An Efficient Real-Time FPGA Implementation of the CCSDS-123 Compression Standard for Hyperspectral Images

Johan Fjeldtvedt, *Student Member, IEEE*, Milica Orlandić, *Member, IEEE* and Tor Arne Johansen, *Senior Member, IEEE*

***Abstract*—Hyperspectral imaging (HSI) can extract information from scenes on the Earth surface acquired by airborne or spaceborne sensors. On-board processing of hyperspectral imaging is characterized by large datasets on one side and limited processing time and communication links on the other. The CCSDS-123 algorithm is a compression standard assembled for space-related application which allows compacted data transmission via a transmission link. In this paper, a low-complexity high-throughput FPGA implementation of CCSDS-123 compression algorithm with BIP ordering is presented. Hardware accelerators implemented in Field-Programmable Gate Arrays (FPGAs) are increasingly used for custom tasks due to their efficiency, low power and reconfigurability. The proposed implementation of CCSDS-123 compression standard has been tested on Zedboard development board containing a Zynq-7020 FPGA. The results are verified against an existing software implementation. The synthesized design can perform on-the-fly processing of hyperspectral images with maximum operating frequency of 147 MHz. The achieved throughput of 147 Msamples/s (2.35 Gb/s) is higher when compared with the throughput reported in recent state of the art FPGA implementations.**

***Index Terms*—CCSDS-123 compression, Field programmable gate arrays (FPGA), HSI, lossless compression, space-related applications**

I. Introduction

Due to the increasing demands of onboard autonomy, onboard computing is one of the needs for future drones and spacecraft. In the recent period, the space development has moved towards small-satellite (SmallSat) missions, low-cost platforms with introduced budget and schedule flexibility. SmallSat computing resides in the use of commercial processors contrarily to expensive flagship satellite missions which primarily rely upon rad-hard devices to safeguard electronics from failing. In this sense, Small satellites are often used as technology demonstrations to reduce time for advancing the state of the art. Space-related applications such as synthetic aperture radar (SAR), image processing and hyperspectral imaging (HSI) require critical data processing to be performed onboard in order to preserve transmission bandwidth. Hyperspectral imaging applications, in particular, are characterized by large amount of data in a 3D cube form. A HSI pixel consists typically of hundreds of components in the spectral domain. The widely used push-broom imagers, such as a lightweight imager of high precision for drone and airborne applications in [1], record spatial dimensions as a perpendicular cross-section of any target along the track direction.

The data compression in onboard HSI processing is performed to lower the amount of data for transmission back to Earth. The challenge is to perform extensive data reduction with minimum requirements of limited onboard resources. In addition, researchers in HSI remote sensing urge for no loss of information in the experiments, making lossless compression the preferable compression mode. Even though lossless compression reduces data volume, it does not compromise data integrity and the image can be fully recovered after decompression. Real-time data processing and reconfigurable hardware solutions have become the standard choice for onboard remote sensing application and compression due to their small size, weight, power consumption and their resistance to damages and malfunctions caused by ionizing radiation [2]. Hybrid processing systems, system-on-a-chip (SoC) platforms, combining different technological solutions such as CPU, GPU, DSPs and FPGAs are attractive for onboard processing. The sequential portion of algorithms run on the processors whereas intensive computational operations suitable for parallel implementation are placed in hardware accelerators (in FPGAs on the programmable logic). FPGAs are desirable for space-directed applications because of high performance achieved by creating custom application-specific architectures. Massive computational speedup and low energy consumption are often achieved by implementing highly optimized data paths. By expanding logic resources in current FPGAs, it is also possible to execute several complex algorithmic task in parallel achieving the throughput required for real-time processing. Common SoC devices for space-related applications (CubeSat and other SmallSat single-board computers) used in ongoing Low Earth Orbit (LEO) missions include the Xilinx Zynq 7000 series, Microsemi Smart Fusion or Xilinx Virtex-5 as stated in [3]. The survey also reports the poor performance of rad-hard FPGAs compared to commercial ones (Xilinx Virtex 5QV FX150 versus Zynq-7020).

A typical compression system in image/video processing consists of decorrelation, quantization and entropy encoding stages. The decorrelation stage contains a prediction process and/or transform process. Most common transforms are

J.Fjeldtvedt and M. Orlandić is with the Departmentof Electronic Systems, Norwegian University of Science and Technology, Norway.

T. A. Johansen is with the Department of Engineering Cybernetics, Norwegian University of Science and Technology, Norway.

Corresponding author: Milica Orlandić, e-mail: milica.orlandic@ntnu.no.

Manuscript received **May 8th**, 2018; revised January –, —-.

Karhunen–Loeve Transform (KLT), Discrete Wavelet Transform (DWT) and Discrete Cosine Transform (DCT). Among algorithms for compressing HSI images, JPEG2000 standard uses DWT for spatial decorrelation, whereas KLT is part of Principle Component Analysis (PCA) widely used for spectral dimensionality reduction [4]. The low complexity JPEG-LS uses a simple non-linear predictor and a context-based entropy coder, whereas a differential version of JPEG-LS algorithm encodes differences of adjacent bands. The LCE algorithm [5], characterized by high tolerance to bit-flips in the encoded stream, is considered as a good candidate for parallel implementation. Recent FPGA implementations of LCE algorithm are presented in [6], [7].

The Consultative Committee for Space Data Systems (CCSDS) defines standards for lossless data compressors applicable to 3D images produced by multispectral and hyperspectral imagers. The CCSDS-121 standard [8] consists of pre-processing unit with prediction and mapping operations and adaptive entropy coder based on Rice coding. A recent FPGA implementation of CCSDS-121 algorithm is presented in [9]. The successor CCSDS-122 [10] shares features with JPEG2000 such as a three-level 2D discrete wavelet transform (DWT), successive quality refinement of an images and a Bit Plane Encoder (BPE). An FPGA implementation of CCSDS-122 standard in [11] is tested in EnMap mission satellite. The latest CCSDS-123 standard [12] is characterized by low complexity suitable for fast real-time hardware implementation. Recent FPGA implementations of the CCSDS-123 standard [13] presented in [14]–[18] target high throughput and take into account limitations in terms of storage utilization.

In this paper, an efficient high-throughput FPGA implementation of the CCSDS-123 compression standard is proposed. The proposed compression core is capable of processing HSI cube sizes of various imagers. The core supports the majority of user-defined compression parameters proposed by the standard. The achieved throughput is higher in comparison with the state of the art implementations [14]–[18].

The paper is structured as follows: Section II presents a overview of the CCSDS-123 standard. The proposed hardware implementation is described in Section III. Results in terms of logic utilization are presented for a variety of compression parameters in Section IV. In addition, performance comparison with the state of the art is described. Finally, the conclusions are given in Section V.

II. Background

The CCSDS-123 standard [13] is a data compression algorithm for multispectral and hyperspectral images. In this section, detailed algorithm description is given for identifying data dependencies affecting the performance.

A HSI cube is a three-dimensional array of size (N_x, N_y, N_z) with sample $s_{z,y,x}$ where z indicates the spectral band and (x, y) are spatial coordinates. The data samples may be also described by the pair of indices (z, t) where $t = y \cdot N_x + x$. In a push-broom imager, the dimensions x and y correspond to scanned line (cross-track) direction and flight (along-track) direction, respectively. A frame F_y is the set of sample values for one y coordinate. The prediction stage computes estimates of each component based on samples in 3D space around the processed sample. The difference between the estimate and the actual component value, prediction residual, is then encoded by the entropy encoder. Cube size, dynamic range, output word size, sub-frame interleaving depth, sample type, scanning order and entropy encoder type are first encoded in the bitstream. The dynamic range D defines bit resolution of a sample within the range $[2, 16]$, whereas the output word size B is within the range $1 \leq B \leq 8$ bytes. The CCSDS-123 standard supports Band Interleaved (BI) and Band Sequential (BSQ) encoding orders for defining the arrangement of encoded samples in the input bitstream. The BI ordering reads the image in (y, x, z) order, whereas BSQ ordering reads the image band by band - (z, y, x). In BI mode, the parameter sub-frame interleaving depth M, defined in the range $(1, N_z)$, partitions frame F_y into sub-frames. Each sub-frame contains M consecutive spectral bands. Special cases of BI ordering are Band Interleaved by Pixel (BIP) for $M = 1$ and Band Interleaved by Line (BIL) for $M = N_z$.

A. Prediction stage

Prediction of the image sample $s_{z,y,x}$ is computed based on values of nearby predicted samples in the current spectral band and in P preceding bands.

1) Local sum and differences: The local sum $\sigma_{z,y,x}$ is a weighted sum of adjacent samples to sample $s_{z,y,x}$ within the spectral band z. The neighbor-oriented local sum is computed as follows:

$$\sigma_{z,y,x} = s_{z,y,x-1} + s_{z,y-1,x-1} + s_{z,y-1,x} + s_{z,y-1,x+1}, \quad (1)$$

for $y > 0$ and $0 < x < N_X - 1$, whereas sample predictions at the edges are given as:

$$\sigma_{z,y,x} = \begin{cases} 4s_{z,y,x-1}, & y = 0, x > 0 \\ 2(s_{z,y-1,x} + s_{z,y-1,x+1}), & y > 0, x = 0 \\ s_{z,y,x-1} + s_{z,y-1,x-1} + 2s_{z,y-1,x}, & x = N_x - 1. \end{cases} \quad (2)$$

The central local differences for $P+1$ spectral bands are then defined as:

$$d^k_{z,y,x} = 4 \cdot (s_{z-k,y,x}) - \sigma_{z-k,y,x} \quad (3)$$

where $k = 0, \dots, P$ is the distance of the previously processed spectral band from the currently processed band z. Reduced prediction mode contains only central local differences $d^k_{z,y,x}$, whereas full mode includes in addition the directional differences:

$$d^N_{z,y,x} = \begin{cases} 4 \cdot s_{z,y-1,x} - \sigma_{z,y,x}, & y > 0 \\ 0, & y = 0 \end{cases} \quad (4)$$

$$d^{NW}_{z,y,x} = \begin{cases} 4 \cdot s_{z,y-1,x-1} - \sigma_{z,y,x}, & x > 0, y > 0 \\ 4 \cdot s_{z,y-1,x} - \sigma_{z,y,x}, & x = 0, y > 0 \\ 0, & y = 0 \end{cases} \quad (5)$$

$$d^W_{z,y,x} = \begin{cases} 4 \cdot s_{z,y,x-1} - \sigma_{z,y,x}, & x > 0, y > 0 \\ 4 \cdot s_{z,y-1,x} - \sigma_{z,y,x}, & x = 0, y > 0 \\ 0, & y = 0. \end{cases} \quad (6)$$

where labels 'N', 'W' and 'NW' define position of neighboring samples with respect to the currently processed sample within the band z. The computed differences are stored in local difference vector $U_z(t)$ with size $C_z = P + 3$ for full mode and $C_z = P$ for reduced prediction mode.

2) Prediction: An adaptive linear prediction computes the integer scaled predicted sample value $\hat{s}^*_z(t)$ as follows:

$$\hat{s}^*_z(t) = \frac{2^{\Omega} \cdot \sigma_z(t) + W_z^T(t) \cdot U_z(t)}{2^{\Omega+1}} \quad (7)$$

where $W_z(t)$ is a weight vector with $(\Omega+3)$-bit signed integer coefficients, and resolution Ω is within range $[4, 19]$. The rounded $\tilde{s}_{round}$ version of scaled predicted sample value is given as:

$$\tilde{s}_{round}(t) = \left\lfloor \frac{\text{mod}^*_R \left[\hat{d}_z(t) + 2^{\Omega}(\sigma_z(t) - 4s_{\text{mid}})\right]}{2^{\Omega+1}} \right\rfloor, \quad (8)$$

where $\bmod{}^*_R[x] = ((x + 2^{R-1}) \mod 2^R) - 2^{R-1}$ and the range of register size R is $[\max(32, D + \Omega + 2), 64]$. A weighted average of differences $\hat{d}_z(t)$ is defined as $\hat{d}_z(t) = W_z(t)^T U_z(t)$. The final scaled predicted sample value $\tilde{s}_z$, for $t > 0$ is clipped version of $\tilde{s}_{round}$:

$$\tilde{s}_z(t) = \text{clip}\left(\tilde{s}_{round}(t) + 2s_{mid} + 1, (2s_{min}, 2s_{max} + 1)\right), \quad (9)$$

whereas for $t = 0$, it is given by:

$$\tilde{s}_z(t) = \begin{cases} 2s_{z-1}(t), & t = 0, P > 0, z > 0 \\ 2s_{\text{mid}}, & t = 0 \text{ and } (P = 0 \text{ or } z = 0). \end{cases} \quad (10)$$

The lower and the upper sample value limit and the mid-range sample value $[s_{\text{min}}, s_{\text{max}}, s_{\text{mid}}]$ are defined as $[0, 2^D - 1, 2^{D-1}]$ and $[-2^{D-1}, 2^{D-1} - 1, 0]$ for unsigned and signed samples, respectively. The parameter $\tilde{s}_z$ is then re-normalized to the range of the input sample (D-bit quantity):

$$\hat{s}_z(t) = \left\lfloor \frac{\tilde{s}_z(t)}{2} \right\rfloor. \quad (11)$$

3) Weight update: The standard defines default and custom weight initialization for selecting the initial weight vector W_z. Default values of vector coefficients ω_z^k are defined as:

$$\omega_z^k = \begin{cases} \frac{7}{8} 2^{\Omega}, & k = 1 \\ \lfloor \frac{1}{8} \omega_z^{k-1} \rfloor, & k = 2, \ldots, P. \end{cases} \quad (12)$$

Initial directional weight coefficients are set to zero. The weights are dynamically updated based on prediction error $e_z(t) = 2s_z(t) - \tilde{s}_z(t)$ as follows:

$$\Delta W_z(t) = \left\lfloor \frac{1}{2} \left(\text{sgn}^+[e_z(t)] \cdot 2^{-\rho(t)} \cdot U_z(t) + 1\right) \right\rfloor. \quad (13)$$

Weight update scaling exponent ρ, controlling convergence speed, is given as:

$$\rho(t) = \text{clip}\left(\nu_{min} + \left\lfloor \frac{t - N_x}{t_{inc}} \right\rfloor, \{\nu_{min}, \nu_{max}\}\right) + D + \Omega. \quad (14)$$

Computation of ρ includes parameters ν_{min}, ν_{max} and weight update change interval t_{inc} which determine the rate at which the predictor adapts to the image content statistics. The ranges of ν_{min} and ν_{max} are $-6 \leq \nu_{min} \leq \nu_{max} \leq 9$, whereas the range of t_{inc} is $(2^4, 2^{11})$. The exponent ρ is incremented at regular intervals determined by the value t_{inc} and its initial and final value are defined as $\nu_{min} + D - \Omega$ and $\nu_{max} + D - \Omega$, respectively. The final update step for pixel at position $t + 1$ is given by:

$$W_z(t + 1) = \text{clip}\left(W_z(t) + \Delta W_z(t), \{\omega_{\text{min}}, \omega_{\text{max}}\}\right) \quad (15)$$

where $\omega_{\text{min}} = -2^{\Omega+2}$ and $\omega_{\text{max}} = 2^{\Omega+2} - 1$.

4) Residual mapping: The prediction residual $\Delta_z(t)$ is the difference between the actual sample value $s_z(t)$ and the predicted sample $\hat{s}_z(t)$, $\Delta_z(t) = s_z(t) - \hat{s}_z(t)$. The residual mapping converts the signed predicted residuals to a D-bit unsigned integer, producting the mapped prediction residual $\delta_z(t)$:

$$\delta_z(t) = \begin{cases} |\Delta_z(t)| + \theta_z(t), & |\Delta_z(t)| > \theta_z(t), \\ 2|\Delta_z(t)|, & 0 \leq (-1)^{\tilde{s}_z(t)} \Delta_z(t) \leq \theta_z(t), \\ 2|\Delta_z(t)| - 1, & \text{otherwise}, \end{cases} \quad (16)$$

where

$$\theta_z(t) = \min\{\hat{s}_z(t) - s_{\text{min}}, s_{\text{max}} - \hat{s}_z(t)\}. \quad (17)$$

B. Entropy Encoding

The CCSDS-123 standard defines a sample-adaptive encoding and a block-adaptive entropy encoder. The focus of the paper is on sample-adaptive encoder which uses a Golomb 2^n variable-length binary coding technique (GPO2). The maximum produced code word length is $U_{max} + D$ for each incoming mapped residual $\delta_z(t)$, where the unary length limit U_{max} is defined in the range $[8, 32]$.

The generation of code words relies on tracking of the average values of the residuals in each band. The average value computation is performed by accumulating the sample values in an accumulator $\Sigma_z(t)$ and dividing the result by the number of processed samples, $\Gamma(t)$. The ratio $\Sigma_z(t)/\Gamma(t)$ is an estimate of the mean mapped prediction residual within a spectral band. The accumulator $\Sigma_z(t)$ is defined as:

$$\Sigma_z(t) = \Sigma_z(t-1) + \delta_z(t-1), \text{ for } \Gamma(t-1) < 2^{\gamma^*} - 1, \quad (18)$$

and counter $\Gamma(t)$ is incremented for each sample:

$$\Gamma(t) = \Gamma(t - 1) + 1, \ \Gamma(t - 1) < 2^{\gamma^*} - 1. \quad (19)$$

The re-scaling counter size γ^* determines the maximum value of counter $\Gamma_z(t)$. The initial value of the counter and the accumulator are given as:

$$\Gamma(1) = 2^{\gamma_0}, \quad (20)$$

$$\Sigma_z(1) = \left\lfloor \frac{1}{2^7} \left(3 \cdot 2^{K+6} - 49\right) \Gamma(1) \right\rfloor. \quad (21)$$

The initial count exponent γ_0 is defined in the range $[1, 8]$, whereas the range of accumulator initialization constant K is in the range $[0, D-2]$. After reaching $\Gamma(t-1) = 2^{\gamma^*} - 1$, the

accumulator and the counter are re-scaled in order to increase impact of more recent sample values as follows:

$$\Gamma(t) = \left\lfloor \frac{\Gamma(t-1)+1}{2} \right\rfloor, \tag{22}$$

$$\Sigma_z(t) = \left\lfloor \frac{\Sigma_z(t-1)+\delta_z(t-1)+1}{2} \right\rfloor. \tag{23}$$

The parameter $k_z(t)$ is defined as the largest non-negative integer $k_z(t) = \max\{1 \leq i \leq D-2\}$ satisfying the following inequality:

$$\Gamma(t)2^i \leq \Sigma_z(t) + \left\lfloor \frac{49}{2^7}\Gamma(t) \right\rfloor, \tag{24}$$

otherwise $k_z(t) = 0$, for $2\Gamma(t) > \Sigma_z(t) + \left\lfloor \frac{49}{2^7}\Gamma(t) \right\rfloor$.

For an input residual $\delta_z(t)$ and a given U_{max}, code word generation involves computation of the quotient and residual pair (u_z, r_z) as follows:

$$u_z(t) = \left\lfloor \frac{\delta_z(t)}{2^{k_z(t)}} \right\rfloor, r_z = \delta_z(t) \mod 2^{k_z(t)}. \tag{25}$$

The parameters $k_z(t)$, $r_z(t)$ and $u_z(t)$ are finally used in code word generation as follows:

- If $t = 0$, the code word is the D-bit unsigned integer binary representation of $\delta_z(t)$.
- If $t > 0$ and $u_z(t) < U_{\text{max}}$, the code word consists of $u_z(t)$ '0's followed by '1' and the $k_z(t)$ least significant bits of $\delta_z(t)$ corresponding to the remainder $r_z(t)$.
- If $t > 0$ and $u_z(t) \geq U_{\text{max}}$, the code word consists of U_{max} '0's, followed by the D-bit unsigned integer binary representation of $\delta_z(t)$.

III. Proposed CCSDS-123 Encoder Implementation

The important aspects of the hardware implementation performance are throughput, algorithmic complexity and memory/buffering requirements. The computational complexity of the CCSDS-123 algorithm is considered low, so focus is on data dependency identification affecting storage requirements and throughput. Due to the causal nature of the prediction stage, the amount of data to be stored varies with parameter P and with the processing order. In particular, the sample ordering affects storage requirements for local differences and weight vectors, data dependencies between various stages, throughput and susceptibility to parallel implementation.

A. Memory Requirements

1) Neighboring samples: When encoding a sample $s_z(t)$, it is required to store locally the neighboring samples used to compute the local sums and differences. Sample distance is defined as the number of clock cycles from the cycle samples are first streamed in until the cycle when these samples are used as neighbors. Table I shows sample distances required for storing neighbors (W, NW, N, NW) for different sample orderings.

TABLE I: Sample distance between current sample and neighboring samples [18]

Order	W	NE	N	NW
BIP	N_z	$(N_x-1)N_z$	N_xN_z	$(N_x+1)N_z$
BIL	1	N_xN_z-1	N_xN_z	N_xN_z+1
BSQ	1	N_x-1	N_x	N_x+1

2) Weight vectors and accumulators: Each band uses its own weight vector in the prediction stage and accumulator in the encoding stage. In BIP ordering, sample $s_z(t+1)$ is processed N_z samples after $s_z(t)$ implying the need for storing the weight vector and accumulator for each band. In BIL ordering, sample $s_z(t+1)$ is processed after sample $s_z(t)$ within the same line of the image. When the end of the line is reached, it is required to store the weight vector and accumulator until processing for the same band starts in the next line. This implies the need for storing the weight vector and the accumulator for each band also in BIL ordering. In BSQ ordering, only one weight vector and accumulator are stored since $s_z(t+1)$ is processed after $s_z(t)$.

3) Previous local differences: For each sample there is a need to store the central local differences computed in P previous bands of the same pixel. The requirements for storing the local differences are summarized in Table II. In BIP ordering, differences are computed during the P previous cycles. In BIL ordering, the local differences are computed for the whole line of width N_x in the P previous bands. The sample distance between the least recent local difference required to be stored and current sample is $P \times N_x$ cycles. In BSQ ordering, the frames are processed sequentially, implying that the largest sampling distance is $P \times N_x \times N_y$ cycles.

TABLE II: Sample distance between the current sample and the sample in the same position in the previous bands [18]

Order	$z-1$	$z-2$	...	$z-P$
BIP	1	2		P
BIL	N_x	$2N_x$		PN_x
BSQ	N_xN_y	$2N_xN_y$		PN_xN_y

A summary of required memory resources in different stages of the compression algorithm for various sample orderings is given in Table III. The BSQ ordering requires large amounts of memory, even for small values of P, for storing the set of $N_y \times N_x$ local difference vectors. Thus, storing local differences in BSQ ordering for a cube of size $512 \times 2000 \times 128$ with 16-bit samples and with a default value of $P = 3$, requires 6.95 MB exceeding available block RAM capacity in a mid-range Zynq-7020 FPGA. For larger P, available block RAM capacity is quickly exceeded even in high-end devices.

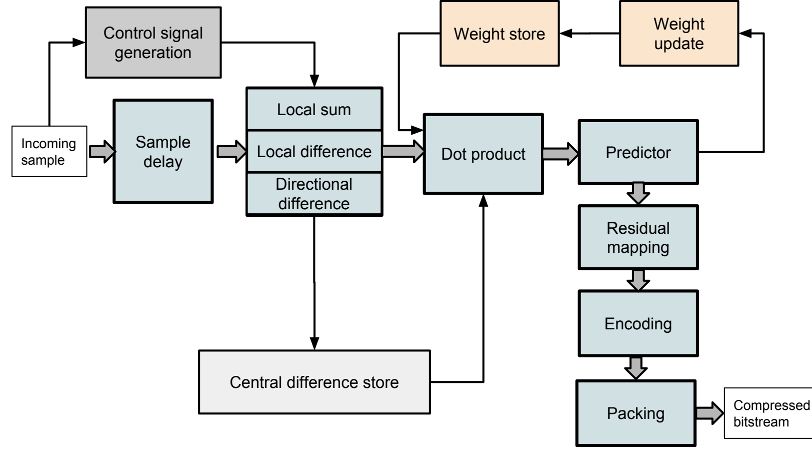


Fig. 1: Overview of the CCSDS-123 implementation with BIP scanning order.

TABLE III: Memory resource requirements for sample orderings

Order	BIP	BIL	BSQ
Samples	$(N_x+1)N_zD$	$(N_xN_z+1)D$	$(N_x+1)D$
Local diff.	$P(D+3)$	$PN_x(D+3)$	$PN_xN_y(D+3)$
Weights	$N_zC_z(\Omega+3)$	$N_zC_z(\Omega+3)$	$C_z(\Omega+3)$
Accum.	$N_z(D+\gamma^*)$	$N_z(D+\gamma^*)$	$D+\gamma^*$

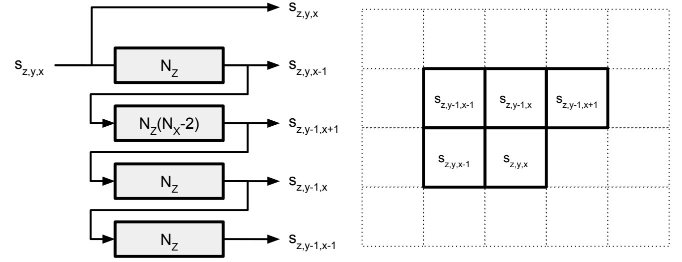


Fig. 2: Sample delay module with chain of FIFOs

B. Data Dependencies

In BIP ordering, the updated weight vectors for all the bands need to be stored. On the other side, the BIP ordering is a good candidate for parallel processing since it is not required to complete prediction of a sample $s_z(t)$ before starting prediction of the next sample $s_{z+1}(t)$. In BIL and BSQ ordering, the prediction and weight update operations have to be completed before starting the prediction of the sample since updated weight vector $W_z(t+1)$ is used for computing $\tilde{d}_z(t+1)$.

C. Architecture overview

In a typical HSI system, samples are either streamed directly from camera sensor of an imager or from the external memory to the accelerator. Based on the previous analysis in terms of memory requirements, possibilities for parallel implementation, and taking into consideration native layout of the HSI image in memory, the BIP ordering is chosen as sample ordering. An overview of the proposed CCSDS-123 implementation is shown in Figure 1.

1) Sample delay: This module delays incoming samples so that the current and the neighboring samples in positions (W, NW, N, NE) are available. This is achieved by chaining FIFO instances of particular lengths, N_z and $N_z \times (N_x-2)$ as shown in Figure 2. Given the sample distance of N_z between sample $s_{z,y,x}$ and its neighbor $s^W_{z,y,x-1}$, sample $s_{z,y,x-1}$ is pushed into a FIFO of length N_z in order to be present at the FIFO's output in the same cycle $s_{z,y,x}$ is streamed in. Sample $s_{z,y-1,x-1}$ is present at the output of the last FIFO $(N_x+1)\cdot N_z$ cycles after its arrival to the sample delay module.

2) Control signal generation: In addition to the sample streaming between the modules in the pipeline, control signals such as x, y and z coordinates of the current sample are also produced by using a set of counters. The additional signals based on the position of the sample are generated as follows:

- flags indicating if the sample is in the first line, in the first pixel of a line, the last pixel of a line, or the last sample in the last pixel,
- weight update scaling exponent $\rho(t)$, used in the weight update in Eq. (14).

3) Local sum and difference computation: These operations are performed in a three-stage pipeline as presented in Fig. 3. The first two stages compute the local sum, whereas the last stage computes the local central difference and the directional differences. The local sum defined by Eq. (2) is split across two pipeline stages in order to reduce delay. In the first stage, expressions $term_1$ and $term_2$ are computed as follows:

$$term_1 = \begin{cases} s_{z,y,x-1} + s_{z,y-1,x-1}, & y>0, 0<x<N_X-1 \\ 4s_{z,y,x-1}, & y=0, 0<x<N_X-1 \\ 2s_{z,y-1,x}, & y>0, x=0 \\ s_{z,y,x-1} + s_{z,y-1,x-1}, & y>0, x=N_X-1 \end{cases} \tag{26}$$

$$term_2 = \begin{cases} s_{z,y-1,x} + s_{z,y-1,x+1}, & y > 0, 0 < x < N_X - 1 \\ 0, & y = 0, 0 < x < N_X - 1 \\ 2s_{z,y-1,x+1}, & y > 0, x = 0 \\ 2s_{z,y-1,x}, & y > 0, x = N_X - 1, \end{cases} \quad (27)$$

whereas the local sum is then produced by summation of $term_1$ and $term_2$.

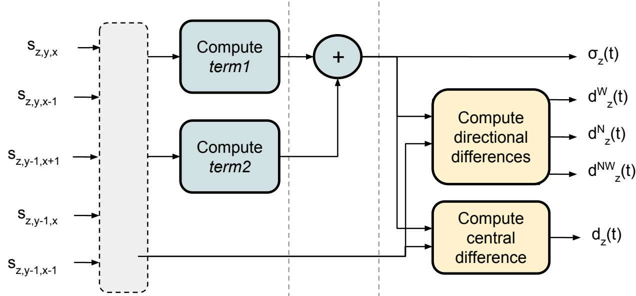


Fig. 3: Local sum, local difference and central difference calculations

4) Central difference store: The central difference store keeps the local central differences $d_{z-k,y,x}$ computed in the previous $k = 1, \ldots, P$ bands in a shift register shown in Fig. 4. The computed local central difference $d_{z,y,x}$ is stored in the first register, while the previous contents is shifted by one position. When $z = N_z - 1$, the contents of the shift register is set to zero so that local differences from the previous pixel are not used for the prediction of a new pixel.

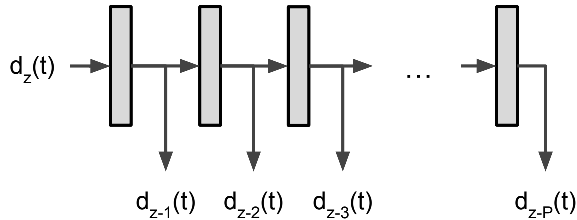


Fig. 4: Central difference store

5) Dot product: The dot product is performed in a pipeline with variable depth depending on C_z. In the first stage, each element in $U_z(t)$ is multiplied with the corresponding element in $W_z(t)$ followed by a tree of adders as presented in Fig. 5. The number of stages in the adder tree is $S = \lceil \log_2(C_z) \rceil$. The adder tree is implemented as follows: $s(2^S + i) = s(2i) + s(2i+1)$ for $0 \leq i \leq 2^S - 2$, whereas the initial $2^S - 1$ indices are reserved for the multiplication operations $s(i) = u_i \cdot \omega_i$ for $0 \leq i \leq 2^S - 1$. The result of the dot product operation is $\hat{d}_z = s(2^{S+1} - 2)$.

6) Predictor: The predictor computes the scaled predicted sample $\tilde{s}_z(t)$ defined in Eq. (9) by splitting the operation across two-pipeline stages. In the first stage, fraction $temp_1$ in the numerator is computed:

$$temp_1 = \text{mod}^*_R \left[\hat{d}_z(t) + 2^{\Omega}(\sigma_z(t)) \right],$$

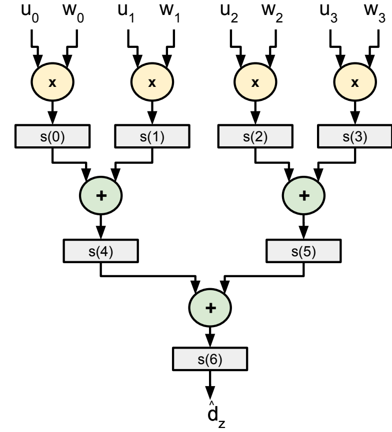


Fig. 5: Dot product for $C_z = 4$

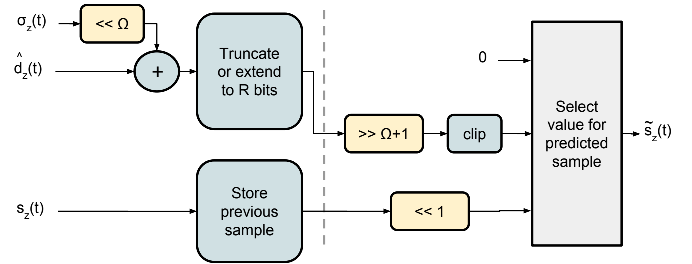


Fig. 6: Implementation of prediction stage

where multiplication by a power of 2 is implemented by shifting. The term $4s_{\text{mid}}$ is set to zero since implementation uses signed numbers. The scaled predicted sample $\tilde{s}_z(t)$ is then computed as follows:

$$\tilde{s}_z(t) = \text{clip}\left(\left\lfloor \frac{temp_1}{2^{\Omega+1}} \right\rfloor + 1, \{2s_{\text{min}}, 2s_{\text{max}} + 1\} \right), \text{ for } t > 0.$$

For $t = 0$, $P > 0$ and $z > 0$, the sample $s_{z-1}(t)$ is used in the computation of $\tilde{s}_z(t)$, otherwise $\tilde{s}_z(t) = 0$. A multiplexer finally chooses the result between three defined cases based on status of the flags, parameter P and band z.

7) Weight update: The weight update from Eq. (15) is performed in three-pipeline stages as presented in Fig. 7. The parameter $\rho(t)$ computed in the control unit is sent along with the input sample. The first stage computes the component-wise product $temp_1 = \text{sgn}^+[e_z(t)] \cdot U_z(t)$, equivalent to the change of the sign of the component depending on whether $e_z(t)$ is positive or negative. Since $e_z(t) = 2s_z(t) - \hat{s}_z(t)$, the condition $e_z(t) < 0$ is equivalent to the inequality $2s_z(t) < \hat{s}_z(t)$. This is implemented by choosing between vectors $U_z(t)$ and $-U_z(t)$ based on the result of the comparison. The second stage computes weight update factor $\Delta W_z(t)$ from Eq. (13). The parameter $\rho(t)$ has a small range of possible values (at most -6 to 9), so the expression $2^{-\rho(t)} temp_1$ for each value of $\rho(t)$ are computed in parallel. The operations are left or right shifts depending on the sign of $\rho(t)$. A multiplexer chooses the value of expression $2^{-\rho(t)} temp_1$ based on the computed value of $\rho(t)$. The selected output vector of the multiplexer is added to 1 and shifted to the right. The final stage is the

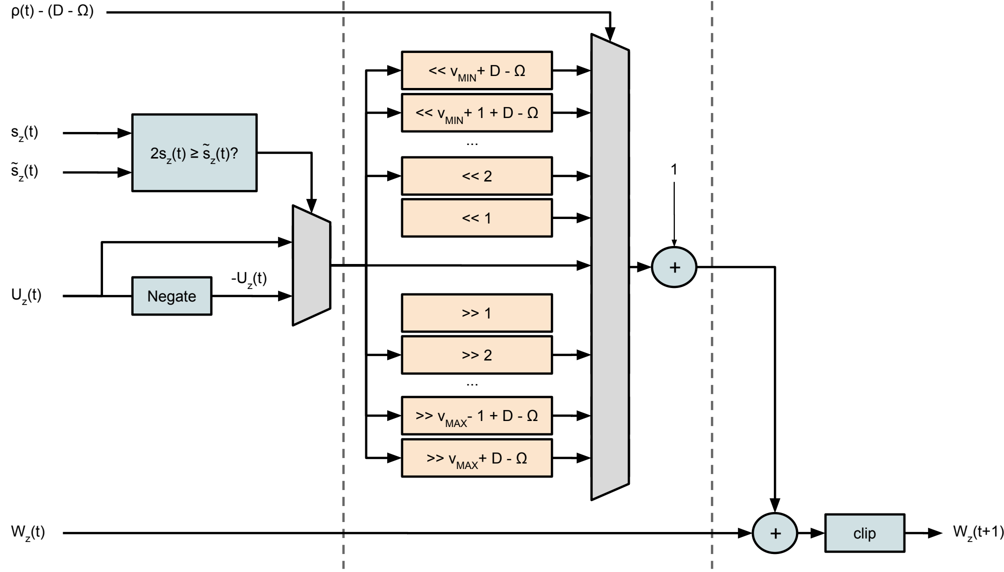


Fig. 7: Weight update stage

update operation of weight vector $W_z(t+1)$ computed by the sum of the weight vector $W_z(t)$ and the weight update factor $\Delta W_z(t)$.

8) Weight store: The weight store keeps the weight vectors in between operations of weight vector updating for band z and reading the weights of the same band for the next sample. A dual port block RAM is used to read a weight vector from one address and simultaneously write the weights at another address. The band z of the incoming sample is used as an address to read the corresponding weights from the weight store. When updating weights, the coordinate z of the new weight vector is used as a write address.

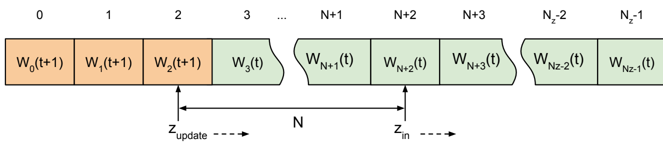


Fig. 8: Weight store data ordering

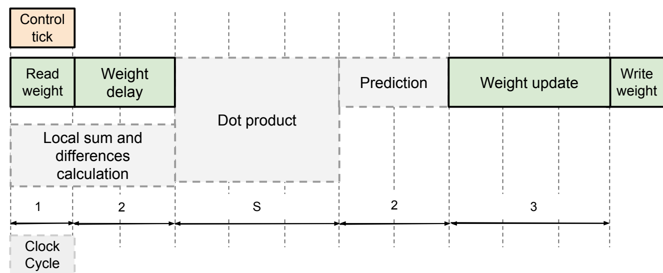


Fig. 9: Time diagram of weight updating and storing pipeline

Fig. 8 illustrates situation when weight vector $W_{z_{in}}(t)$ in band z_{in} is read and the weight vector $W_{z_{update}}(t+1)$ in band z_{update} is updated. The relation between the band z_{update} and the band z_{in} of the currently processed sample is given as:

$$z_{update} = (z_{in} - N) \bmod N_z \tag{28}$$

where $N = 1 + 2 + S + 2 + 3$ is a delay which equals to the number of pipeline stages from reading operation in weight store to the final stage of weight update operation. The value of N depends on the number of stages S in tree adder of the dot product module. The time diagram of the described pipeline in Fig. 9 shows that read operation of a weight vector is performed in parallel with the local sum and difference computations. The weight vector from the block RAM is delayed by two cycles, such that the local difference vector and weight vector for the same sample arrive simultaneously at the dot product module.

9) Residual mapping: The residual mapping is computed in two-pipeline stages as presented in Fig. 10. The first stage computes $\Delta_z(t)$ and $\theta_z(t)$ from Eq. (17). The mapped

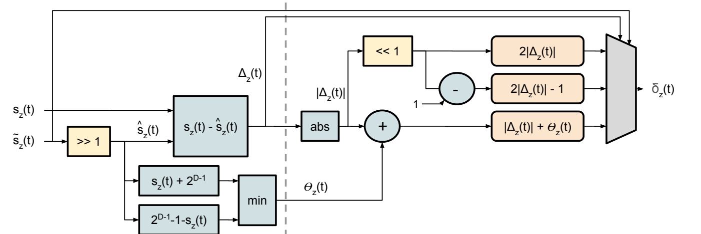


Fig. 10: Implementation of residual mapping stage

prediction residual $\delta_z(t)$, defined in Eq. (16) is computed then by using a multiplexer to select between three cases. The inequality $0 \leq (-1)^{\tilde{s}_z(t)} \Delta_z(t) \leq \theta_z(t)$ holds in one of the cases. The expression $(-1)^{\tilde{s}_z(t)}$ is equivalent to 1 when $\tilde{s}_z(t)$ is even, and -1 when $\tilde{s}_z(t)$ is odd. Then, the re-stated inequality given as $\tilde{s}_z(t)$ is even and $\Delta_z \geq 0$ or $\tilde{s}_z(t)$ is odd and $\Delta_z(t) \leq 0$

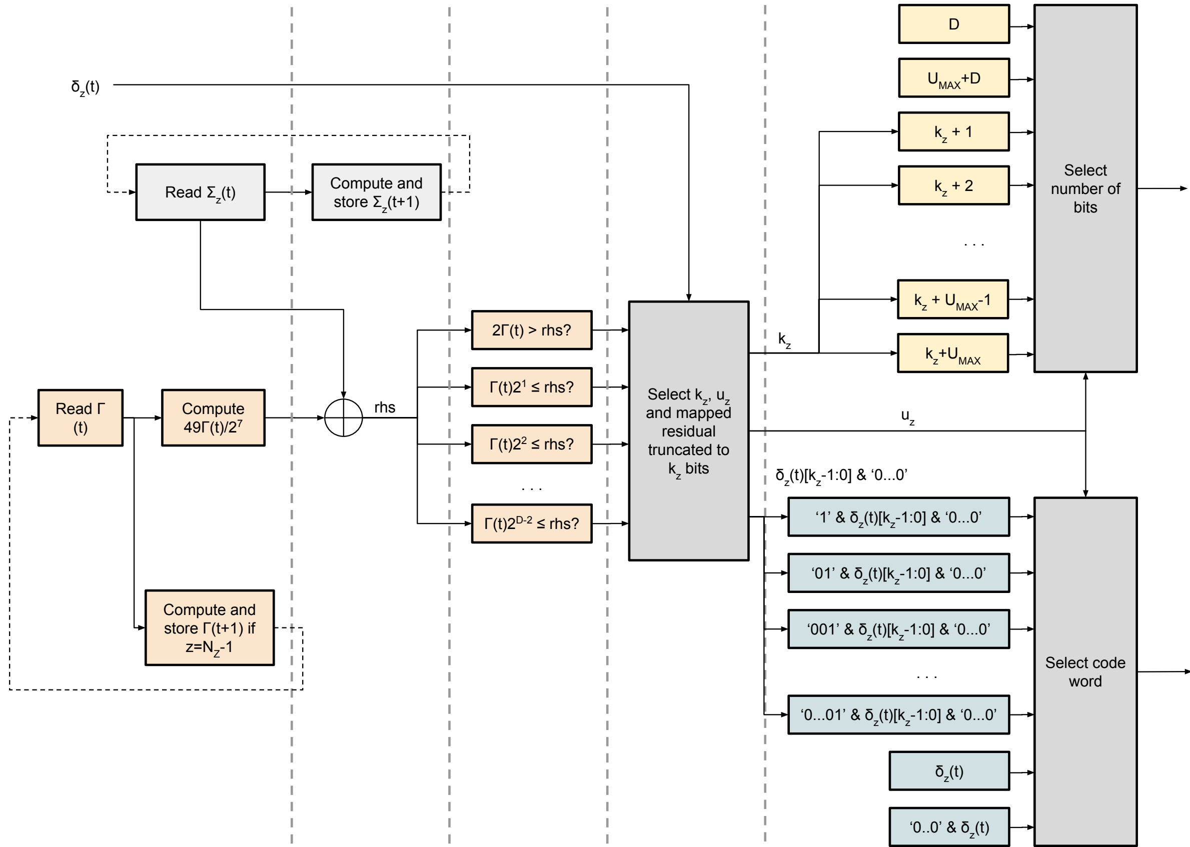


Fig. 11: Overview of sample adaptive encoder implementation

is implemented in hardware as checking whether the LSB of $\tilde{s}_z(t)$ is 0 or 1.

10) Encoding: The sample adaptive encoder implementation with five-stage pipeline is shown in Fig. 11. The first and second stages compute the right hand side (rhs) of Ineq. (24). In the third stage, each of the inequalities for $i = 1, \dots, D-2$ are evaluated in parallel. Then, parameters k_z and u_z are chosen based on the results from inequality evaluation by using priority encoder. The highest integer i, for which the inequality with left hand side $\Gamma(t)2^i$ holds, is chosen as the value for k_z, whereas expression $\delta_z(t)/2^i$ for chosen i is assigned to $u_z(t)$. In addition, a truncated version of $\delta_z(t)$ with the k_z least significant bits is created. The bits are right-shifted so that the most significant bits are taken from the truncated $\delta_z(t)$. In the last stage, the code word is generated based on the defined rules determined by values of U_{max}, $u_z(t)$ and position t. Both code word and its length are output of the encoding stage.

11) Bit packing: The bit packing module collects variable-length encoded words into packets of a given configurable output word size B. The packing operation is centered around two registers of the size B. The registers alternate between being the *current* register and *next* register. Incoming words from the encoder are stored in the current register. When the current register is full, leftover bits are put in the next register, and the data from the current register are sent to the output. In the following cycle, the registers switch roles. Due to the variable length of the incoming code words, the bit position in the current register for storing an incoming word can be any value ranging from the most significant bit $N-1$ to the least significant bit 0. Creation of N different candidates for register layout is done by creating words which consist of the i most significant bits of the current register followed by the $(U_{max} + D)$-bit padded input word for each $i \in [0, N-1]$. Selection of the correct candidate is performed by using a *write pointer* which keeps track of the first non-occupied bit position in the current register. Fig. 12 illustrates an example of the packing process. The left-most register is assigned to be the current register, and the incoming words in the first 4 cycles are stored into this register. The fifth word is larger than the remaining space in the current register, so the leftover bits are stored as the most significant bits of the next register. In the next clock cycle, the content of the current register is written to the output FIFO, and the next and current registers swap roles. The next input words are written to the newly assigned current register, until it gets full in the eighth cycle.

IV. Results

The factors affecting computational complexity and timing of hardware implementation of CCSDS-123 algorithm are size of the image to be compressed and the configuration parameters, in particular number of bands P for prediction and the sample ordering. The proposed architecture of the CCSDS-123 algorithm is described at RTL level using the VHDL language,

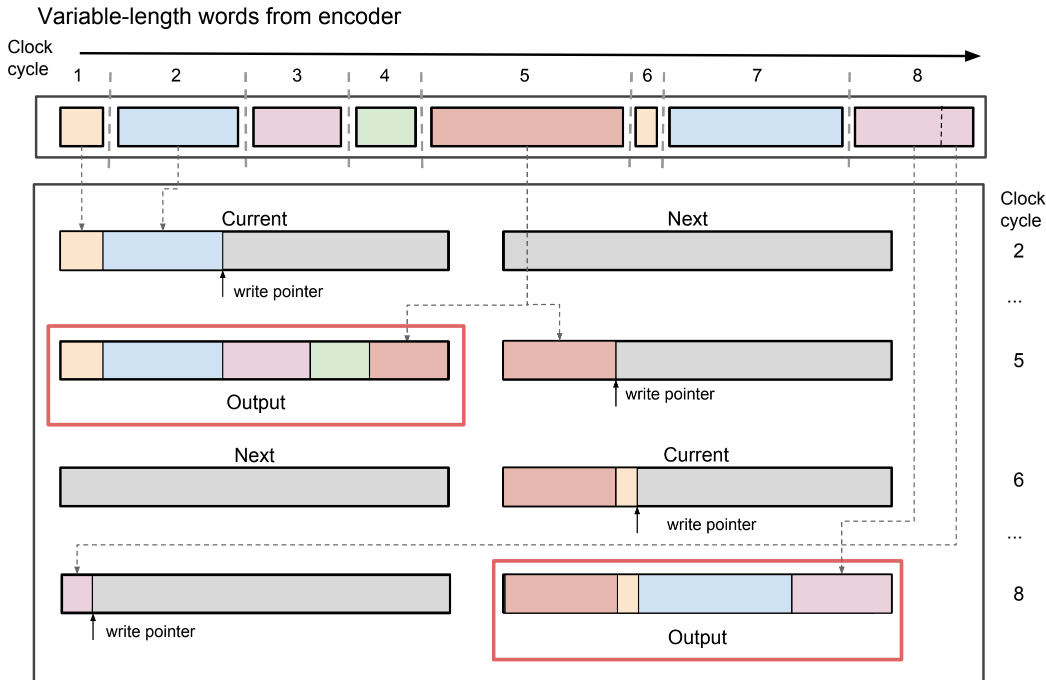


Fig. 12: Packing of variable length code words into fixed-size packets

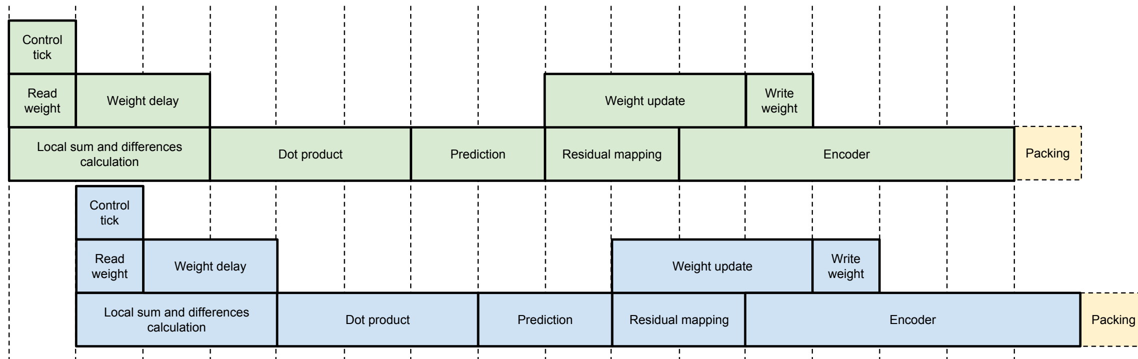


Fig. 13: Timing diagram of complete CCSDS-123 pipeline in the proposed implementation

whereas the Vivado tool is used for synthesis, implementation, power estimation, testing and verification of the proposed implementation on a Zedboard development board with a Zynq-7020 FPGA. The proposed hardware implementation supports the majority of the parameter settings defined by the compression standard, including full ranges of bit-depths, number of previous bands for prediction P and output word size B. The proposed hardware implementation supports both on the fly processing of one input sample per clock cycle and offline processing from the external memory. For on the fly processing, N_z must be larger than number of stages N to avoid data corruption. This is due to the fact that the sample $s_z(t+1)$ arrives N_z clock cycles after $s_z(t)$, and it takes N clock cycles from $s_z(t)$ arrives until the weight for $s_z(t+1)$ is stored. The block-adaptive entropy encoding and custom initialization of weight vectors are not supported. The chosen scanning order is BIP based on the memory requirement analysis performed in the previous section. Fig. 13 shows the complete pipeline of the proposed implementation. It is observed that the processing chain after a sample is input until the code word is generated takes 15 clock cycles. The packing stage takes a variable number of clock cycles depending on the size of encoded words and output word size B.

A. Logic Utilization Results

Table IV shows the default values of the CCSDS-123 parameters used in performance and logic utilization analysis. The cube sizes of multispectral and hyperspectral imagers - Moderate Resolution Imaging Spectroradiometer (MODIS) [19], Hyperion [20], Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) [21] and Hyperspectral Imager for the Coastal Ocean (HICO) [22] and their total resource utilization in terms of LUTs, registers, BRAM and DSP blocks of

TABLE IV: Default CCSDS-123 parameters used in logic utilization analysis

Parameter	Description	Value
D	Sample bit resolution	16
P	Number of previous bands to use in prediction	3
Ω	Weight resolution (weight bit resolution is $\Omega + 3$)	19
ν_{min}	Weight update scaling exponent initial parameter	-1
ν_{max}	Weight update scaling exponent final parameter	3
t_{inc}	Weight update scaling exponent change interval	2^6
R	Register size	64
U_{max}	Unary length limit	18
γ_0	Initial count exponent	1
γ^*	Re-scaling counter size	6
K	Accumulator initialization constant	3

the proposed CCSDS-123 implementation are presented in Table V.

Detailed block-level resource utilization for the HICO cube size is further elaborated. The area resources are dependent on a number of parameters. In particular, the number of bands used for prediction P defined within the range $[0, 15]$ can significantly affect the area utilization. The area utilization in terms of LUTs and registers for the modules dot product, predictor and weight update dependent on parameter P are shown in Table VI and Table VII, respectively. Both the number of LUTs and the number of registers scale linearly with P as illustrated in Fig. 14 and Fig. 15. The LUTs and register resources used in other modules which are not dependent on P are presented in Table VIII. For the default value of $P = 3$, 2952 LUTs and 2469 registers are used corresponding to 5.5% and 2.3% utilization on a Zynq-7020 FPGA, respectively. Block RAM utilization is shown in Fig. 16, for different values of P. For $P = 3$, 36 block RAMs are used corresponding to 26% of available block RAM of a Zynq-7020 FPGA. The area utilization in terms of LUTs and registers for various bit-depth D are also presented in Table IX and Table X, respectively.

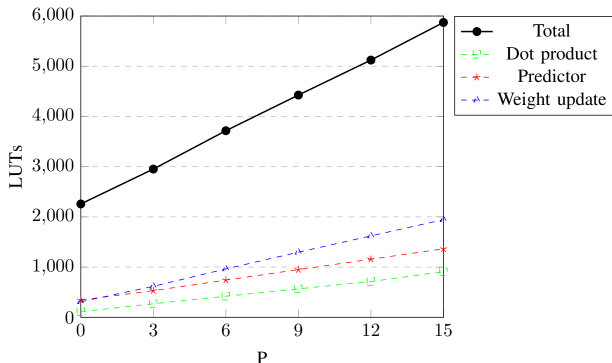


Fig. 14: LUT utilization for different values of P

The estimated maximum operating frequency for HICO size images is approximately 147 MHz, where the top ten critical paths of the design are in the first pipeline stage of the local sum computation. The pipeline implementation allows on the fly processing of one sample per clock cycle, achieving a throughput of 147 Msample/s. The power estimation performed in Vivado tool using default parameters and HICO cube size is 0.295 W.

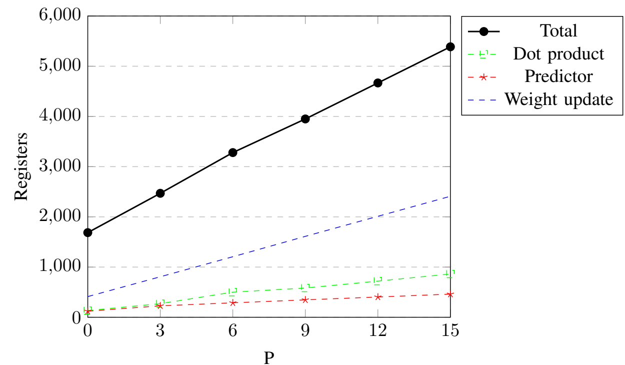


Fig. 15: Register utilization for different values of P

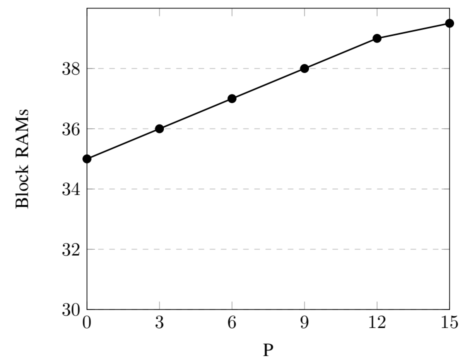


Fig. 16: Block RAM utilization for different number of previous bands P

B. Verification

Testing of the complete core was initially performed in simulations against the Emporda compressor [23] for a selection of parameters and for a number of edge cases with cubes in all three dimensions restricted to less than 100 elements to reduce simulation time. The core was tested on a set of special test pattern images produced by CCSDS for verifying that certain edge cases with overflows are handled correctly.

An automated verification system is also created comparing the compressed bitstream from the implemented design with the compressed bitstream from the Emporda software [23]. The proposed verification flow shown in Fig. 17 allows the design to be automatically checked for different sets of generic parameters yielding greater confidence in the robustness of the design.

The last verification stage for assessment of processing performance and cross-testing of the proposed FPGA implementation with Emporda reference software uses well-known hyperspectral datasets collected by imagers listed in Table V. The core was also tested on a ZedBoard development board by loading these cubes in BIP ordering into the external memory and by using Xilinx AXI DMA core [24] for streaming data to the proposed accelerator. The resulting bitstream was successfully compared to the expected compressed image bitstream.

TABLE V: Resource utilization for various multispectral and hyperspectral imagers

Imagers	D	N_x	N_y	N_z	LUTs	Registers	BRAM	DSP	$f_{\max}$
MODIS	12	1354	2030	17	2414	2303	14	7	154
Hyperion	12	256	3242	242	2457	2303	28	7	147
AVIRIS	16	680	512	224	3012	2528	84	6	147
HICO	16	512	2000	224	2952	2469	36	6	147

TABLE VI: LUT utilization for different number of previous bands P

P	Dot prod.	Predictor	Weight upd.	Total
0	111	344	310	2258
3	270	531	617	2952
6	420	741	964	3716
9	567	950	1297	4426
12	712	1157	1621	5123
15	909	1364	1945	5872

TABLE VII: Register utilization for different number of previous bands P

P	Dot prod.	Predictor	Weight upd.	Total
0	133	121	412	1686
3	272	228	807	2469
6	501	288	1206	3280
9	583	347	1612	3950
12	721	404	2011	4667
15	862	461	2410	5387

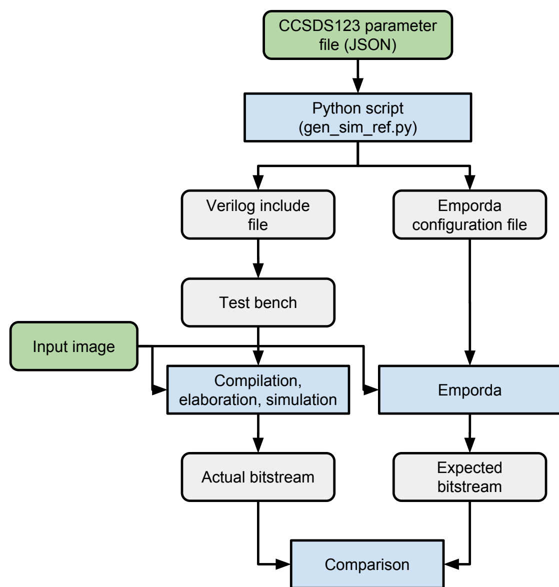


Fig. 17: Overview of automatic verification of the proposed design

C. Comparison

The proposed CCSDS-123 implementation is compared with recent state of the art FGPA implementations [14]–[18] as presented in Table XI. The sensor maximum data rates for AVIRIS, AVIRIS NG and HICO imagers are listed representing real-time constraint for data processing. The lack of implementation details of various stages in state of the art works limits the comparison to the maximum frequency, the throughput performance, power and to ability to perform on-the-fly compression by streaming one sample per clock cycle directly from the sensor without use of external memory. The data dependencies in BSQ ordering limits significantly the throughput in hardware implementation, whereas BIP ordering provides theoretical throughput of one compressed sample per clock cycle.

In HyLoC implementation [15], the maximum throughput depends on the selected configuration parameters. Parameter P affects the number of cycles per sample processing. The maximum operating frequency varies between 43.0 - 43.9 MHz for RTAX1000S FPGA, achieving throughput of 1.75 MSamples/s and 3.5 MSamples/s for $P = 15$ and $P = 3$, respectively. On Virtex-5 FPGA, the achieved throughput is 11.3 MSamples/s for $P = 3$. The advantage of this implementation is removal of internal storage requirements for neighboring samples for BSQ ordering. Instead of storing intermediate results, the data are re-calculated for each input sample requiring also neighboring samples to be fetched from the external memory. In this manner, the irregular data access patterns are introduced in communication with external memory.

SHyLoC implementation [16] consists of two IP cores - the CCSDS-121 and the CCSDS-123 supporting all three compression orderings. There is a need for different architectures of the CCSDS-123 standard for each ordering. The design is optimized for default value of P. The achieved throughput is higher in BIP than in other two orderings. The implementation requires the use of external memory.

The approach in [18] provides both the use of external memory as a buffer for storing data cube and on-the-fly data streaming directly from the sensor. The achieved throughput for Virtex-4 and Virtex-7 FPGAs are 23 MSamples/s and 48 MSamples/s, respectively. FPGA implementation proposed in [17] achieves throughput of 110 MSamples/s on Virtex-5 FPGA, streaming both current sample to process and its N and NW neighbors from the external memory. This disables further improvements which can be introduced by parallel implementation due to limited I/O capabilities of the supporting system.

The proposed implementation achieves throughput of 147 MSamples/s, with negligible initial latency of 15 clock cycles,

TABLE VIII: LUT and register utilization for modules independent of P

Sample delay	Local Diff	Diff. store	Control	Weight store	Res. mapping	Encoder	Packer
LUTs							
209	167	3	115	2	99	577	365
Registers							
122	337	57	47	132	71	250	146

TABLE IX: LUT utilization for different sample widths D

D	Dot prod.	Prediction	Weight upd.	Coder	Total
8	77	337	476	281	1892
10	83	414	513	355	2184
12	89	457	549	423	2439
16	270	531	617	577	2952

TABLE X: Register utilization for different sample widths D

D	Dot prod.	Prediction	Weight upd.	Coder	Total
8	74	139	682	168	1758
10	76	162	707	193	2143
12	82	184	755	213	2285
16	272	228	807	250	2469

after which the data is output after bit packing stage in B-byte words. On-the-fly compression of data streamed in BIP ordering directly from the sensor is supported. Processing of data stored in external memory in the BIP ordering is also supported by the use of AXI DMA core. The improved performance is achieved at the expense of increased BRAM utilization compared to [15]. However, the BRAM utilization corresponding to 1.327 Mbits occupies only 26% of the available resources on Zynq-7020 FPGA for $D = 16$ and $P = 3$.

V. Conclusion

In this paper an efficient FPGA implementation of the CCSDS-123 compression algorithm with BIP ordering is presented. The complete pipeline processing one input sample per clock cycle is described in detail and efficient hardware solutions are presented for a number of stages in the algorithm. The implementation is tested against available reference software and it is fully compliant with standard allowing user-defined parameter adjustments. The achieved throughput is 147 MSamples/s for 16-bit samples, which represents better performance than the proposed state of the art implementations.

Acknowledgement

This work was supported by the Research Council of Norway (RCN) through the MASSIVE project, grant number 270959, and the AMOS project, grant number 223254, as well as by the Norwegian Space Center.

References

[1] F. Sigernes, M. Syrjäsuo, R. Storvold, J. Fortuna, M. E. Grøtte, and T. A. Johansen, “Do it yourself hyperspectral imager for handheld to airborne operations,” *Optics express*, vol. 26, no. 5, pp. 6021–6035, 2018.

[2] S. Lopez, T. Vladimirova, C. Gonzalez, J. Resano, D. Mozos, and A. Plaza, “The promise of reconfigurable computing for hyperspectral imaging onboard systems: A review and trends,” *Proceedings of the IEEE*, vol. 101, no. 3, pp. 698–722, 2013.

[3] A. D. George and C. M. Wilson, “Onboard processing with hybrid and reconfigurable computing on small satellites,” *Proceedings of the IEEE*, vol. 106, no. 3, pp. 458–470, 2018.

[4] C. Rodarmel and J. Shan, “Principal component analysis for hyperspectral image classification,” *Surveying and Land Information Science*, vol. 62, no. 2, p. 115, 2002.

[5] A. Abrardo, M. Barni, and E. Magli, “Low-complexity predictive lossy compression of hyperspectral and ultraspectral images,” in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE, 2011, pp. 797–800.

[6] L. Santos, J. F. López, R. Sarmiento, and R. Vitulli, “FPGA implementation of a lossy compression algorithm for hyperspectral images with a high-level synthesis tool,” in *Adaptive Hardware and Systems (AHS), 2013 NASA/ESA Conference on*. IEEE, 2013, pp. 107–114.

[7] A. García, L. Santos, S. López, G. Callicó, J. F. López, and R. Sarmiento, “FPGA implementation of the hyperspectral Lossy Compression for Exomars (LCE) algorithm,” in *High-Performance Computing in Remote Sensing IV*, vol. 9247. International Society for Optics and Photonics, 2014, p. 924705.

[8] Consultative Committee for Space Data Systems, “Lossless Multispectral and Hyperspectral Image Compression - CCSDS 121.0-B-1,” 2012.

[9] N. Kranitis, I. Sideris, A. Tsigkanos, G. Theodorou, A. Paschalis, and R. Vitulli, “Efficient field-programmable gate array implementation of CCSDS 121.0-b-2 lossless data compression algorithm for image compression,” *Journal of Applied Remote Sensing*, vol. 9, no. 1, p. 097499, 2015.

[10] Consultative Committee for Space Data Systems, “Lossless Multispectral and Hyperspectral Image Compression - CCSDS 122.0-B-1,” Tech. Rep., 2005.

[11] L. Li, G. Zhou, B. Fiethe, H. Michalik, and B. Osterloh, “Efficient implementation of the CCSDS 122.0-B-1 compression standard on a space-qualified field programmable gate array,” *Journal of Applied Remote Sensing*, vol. 7, no. 1, p. 074595, 2013.

[12] Consultative Committee for Space Data Systems, “Lossless Multispectral and Hyperspectral Image Compression - CCSDS 123.0-B-1,” *Green Book*, 2015.

[13] CCSDS, “Lossless Multispectral and Hyperspectral Image Compression - CCSDS 123.0-B-1,” *Blue Book*, 2012.

[14] 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 *Adaptive Hardware and Systems (AHS), 2014 NASA/ESA Conference on*. IEEE, 2014, pp. 278–284.

[15] L. Santos, L. Berrojo, J. Moreno, J. F. López, and R. Sarmiento, “Multispectral and hyperspectral lossless compressor for space applications (HyLoC): A low-complexity FPGA implementation of the CCSDS 123 standard,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 2, pp. 757–770, 2016.

[16] University of Las Palmas de Gran Canaria. (2017) SHyLoC IP Core. [Online]. Available: http://www.esa.int/Our_Activities/Space_Engineering_Technology/Microelectronics/SHyLoC_IP_Core

[17] 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

TABLE XI: Performance comparison of CCSDS-123 implementations

Implementation	Order	P	D	Platform	f_{max} [MHz]	Throughput [Msamples/s]	Throughput [Mb/s]	Power [mW]	On-the-fly support
UAB Emporda [23]	All	15	16	Software(i-7 7500U)	-	4.928 [18]	78	-	No
AVIRIS Classic [21]	-	-	12	Sensor Maximum	-	1.7	20.4	-	-
AVIRIS-NG [21]	-	-	16	Sensor Maximum	-	1.92	30.72	-	-
HICO [25]	-	-	16	Sensor Maximum	-	4.78	76.5	-	-
Keymeulen et al [14]	BIP	3	13	Virtex-5	40	40	520	-	Yes
HyLoC, Santos et al [15]	BSQ	3	16	Virtex-4	134	11.2	179	1488	No
SHyLoC, Santos et al [16]	All	15	16	Virtex5	140	140	224O	-	No
Bascones et al [18]	BIP	15	16	Virtex-4	50	23.3	379	450	Yes
Bascones et al [18]	BIP	15	16	Virtex-7	50	47.6	760	450	Yes
Theodorou et al [17]	BIP	3	16	Virtex-5	110	110	1790	-	No
Proposed work	BIP	15	16	Zynq-7020	147	147	2350	295	Yes

Proceedings of the 5th International Workshop on On-Board Payload Data Compression, Frascati, Italy, 2016, pp. 28–29.

[18] D. Báscones, C. González, and D. Mozos, “FPGA Implementation of the CCSDS 1.2.3 Standard for Real-Time Hyperspectral Lossless Compression,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2017.

[19] NASA. Moderate Resolution Imaging Spectroradiometer (MODIS). [Online]. Available: https://modis.gsfc.nasa.gov/

[20] USGS. Hyperion - USGS EO-1. [Online]. Available: https://eo1.usgs.gov/sensors/hyperion

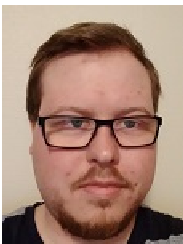
[21] NASA. Airborne Visible InfraRed Imaging Spectrometer (AVIRIS). [Online]. Available: https://aviris.jpl.nasa.gov/

[22] Naval Research Laboratory. Hyperspectral Imager for the Coastal Ocean (HICO). [Online]. Available: http://hico.coas.oregonstate.edu/

[23] GICI group, Universitat Autonoma de Barcelona, “Emporda software,” 2011. [Online]. Available: http://www.gici.uab.es

[24] Xilinx, “LogiCORE IP Product Guide, AXI DMA v7.1,” Tech. Rep., 2017.

[25] M. D. Lewis, R. Gould, R. Arnone, P. Lyon, P. Martinolich, R. Vaughan, A. Lawson, T. Scardino, W. Hou, W. Snyder *et al.*, “The hyperspectral imager for the coastal ocean (hico): Sensor and data processing overview,” in *OCEANS 2009, MTS/IEEE Biloxi-Marine Technology for Our Future: Global and Local Challenges*. IEEE, 2009, pp. 1–9.



Johan Fjeldtvedt received the B.Sc. degree in mathematics at The Norwegian University of Science and Technology (NTNU) in 2014. He is now working toward the M.Sc. degree in Electronics at NTNU. His research interest is high efficiency streaming and compression related to hyperspectral image processing.

Milica Orlandić received the B.Sc. and M.Sc. degrees in electrical engineering from University of Montenegro in 2007 and 2009. She received the Ph.D. degree from Norwegian University of Science and Technology (NTNU) in 2015 in electronics. She is currently holding a position of Postdoctoral Research Associate at NTNU. Her research interests include digital hardware design of video and image processing algorithms, hyperspectral imaging processing in satellite applications and reconfigurable systems on FPGA.

Tor Arne Johansen received the M.Sc. and Ph.D. degrees in electrical and computer engineering from the Norwegian University of Science and Technology (NTNU), Trondheim, Norway, in 1989 and 1994, respectively. From 1995 to 1997, he was with SINTEF, Trondheim, Norway, as a Researcher. He was an Associate Professor at NTNU in 1997 and a Professor in 2001. In 2002, he co-founded the company Marine Cybernetics AS, where he was a Vice President until 2008. He is currently a Principal Researcher with the Center of Excellence on Autonomous Marine Operations and Systems, and the Director of the Unmanned Aerial Vehicle Laboratory at NTNU. He has authored several articles in the areas of control, estimation, and optimization with applications in the marine, automotive, biomedical, and process industries.

Prof. Johansen was a recipient of the 2006 Arch T. Colwell Merit Award of the SAE.