Image compression on reconfigurable FPGA for the SO/PHI space instrument

D. Hernández Expósito, J. P. Cobos Carrascosa, J. L. Ramos Mas, M. Rodríguez Valido, D. Orozco Suárez, et al.

D. Hernández Expósito, J. P. Cobos Carrascosa, J. L. Ramos Mas, M. Rodríguez Valido, D. Orozco Suárez, J. Hirzberger, J. Woch, S. Solanki, J. C. del Toro Iniesta, "Image compression on reconfigurable FPGA for the SO/PHI space instrument," Proc. SPIE 10707, Software and Cyberinfrastructure for Astronomy V, 107072F (6 July 2018); doi: 10.1117/12.2312701

SPIE.

Event: SPIE Astronomical Telescopes + Instrumentation, 2018, Austin, Texas, United States

Image compression on reconfigurable FPGA for the SO/PHI space instrument

D. Hernández Expósito*[a], J.P. Cobos Carrascosa[a], J.L. Ramos Mas[a], M. Rodríguez Valido*[b], D. Orozco Suárez[a], J. Hirzberger*[c], J. Woch[c], S. Solanki[c], J.C. del Toro Iniesta[a]

[a]Instituto de Astrofísica de Andalucía (IAA-CSIC), Apartado 3004, 18080 Granada, Spain; [b]Universidad de La Laguna, Apartado 456, 38200 San Cristóbal de La Laguna, Spain; [c]Max Planck Institute for Solar System Research, Justus-von-Liebig-Weg 3, Göttingen, Germany

ABSTRACT

In this paper we present a novel FPGA implementation of the Consultative Committee for Space Data Systems Image Data Compression (CCSDS-IDC 122.0-B-1) for performing image compression aboard the Polarimetric and Helioseismic Imager instrument of the ESA's *Solar Orbiter* mission. This is a System-On-Chip solution based on a light multicore architecture combined with an efficient ad-hoc Bit Plane Encoder core. This hardware architecture performs an acceleration of ~30 times with respect to a software implementation running into space-qualified processors, like LEON3. The system stands out over other FPGA implementations because of the low resource usage, which does not use any external memory, and of its configurability.

Keywords: Image compression, CCSDS-IDC 122.0

1. INTRODUCTION

The *Solar Orbiter* (SO) spacecraft of the European Space Agency will orbit the Sun with a perihelion radius of 0.28 astronomical units. The Polarimetric and Helioseismic Imager instrument (SO/PHI), aboard the SO spacecraft, will make full-disk and high-resolution magnetic field and velocity measurements of the Sun. One of the major challenges of the instrument is to maximize the scientific return with the very limited telemetry allocated. To comply with this constraint, a great deal of data reduction and analysis must be made on board.

The onboard data processing pipeline is sketched in Figure 1. Each observation involves the acquisition of 24 images (a combination of six wavelengths and four states of light), with a sensor of 2048x2048 pixels. This set of images goes through a fairly involved pre-processing task, which gives the images the necessary quality to allow performing the calibration and scientific analysis. By means of the Radiative Transfer Equation (RTE) inversion, maps of the three components of the magnetic field vector (strength, inclination, and azimuth) and the plasma velocity along the line-of-sight are obtained. These four maps, along with a continuum image compose a nominal observation. Eventually, these images are bit-truncated to 16 bits per pixel and compressed. Thanks to this strategy, the final amount of data is reduced by more than 30 times.

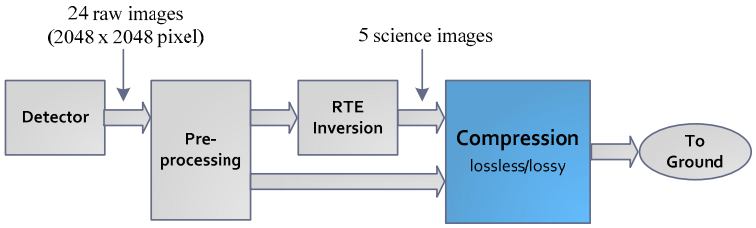


Figure 1. SO/PHI data processing pipeline.

Images in the range of 128x128 to 2048x2048 pixels have to be compressed while fulfilling a set of requirements depending on the different observing modes.. In nominal mode, a lossless compression scheme with an expected

Software and Cyberinfrastructure for Astronomy V, edited by Juan C. Guzman, Jorge Ibsen, Proc. of SPIE
Vol. 10707, 107072F · © 2018 SPIE · CCC code: 0277-786X/18/$18 · doi: 10.1117/12.2312701

compression ratio (*cr*) of 2 shall be applied to the RTE inversion outputs. On the other hand, for particular purposes, a lossy compression with a *cr* of up to 10 shall be applied to either the pre-processed data or the raw data directly. The compression stage is always performed offline in pre-established processing windows along the orbits. Due to the stringent electromagnetic compatibility requirements of the mission, such processing windows are very short for all the PHI data-processing tasks. Specifically, the compression time is expected to take around 10 seconds per image in average.

To perform the SO/PHI compression stage the *Consultative Committee for Space Data System – Image Data Compression* recommendation 122.0-B-1 [1] (CCSDS-IDC) has been selected. This algorithm is ideal in our case because it is able to perform both lossless and lossy compression with an effectiveness close to that of JPEG2000, which is considered the state of the art [2]. However, the standard relies on the Discrete Wavelet Transform (DWT), which makes its computational complexity significant higher than other lossless algorithms as the JPEG-LS or the CCSDS-LDC [3]. The DWT is not very suitable for running into general purpose processors. For example, the reference software implementation TER [4] takes near 2 seconds for compressing a SO/PHI image when running in a current cutting-edge computer (Intel Xenon Octa-core). In addition, in a space instrument context, the qualified computing devices do not provide comparable performance capabilities. This is due to their design for harsh environments, with reduced mass and power consumption. These are the main reasons to address an ad-hoc compression architecture in this work.

The SO/PHI instrument has been endowed with a Data Processing Unit (DPU) that utilizes a smart combination of a LEON3-FT-based main processor (*System Controller*) together with a pair of in-flight reconfigurable SRAM FPGAs Xilinx Virtex-4 (XQR4VSX55) [5]. A software implementation running into the LEON-3-FT main processor takes more than a minute per image. In such a scenario, we propose a hardware CCSDS-IDC compression core that accelerates the compression time respect to the LEON3-FT taking advantage of one of the Virtex 4 FPGAs.

There are already existing CCSDS-IDC FPGA implementations in space contexts. However, they use either more powerful devices than the ones available for SO/PHI or make use of additional external memory. For example, the FORMAT-5/RSI implementation employs three Virtex-5 devices with external memory [6]. On the other hand, in the proposal by [7], the authors also make use of external memory. In contrast, our proposal consists of a System-On-Chip (SoC) implementation that does not require any external memory. Furthermore, it implements other features of the standard that are not available in the existing implementations.

This contribution is organized as follows. Section 2 addresses the most important features of the compression standard algorithm. In Section 3, our proposed hardware implementation is explained. The strategy to exploit the FPGA resources for generating an optimal architecture is described as well. In Section 3, we show different compression tests using both dummy and real images. We conclude with some remarks about the most important contribution of this paper.

2. CCSDS-IDC ALGORITHM DESCRIPTION

The CCSDS-IDC compressor consists of two functional parts, depicted in Figure 2: the Discrete Wavelet Transform (DWT) module, which performs a two-dimensional decorrelation of the image, and a Bit Plane Encoder (BPE) module, which encodes the decorrelated data forming a bitstream.

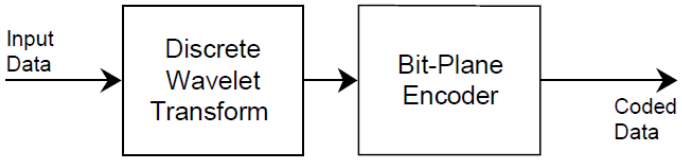
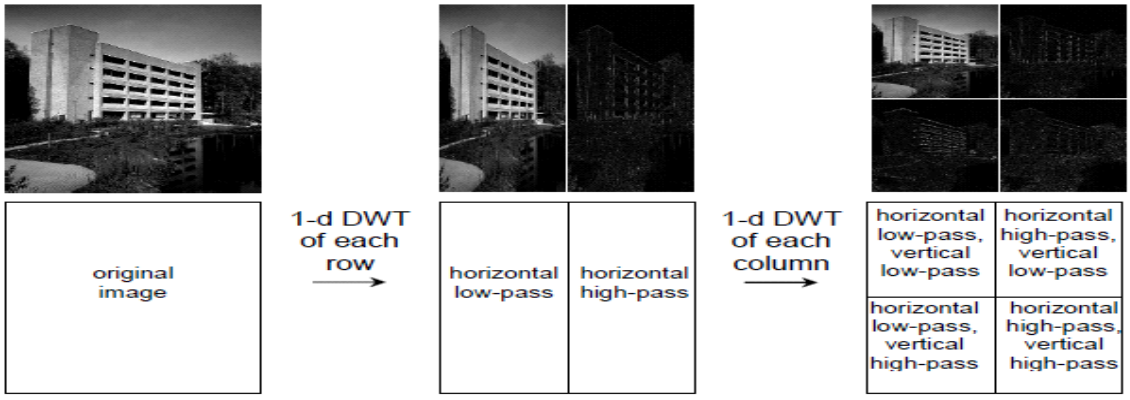
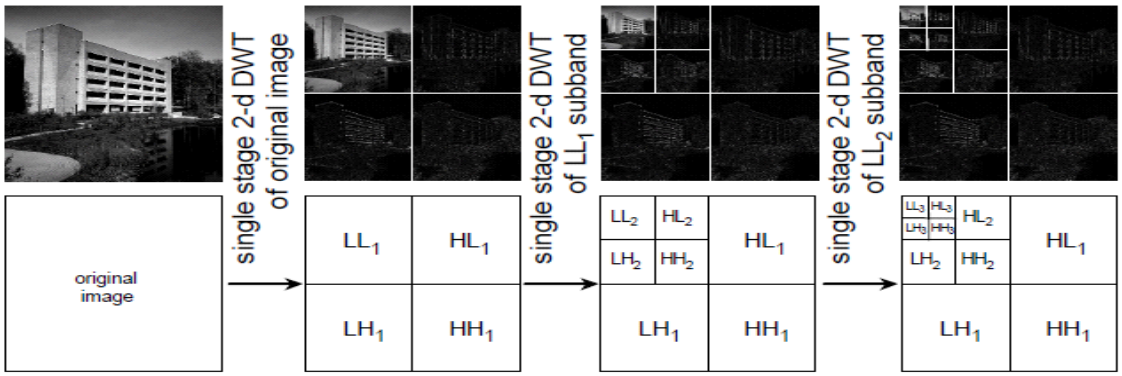


Figure 2. CCSDS-IDC block diagram [1].

The DWT module relies on carrying out iteratively the one-dimensional DWT (1D-DWT) in order to perform a 3-level 2-dimensional DWT (3-level 2D-DWT) decomposition of the input image. The 1D-DWT can be interpreted as a pair of FIR filters, a low-pass and a high-pass. The standard recommends two kinds of DWT filter kernels: an integer (IDWT) and a floating point DWT (FDWT). The IDWT involves only integer arithmetic operations and is able to produce lossless and lossy compression. On the other hand, the FDWT requires floating point arithmetic operations and is able to produce just lossy compression but outperforming the image quality of the IDWT in low bit rate lossy mode.



a) 2D-DWT



b) 3-level 2D-DWT

Figure 3. DWT decomposition of an image [1].

As shown in Figure 3 a), the 2D-DWT is obtained by performing the 1D-DWT to each row of the input image, and then, performing the 1D-DWT in column order to the resulting image. As a result, the 2D transformed image consists of four sub-images that are referred to as LL, LH, HL and HH subbands. The three-level decomposition consists on applying the 2D-DWT to the previous LL subband as shown in Figure 3 b). The sub-index included now to each subband name corresponds to the 2D-DWT iterations, so LL_1 is the LL subband after being 2D-transformed once, LL_2 two and LL_3 three times. Eventually, the DWT decomposition results in ten subbands, each representing the original image in a given frequency domain.

Before being encoded, the image in the wavelet domain is arranged in *blocks*. Each block contains a single pixel – *coefficient* – of the LL_3 subband referred to as *DC* coefficient, and 63 coefficients from the other subbands referred to as *AC* coefficients. Therefore, each block represents a spatial-slice of the image in every DWT subband.

These *blocks* are then grouped into *segments* and processed independently by the BPE. The size of the segments, S, is user-configurable but has to be carefully selected because it has implications in the protection against errors, compression effectiveness and needed memory resources of the BPE implementation. The Informational Report [2] defines a smart selection of the S value as one eighth of the of the input image width. When an image is compressed with this segment size, which is called strip compression, a minimum of memory resources is needed and a good balance with the other above mentioned factors is achieved.

The *dynamic range* of each segment must be calculated before being encoded by the BPE. This dynamic range is defined by four parameters so that each one determines the number of bits needed to represent, in binary representation, a group of coefficients within the segment: *BitDepthDC,* referred to the group of DC coefficient in the segment; $BitDepthAC\_block_m$ $m = [0 \ldots S\text{-}1]$, referred to the group of AC coefficients belonging to each block; and the *BitDepthAC*, defined as the maximum between every $BitDepthAC\_block_m$*'s.*

The BPE encodes each segment in several stages, as is shown in Figure 4. First, a *Segment header* is included with the necessary information for the decoding process such as image dimensions, dynamic range parameters and compression parameters. Then, a quantized version of all DC coefficients in the segment is encoded with a predictor scheme and a Variable Length Coding (VLC). Afterward, with the same compression method, *BitDepthAC_block_m* parameters of every block are encoded. Finally, the AC coefficients are progressively encoded in *bitplane* flavor: all b^{th} most significant bits of the binary representation of each AC coefficients make up a given *bitplane b*, which is encoded independently. The number of bitplanes in a segment depends on the *BitDepthAC* parameter. Each bitplane is further composed of five stages. The first one includes non-coded data relative to the DC coefficients. Stages from 1 to 3 perform a codification based on creating words that collect information relative to a group of coefficients, and coding them by means of VLC. Finally, Stage 4 adds non-coded data that performs the refinement of the reconstructed image.

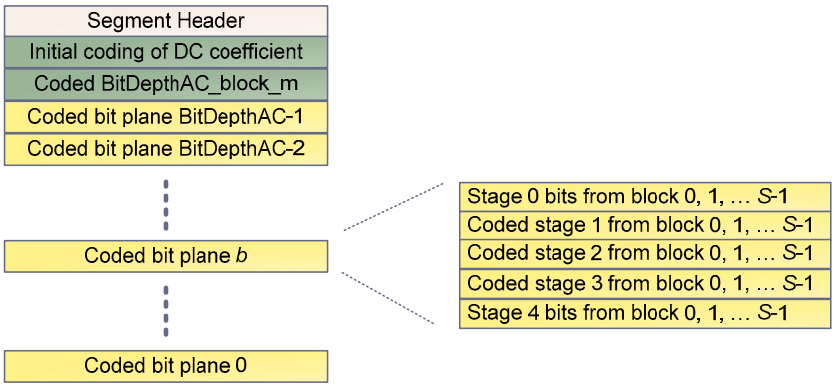


Figure 4. Coded segment structure.

A given segment is losslessly compressed when the IDWT is used and all sections of the segments are included. Otherwise, the segment can be truncated to provide lossy compression. The standard offers four user configurable parameters to this purpose: *SegByeLimit*, which defines the maximum number of bytes in the segment; *DCStop*, *BitplaneStop* and *StageStop*, where each of them defines the stop point after which the following sections will be not included [1].

3. CCSDS-IDC FPGA IMPLEMENTATION

In this work, we propose a CCSDS-IDC FPGA Coder that implements the whole algorithm within one of the SO/PHI DPU's Xilinx Virtex-4 device. Figure 5 shows the block diagram of this FPGA configuration. This device includes a SocWire driver [8], which interconnects the Coder within the DPU network, and the CCSDS-IDC Coder itself. Through the SocWire driver, the DPU System Controller operates the compression Coder sending images and receiving the compressed ones, configuring and monitoring the device.

The Coder itself counts with three main blocks, namely, the DWT, the BPE and a Configuration and Control module. The DWT module efficiently performs the 3-Level 2D-DWT by means of a light multi-processor architecture combined with a smart structure of buffers, so that all DWT levels are calculated in parallel. On the other hand, the BPE core has been carefully designed ad-hoc to this machine, in a manner in which it processes the segments sufficiently fast to prevent the halting of the DWT execution. On top of that, the Configuration and Control module is responsible for managing the data flow, reporting the system status to the DPU and configuring the DWT and BPE blocks. From the DPU System Controller side, the configuration relies on a set of Read/Write registers through which parameters such as image shape (width and height), input data format (signed/unsigned), and lossless or lossy compression mode (SegByeLimit, DCStop, BitplaneStop, StageStop) can be configured.

The following subsections describe the DWT and the BPE in detail.

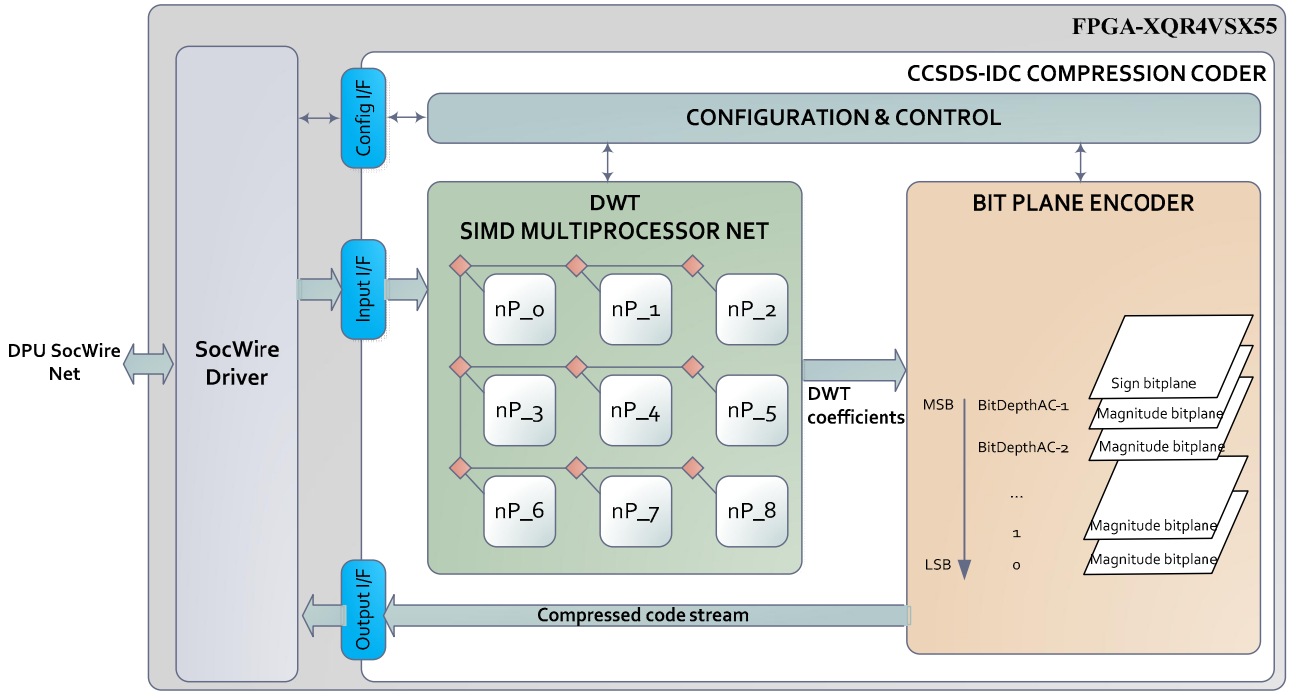


Figure 5. CCSDS-IDC FPGA Coder block diagram.

3.1 Discrete Wavelet Transform – SIMD architecture

In this work, and inspired by [9], we propose to use a SIMD -Single Instruction, Multiple Data- multiprocessor architecture to carry out the 3-level 2D-DWT algorithm defined in the standard. The SIMD multiprocessor topology used in this work is proposed by [10] and involves multiple processing elements -nProcessors- that perform the same operation on multiple data simultaneously. This topology exploits the data level parallelism in those algorithms that are susceptible of being parallelized. This is clearly the case of study here since it consists of performing iteratively the 1D-DWT.

As Figure 6 shows, our proposal counts with nine nProcessors connected between them by means of buffers (*xxx_buff* in the Figure), creating a peculiar structure. Such a structure follows the buffer assembly recommended in [2] for Memory Efficient DWT Calculation. Thanks to this approach, the system saves memory resources based on the fact that a single DWT coefficient depends only on a relatively small cluster of the input image, therefore it prevents from storing the whole image.

Each nProcessor performs the 1D-DWT on a different input data set. The unique instruction set, corresponding to the 1D-DWT program, is broadcast by the nProcessor net Control Unit (nPCU) to every nProcessor throughout a unique instruction bus. The nProcessor's Arithmetic Logic Unit (ALU) can be configured in synthesis time to implement either a 32-bit single Floating-point, compliant with both the IDWT and the FDWT, or 20.4 fixed-point arithmetic logic, only IDWT compliant. Hence, the architecture can be configured for the maximum versatility with the Float-Point version, or in saving-resources mode.

At a given DWT-level, the 2D-DWT is calculated with three processors. The first one performs the 1D-DWT in row order with the help of a *row buffer* (*LLx* in the Figure). The others two processors perform the 1D-DWT in column order, thanks to the *row-to-columns* buffers that precede them, where the row-to-column transposition takes place. This 2D-DWT structure is replicated three times, where the input to the second and the third DWT levels are the LL output of the previous 2D-DWT level. As a result, the ten DWT subbands are calculated in parallel.

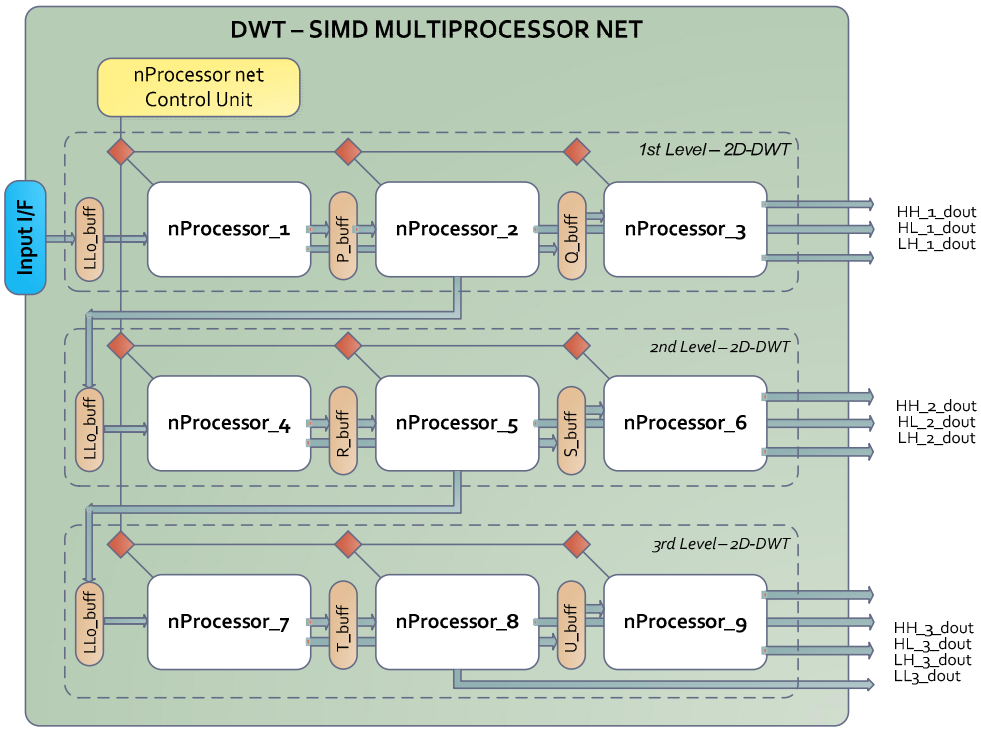


Figure 6. DWT – SIMD multiprocessor block diagram.

The row-to-column buffers are designed with a smart structure of *line buffers* connected between them as proposed in [11], so they perform the row-to-column transposition with a minor logic resources usage. In this application, eight line buffers plus a register compose the buffer structure, as Figure 7 shows. Each line buffer works as a pixel shift-register delay with N positions, N being the width of a given DWT subband. Hence, after a filling stage of $8*N + 1$ pixels, the buffer provides at the output multiplexer the necessary pixels to perform an iteration of the 1D-DWT in column order. It should be highlighted that the line buffers are entirely implemented using internal Block RAM (BRAM) resources configured as double-port RAM. Table 1 summarizes the BRAMs usage for input image width up to 2048.

Table 1. DWT internal buffer BRAM usage.

Buffer	Buffer size (32 bit words)	BRAM usage
P	8x1024 = 8192	16
Q	8x1024 = 8192	16
R	8x512 = 4096	8
S	8x512 = 4096	8
T	8x256 = 2048	8
U	8x256 = 2048	8

The row buffers just hold the necessary number of pixels of a horizontal filtering mask. Therefore, its internal structure is implemented with 32-bit registers as Figure 7 shows.

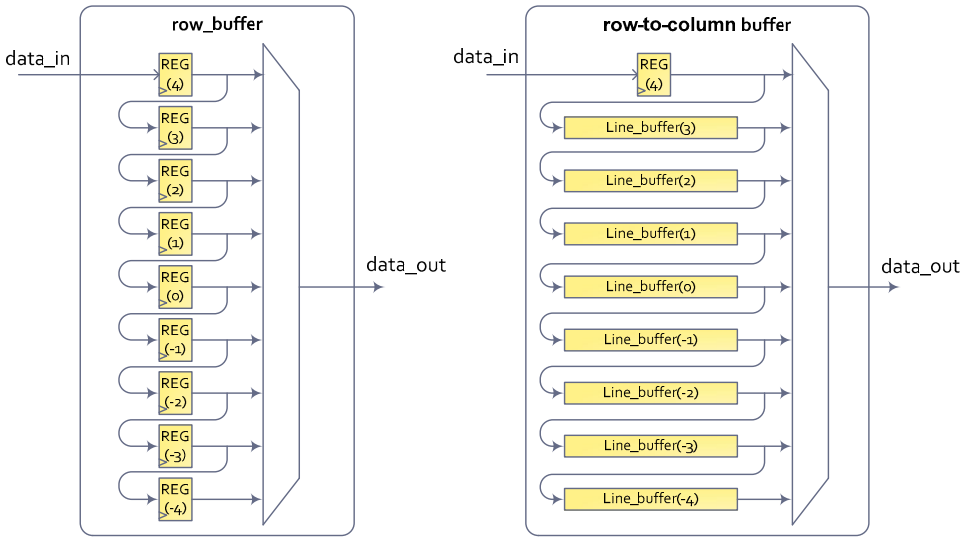


Figure 7. DWT Internal buffer structure.

3.2 Bit Plane Encoder

The Bit Plane Encoder module (BPE) was designed to perform the maximum throughput with the minimum resources. It combines pipeline and parallel execution techniques. Its internal architecture is depicted in Figure 8. The different segment sections are generated by three main blocks that work in parallel: the *Header generator*, the *DC Coder*, and the *AC Coder*. All parts of the segment are properly merged in the *BitStream Organizer* module, which generates a coded segment stream of 16-bit words. Finally, the *Bitrate Control* block is in charge of truncating the segment depending on the *SegByteLimit* parameter.

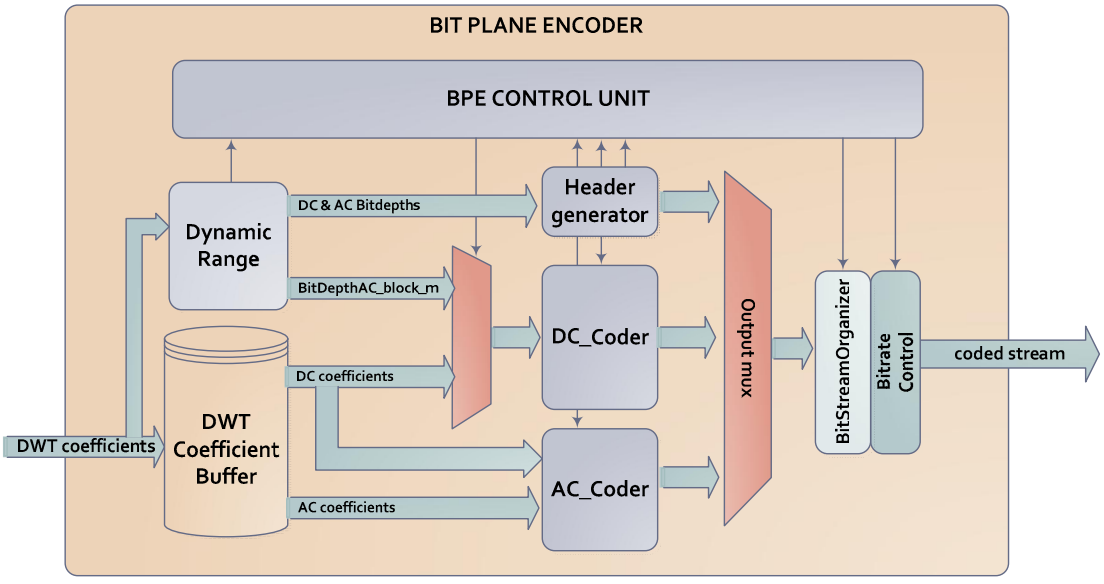


Figure 8. BPE block diagram.

As explained in the previous section, the segment codification depends on the dynamic range of different groups of coefficients that belong to it. In our proposal, the **Dynamic Range** block performs such a calculus seamlessly, generating the set of dynamic range parameters as soon as the DWT module produces the necessary coefficients. It means that,

without halting the DWT core, it processes coefficients from the DWT module, groups it internally for creating *blocks* and *segments* structures, and calculates these parameters.

In parallel with the Dynamic Range block, the **DWT Coefficient Buffer** performs a double-buffering of the incoming segments, allowing the BPE to perform the codification of the previous segment at the same time that the next one is being generated by the DWT core. The DWT Coefficient buffer is able to store up to two segments of 256 blocks each. Internally, the buffer is implemented with internal BRAMs organized in ten double-port RAMs, one per subband. Its capacity is actually greater than two segments due to the order in which the DWT generates a segment, where there is some delay between the initial storage of subbands and the arrival of associated DC coefficients. Table 2 summarizes the BRAMs utilization for each subband for a maximum segment size of S = 256 blocks.

Table 2. DWT Coefficient buffer BRAM usage.

Subband	Word width (bits)	Depth (number of words)	BRAM usage
HL1	18	17x1024	17
LH1	18	17x1024	17
HH1	19	17x1024	19
HL2	19	7x512	5
LH2	19	7x512	5
HH2	19	7x512	5
HL3	20	2x256	1
LH3	20	2x256	1
HH3	20	2x256	1
LL3	21	2x256	1

The **BPE Control Unit** is responsible for configuring and managing the BPE data flow. It triggers the compression process whenever a new segment is available in the DWT Coefficient Buffer. From that time on, the Header Generator, the DC Coder and the AC Coder run in parallel. These three blocks are synchronized so that they add its segment part in the proper time. First, the Header Generator adds the segment header to the output bitstream. Then, a unique instance of the **DC Coder** block performs both the *Initial coding of DC coefficients,* and then, the codification of the *BitDepthAC_block_m* parameters. The DC Coder implements the *Optimum code Option* scheme, that performs the better compression effectiveness in each group of coefficients (*gaggle*) [1].

The **AC Coder** block carries out the most complex task within the segment encoding process. In contrast with the DC Coder, it has to process all coefficients of the segment in a peculiar block order. In addition, the whole segment has to be processed *BitDepthAC* times in order to generate all the bitplanes. The proposed architecture, depicted in Figure 9, is designed to performing the bitplane coding sequentially, from bitplane *BitDepthAC-1* to *0*. Within each bitplane, the AC Coder reads all blocks of the segment, generates the coded stages 0-4 and merges them into the coded bitplane. It counts with two different parts: the stage 0 block, and the block that performs the coding of stages 1-4. Every block in the AC Coder are commanded by the *Bitplane control*, which indicates the index of the bitplane that it is being coded.

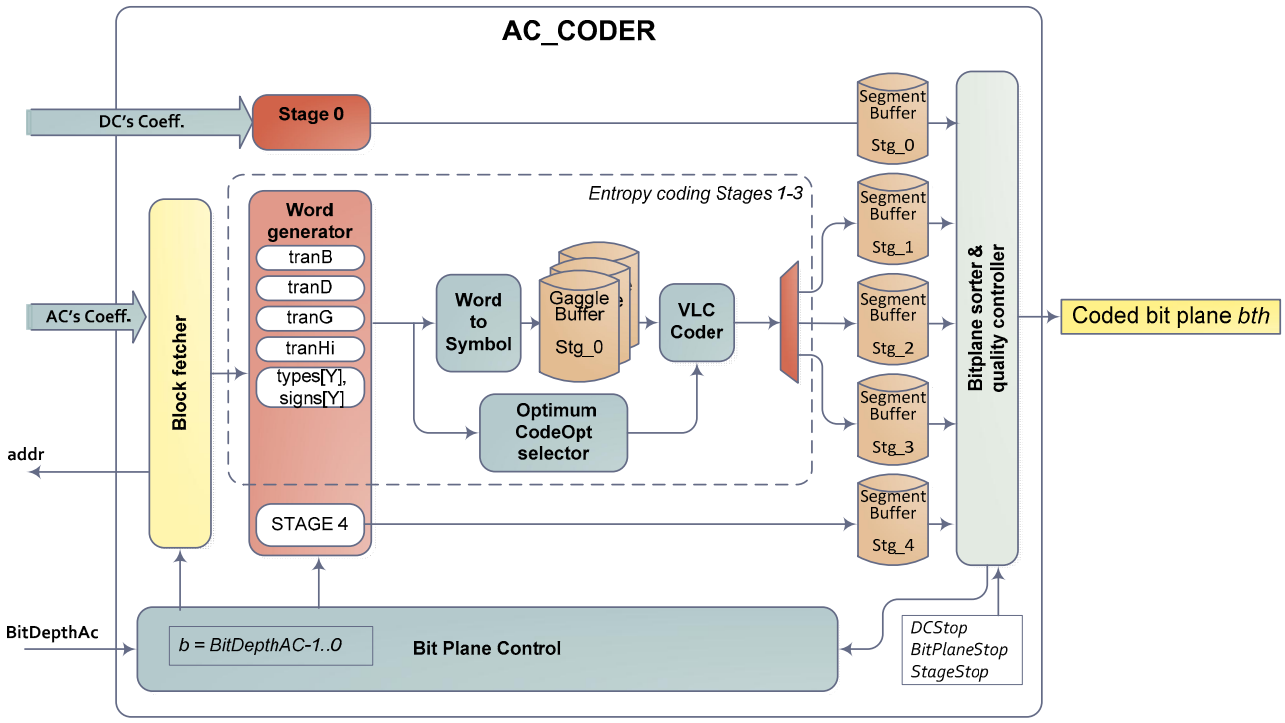


Figure 9. AC Coder block diagram.

The *Stage 0* block generates the coded stage 0 stream. For each bitplane, it reads the DC's coefficients of each block in the segment from the LL_3 *coefficient buffer*, takes the b_{th} bit of each coefficient and stores them in the *Stage 0 segment buffer*.

The coded stages 1-3 are produced by the blocks enclosed by the dash-line in Figure 9. Here the coding process is executed sequentially, from block 0 to block *S-1*. The *block fetcher* generates the proper addresses to read the 63 AC's coefficients of each block from the *DWT coefficient buffers*. These coefficients are converted into 38 *words* in the W*ord generator* module. In order to be coded, these words pass through several steps. First, they are translated into *symbols*, and buffered in the *gaggle buffer*. This buffer is used to gather the words belonging to 16 blocks, referred to a *gaggle*, meanwhile the optimum code option for encoding the *gaggle* is calculated by the *Optimum Code Option* selector block. Finally, the words of the gaggle are VLC coded using this code option and buffered in its respective *segment buffer*. Once every block in the segment has been processed, the stages 1-3 segment buffers contain each coded stage. The stage 4 bits are also generated in the *Word generator* module. As these bits are included without coding, they are stored directly in the Stage 4 segment buffer.

At the end of a bitplane cycle, all stages of a given bitplane are calculated and stored in their respective segment buffers. Therefore, the *Bitplane Sorter and Quality Controller* block reads out these buffers in order, from stages 0 to 4, and multiplexes the five inputs into a unique output, generating in this way the stream of coded words that compose a coded bitplane. The generated segment can be truncated as a function of the user configurable parameters *DCStop, BitPlaneStop and StageStop*.

The pipelined architecture of the AC Coder allows it to process one coefficient per clock cycle, which implies 63 clock cycles per block and bitplane.

3.3 Work flow

The DWT – SIMD multiprocessor machine joined to the BPE module performs a pipeline execution. Figure 10 shows a timeline of the compression at the beginning of the process. Each rectangle at the input image represents the time the Coder takes to process an image row of *2N* pixels. On the other hand, the rectangles in each DWT level output represents the time it takes to generate every subband belonging to a DWT level, so as their coefficients are generated in parallel. The DWT outputs are marked with different patterns depending on the segment they belong to, taking into account that

they are arranged for performing *strip compression*. The BPE output shows the time it takes to perform the different processes that take place in parallel: the header generation, the DC coding and AC coding.

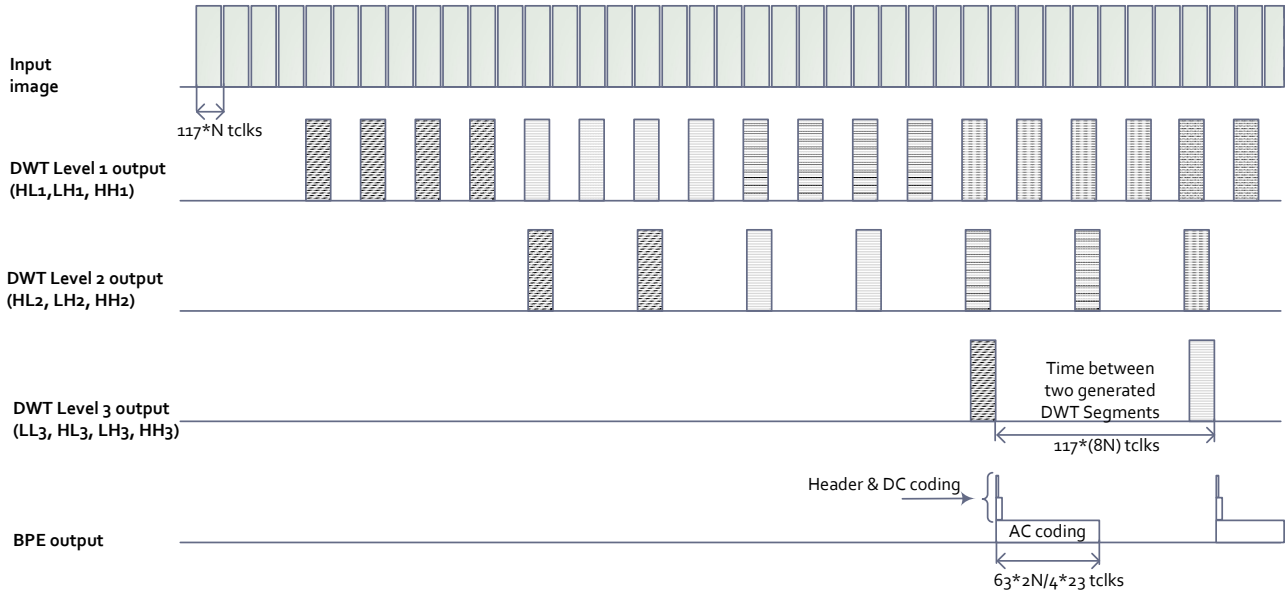


Figure 10. Compression timeline.

The DWT core takes 117 clock cycles (tclk) on executing the 1D-DWT program that processes 2 pixels at a time, so it takes 117*N tclk on processing an image row. After a filling stage, DWT level 1, 2 and 3 start to produce their coefficients. Due to the data dependency, there is a delay between the generated coefficients of the level 3 subband for a given segment. Once the level 3 subband is generated, the whole segment is available for coding within the BPE and, thanks to the Dynamic Range module, it is already characterized.

Within the BPE coding, the AC coding is the critical path because it has to process the AC coefficients at every bitplane. Thanks to its pipeline structure, it processes one coefficient per tclk, so it takes 63 tclk to processa *block*. Hence, the number of blocks in a segment, S, and number of bitplanes that are to be coded, BitDepthAC, define the required number of tclk to process a whole segment. In strip compression, the segment size is S = 2N/8. On the other hand, the maximum number of bitplanes due to the dynamic range expansion of the DWT coefficients is BitDepthAC = 23 [2]. Therefore, it takes 63*2N/4*23 tclk to process a segment in the worst case.

From the performance of the DWT and BPE modules, it can be inferred that the BPE performs the segment encoding faster than the DWT coefficients generation. Therefore, the DWT actually defines the maximum data rate of the device with the following expression:

$$\text{Maximum Pixel Rate} = 2/117/\text{tclk} \quad (1)$$

4. RESULTS

The CCSDS-IDC Coder with the SocWire communication driver was implemented with the Xilinx ISE Project Navigator tools for the target Xilinx Virtex 4 XQR4VSX55 device. Table 3 lists the resource utilization, which stands out for its low occupancy in spite of being implemented in a medium-capacity FPGA. The occupied slices are near 50%, but looking at its actual usage, Flip-Flops and LUTs, it is about the 30%. The BRAM usage is the most critical resource, so that DWT internal buffers and the DWT Coefficient buffer make intensive usage of them. However, the whole system fits within the device using less than 60% of the BRAMs. Is also remarkable the low usage of DSP due to the light layout of nProcessor ALUs, that use less resources than specific hardware DWT kernels.

The power consumption has been measured within the development board, so it might have deviations due to other devices in the board (led diodes, power regulator efficiency, etc). For that reason, Table 3 presents the overall consumption as the sum of the static consumption, when the FPGA is not programmed, and the dynamic consumption, when the device is programmed and running. The total power consumption of just 2.6 Watt reflects the advantage of using a single chip without external memory with respect to other implementations such a [6], with 5 Watt and [7], with 4 Watt of total consumption.

Table 3. XQR4VSX55 implementation statistics.

Number of occupied Slices	12,220 (49%)
- Number of Slice Flip Flops	12,761 (25%)
- Total Number of 4 input LUTs	15,182 (30%)
Number of used BRAMs	186 (58%)
Number of DSP48s	53 (10%)
Maximum frequency	150 MHz
Throughput @ 100 MHz	27.3 Mbps
Power consumption @ 100 MHz	2.58 Watt RMS
- Static power consumption	1.54 Watt RMS
- Dynamic power consumption	1.04 Watt RMS

The implementation in the target device reports a maximum frequency of 150 MHz. Nevertheless, the Flight Model implementation will run at 100 MHz. This means that it carries out the compression of a SO/PHI image, 2048x2048 pixel 16 bits per pixel, in 2.45 seconds. Hence, as Table 4 shows, our FPGA implementation compresses a whole SO/PHI observation of five images in just 12.25 seconds, around 28 times faster than the same operation carried out in the SO/PHI System Controller (GR712 - LEON3-FT), which takes more than 5 minutes.

Table 4. Compression times for a nominal SO/PHI observation of 5 images.

Device	**Time (seconds)**
GR712 - LEON 3-FT	~340
FPGA – proposal implementation	12.25

5. COMPRESSION CORE VALIDATION

The system has been validated by means of simulations and functional tests. The setup to perform the test is depicted in Figure 11. A set of real and synthetic images are used as input data set. These images are encoded by both, the reference software TER and the proposed FPGA coder under the same compression parameters. The resulted compressed files are compared and the test is considered *passed* if the compared files are equal, bit-to-bit.

The carried out tests involve most of the possible conditions based on a wide combination of images and compression parameter, including edge conditions as defined in [12]. Table 4 describes the functional test performed and its results.

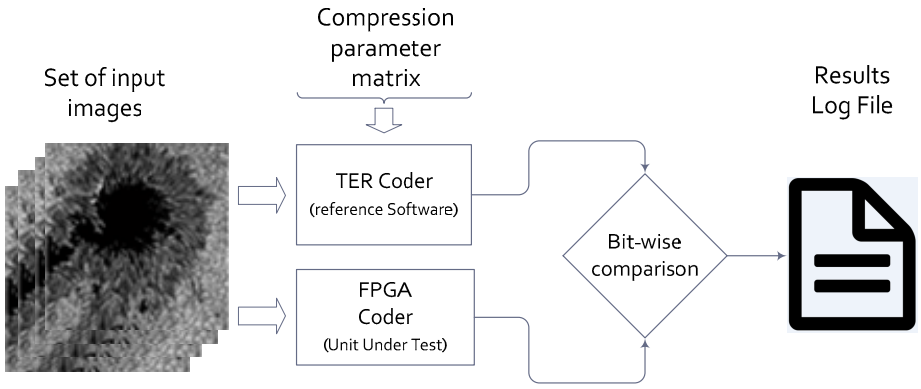


Figure 11. Core validation setup.

Table 5. List of validation test.

Test number	**Description**	**Result**
1	Lossless compression on all possible image sizes: 512 images with different shape resulted of the combination of the image width from 128:128:2048 and image height from 64:64:2048	Passed
2	Lossless compression on CCSDS image library: 21 images from different instruments collected in [13]	Passed
3	Lossless compression on image with all-zero data	Passed
4	Lossless compression on image with random data	Passed
5	Lossless compression on uniform intensity image	Passed
6	Lossless compression on image which produces maximum dynamic range after integer DWT	Passed
7	Lossless compression on image which produces data in section 4.3.3 of compressed bitstream	Passed
8	Lossless compression on Integer 16 bits image with negative values (signedPixels = '1')	Passed
9	Lossy compression at different image quality parameters: 40 lossy compression resulted of different combination of the DCStop, BitplaneStop and StageStop	Passed
10	Lossy compression at different coding rates: 6 lossy compression resulted of different values the SegByteLimit parameter	Passed

6. CONCLUSIONS

In this paper we present a System-On-Chip implementation of the CCSDS-IDC 122.0-B.1 standard for the SO/PHI space instrument. The hardware compression coder is in-flight reconfigured within one of the Xilinx Virtex-4 FPGA available on the SO/PHI DPU. It fulfills all SO/PHI requirements regarding the image compression stage, performing lossless and lossy compression over different image shapes in a very short time and within a very small power budget.

The proposed FPGA architecture performs lossless compression of 2048x2048 pixel images with full dynamic range of 16 bit/pixel in less than 3 seconds, which represents an acceleration of ~30 times with respect to a software implementation running on the LEON3-FT based processor also included into the SO/PHI's DPU.

The architecture stands out because of its low FPGA resource utilization. Thanks to the lightweight layout of the DWT core, based on SIMD multiprocessor architecture, and the optimized BPE core, the whole system fits into an average FPGA using less than 50% of the logic and 60% of BRAM memory elements. This is an advantage over other FPGA implementations because it does not use any external memory, hence saving mass, power consumption, and possible effects due to radiation.

Our implementation is also remarkable because of its configurability. In contrast with other implementations, it can perform both the Integer and the Float DWT. It is completely compliant with a wide range for image sizes, from 128x128 to 2048x2048 pixels, and a maximum dynamic range of 16 bit per pixel. Furthermore, it is able to manage large segments of up to S=256 blocks.

The lightness and modularity of the proposed architecture makes it extensible for future uses in different instruments with more demanding data rate. Using newer and more powerful FPGA devices would enable the possibility of increasing the number of nProcessors in the DWT core and including several instantiations of BPE cores, enhancing in this way the overall performance.

ACKNOWLEDGEMENT

This work has been partially funded by the Spanish Ministerio de Economía y Competitividad, through Project No. ESP2016-77548-C5-1-R, including a percentage from European FEDER funds.

REFERENCES

[1] Consultative Committee for Space Data Systems, "IMAGE DATA COMPRESSION Recommendation for Space Data System Standards 122.0-B-1," 2005.

[2] Consultative Committee for Space Data Systems, "IMAGE DATA COMPRESSION Informational Report 120.1-G-1," 2007.

[3] G. Yu, T. Vladimirova, and M. N. Sweeting, "Image compression systems on board satellites," *Acta Astronaut.*, vol. 64, no. 9–10, pp. 988–1005, 2009.

[4] GICI group, "TER software, an implementation of CCSDS Recommendation for Image Data Compression." [Online]. Available: http://www.gici.uab.cat/TER/.

[5] B. Fiethe, F. Bubenhagen, T. Lange, H. Michalik, H. Michel, J. Woch, and J. Hirzberger, "Adaptive Hardware by Dynamic Reconfiguration for the Solar Orbiter PHI Instrument," pp. 31–37, 2012.

[6] A. Lin, C. F. Chang, N. S. Organization, M. C. Lin, and L. J. Jan, "High-performance computing in remote sensing image compression," vol. 8183, no. Mc, pp. 1–11, 2011.

[7] 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," *J. Appl. Remote Sens.*, vol. 7, no. 1, p. 074595, 2013.

[8] C. Hilton and B. Nelson, "PNoC: a flexible circuit-switched NoC for FPGA-based systems," *IEE Proc. - Comput. Digit. Tech.*, vol. 153, no. 3, p. 181, 2006.

[9] A. Shahbahrami, "Improving the Performance of 2D Discrete Wavelet Transform using Data-Level Parallelism," *Image (Rochester, N.Y.)*, pp. 362–368, 2011.

[10] J. P. Cobos Carrascosa, J. L. Ramos Mas, B. Aparicio del Moral, M. Balaguer, A. C. López Jiménez, and J. C. del Toro Iniesta, "SIMD architecture on FPGA for scientific computing aboard a space instrument," *J. Syst. Archit.*, vol. 62, pp. 1–11, Jan. 2016.

[11] D. Ridgeway, *Designing Complex 2-Dimensional Convolution filters. The programmable Logic DataBook, Xilinx. 1994N. Palacios, Word 2003, Anaya multimedia, 2004*. .

[12] Yeh Pen-Shu, "BB122TestData - All Documents," 2007. [Online]. Available: https://cwe.ccsds.org/sls/docs/Forms/AllItems.aspx?RootFolder=%2Fsls%2Fdocs%2FSLS-DC%2FBB122TestData&FolderCTID=0x012000439B56FF51847E41B5728F9730D7B55F&View=%7BAE8FB44C-E80A-42CF-8558-FB495ABB675F%7D.

[13] Yeh Pen-shu, "BB122TestImage - All Documents," 2007. [Online]. Available: https://cwe.ccsds.org/sls/docs/Forms/AllItems.aspx?RootFolder=%2Fsls%2Fdocs%2FSLS-DC%2FBB122TestImage&FolderCTID=0x012000439B56FF51847E41B5728F9730D7B55F&View=%7BAE8FB44C-E80A-42CF-8558-FB495ABB675F%7D.