santos, A. Gomez, R. Sarmiento, L. Fossati, and D. Merodio, “Architectural design and FPGA implementation of CCSDS-123 and CCSDS-121 IP cores for lossless satellite data compression,” in *Informal Proc. 5th ESA Int. Workshop On-Board Payload Data Compression*, Sep. 2016.
[12] G. Theodorou, N. Kranitis, A. Tsigkanos, and A. Paschalis, “High performance CCSDS 123.0-B-1 multispectral & hyperspectral image compression implementation on a space-grade SRAM FPGA,” in *Informal Proc. 5th ESA Int. Workshop On-Board Payload Data Compression*, Sep. 2016.
[13] D. Keymeulen, N. Aranki, A. Bakhshi, H. Luong, C. Sarture, and D. Dolman, “Airborne demonstration of FPGA implementation of Fast Lossless hyperspectral data compression system,” in *Proc. NASA/ESA Conf. Adaptive Hardware Syst.*, Jul. 2014, pp. 278–284.
[14] C. Leiserson, F. Rose, and J. Saxe. “Optimizing synchronous circuitry by retiming,” in *Proc. 3rd Caltech Conf. VLSI*, Mar. 1983, pp. 87–116.
[15] N. Kranitis, I. Sideris, A. Tsigkanos, G. Theodorou, A. Paschalis, and R. Vitulli, “An efficient FPGA implementation of CCSDS 121.0-B-2 lossless data compression algorithm for image compression,” *J. Appl. Remote Sens.*, vol. 9, no. 1, May 2015, Art. no. 097499.
[16] E. Auge, J. E. Sanchez, A. Kiely, I. Blanes, and J. Serra-Sagrista, “Performance impact of parameter tuning on the CCSDS-123 lossless multi- and hyperspectral image compression standard,” *J. Appl. Remote Sens.*, vol. 7, no. 1, Aug. 2013, Art. no. 074594.
[17] The Consultative Committee for Space Data Systems, Corpus of Hyperspectral and Multispectral Images, [Online]. Available: http://cwe.ccsds.org/sls/docs/sls-dc/123.0-B-Info/TestData, Accessed on: Sep. 2017.
[18] ESA TEC-EDP Data Compression Tools, CCSDS 123.0-B-1 Multispectral and Hyperspectral Lossless Data Compression SW, [Online]. Available: http://www.esa.int/Our_Activities/Space_Engineering/Onboard_Data_Processing/Data_compression_tools, Accessed on: Sep. 2017.
[19] VUnit testing for VHDL, [Online]. Available: http://vunit.github.io, Accessed on: Sep. 2017.

[20] Xillibus IP core product brief v1.9, Jan. 28, 2016. [Online]. Available: http://xillybus.com/downloads/xillybus_product_brief.pdf, Accessed on: Sep. 2017.

[21] SpaceFibre IP Core data sheet, Star Dundee, [Online]. Available: https://www.star-dundee.com/sites/default/files/SpaceFibre%20IP%20Core%20Data%20Sheet.pdf, Acessed on: Sep. 2017.

[22] Continuous Integration software, Gitlab, Inc. [Online]. Available: https://about.gitlab.com/features/gitlab-ci-cd/, Accessed on: Sep. 2017.

[23] B. Ronak and S. A. Fahmy, "Mapping for maximum performance on FPGA DSP blocks," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 35, no. 4, pp. 573–585, Apr. 2016.

[24] B. Hopson, K. Benkrid, D. Keymeulen, and N. Aranki, "Real-time CCSDS lossless adaptive hyperspectral image compression on parallel GPGPU & multicore processor systems," in *Proc. NASA/ESA Conf. Adaptive Hardware Syst*, 2012, pp. 107–114.

ANTONIS TSIGKANOS (M'17) received the BSEE degree from the Electrical and Computer Engineering School of the National Technical University of Athens. He is working toward the PhD degree from the Department of Informatics and Telecommunications, NKUA. His current research interests include on-board payload data processing systems, SoC design, dependable and reconfigurable computing, and reliability. He is a member of the IEEE.

NEKTARIOS KRANITIS (M'03) received the BSc degree from the Department of Physics, University of Patras, Patras, Greece, in 1997, and the PhD degree from the Department of Informatics and Telecommunications, University of Athens, Athens, Greece, in 2005. Currently, he is a postdoctoral researcher with the Department of Informatics and Telecommunications. He has published more tahn 50 papers in peer reviewed transactions, journals, and conference proceedings. His current research interests include on-board payload data processing systems and dependable computer architecture. He is a member of the IEEE.

GEORGE THEODOROU (M'06) received the BSc, MSc, and PhD degrees from the Department of Informatics and Telecommunications, NKUA. Currently, he is a senior design engineer in Xilinx at Optical Network Solutions Group. He has published more than 10 papers in peer reviewed transactions, journals, and conference proceedings. His current research interests include high-speed IP core solutions for OTNs, on-board payload data processing systems, FPGA-based network architectures, microprocessor and SoC design, dependable computer architecture, memory testing, and reliability. He is a member of the IEEE.

ANTONIS PASCHALIS (M'97) received the BSc degree in physics, the MSc degree in electronics and computers, and the PhD degree in computers, all from the Department of Physics, National and Kapodistrian University of Athens, Athens, Greece, in 1983, 1986, and 1987, respectively. He is a professor with the Department of Informatics and Telecommunications, University of Athens, Athens. He has published more than 150 papers (22 IEEE Transactions) and holds a U.S. patent. His current research interests include reconfigurable payload data processing units and high-speed IP cores for space applications, VLSI design and testing, and dependable computer architecture. He is a "Golden Core Member" of IEEE Computer Society and has participated in several tens of organizing and program committees of international events (9 times as General Chair) in the area of design and test. He is a member of the IEEE.