SHyLoC

EXPRO+ES AO/1-8032/14/NL/AK

IUMA/1410/AO8032

SHyLoC 2.0 IP Datasheet

By

University of Las Palmas de Gran Canaria
Institute for Applied Microelectronics (IUMA)
Spain

19 August 2020

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada

DISCLAIMER

DOCUMENT HISTORY

Date	Version	Author	Description	Status
16-10-2017	1.0	Lucana Santos	Product Datasheet – from project deliverable	Release
11-08-2020	2.0	IUMA	Product Datasheet for v2.0 of SHyLoC, includes following improvements: CCSDS 121 IP – Unit delay predictor CCSDS 123 IP – BIL-Mem architecture and AMBA AHB burst capabilities	Release

LIST OF AUTHORS

Partner	Authors
ESA	Lucana Santos
IUMA	Roberto Sarmiento
IUMA	Antonio Sánchez
IUMA	Yúbal Barrios

TABLE OF CONTENTS

FIGURES

TABLES

GLOSSARY

ACRONYM	MEANING
CCSDS	Consulting Committee for Space Data System
ASIC	Application-Specific Integrated Circuit
ITT	Invitation to Tender
IP	Intellectual Property
ADC	Analog-to-Digital Converter
AMBA	Advanced Microcontroller Bus Architecture
AHB	Advanced High-Performance Bus
SpW	SpaceWire
EDAC	Error Detection and Correction
EGSE	Electrical Ground Support Equipment

1 INTRODUCTION

1.1 Document scope

This document corresponds to the datasheets of the ESA Contract No. 4000113182/15/NL/LF entitled CCSDS Lossless Compression IP-Core Space applications. The document was updated based on the "Extension of the SHyLoC IP Cores: Improving Lossless Compression for Space Application" approved by ESA on December 2017.

1.2 Applicable documents

[AD-1] *Lossless Multispectral & Hyperspectral Image Compression*. Recommendation for Space Data System Standards, CCSDS 123.0-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, May 2012.

[AD-2] *Lossless Data Compression*. Recommendation for Space Data System Standards, CCSDS 121.0-B-2. Blue Book. Issue 2. Washington, D.C.: CCSDS, May 2012.

[AD-3] ESA ITT AO/1-8032/14/NL/AK, CCSDS Lossless Compression IP-Core Space Applications, Statement of Work, ESA, September 2014.

[AD-4] ESA ITT AO/1-8032/14/NL/AK, CCSDS Lossless Compression IP-Core Space Applications, Proposal, IUMA-TELETEL, October 2014.

[AD-5] "ASIC Design and Manufacturing Requirements", ESA document WDN/PS/700: http://microelectronics.esa.int/asic/DesignReq.pdf

[AD-6] ARM AMBA 2 Specification: http://www.arm.com/products/system-ip/amba/amba-open-specifications.php

[AD-7] AMBA synthesizable VHDL package, version 0.5, February 2002, amba.vhd

[AD-8] Aeroflex Gaisler GRLIB

http://gaisler.com/index.php/products/ipcores/soclibrary

1.3 Reference documents

[RD-1] D2, Requirements Specifications, IUMA, October 2018.

[RD-2] Single Event Effect Mitigation Plan, TASE, March 2015.

[RD-3] ESA Microelectronics: EDAC – HDL

http://www.esa.int/Our_Activities/Space_Engineering/Microelectronics/EDAC_-_HDL

[RD-4] CCSDS 123 software implementation from Universitat Autonoma de Barcelona http://www.gici.uab.cat/GiciWebPage/downloads.php#emporda

[RD-5] WhiteDwarf compression tool, CCSDS 123 and 121 software implementation from ESA

http://www.esa.int/TEC/OBDP/SEM069KOXDG_0.html

[RD-6] CCSDS 121 software implementation from HDF group

http://www.hdfgroup.org/doc_resource/SZIP

[RD-7] D5, Verification and Validation Plan, IUMA, October 2018.

1.4 Document description

This document presents the datasheet of the IP cores (CCSDS121 and CCSDS123) developed under the TRPAO8032. The document follows the guidelines in Appendix C.5 of [AD-4].

1.5 Cross-reference

This deliverable D3 (Extended SHyLoC IP Datasheet) has been created based on the document TRP-AO8032_D10_Datasheet_v4.10.docx, delivered on July 2017 for the Final Review of the project EXPRO+ES AO/1-8032/14/NL/AK.

2 OVERALL FUNCTIONALITY OF THE CCSDS121 AND CCSDS123 COMPRESSION IP CORES

This document presents the datasheet of two different IP-cores that are compliant with the CCSDS 121 [AD-2] and CCSDS 123[AD-1] compression standards respectively.

The CCSDS121 IP is compliant with the CCSDS 121 [AD-2] standard, which defines a lossless universal compressor based on Rice adaptive coding. Additionally, it allows the addition of a pre-processing stage. The CCSDS 121 standard proposes a Unit-Delay predictor as the pre-processing stage (from now on, *CCSDS121 predictor*, or *CCSDS121 pre-processor*). On the other hand, the logic in charge of performing the Rice adaptive coding will be denoted as *CCSDS121 block coder* in this document. The combination of the CCSDS121 predictor and the CCSDS121 block coder conforms the full *CCSDS121 IP* core, as shown in the bottom half of Figure 2-1

The CCSDS123 IP is compliant with the CCSDS 123 standard [AD-1], which describes a 3D predictive lossless compressor for hyperspectral and multispectral data. It describes the compressor as a two-part functional system: 3D prediction and entropy coder. It offers two options for the entropy coding stage: the sample-adaptive entropy coding and the block-adaptive coding, which corresponds to the specifications of the CCSDS 121 encoder. This fact makes it possible to reuse the CCSDS121 block coder to perform the block-adaptive encoding described by the CCSDS 123 standard.

The CCSDS123 IP and CCSDS121 IP are independent compressors; however, they have compatible interfaces, making possible to combine them as shown in Figure 2-1.

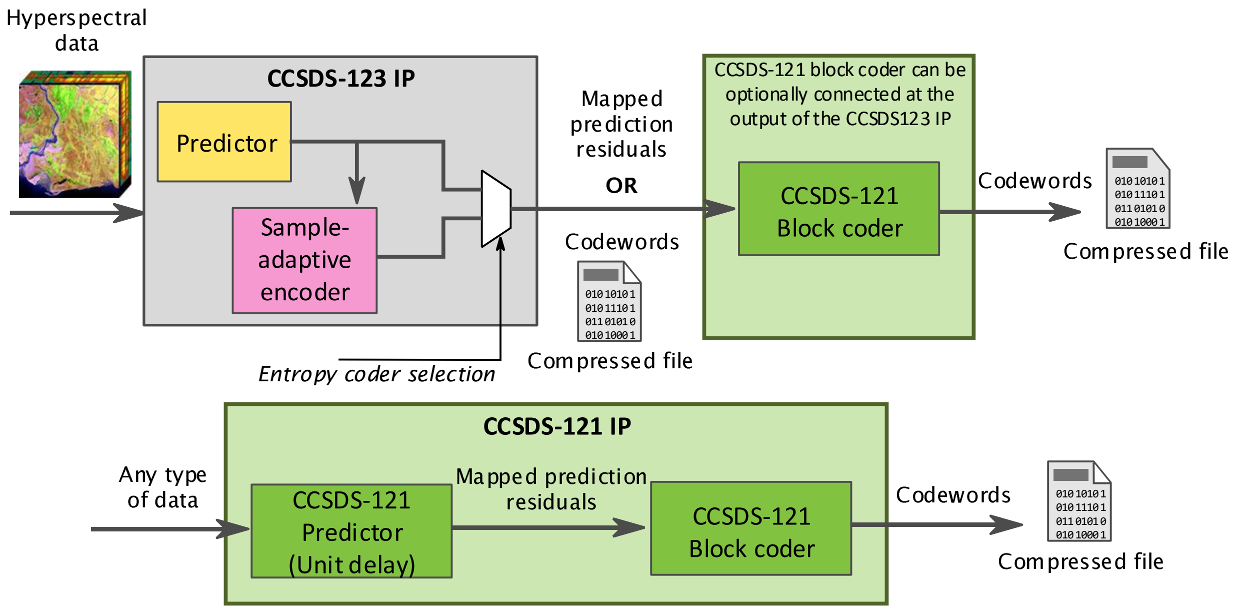


Figure 2-1: Designed IP cores and connectivity between them.

3 CCSDS121 IP CORE DATSHEET

3.1 System overview

The CCSDS121 IP Core performs the compression of a set of input samples according to the specifications of the CCSDS 121 standard [AD-2].

The input samples might be received from different sources: a mass memory, a SpW interface, an ADC or another IP core. The CCSDS121 IP core does not implement any standard communication protocol. The control signals implement a simple handshake protocol to ensure synchronization with the source.

A typical system including the CCSDS121 IP core looks like the one displayed in Figure 3-1.

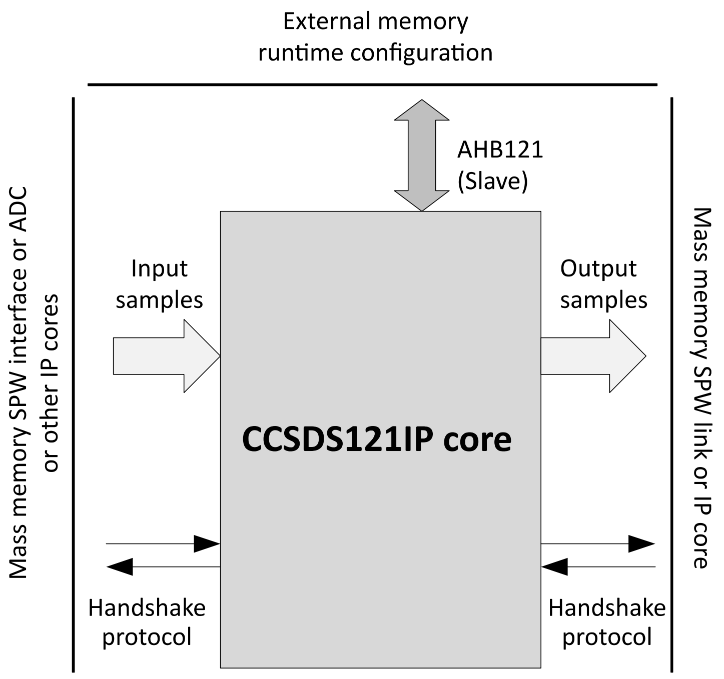


Figure 3-1: CCSDS121 IP - Compression core in a system.

The CCSDS121 IP core can be configured with a wide set of configuration parameters which can be selected either at implementation time or at runtime. Those parameters that can be selected at runtime are received by the IP core through the AMBA AHB slave interface.

The compressed samples are sent through the output data interface. Similarly to the input, the output might send values to a mass memory, SpW interface or another IP core. The output data are sent together with a valid flag. The output data represent the bitstream (compressed input data), according to the specifications in the CCSDS 121 standard, including the necessary side-information (header) to allow the decompression of the data.

The CCSDS121 block coder can be easily connected at the output of the CCSDS123 IP to implement the block-adaptive encoding option as described by the CCSDS 123 standard. More details about the CCSDS123 IP implementation and how to connect the CCSDS121 block coder as encoding stage can be found in Section 4.

3.2 Functional description

3.2.1 Applicable algorithms

Figure 3-2 represents the CCSDS 121 compressor, as defined in [AD-2] , including the pre-processing stage described in Section 4 of the CCSDS 121 standard. The pre-processor contains two functions, prediction and mapping. When enabled, the pre-processing stage attempts to remove the correlation among data samples (predictor) and reformat them into a more suitable probability distribution (mapper). The proposed predictor in the CCSDS 121 standard is a Unit-Delay predictor, where the previous sample x_{i-1} is used as an estimator $\hat{x}_i$ of the current sample x_i. Then, the prediction error $\varDelta_i$ with respect to the current sample is computed, and it is mapped into a non-negative integer δ_i. To be able to recover the sample values from the decoded prediction errors, it is necessary to insert some reference samples in the compressed bitstream. Therefore, the pre-processing stage is reset with a periodicity specified by the user. Later, the entropy coder chooses one of a set of code options to represent an incoming block of pre-processed data samples, δ. A unique identifier (ID) bit sequence is attached to the code block to indicate to the decoder which decoding option to use.

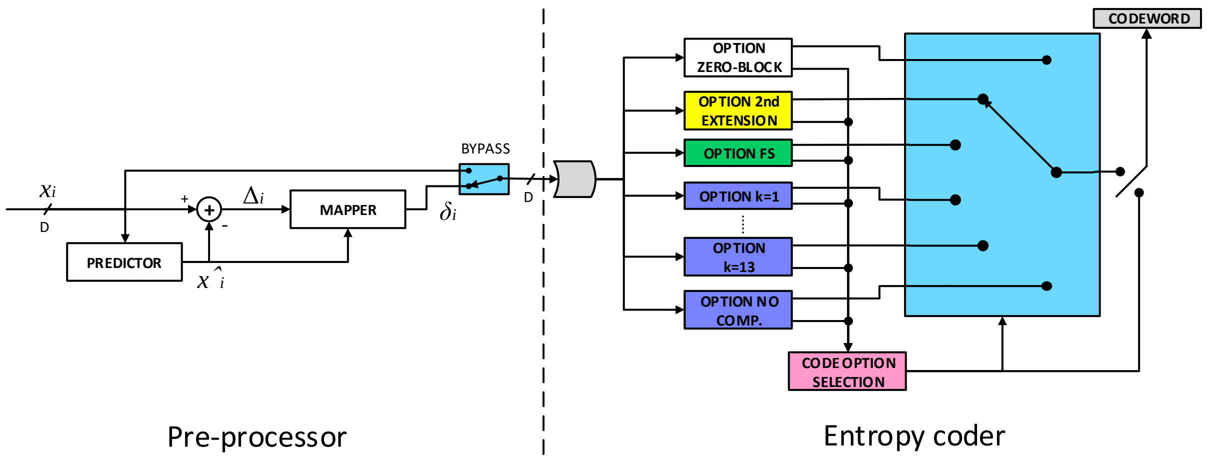


Figure 3-2: CCSDS121 IP - Basic block-adaptive coder with pre-processing stage.

The basic code selected is a variable-length code that utilizes Rice's adaptive coding technique. In Rice's coding technique, several algorithms are concurrently applied to a block of J consecutive preprocessed samples, specifically:

- *Fundamental sequence (FS)*: a codeword consists of δ_i zeros followed by a one, where δ_i is the mapped prediction residual.
- *Sample splitting*: the codeword is obtained by removing the *k* least significant bits from the binary representation of each mapped prediction residual, δ_i, encoding the remaining bits with an FS codeword.
- *Second-extension option*: each pair of pre-processed samples δ_i and $\delta_{(i+1)}$ is transformed into a single new symbol γ, which is encoded using an FS codeword.
- *Zero-block option*: this option is selected when one or more blocks of pre-processed samples are all zeros.

- *No compression*: the mapped prediction residuals are unaltered.

The algorithm option that yields the shortest encoded length for the current block of data is selected for transmission. The zero-block option is a special case in which a single codeword sequence represents one or more consecutive blocks of J preprocessed samples. In all other options, the codeword sequence represents a single block of J consecutive preprocessed samples.

3.2.2 General description

The IP core receives first the configuration through the AMBA AHB Slave interface. It informs then the source that it is ready to receive the input samples. If the CCSDS121 predictor is enabled, the input samples are pre-processed prior to being coded. The block coder calculates the amount of bits taken by the compressed block for all possible encoding options and selects the option that yields the shortest compressed block, as specified by the standard. The complete block is then encoded with the selected coding option and packed to be sent through the output interface.

The CCSDS121 IP may work as an independent compressor. In addition, it is designed in such a way that the CCSDS121 block coder can be easily connected to the CCSDS123 IP to perform only the block-adaptive entropy coding option defined in the CCSDS 123 standard. For that purpose, the IPs are designed with compatible interfaces. Moreover, pre-processors other than the CCSDS123 IP core or the pre-processing stage of the CCSDS 121 standard can be easily connected. Configuration options are given for the user to indicate to the CCSDS121 IP which pre-processor is present.

The CCSDS121 IP core creates a header and then the generated codewords are attached and packed. For verification or validation purposes, it is possible to inhibit the generation of headers. The configuration option DISABLE_HEADER is provided to disable the header generation.

Additionally, when the block coder is working as an external encoder of a pre-processor such as the CCSDS123 IP, the CCSDS121 block coder generates its corresponding part of the header, which is then attached to the header of the pre-processor, sent by the CCSDS123 IP.

The CCSDS121 IP core might be configured at runtime or at implementation time. The generic parameter EN_RUNCFG is used to select between the two options.

When EN_RUNCFG = 1, the IP core configuration is received through the AMBA AHB Slave interface. The CCSDS121 IP clock is independent from that of the AMBA interface. The received configuration data are stored in internal registers, read by the IP and then made available for all the modules that request them.

In addition, the possibility of setting some parameters at implementation time (before synthesis or compilation of the model) is allowed. In that case, the design is tailored to the user's needs. The different constants are stored in a configuration package, and then propagated to the generics of the sub-modules or used directly by the sub-modules as constants.

3.2.3 SEE Mitigation

The following SEE Mitigation Techniques are considered:

- Triple Modular Redundancy and Majority Voting in all flip-flops and latches (TMR) can be applied at logic level during synthesis with the appropriate synthesis attribute according to the target technology.
- Error Detection and Correction (EDAC) for embedded memories, with capability for correcting one error and detecting double errors. EDAC is enabled by setting the corresponding generic parameter.
- States in the FSM are encoded using Gray coding. All FSM include the <<when others>> clause, and return to the IDLE state in case of errors.

3.2.4 Functions not included

The segment size (s), described in Section 3.4.3.2 of [AD-2] is set to the recommended value of 64 block, and cannot be configured by the user.

In Section 3.4.3.3 of the Recommendation [AD-2], the Remainder-Of-Segment (ROS) is used when the remainder of the last segment in the data to be compressed consists of five or more all-zero blocks. This applies even if the last segment is shorter than the segment size (s).

The possibility of generating the optional secondary header in Section 5.2 of [AD-2] is not implemented, nor the possibility of choosing CIP (Section 6 of [AD-2]).

3.2.5 Functional specification

3.2.5.1 Design options

The design options are set by VHDL constants that are stored in a package and then propagated to the generics of the sub-modules or passed directly as constants. The provided constants are summarized in the following sections. These constant parameters affect or limit the runtime configuration options of the design, as it is detailed in Section 3.2.5.2.

3.2.5.1.1 System parameters

The constant EN_RUNCFG in Table 3-1 is used to enable or disable the configuration of the compressor's parameters at runtime. When EN_RUNCFG = 0, the design is tailored according to the user-selected configuration constants described in Section 3.2.5.1.2. EN_RUNCFG = 1 enables the user to set the configuration parameters at runtime.

The constant PREPROCESSOR_GEN indicates whether the CCSDS121 IP is being used as an independent compressor, or there is a pre-processor attached to it. This affects mainly the generation of the header, since such header will be different for each case. In particular, four scenarios are foreseen:

- The CCSDS121 block coder is working as an independent compressor, without any pre-processing stage (PREPROCESSOR_GEN = 0). In this case, the CCSDS121 block coder generates a header according to the specifications in Section 6.3.3.5.3 of [AD-2].

- The CCSDS121 block coder is working as the block-adaptive entropy coder of the CCSDS123 IP (PREPROCESSOR_GEN = 1). The CCSDS121 block coder bypasses any header sent by the CCSDS123 IP and attaches then its own header, generated as specified in Table 5-4 of [AD-1].

- The full CCSDS121 IP is implemented (PREPROCESSOR_GEN = 2). In this case, the header is generated according to the specifications in Section 6.3.3.5.2 of [AD-2].

- The CCSDS121 block coder is working as the entropy coder of any pre-processor, other than the CCSDS123 IP core (PREPROCESSOR_GEN = 3). Any header sent by the pre-processor is bypassed and no header is generated by the CCSDS121 IP.

TABLE 3-1: CCSDS121 IP - SYSTEM CONSTANTS

Constant	Allowed	Description
EN_RUNCFG	0	Disables runtime configuration (the configuration is set at implementation time) .
	1	Enables runtime configuration (the configuration is set at runtime reading the memory-mapped register).
RESET_TYPE	0	Implement asynchronous reset.
	1	Implement synchronous reset.
EDAC	0	Inhibits EDAC implementation.
	1	EDAC is implemented for embedded memories. *NOTE: this parameter is forced to '0' in the current implementation, because the memory use is so limited that BRAMs are not inferred and therefore only FFs are synthesized. It is up to the users to change the GENERIC assignment if they prefer to pass the EDAC parameter to a memory instance.*
PREPROCESSOR_GEN	0	The pre-processor is not present, the CCSDS121 IP is working as an independent entropy encoder, generating its own header as specified in Section 6.3.3.5.3 of [AD-2].
	1	The CCSDS123 IP pre-processor is present, and the CCSDS121 IP is working as entropy coder of the CCSDS123 IP. Any header sent from the CCSDS123 pre-processor is bypassed; and afterwards the CCSDS121 IP generates its own header according to Table 5-4 of [AD-1].
	2	The CCSDS121 pre-processor (unit-delay predictor) is present, working as a complete compressor together with the block-adaptive encoder. The header is generated according to Section 6.3.3.5.2 of [AD-2].
	3	Any other pre-processor is present, and the CCSDS121 block coder is working as entropy coder of another pre-processor, different from the CCSDS123 IP or the CCSDS121 pre-processor. Any header sent from the pre-processor is bypassed.

3.2.5.1.2 Configuration parameters

The implementation time parameters in Table 3-2 and Table 3-3 affect the configuration of the CCSDS 121 compressor. The aforementioned tables present a description of the parameters.

Specific baseline values are proposed for each constant and displayed in the second column ("Baseline") of the tables. The values of the implementation time parameters put limitations in the selectable runtime configuration values, as it is shown later in detail in Section 3.2.5.2. The "Baseline" values will generate an implementation of the compressor that allows the user to set the runtime configuration parameters to the most commonly used figures. Additionally, the third column ("Max.") displays the values that allow the user to select any of the possible configuration values permitted by the CCSDS 121 standard.

The image metadata in Table 3-2 include three parameters to set the number of samples to be compressed, which correspond to the three dimensions of an image cube (Nx, Ny, Nz). The CCSDS121 IP compressor is a universal encoder and as such, it can be used to compress three-dimensional as well as bi-dimensional or mono-dimensional data. In the latter cases, if the data to be compressed are mono-dimensional, Ny and Nz shall be set to 1; likewise, if the data are bi-dimensional, Nz shall be set to 1. The CCSDS121 IP then uses the given information to compute the total amount of samples in the image as Nz×Ny×Nz.

TABLE 3-2: CCSDS121 IP - IMAGE METADATA CONSTANTS

Constant	Baseline	Max.	Allowed values	Description	Reference in [AD-2]
Nx_GEN	1024	65535	[1:2^{16}-1]	Maximum allowed number of samples in a line.	
Ny_GEN	1024	65535	[1:2^{16}-1]	Maximum allowed number of samples in a row.	
Nz_GEN	1024	65535	[1:2^{16}-1]	Maximum allowed number of bands.	
D_GEN	16	32	[2:32] when used as standalone encoder. [8,16] when connected to the CCSDS123 IP as block-adaptive encoder.	Maximum dynamic range of the input samples.	
IS_SIGNED_GEN	0	1	[0, 1]	(0) unsigned or (1) signed input samples.	
ENDIANESS_GEN	0	1	[0,1]	(0) little endian; (1) big endian.	

TABLE 3-3: CCSDS121 IP - ENCODING CONSTANTS

Constant	Baseline	Max.	Allowed values	Description	Reference in [AD-2]
DISABLE_HEADER_GEN	0	0	[0,1]	Selects whether to disable (1) or not (0) the header generation.	
J_GEN	16	64	[8,16,32,64]	Block size.	3.1.4
CODESET_GEN	0	0	(0) Basic; (1) Restricted	Code option.	5.1.2
REF_SAMPLE_GEN	4096	4096	≤ 4096	Reference sample interval.	4.3

W_BUFFER_GEN	32	32	N/A	Number of bits in the output buffer.	

3.2.5.1.3 AMBA AHB parameters

The generic parameters in Table 3-4 are included for AMBA AHB configuration.

TABLE 3-4: CCSDS121 IP - AHB CONSTANTS

Constant	Allowed values	Description
HSINDEX	0-NAHBSLV-1	AHB slave index.
HSCONFIGADDR	0-16#FFF#	ADDR field of the AHB Slave. Sets the 12 most significant bits in the 32-bit AHB address.
HSADDRMASK	0-16#FF0#	MASK field of the AHB Slave.

3.2.5.2 Runtime configuration

3.2.5.2.1 Registers

The CCSDS121 IP core allows for runtime configuration of image metadata and encoding parameters, when the parameter EN_RUNCFG = 1.

In this case, the configuration values used by the IP core during the compression are the ones in the memory-mapped configuration registers, as shown in Table 3-5 and Table 3-6.

The “Max.” values displayed in Table 3-2 and Table 3-3 for the implementation time parameters when the configuration is set at runtime (EN_RUNCFG = 1) ensure the possibility for the user to select among all the possible range of values allowed by the CCSDS 121 standard. The user might change the values of the implementation time parameters at convenience, taking into account that the runtime configuration values allowed might be affected and limited, as it is shown in “Allowed values” column of Table 3-5 and Table 3-6.

TABLE 3-5: CCSDS121 IP - RUNTIME CONFIGURATION OF THE IMAGE METADATA

Parameter name	Allowed values	Description
Nx	[1 : Nx_GEN]	Number of samples in a line.
Ny	[1 : Ny_GEN]	Number of samples in a row.
Nz	[1 : Nz_GEN]	Number of bands.
D	[1 : D_GEN]	Dynamic range of the input samples.
IS_SIGNED**	[0, 1]	(0) unsigned or (1) signed input samples.
ENDIANESS*	[0,1]	(0) little endian;(1) big endian.
PREPROCESSOR*	[0,1,2,3]	(0) the pre-processor is not present, (1) the CCSD123 pre-processor is present, (2) the CCSDS121 pre-processor is present, (3) any other pre-processor is present.
BYPASS	[0,1]	(0) compression; (1) bypass compression.

TABLE 3-6: CCSDS121 IP - RUNTIME CONFIGURATION OF COMPRESSION PARAMETERS

Parameter name	Allowed values	Description
DISABLE_HEADER*	[0,1]	Selects whether to send or not the header.
J	[0,8,16,32,64] <=J_GEN	Block size.
CODESET	[0, 1]	Code option.
REF_SAMPLE	≤ REF_SAMPLE_GEN	Reference sample interval.
W_BUFFER	≤ W_BUFFER_GEN	Number of bits in the output buffer.

**Runtime configuration values override the values of the constants implementing the same functionality.*

***When the unit-delay predictor is not present (PREPROCESSOR generic different to 2), the IS_SIGNED option is ignored, and all samples are treated as unsigned. If any other pre-processor is used, it should include its own parameter to manage signed samples.*

Invalid configuration values are detected when reading the runtime configuration values; in such an event, an error is signalled as specified in Section 3.4.6 .

NOTE: It is assumed EDAC protection for these external memory-mapped configuration registers is managed by an external entity, and therefore the IP Core does not actively check for EDAC errors. It is up to the user to evaluate if protection is necessary considering the short life time of the information. Configuration information is read by the IP Core immediately after it is written in the memory-mapped registers and only once, and then stored in internal registers.

3.2.5.2.2 Memory mapping

The values of the memory-mapped registers are accessed by the IP core using the AMBA AHB Slave interface. The IP core includes the following memory-mapped registers:

TABLE 3-7: CCSDS121 IP - MEMORY-MAPPED REGISTERS

AHB Offset	Register
0x00	Control&Status
0x04	121CFG0
0x08	121CFG1
0x0C	121CFG2

The Control&Status register is used to enable the IP core and to write information about the IP core's status, storing the values in the control interface and an error code.

TABLE 3-8: CCSDS121 IP - CONTROL&STATUS REGISTER

AHB Offset	Bit Number	Mode	Description	Example value	Bits taken
0x00	31	w	Value of the **AwaitingConfig** signal.	0	1
	30	w	Value of the **Ready** signal.	0	1
	29	w	Value of the **FIFO_Full** signal.	0	1
	28	w	Value of **EOP** signal.	0	1
	27	w	Value of **Finished** signal.	0	1

	26	w	Value of **Error** signal.	0	1
	25:22	w	Error code.	0	4
	21:1	r	reserved	0	22
	0	r/w	ENABLE	0	1

When ENABLE is set to 1, the IP core starts running. This also indicates the IP that all the necessary configuration values have been sent.

The memory-mapped registers shall be of 32-bits, with the following configuration registers:

TABLE 3-9: CCSDS121 IP - CONFIGURATION REGISTER 121CFG0

AHB Offset	Bit Number	Mode	Parameter name	Example value	Bits taken
0x04	31:16	r	Nx	512	16
	15	r	CODESET	1	1
	14	r	DISABLE_HEADER	0	1
	13:7	r	J	16	7
	6:0	r	W_BUFFER	32	7

TABLE 3-10: CCSDS121 IP - CONFIGURATION REGISTER 121CFG01

AHB Offset	Bit Number	Mode	Parameter name	Example value	Bits taken
0x08	31:16	r	Ny	512	16
	15:3	r	REF_SAMPLE	4096	13
	2:0	r	reserved	all zeros	3

TABLE 3-11: CCSDS121 IP - CONFIGURATION REGISTER 121CFG02

AHB Offset	Bit Number	Mode	Parameter name	Example value	Bits taken
0x0C	31:16	r	Nz	512	16
	15:10	r	D	16	6
	9	r	IS_SIGNED	0	1
	8	r	ENDIANESS	0	1
	7:6	r	PREPROCESSOR	0	2
	5	r	BYPASS	0	1
	4:0	r	reserved	all zeros	5

3.3 Conventions

The following conventions shall be considered to correctly interpret the raw and compressed data values:

- The most significant bits of the compressed bitstream are located to the left and are the first to be transmitted.
- When the number of valid bits in a signal or register is smaller than the bit width, the valid bits are located to the right, i.e. using the least significant bits.
- The bitstream is packed according to the size of the output buffer and the user-defined parameter W_BUFFER. When the compression finishes, the remaining bits of the bitstream shall be flushed. Zeros shall be appended to the right, up to the W_BUFFER boundary.
- The input data samples are interpreted as specified according to the endianness specified by the user. Once received by the IP, they are always treated as signed values stored in big-endian order.

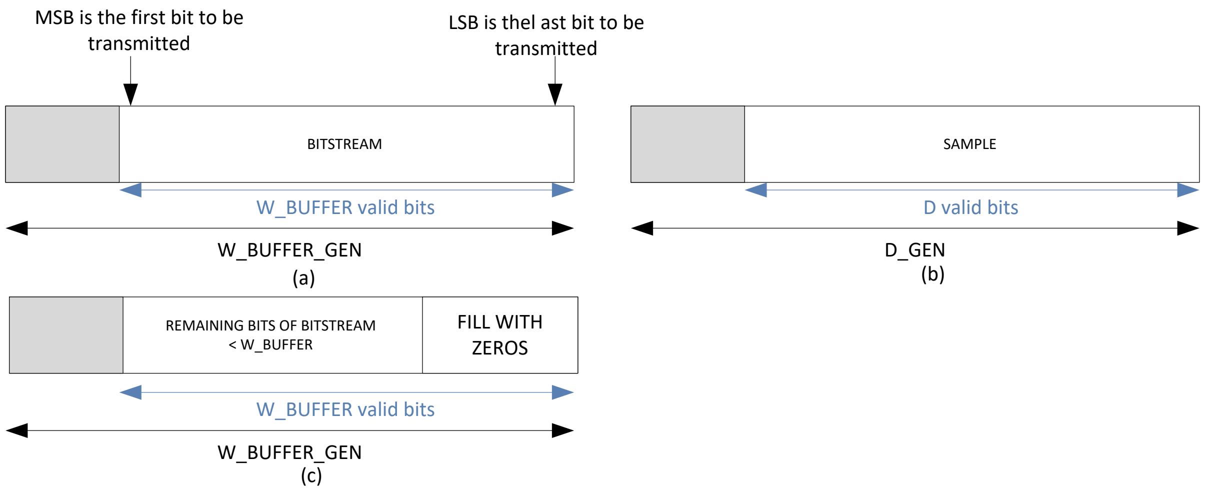


Figure 3-3: Location of valid bits in the i/o signals. (a) Bitstream (b) Input sample (c) Remaining bits of the bitstream after flushing the output buffer.

3.4 Interfaces and signal description

3.4.1 Interfaces general description

3.4.1.1 System Interface

- **Rst_N**: Active low input signal that resets the IP to its initial state. **Rst_N** might be synchronous or asynchronous according to the value of the generic RESET_TYPE.
- **Clk_S**: System clock signal. The registers internal to the IP are clocked on the rising **Clk_S** edge.

3.4.1.2 Input data interface

The input data interface includes the following signals:

- **DataIn**: Input uncompressed samples, or header values from a pre-processor, D_GEN bits wide. The actual number of valid bits is determined as follows:
    - When **DataIn** is a header (**IsHeaderIn** high):

        - The number of valid bits is specified by the input signal **NBitsIn.**
    - When **DataIn** is a raw or pre-processed sample to be encoded (**IsHeaderIn** low):
        - D_GEN when EN_RUNCFG = 0.
        - The configurable value D when EN_RUNCFG = 1. If D_GEN > D, the valid bits are placed in the least significant bits of **DataIn**.
- **DataIn_NewValid**: Indicates the presence of new valid values (active high) in **DataIn** for one **Clk_S** cycle.
- **NbitsIn: I**nforms the IP core about the number of valid bits in the **DataIn** signal. **NbitsIn** is only used when the IP block coder is used as an external entropy coder of a pre-processor such as the CCSDS123 IP.
- **IsHeaderIn:** Indicates the IP core that the data in **DataIn** corresponds to a header, with **NbitsIn** valid bits. **IsHeaderIn** is only used when the IP block coder is used as an external entropy coder of a pre-processor such as the CCSDS123 IP.

3.4.1.3 Output data interface

The output data interface includes the following signals:

- **DataOut:** Output bitstream, W_BUFFER_GEN bits wide. The actual number of valid bits is determined as follows:
    - W_BUFFER_GEN when EN_RUNCFG = 0.
    - The configurable value W_BUFFER when EN_RUNCFG = 1. If W_BUFFER_GEN > W_BUFFER, the valid bits are placed in the least significant bits of **DataOut**.
- **DataOut_NewValid:** Indicates the presence of new valid values (active high) in **DataOut** for one **Clk_S** cycle.

3.4.1.4 Control interface

The signals in the control interface inform the source about the state of the IP core.

- **ForceStop:** If asserted (high) the IP core shall stop compressing the current image and go back to the IDLE state. The source might then start the compression of new data.
- **AwaitingConfig**: If asserted (high) the IP core is waiting to be configured. The output is clocked out on the rising **Clk_S** edge.
- **Ready**: If asserted (high) the IP core has been configured and is able to receive new samples for compression. If de-asserted (low) the IP is not available to receive new samples. If the IP is unavailable to receive new samples, **Ready** is de-asserted with sufficient time to prevent data losses. The output is clocked out on the rising **Clk_S** edge.
- **FIFO_Full**: If asserted (high) there was an attempt to write in a full input FIFO. The output is clocked out on the rising **Clk_S** edge.
- **EOP**: If asserted (high) the compression of the last sample has started. The output is clocked out on the rising **Clk_S** edge.

- **Finished**: If asserted (high) the IP core has finished processing all samples in the image. The output is clocked out on the rising **Clk_S** edge.
- **Error**: If asserted (high), there has been an error during the compression, for instance, a configuration error. The output is clocked out on the rising **Clk_S** edge.
- **Ready_Ext**: If asserted (high), the output receiver is ready to receive the output samples. If de-asserted the compression IP shall stop the compression and inform the source that it is unavailable to receive more samples using signal **Ready**.

3.4.1.5 AMBA AHB slave interface

The AMBA AHB slave interface is used to receive the runtime configuration parameters of the IP core. For detailed information of the records used for the AMBA AHB interface see [AD-6] and [AD-7].

The AMBA SLAVE interface shall respond with either HRESP = “00” (OKAY) or “01” (ERROR). Error, Split and Retry are not supported. The width of the AMBA AHB data vectors is the default value (CFG_AHBDW = 32).

3.4.2 Signal overview figure

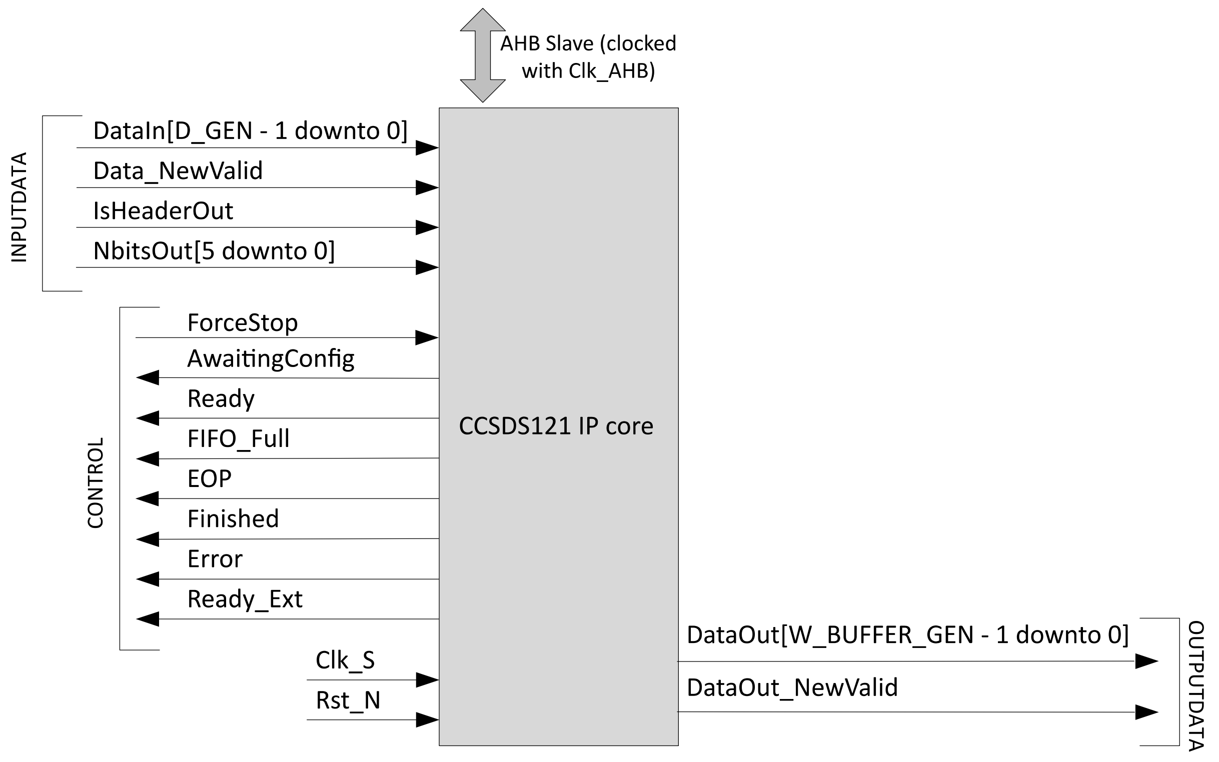


Figure 3-4: CCSDS121 IP - Core signal overview figure.

3.4.3 Signal description

TABLE 3-12: CCSDS121 IP - CORE INTERFACES

Signal	Type	Mode	Description	Clock domain	Active	Value after reset
System Interface						
Rst_N	Std_Logic	In	Synchronous/Asynchronous reset.	Clk_S	Low	High
Clk_S	Std_Logic	In	System clock.	Clk_S	N/A	N/A*
Data Input Interface						
DataIn	Std_Logic_Vector (D_GEN - 1 downto 0)	In	Uncompressed samples.	Clk_S	N/A	N/A
DataIn_NewValid	Std_Logic	In	Flag to validate input signals.	Clk_S	High	N/A
NbitsIn	Std_Logic_Vector(5 downto 0)	In	Number of valid bits in **DataOut** signal.	Clk_S	N/A	N/A
IsHeaderIn	Std_Logic	In	The received data is a header.	Clk_S	High	N/A
Data Output Interface						
DataOut	Std_Logic_Vector (W_BUFFER_GEN - 1 downto 0)	Out	Compressed samples.	Clk_S	N/A	Low
DataOut_NewValid	Std_Logic	Out	Flag to validate output data.	Clk_S	High	Low
Control Interface						
ForceStop	Std_Logic	In	Force the stop of the compression.	Clk_S	High	N/A
AwaitingConfig	Std_Logic	Out	The IP core is waiting to receive the configuration through the AMBA AHB Slave bus.	Clk_S	High	Low
Ready	Std_Logic	Out	If asserted (high) the IP core has been configured and is able to receive new samples for compression. If de-asserted (low) the IP is not ready to receive new samples.	Clk_S	High	Low
FIFO_Full	Std_Logic	Out	Signals that there was an attempt to write in a full input FIFO.	Clk_S	High	Low
EOP	Std_Logic	Out	Indicates that the compression of the last sample has started.	Clk_S	High	Low
Finished	Std_Logic	Out	The IP has finished compressing all samples in the image.	Clk_S	High	Low
Error	Std_Logic	Out	There was an error during the compression	Clk_S	High	Low

Ready_Ext	Std_Logic	In	The output is ready to receive more compressed samples.	Clk_S	High	N/A
AMBA AHB Slave Interface						
Clk_AHB	Std_Logic	In	AHB clock.	N/A	N/A	N/A
Reset_AHB	Std_Logic	In	AHB reset.	N/A	Low	N/A
AHBSlave121_In	AHB_Slv_In_Type	In	AHB slave input signals.	Clk_AHB	N/A	N/A
AHBSlave121_Out	AHB_Slv_Out_Type	Out	AHB slave output signals.	Clk_AHB	N/A	Low

**N/A stands for Not Applicable.*

3.4.4 State after reset

The IP core is in IDLE state and asserts signal **AwaitingConfig** after reset, indicating that it is ready to receive the configuration values. It stays in that state until the configuration is received through the AMBA AHB Slave interface. Reset is active low and might be synchronous or asynchronous, according to the user-defined generic parameter RESET_TYPE (0 for asynchronous; 1 for synchronous).

3.4.5 Initialization

The IP core is in IDLE state and assert signal **AwaitingConfig** after reset, indicating that it is ready to receive the configuration values. The IP receives first the configuration through the AMBA AHB Slave interface. The read configuration registers are stored in internal registers that are shared among the different sub-modules that conform the IP. After the configuration, the IP shall be given the command ENABLE = 1 in the least significant bit of the Control&Status memory-mapped register (see Section 3.2.5.2.2). Once the IP is configured, it asserts signal **Ready**, indicating that it is ready to receive samples. Additionally, it starts issuing the header values, which are packed and sent through the output interface using W_BUFFER bits. If there are header bits remaining in the packer, the IP attaches them to the first bits of the bitstream. The IP core compresses the received samples block by block until all the necessary samples have been processed. The explained process is shown in the chronogram in Figure 3-5.

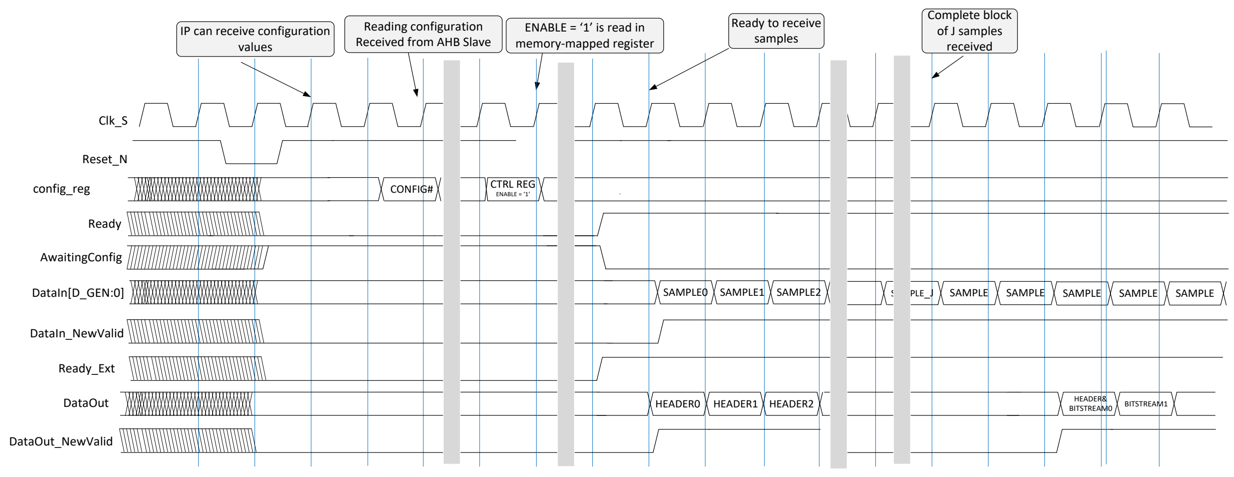


Figure 3-5: Initialization with synchronous reset.

3.4.6 Error handling

The CCSDS121 IP core detects and inform about the following errors:

- Invalid configuration value is detected:
    - Nx, Ny or Nz = 0.
    - Values are not in the range allowed by the corresponding constants.
    - W_BUFFER < D*.
    - W_BUFFER_GEN < D_GEN*.
- FSM enters an invalid state.

** These restrictions constitute the theoretical worst case in order to process a set of input samples with dynamic range D, assuming a processing performance of one input sample per clock cycle. However, it has been observed that with W_BUFFER values equal or slightly greater than D, the compressor may generate a wrong compressed bitstream under certain circumstances, namely when the coding options "no compression" or "sample splitting" with values of k close to D are selected. In these situations, the performance of the output packing for a given block of input samples may be slower than the rate at which input samples are read, making some blocks of samples to be skipped. This is further discussed in section 3.5.4.1. In summary, it is recommended using values of W_BUFFER greater than D.*

When an error is detected, the IP produces a high value in signal **Error** in the next rising edge of **Clk_S**, and shall then goes to the IDLE state and asserts the **AwaitingConfiguration** signal. A new configuration can be then set.

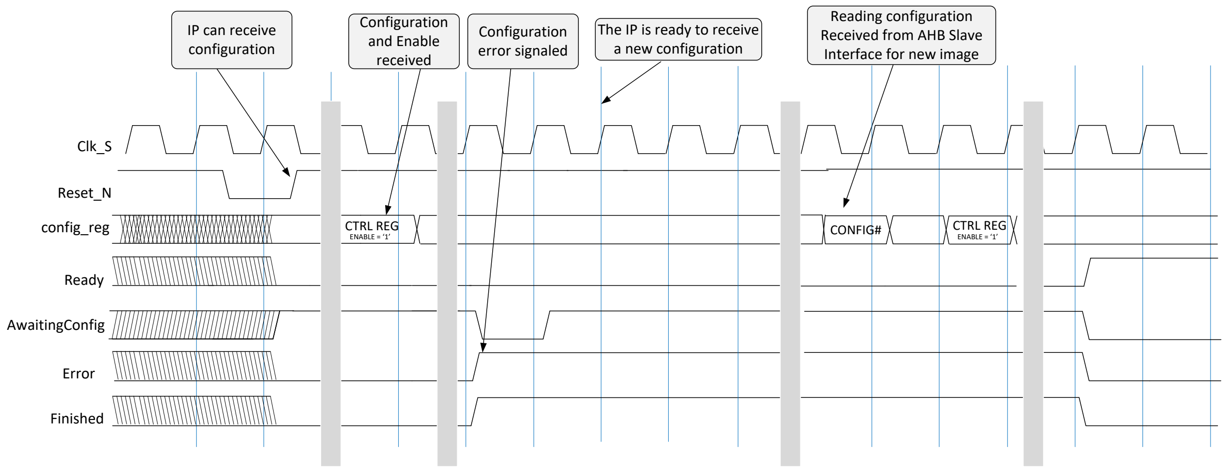

IP can receive configuration
Configuration and Enable received
Configuration error signaled
The IP is ready to receive a new configuration
Reading configuration Received from AHB Slave Interface for new image
Clk_S
Reset_N
config_reg
Ready
AwaitingConfig
Error
Finished
CTRL REG
ENABLE = '1'
CONFIG#
CTRL REG
ENABLE = '1'

Figure 3-6: A configuration error is detected and the compression of a new image is started.

The error code field in the Control&Status memory-mapped register informs about the cause of the error.

TABLE 3-13: CCSDS121 IP - ERROR CODES

Error code	Meaning
0000	No error.
0001	Nx, Ny or Nz = 0.

0010	Values are not in the range allowed by the corresponding constants.
0011	W_BUFFER < D.
0011	W_BUFFER_GEN < D_GEN.
0100	Invalid FSM state.

3.4.7 Forcing the IP core to stop

The control signal **ForceStop** can be used to stop the compression at any moment. If a high value is detected in the positive edge of the clock, the IP asserts signal **Finished** in the next clock cycle and then returns to the IDLE state. No more compressed data are produced after **ForceStop**. The explained process is shown in the chronogram in Figure 3-7.

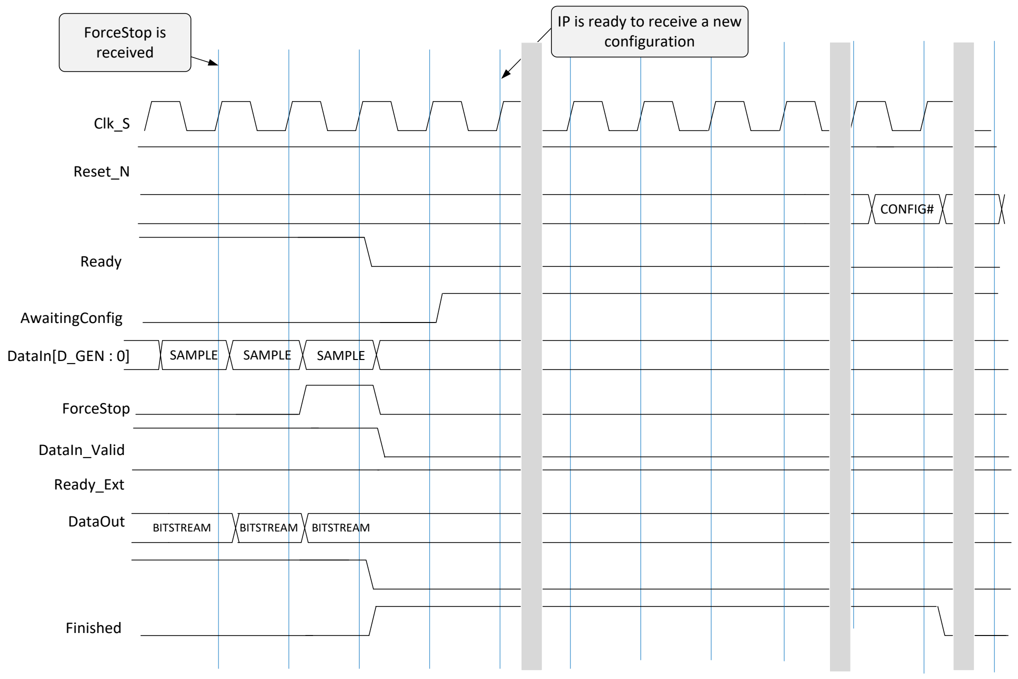


Figure 3-7: ForceStop is detected by the IP. It asserts Finished and one cycle after it is ready to receive a new configuration.

3.4.8 Output not ready to receive compressed samples

If the output receiver is not ready to receive compressed samples, it shall de-assert signal **Ready_Ext.** This value is then propagated to the **Ready** signal and the compression is stopped. An example is shown in Figure 3-8.

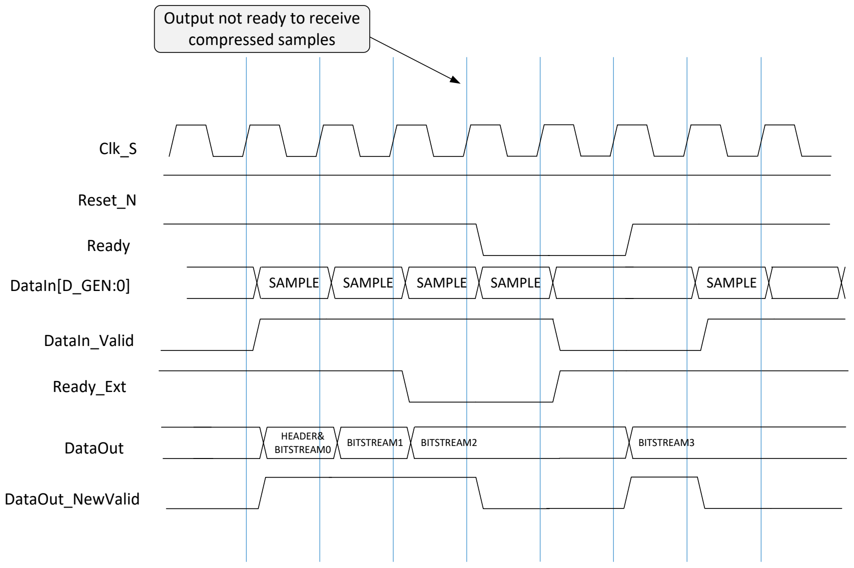


Figure 3-8: Output is not ready to receive compressed samples for two clock cycles.

3.4.9 Finishing the compression without errors

When the IP has received all the samples to be compressed, **Ready** is de-asserted. When the last sample received starts begin compressed **EOP** is asserted. Later, when such sample has been processed, the IP sends the encoded sample and flushes the output packer buffer, which might contain remaining bits. After flushing the buffer, signal **Finished** is asserted, indicating that no more valid bitstream values are produced at the output. The IP goes then to IDLE state and asserts the **AwaitingConfig** signal to receive a new configuration. The value in signal **Finished** is kept until the new configuration values are received.

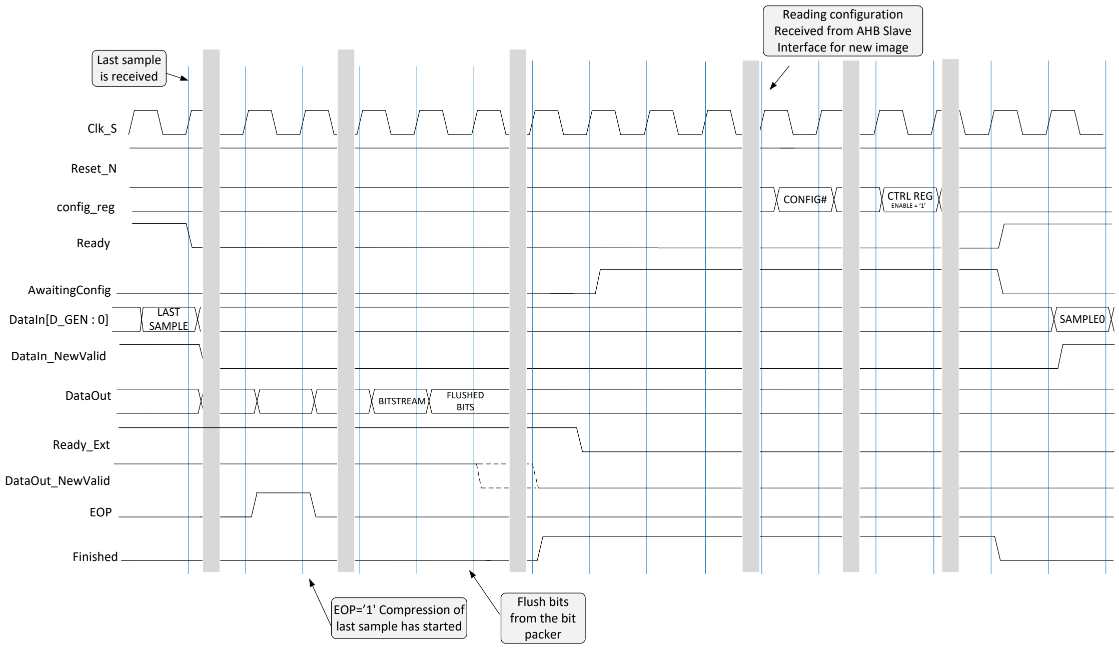


Figure 3-9: Last sample is received and the compression of a new image is started.

3.5 Design description

This section presents a high-level representation in the form of block diagrams of the VHDL description of the CCSDS121 IP core, in order to facilitate the understanding of the delivered sources and the interpretation of the results when mapping the IP core to different technologies.

3.5.1 Simplified block diagram of the IP top module

First, an overview of the full CCSDS121 IP connectivity can be seen in Figure 3-10. Two main modules compound the IP core: the **predictor** and the **block coder**.

The block coder performs the entropy coding of the samples (which may be pre-processed or not, depending on the PREPROCESSOR configuration option). In addition, the block coder receives the runtime configuration from the AHB bus and disseminates the configuration to the predictor module (if present). It is also in charge of generating the output bitstream.

On the other hand the predictor, which is included only when the PREPROCESSOR configuration option is set to 2, is in charge of pre-processing the incoming input samples. The runtime configuration of this module is received from the AHB bus through the block coder. If the predictor is present, the data input of the block coder module is connected to the output of the predictor. Otherwise, the IP data inputs are directly connected to the block coder inputs.

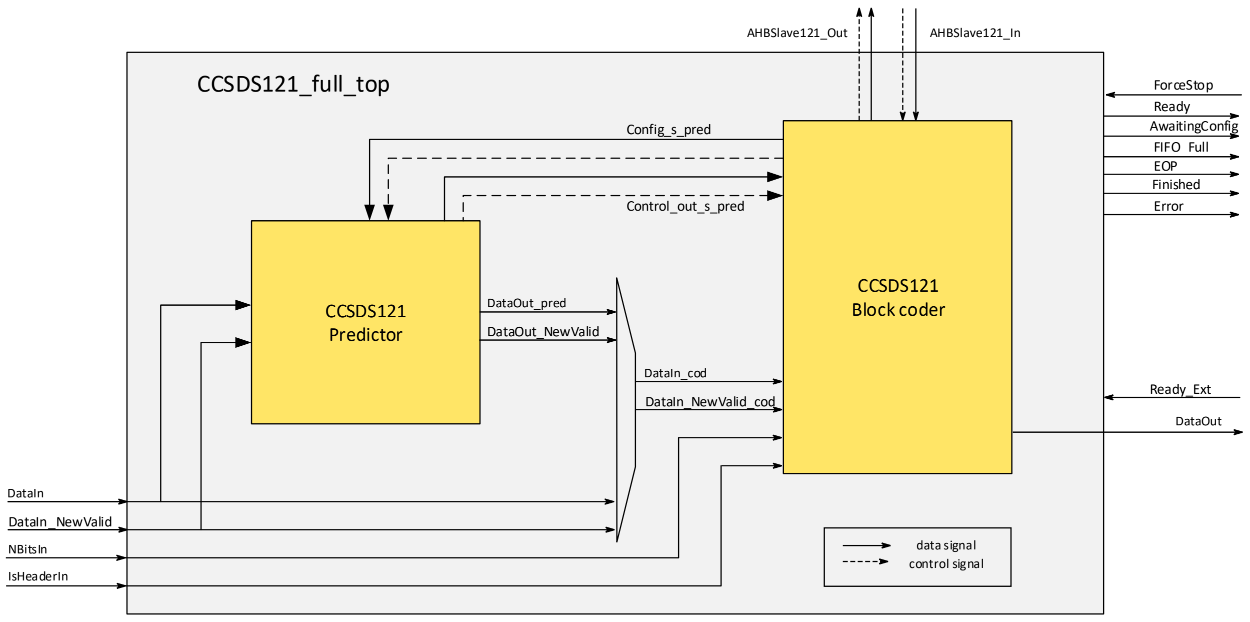


Figure 3-10: CCSDS121 IP - Simplified block diagram of the top module

The following subsections explain in more detail each one of these modules.

3.5.2 Simplified block diagram of the unit-delay predictor top module

The top module of the CCSDS121 unit-delay predictor core, depicted in Figure 3-11, includes the necessary logic to bind the components that perform the reception of input samples, the prediction step and the flow control of the output mapped prediction residuals and periodic reference samples.

The runtime configuration is received from the AHB bus through the block-coder top module (section 3.5.3), together with a valid flag. According to the received configuration, the **fsm** module controls the operation of the rest of the modules in the design.

The **components** module performs the pre-processing step. It contains a **unit-delay predictor** which computes the predicted samples based on the input data, and a **mapper** module which maps the prediction residuals. In order to insert the reference sample periodically according to the CCSDS 121 standard, the **fsm** may bypass the pre-processing step by activating a **Bypass** signal.

The predictor top includes input and output FIFOs in order to adapt the data transfers between modules. The input FIFO stores the incoming input data until they are pre-processed, while the output FIFO stores the pre-processed data until the block coder requires them.

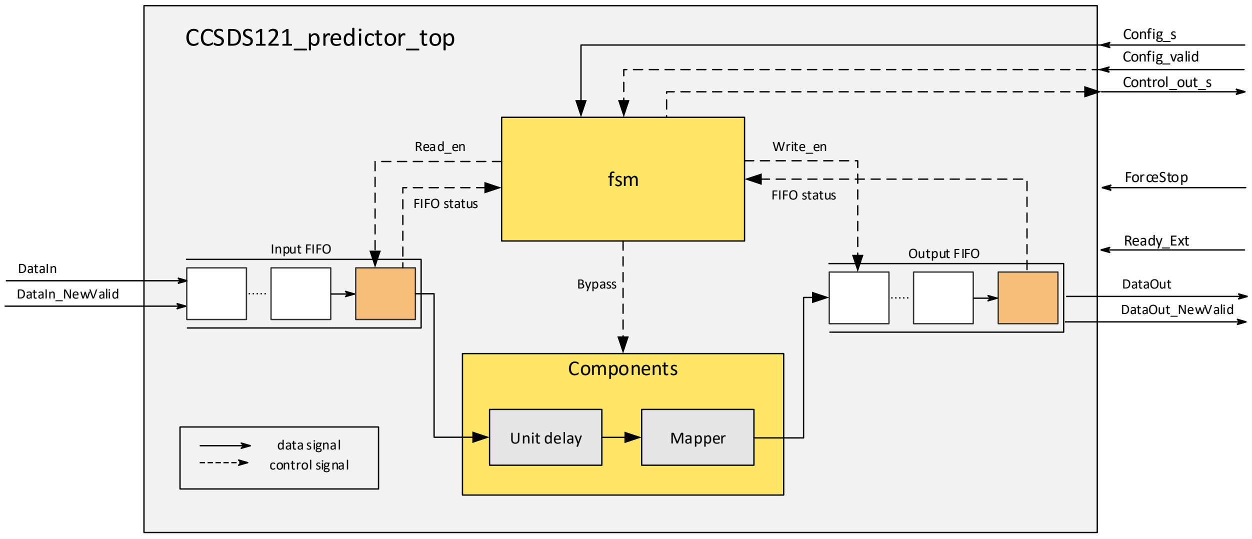


Figure 3-11: CCSDS121 IP – Simplified block diagram of the unit-delay predictor top module

3.5.3 Simplified block diagram of the block-coder top module

The top module of the CCSDS121 block-coder core, depicted in Figure 3-12, includes the necessary logic to bind the components that perform the reception of runtime configuration values and input samples, the entropy coding stage and the flow control of the output bitstream.

Runtime configuration values are received by the **ahbslv** module, and then adapted to the CCSDS121 IP core's clock frequency by the **clk_adapt** module. These values are sent to the rest of modules in the design (including the pre-processor, if present), together with a valid flag.

The **DataIn** input signal might contain header values from a pre-processor or samples to be compressed (either raw or pre-processed samples). In the former case, the **IsHeaderIn** input signal is asserted, and the **NBitsIn** signal indicates the number of valid bits in the header. The input values are arranged in different FIFOs, and read by the compression components when necessary. It must be noted that both **NBitsIn** and **IsHeaderIn** inputs are used only when the CCSDS121 predictor is not present (PREPROCESSOR_GEN value different to 2).

The **components** module includes the units that validate the correctness of the received configuration values, generate the header and perform the compression of the mapped residuals according to the CCSDS 121 standard. The scheduling of the operations is performed by the **fsm** module, which contains several finite-state machines in charge of enabling the different compression units in the **components** module, creating a highly-optimized pipeline which allows for compressing a sample every clock cycle.

The output bitstream is finally stored in a FIFO and sent to the output upon availability of the receiver, determined by the assertion of the **Ready_Ext** input.

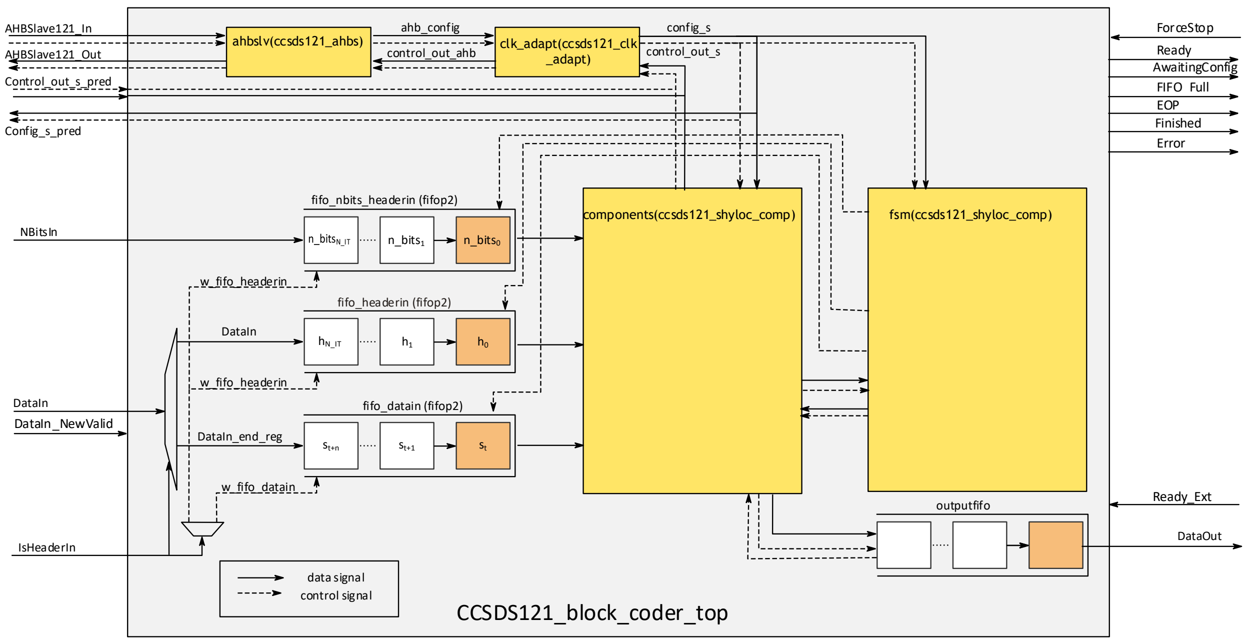


Figure 3-12: CCSDS121 IP – Simplified block diagram of the block-coder top module.

3.5.4 Simplified block diagram of the block-coder core

In Figure 3-13, an abstract representation of the components that perform the entropy coding of the input samples can be found.

The runtime configuration values, after being adapted to the IP core's clock frequency, are read and validated by the **int** module. The **int** module has a twofold purpose. On one hand, it assigns the configuration values, either from the ones received by the AHB slave interface (input port **config_in**), if runtime configuration is enabled (EN_RUNCFG = 1); or from the user-selected parameters (see Section 3.2.5.1), if the runtime configuration is disabled (EN_RUNCFG = 0). On the other hand, it checks that the selected configuration values are correct, and raises an error otherwise.

The **header_gen** module will generate the header values and then send them to the **packing_final** module. This module contains a register buffer of the bit width of the **DataOut** signal where the input values are packed. Once the buffer is full, the packing module generates a valid flag so that the value can be captured at the output.

The rest of the modules in Figure 3-13 constitute the compression engine, which performs the Rice coding functionality as described in Section 3.2.1. The input samples, which can be either raw samples internally pre-processed or already mapped prediction residuals resulting from a previous pre-processing stage, are compressed in a block by block basis. For each block of J samples, several compression options are evaluated concurrently and the length of the encoded block for each option is calculated. The following units are shown in the block diagram:

- **snd_extension**: computes the length of a block when encoded with the second extension option, and transforms each pair of pre-processed samples δ_i and $\delta_{(i+1)}$ into a single new symbol γ.
- **compute_l_k**: computes the length of a block when encoded with the several sample splitting option. The number of options to be evaluated depends on the dynamic range of the input samples and the user-selected codeset parameter. The VHDL module is scaled according to this values. The length of consecutive k-options is subtracted in order to find the first negative value, which marks the location of the minimum. The minimum length is stored in a register, denoted $L_k(winner)$ in Figure 3-13.
- **optioncoder**: this module compares the length of the different encoding options and selects the one that yields the shortest codeword.
- **mapped_fifo and gamma_fifo**: these FIFOs store the residuals to be compressed and the gamma values while the several encoding options are evaluated, to be then encoded with the selected option.
- **fscoder**: encodes the mapped or gamma values using the FS sequence defined in Section 3.2 of the CCSDS 121 standard [AD-2].
- **split**: splits the FS sequence in several chunks, each with a bit width smaller than the bit width of the output buffer.
- **splitpacker**: packs the spit bits of the sample-splitting option, so that they are correctly arranged according to the coded data set format described in Section 5.1.4 of the CCSDS 121 standard [AD-2].
- **fifosplit**: stores the split bits of the sample-splitting option while the FS sequence is sent to the output, so that they are attached afterwards, as described in Section 5.1.4 of the CCSDS 121 standard [AD-2].

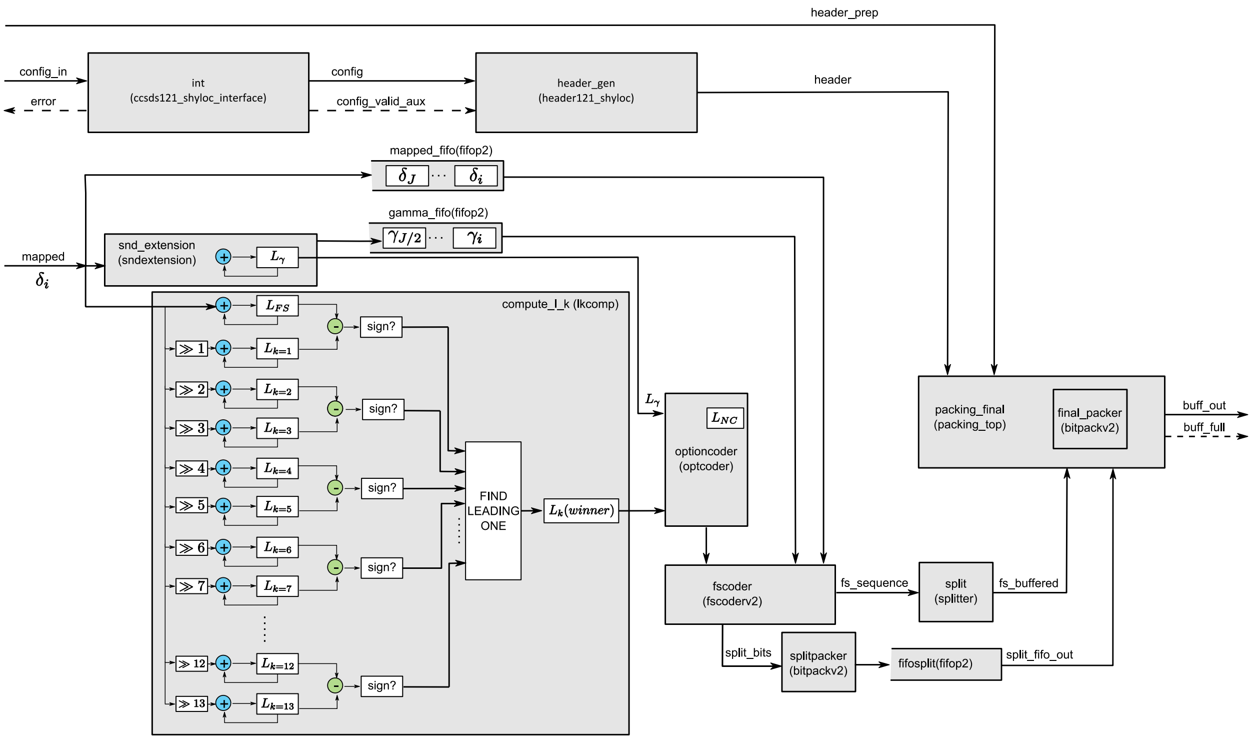


Figure 3-13: CCSDS121 IP – Simplified block diagram of the compression core.

3.5.4.1 Some considerations on the block-coder processing performance

The scheduling of the different units in the block-coder core (**components** module) is performed by several finite-state machines implemented in the **fsm** module. In particular, the datapath for the pre-processed input samples is governed by 3 FSMs:

- FSM1, which controls the reading of input samples and the selection of the best coding option. This FSM has a fixed operation period of *J* clock cycles.
- FSM2, which controls the reading of the mapped and gamma FIFOs, as well as the computation of the FS sequence for the chosen coding option. This FSM has a fixed operation period of *J* clock cycles too.
- FSM3, which controls the splitter modules and the final packing. This FSM has a variable operation period, depending on the length of the output data in relation with the width of the output buffer.

These FSMs operate in a cascade fashion, in such a way that when the operation of an FSM finishes, the operation of the next FSM is triggered. At the same time, when an FSM completes its operation, it immediately begins processing the next block of samples with the aim of optimizing performance. As there is no backward synchronization, it is required the operation period of FSM3 to be equal or shorter than *J* in order to guarantee a correct operation.

Let us take a closer look on the operation period of FSM3. It has a fixed and variable component, corresponding to the packing of the FS sequence and the sample splits, respectively. For the packing of the FS sequence, a number of clock cycles high enough to pack the longest possible sequence is reserved. This value depends just on configuration parameters (namely block size *J,* dynamic range *D* and the output buffer *W_BUFFER*) and it is typically between 2 and 5. On the other hand, the number of cycles required to pack the sample splits depends on the actual coding option, being equal to $\left\lceil\frac{k\cdot J}{W\_BUFFER}\right\rceil$, where $\lceil\,\rceil$ denotes the ceiling operation. For this computation, it is considered a value $k = 0$ when either the "FS sequence", "second extension" or "zero block" coding options are chosen, and $k = D$ is assumed with the option "no compression". From here can be deduced that, when the packing of the sample splits lasts a number of cycles close to J (that is, when both k and *W_BUFFER* are close to *D*) the operation period of FSM3 becomes longer than FSM1, making that FSM3 will not be ready when the next block of samples arrives, skipping it in the process.

In order to avoid this situation, it is sufficient to use configurations with *W_BUFFER* wide enough. However, if the user needs a specific configuration with *W_BUFFER* close to *D*, this situation could be solved by means of slight changes in the fsm module. In particular, once a coding option is chosen, the operation period of FSM3 can be easily computed. If this period is greater than J, then the FSM1 should be interrupted an amount of cycles enough to synchronize the period of all the three FSMs.

3.6 Implementation performance

This section includes implementation results of the CSCSDS121 IP core in different technologies with different sets of generic parameters, defined in Table 3-14. All these results have been obtained using Synopsys Synplify Premier N-2018.03, except for NanoXplore technology where most recent version of NanoXmap tools was used (2.9.2).

TABLE 3-14: CCSDS121 IP - SET OF PARAMETERS USED TO STUDY THE IMPLEMENTATION PERFORMANCE

Generic	Set1	Set2	Set3	Set4	Set5	Set6	Set7
EN_RUNCFG	0	0	0	1	0	1	1
RESET_TYPE	1	1	1	1	1	1	1
PREPROCESSOR_GEN	0	0	0	0	2	2	2
Nx_GEN	1024	1024	1024	1024	1024	1024	1024
Ny_GEN	1024	1024	1024	1024	1024	1024	1024
Nz_GEN	1024	1024	1024	1024	1024	1024	1024
D_GEN	16	16	16	16	32	16	32
ENDIANESS_GEN	0	0	0	0	0	1	0
IS_SIGNED_GEN	0	0	0	0	0	0	1
DISABLE_HEADER_GEN	0	0	0	0	0	0	0
J_GEN	16	32	64	16	32	16	16
CODESET_GEN	0	0	0	0	0	0	0
REF_SAMPLE_GEN	256	256	256	256	128	256	256
W_BUFFER_GEN	32	32	32	32	64	32	64

3.6.1 Implementation performance on one-time programmable FPGAs

Implementation results of the CSCSDS121 IP core in one-time programmable RTAX4000S are shown in Table 3-15.

TABLE 3-15: CCSDS121 IP - IMPLEMENTATION PERFORMANCE ON RTAX4000S.

Parameters	Total Resources	Set 1	Set 2	Set 3	Set 4	Set5	Set6	Set7
I/O	840	168	168	168	168	209	161	209
Combinatorial	40320	5435	6455	7891	7494	12971	8974	14461
Sequential	20160	1211	1385	1690	1565	2062	1684	2333
Core RAM Blocks	120	6	6	6	8	10	8	10
Maximum Frequency (Clk_AHB) (MHz)		162.0	163.2	160.9	153.4	162.0	148.9	132.9
Maximum Frequency (Clk_S) (MHz)		41.8	38.3	34.7	37.3	35.7	39.0	35.6

3.6.2 Implementation performance on programmable FPGAs

Implementation results of the CSCSDS121 IP core in Virtex5 FX 130 and Virtex5QR are shown in Table 3-16 and Table 3-17 respectively.

TABLE 3-16: CCSDS121 IP - IMPLEMENTATION PERFORMANCE ON VIRTEX5 (XC5VFX130T).

Parameters	Total Resources	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7
I/O	840	204	204	204	204	252	204	252
BUFGs	32	2	2	2	2	3	3	3
Block RAMs	298	0	1	5	0	1	0	0

DSP48	320	1	1	1	3	1	4	5
Registers	81920	1170	1267	1597	1507	1976	1584	2291
LUTs	81920	2988	3541	4675	3573	6646	4395	7660
Maximum Frequency (Clk_AHB) (MHz)		290.4	250.2	245.2	249.3	261.1	227.2	248.6
Maximum Frequency (Clk_S) (MHz)		115.0	107.6	94.6	107.6	97.3	104.0	81.4

TABLE 3-17: CCSDS121 IP - IMPLEMENTATION PERFORMANCE ON VIRTEX5QR (XQR5VFX130).

Parameters	**Total Resources**	**Set 1**	**Set 2**	**Set 3**	**Set 4**	**Set 5**	**Set 6**	**Set 7**
I/O	836	204	204	204	204	252	204	252
BUFGs	32	2	2	2	2	3	3	3
Block RAMs	298	0	1	5	0	1	0	0
DSP48	320	1	1	1	3	1	4	5
Registers	81920	1160	1269	1597	1505	1970	1563	2291
LUTs	81920	2952	3524	4601	3551	6637	4381	7670
Maximum Frequency (Clk_AHB) (MHz)		290.4	250.2	245.1	249.3	273.9	228.2	248.6
Maximum Frequency (Clk_S) (MHz)		115.0	107.6	93.4	106.3	98.1	104.0	79.9

Synthesis results for Microsemi devices have been also considered for ProASIC3E and RTG4, and summarized in Table 3-18 and Table 3-19.

TABLE 3-18: CCSDS121 IP - IMPLEMENTATION PERFORMANCE ON PROASIC3E (A3PE3000).

Parameters	Total Resources	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7
I/O Cells	620	168	168	168	168	209	161	209
Core Cells	75264	9357	11157	13816	12330	23984	15248	26857
Block RAMs	112	11	11	11	11	19	11	19
Maximum Frequency (Clk_AHB) (MHz)		86.3	92.3	85.3	85.5	88.7	80.4	78.7
Maximum Frequency (Clk_S) (MHz)		33.7	32.2	27.1	17.8	28.3	15.5	15.2

TABLE 3-19: CCSDS121 IP - IMPLEMENTATION PERFORMANCE ON RTG4 (RTG4 150).

Parameters	Total Resources	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7
IO Cells	720	162	162	168	168	209	161	209
Carry Cells	151824	786	983	1138	962	2373	1290	2865
Sequential Cells	151824	974	1150	1504	1319	1672	1342	1918
Block RAMs (RAM64x18)	209	9	10	19	11	21	11	19

DSP Blocks	462	3	3	3	4	4	5	5
LUTs	151824	4312	4737	6132	5365	9060	6648	11055
Maximum Frequency (Clk_AHB) (MHz)		134.9	141.4	144.0	138.0	131.1	125.1	140.0
Maximum Frequency (Clk_S) (MHz)		66.5	68.4	60.0	62.2	56.8	52.5	46.0

Finally, implementation results for NanoXplore technologies are obtained for NG-MEDIUM NX1H35S and NG-LARGE NX1H140TSP and shown in Table 3-20 and Table 3-21, respectively.

TABLE 3-20: CCSDS121 IP - IMPLEMENTATION PERFORMANCE ON NG-MEDIUM NX1H35S.

Parameters	Total Resources	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7
Carry Cells	8064	1714	1795	1804	1925	5207	2297	5660
Registers	32256	1194	1366	1683	1463	2575	1550	2670
Block RAMs (48Kb)	56	9	9	9	11	20	13	20
DSP Blocks	112	1	1	1	2	1	2	2

LUTs	32256	4063	4616	5774	4747	9489	5916	10309
Maximum Frequency (Clk_AHB) (MHz)		106.4	118.6	124.0	95.6	100.8	102.2	80.9
Maximum Frequency (Clk_S) (MHz)		30.2	32.7	29.0	28.1	24.9	25.8	21.9

TABLE 3-21. CCSDS121 IP- IMPLEMENTATION PERFORMANCE ON NG-LARGE NX1H140TSP.

Parameters	Total Resources	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7
Carry Cells	32256	1714	1795	1804	1925	5207	2297	5660
Registers	129024	1194	1366	1683	1463	2575	1550	2670
Block RAMs (48Kb)	192	9	9	9	11	20	13	20
DSP Blocks	384	1	1	1	2	1	2	2
LUTs	129024	4063	4616	5774	4747	9489	5916	10309

Maximum Frequency (Clk_AHB) (MHz)		99.0	116.1	104.5	87.3	79.6	77.8	83.4
Maximum Frequency (Clk_S) (MHz)		30.7	32.9	29.4	27.6	25.4	25.2	24.1

3.6.3 Implementation performance on ASIC

Table 3-22 shows implementation results for the DARE 180 Standard Cell ASIC Library. These results correspond only to the CCSDS121 block coder implementation and they are not updated in the scope of this extension. The conditions where these results were obtained are:

- Synopsys Design Compiler M-2016.12-SP4 for 64-bit Linux, with conservative optimizations;
- Clk_AHB is constrained to 10 ns (100 MHz);
- Clk_S is constrained to 2.5 ns (400 MHz);
- Memories smaller than 64 words are mapped to Flip-Flops;
- Memories larger than 64 words are mapped into SRAM macros;
- No DFT nor BIST mechanisms have been synthesized;
- No EDAC circuitry has been included for the memories.

Table 3-22 shows results for synthesis with and without SRAM memories (e.g. FIFOs). These results are presented in order to give an idea of the complexity of the IP core, and the amount of memory required for each Set. Discrepancies in gate count between the scenarios are due to memories smaller than 64 words being mapped into FFs. Only a few SRAM macros were available for synthesis, therefore all the needed memory aspect ratios had to be built from this limited set. This means there is an area overhead on the area results below, which should be taken as an upper bound, i.e. synthesis with appropriate memory views should yield lower area.

TABLE 3-22: CCSDS121 IP - IMPLEMENTATION AND PERFORMANCE FIGURES ON DARE 180 STANDARD CELL ASIC LIBRARY

	Parameters	**Set 1**		**Set 2**		**Set 3**		**Set 4**	
		Without Memories	**With Memories**	**Without Memories**	**With Memories**	**Without Memories**	**With Memories**	**Without Memories**	**With Memories**
	Gate Count (kGates)	30.41	55.42	34.57	63.02	30.41	55.42	34.57	63.02
	Area (mm^2)	1.15	2.55	1.31	3.28	1.15	2.55	1.31	3.28
Clk_S Max. Freq. (MHz)	234.19	194.17	231.48	153.85		234.19	194.17	231.48	153.85

4 CCSDS123 IP CORE DATASHEET

4.1 System overview

The CCSDS123 IP Core performs the compression of a set of input samples according to the specifications of the CCSDS 123 standard [AD-1].

The input samples might be received from different sources: a mass memory, a SpW interface, an ADC or another IP core. The CCSDS123 IP core does not implement any particular communication protocol. The control signals implement a simple handshake protocol to ensure synchronization with the source.

A typical system using the designed IP core looks similar to Figure 4-1.

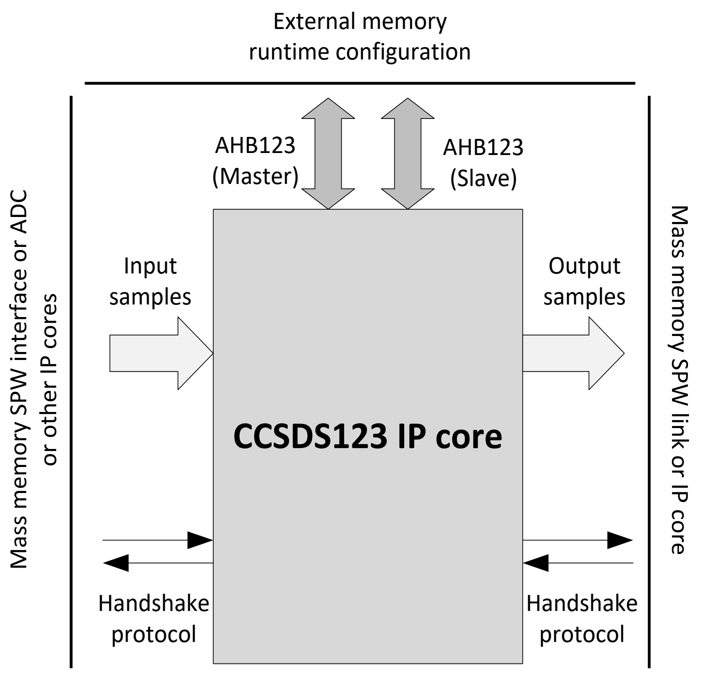


Figure 4-1: CCSDS123 IP - Compression core in a system.

The CCSDS123 IP core can be configured with a wide set of configuration parameters which can be selected at implementation or at runtime. Those parameters that can be selected at runtime are received by the IP core through the AMBA AHB slave interface.

Additionally, an AMBA AHB master interface is provided to connect to an external memory to store intermediate values for compression.

The data stored in the external and internal memories can be protected by EDAC by setting a configuration parameter.

The compressed samples are sent through the output data interface. Similarly to the input, the output might send values to a mass memory, SpW interface or IP core. The output data are sent together with a valid flag.

The output data represent the bitstream (compressed input data), according to the specifications in the CCSDS 123 standard, including the necessary side-information (header) to allow the decompression of the data.

The CCSDS123 IP includes the implementation of the predictor and sample-adaptive entropy coding option as described by the CCSDS 123 standard algorithm. Additionally, it is possible to connect the CCSDS121 block coder to perform the block-adaptive entropy coding option. In this case, the CCSDS123 IP is configured at implementation time to perform only the prediction stage of the CCSDS 123 standard algorithm. The CCSDS123 IP produces the corresponding header and the mapped prediction residuals that are then received by the CCSDS121 block coder, which encodes the residuals and produces the bitstream.

4.2 Functional description

4.2.1 Applicable algorithms

The Recommended Standard defines a payload lossless data compressor that has applicability to multispectral and hyperspectral imagers and sounders. The input to the compressor is a multispectral/hyperspectral image, which is represented as a three-dimensional array of integer sample values.

The output from the compressor is an encoded bitstream from which the input image can be fully recovered. The compressed image is variable-length (depends on the variations in the image contents).

The compressor consists of two functional parts: a predictor and an encoder, as seen in Figure 4-2.

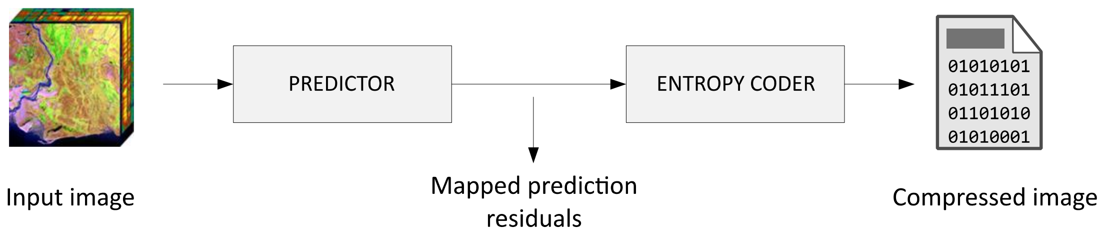


Figure 4-2: CCSDS123 IP - Compressor concept and functional blocks.

The compressed image consists of:

- A header that encodes image constants and compressor parameters.
- A body, produced by an entropy coder which encodes without lossless the mapped prediction residuals.

Within each spectral band, the predictor computes a *local sum* of neighbouring sample values. Each local sum is used to compute a *local difference*. Predicted sample values are calculated using the local sum in the current spectral band and a weighted sum of local difference values from the current and *P* previous spectral bands, where *P* is a tuneable parameter between 0 and 15. The weights used in this calculation are adaptively updated following the calculation of each predicted sample value.
Each prediction residual, that is, the difference between a given sample value $s_{x,y,z}$ and the corresponding predicted sample value $\hat{s}_{x,y,z}$, is mapped to an unsigned integer $\delta_{x,y,z}$, the mapped prediction residual.

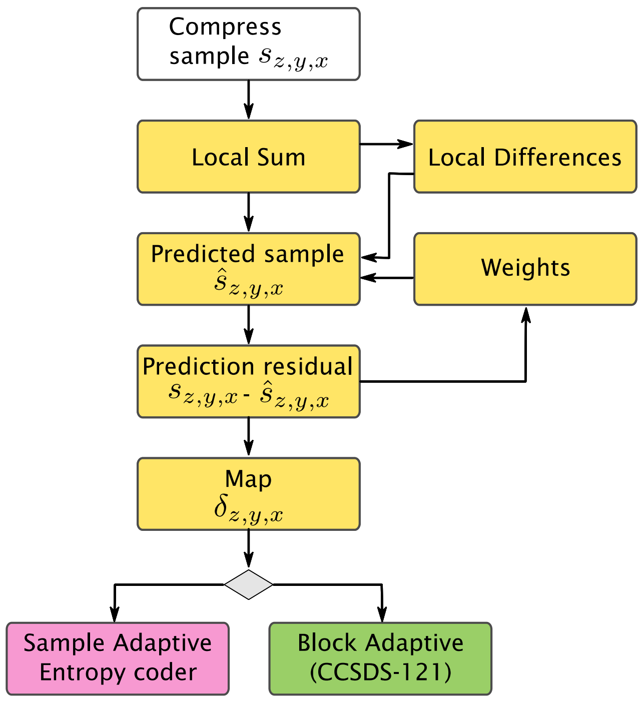

Compress sample $s_{z,y,x}$
Local Sum
Local Differences
Predicted sample $\hat{s}_{z,y,x}$
Weights
Prediction residual $s_{z,y,x}$ - $\hat{s}_{z,y,x}$
Map $\delta_{z,y,x}$
Sample Adaptive Entropy coder
Block Adaptive (CCSDS-121)

Figure 4-3: CCSDS123 IP - Flowchart.

For the compression of a specific sample, $s_{x,y,z}$, a *local sum* $\sigma_{x,y,z}$ of the neighbouring sample values is computed first. A user-defined parameter is employed to select between two possible configurations for the local sum calculation: *column-oriented* and *neighbour-oriented*. The column-oriented local sum is computed using the neighbour on top of the current sample, $s_{x,y-1,z}$; on the other hand, the neighbour-oriented local sum is calculated using 4 neighbouring samples, $s_{x-1,y-1,z}$, $s_{x,y-1,z}$, $s_{x+1,y-1,z}$ and $s_{x-1,y,z}$.

The local sums are used to calculate the *central local differences* values $d_{x,y,z}$ and the *directional local differences* $d^{N}_{x,y,z}$, $d^{W}_{x,y,z}$ and $d^{NW}_{x,y,z}$. The user can choose to perform prediction in *full* or *reduced* mode by selecting the appropriate parameter. Under reduced mode, the prediction is computed from a weighted sum of the central local differences calculated in *P* preceding bands. The directional local differences are not used under reduced mode and therefore do not need to be calculated. On the other hand, under full prediction mode, the prediction depends not only on the central local differences in the *P* previous bands, but also on the weighted sum of the directional local differences in the current band. The three-dimensional neighbourhood used for prediction is depicted in Figure 4-4. The weight values are updated adaptively according to the resulting prediction residual.

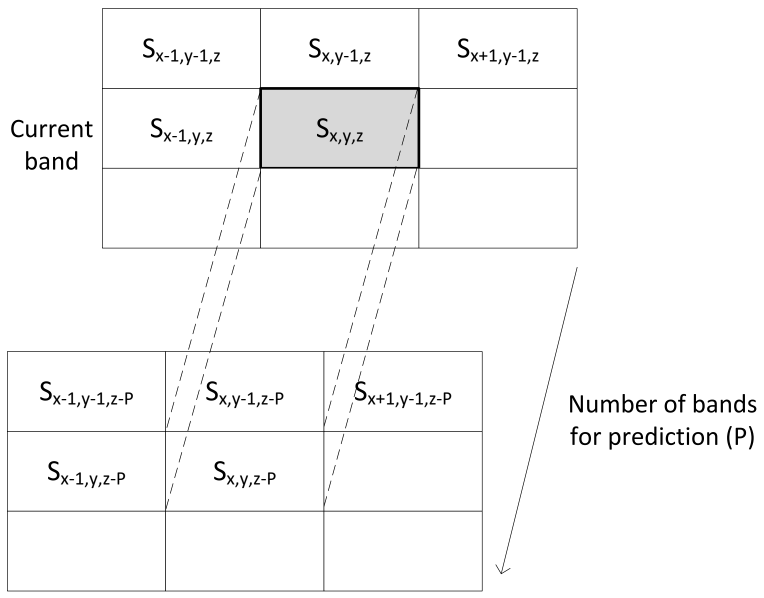


Figure 4-4: CCSDS123 IP - Current sample and neighbours used for computing the local sum and local and directional local differences.

The central local differences in the *P* previous bands together with the three directional local differences conform a vector $U_{x,y,z}$ whose elements are computed according to the selected configuration, as it is shown in Table 4-1.

TABLE 4-1: CCSDS123 IP - LOCAL DIFFERENCES VECTOR FOR FULL AND REDUCED CONFIGURATION OPTIONS

				Local differences vector for band z	
				Neighbour oriented	*Column oriented*
REDUCED PREDICTION	FULL PREDICTION	CENTRAL LOCAL DIFFERENCES		$(4s_{x,y} - s_{x-1,y} + s_{x-1,y-1} + s_{x,y-1} + s_{x+1,y-1})_{z-1}$	$(4s_{x,y} - 4s_{x,y-1})_{z-1}$
				$(4s_{x,y} - s_{x-1,y} + s_{x-1,y-1} + s_{x,y-1} + s_{x+1,y-1})_{z-2}$	$(4s_{x,y} - 4s_{x,y-1})_{z-2}$
				…	…
				$(4s_{x,y} - s_{x-1,y} + s_{x-1,y-1} + s_{x,y-1} + s_{x+1,y-1})_{z-P}$	$(4s_{x,y} - 4s_{x,y-1})_{z-P}$
		DIRECTION LOCAL DIFFERENCES	**N**	$(4s_{x,y-1} - s_{x-1,y} + s_{x-1,y-1} + s_{x,y-1} + s_{x+1,y-1})_{z}$	$(4s_{x,y-1} - 4s_{x,y-1})_{z}$
			W	$(4s_{x-1,y} - s_{x-1,y} + s_{x-1,y-1} + s_{x,y-1} + s_{x+1,y-1})_{z}$	$(4s_{x-1,y} - 4s_{x,y-1})_{z}$
			NW	$(4s_{x-1,y-1} - s_{x-1,y} + s_{x-1,y-1} + s_{x,y-1} + s_{x+1,y-1})_{z}$	$(4s_{x,y-1} - 4s_{x,y-1})_{z}$

The predicted sample, $\hat{s}_{x,\,y,\,z}$, is calculated by performing the dot product of the local differences vector $U_{x,y,z}$ and a weight vector $W_{x,y,z}$. Then, the predicted sample is mapped to positive integer values. Subsequently, the mapped prediction residuals $\delta_{x,\,y,\,z}$ are sequentially encoded. The CCSDS 123 standard allows the selection between a sample-adaptive entropy coder and a block-adaptive entropy coder.

Under the sample-adaptive entropy coding approach, each mapped prediction residual is encoded using a Golomb power-of-two variable-length binary codeword. The adaptive code selection statistics consist of an *accumulator* $\sum_z(t)$ and a counter $\Gamma(t)$ that are adaptively updated during the encoding process. The accumulator $\sum_z(t)$ and the counter $\Gamma(t)$ are rescaled periodically, according to the user-defined *rescaling* *counter size* parameter. Values k and U are computed from the accumulator $\sum_z(t)$ and the counter $\Gamma(t)$ results and are used to create the Golomb power-of-two codewords.

The block-adaptive entropy coder utilizes the Rice coder defined in the CCSDS 121 standard [AD-2].

4.2.2 General description

The IP core receives first the necessary configuration parameters through the AMBA AHB slave interface. It informs then the source that it is ready to receive the input samples, which are stored in a FIFO. First, the prediction stage is performed as specified by the CCSDS 123 standard. Afterwards the samples are encoded by any of the two encoding options: sample-adaptive encoding or block adaptive encoding. If the block-adaptive encoding option is selected, the CCSDS121 block coder has to be connected as an external encoder at the output of the CCSDS123 IP core.

The CCSDS123 IP core might be configured at runtime or at implementation time. The parameter EN_RUNCFG is used to select between the two options.

When EN_RUNCFG = 1, the IP core configuration is received through the AMBA AHB Slave interface. The CCSDS123 IP clock is independent from that of the AMBA interface. The received configuration data are stored in an internal FIFO, read by the IP and then stored in registers that are available for all the modules that request them.

Additionally, the possibility of setting some parameters at implementation time (before synthesis or compilation of the model) is allowed. In that case, the design is tailored to the user's needs. The different constants are stored in a configuration package, and then propagated to the generics of the sub-modules or used directly by the sub-modules as constants.

The CCSDS123 IP core is able to compress samples in that are sent in BIP, BSQ and BIL order. The order is selected at implementation time with a generic parameter that determines the hardware architecture that is implemented, i.e. is not selectable at runtime.

The CCSDS123 IP core might require external memory storage for the intermediate values generated during the compression. An AMBA AHB Master interface is included to store and receive intermediate values for compression from an external memory. The possibility of enabling or disabling EDAC at implementation time is included. EDAC is able to correct single errors and detect double errors. The user can select to use EDAC for internal, external or both types of memories.

When the block-adaptive encoding option is used, the CCSDS123 IP core needs to be connected to the CCSDS121 block coder using the available interfaces. In such a case, it is necessary to configure both IPs independently, i.e. the configuration parameters of the CCSDS121 block coder shall be sent through its own AMBA AHB slave interface.

For verification or validation purposes, it is possible to disable the generation of headers. The configuration option DISABLE_HEADER configuration is provided to disable the header generation.

4.2.3 SEE Mitigation

The following SEE Mitigation Techniques are considered:

- Triple Modular Redundancy and Majority Voting in all flip-flops and latches (TMR) can be applied at logic level during synthesis with the appropriate synthesis attribute according to the target technology.

- Error Detection and Correction (EDAC) for embedded and external memories, with capability for correcting one error and detecting double errors. EDAC is enabled by setting the corresponding generic parameter.

- States in the FSM are encoded using Gray coding. All FSM include the <<when others>> clause, and return to the IDLE state in case of errors.

4.2.4 Functions not included

The extent of compliance with the standard is shown by completing the PRL (Protocol Implementation Conformance Statement Requirements List), See Annex A of [AD-1]. All the functionalities defined in the CCSDS-123 standard are implemented, with the following exceptions:

- The Accumulator Initialization Table cannot be configured at runtime. The values in such table shall only be introduced using constant parameters.
- The option to include the Accumulator Initialization Table in the header described in Section 5.3.4.2.1 of [AD-1] is not implemented.
- The option to select Custom Weight Initialization described in Section 4.6.3.3 of [AD-1] is not implemented. However, the necessary parameters to do it in future extensions are defined.
- The sub-frame interleaving depth (M) is restricted to two possible values: M=1, which corresponds to BIL order; or M = Nz, which corresponds to BIP order (see Section 5.4.2.2 [AD-1]).

4.2.5 Functional specification

4.2.5.1 Design Options

The design options are set by VHDL constants that are stored in a package and then propagated to the generics of the sub-modules or passed directly as constants. The provided constants are summarized in the following sections. These constant parameters affect or limit the runtime configuration options of the design, as it is detailed in Section 0.

4.2.5.1.1 System parameters

The constants in Table 4-2 affect parts of the design which might be implemented or inhibited. If the parts are implemented their functionality might be enabled, disabled or configured by selecting the appropriate runtime configuration parameter.

The constant EN_RUNCFG in Table 4-2 is used to enable or disable the configuration of the compressor's parameters at runtime. When EN_RUNCFG = 0, the designed is tailored according to the user-selected configuration constants described in Section 4.2.5.1.2. EN_RUNCFG = 1 makes the user able to set the configuration parameters at runtime.

The constant PREDICTION_TYPE shall be set according to the order in which the samples are arranged in the raw multispectral/hyperspectral image. The implementation of the prediction stage of the CCSDS123 IP core is tailored according to this parameter. The supported orders are: band-interleaved per pixel (BIP); band-interleaved per line (BIL) and band-sequential (BSQ). See Section 5.4.2 in [AD-1] for detailed information about how the samples can be arranged in a multispectral/hyperspectral image. It is assumed that the samples are received, predicted and entropy coded in the same order.

TABLE 4-2: CCSDS123 IP - SYSTEM CONSTANTS

Constant	Allowed	Description
EN_RUNCFG	0	Disables runtime configuration. (Parameters set at implementation time reading the constants.)
	1	Enables runtime configuration. (Parameters set at runtime reading the memory-mapped configuration registers.)
EDAC	0	Inhibits EDAC implementation.
	1	EDAC is implemented for embedded memories. *NOTE: this parameter is forced to '0' in some FIFOs for the current implementation, because the memory use is so limited that BRAMs are not inferred and therefore only FFs are synthesized. It is up to the users to change the GENERIC assignment if they prefer to pass the EDAC parameter to a memory instance. EDAC is not supported in asynchronous FIFOs (used when PREDICTION_TYPE is 1, 2 or 4). The recommendation is to implement these FIFOs using FFs due to their limited size.*
	2	EDAC is implemented for external memories storing intermediate values (used when PREDICTION_TYPE is 1, 2 or 4)
	3	EDAC is implemented for both embedded and external memories.
RESET_TYPE	0	Implement asynchronous reset.
	1	Implement synchronous reset.
ENCODING_TYPE	0	Only pre-processor is implemented, an external encoder IP core (CCSDS121 block coder) can be attached.
	1	Sample-adaptive encoder implemented.
ENCODER_SELECTION_GEN	0	Disables encoding.
	1	Selects sample-adaptive coder*.
	2	Selects external encoder (block-adaptive).
PREDICTION_TYPE	0	BIP-Base architecture implemented for predictor (samples are arranged in BIP order and intermediate results are stored in the internal memory).
	1	BIP-Mem architecture implemented for predictor (samples are received in BIP order and intermediate results are stored in an external memory).
	2	BSQ architecture implemented for predictor (samples are received in BSQ order).
	3	BIL-Base architecture implemented for predictor samples are received in BIL order and intermediate results are stored in the internal memory).
	4	BIL-Mem architecture implemented for predictor (samples are received in BIL order and intermediate results are stored in an external memory).
ExtMemAddress_GEN	0-16#FFF#	External memory address. Sets the 12 most significant bits in the 32-bit AHB address.

**Only effective if generic ENCODING_TYPE = 1*

4.2.5.1.2 Configuration parameters

The implementation time constants in Table 4-3, Table 4-4 and Table 4-5 affect the configuration of the CCSDS 123 compressor. The aforementioned tables present a description of the parameters and their default values.

Specific baseline values are proposed for each parameter and displayed in the second column ("Baseline") of the tables. The values of the implementation time parameters put limitations in the selectable runtime configuration values, as it is shown later in detail in Section 0. The "Baseline" values will generate an implementation of the compressor that allows the user to set the runtime configuration parameters to the most commonly used figures. Additionally, the third column ("Max.") displays the values that allow the user to select any of the possible configuration values permitted by the CCSDS 123 standard.

TABLE 4-3: CCSDS123 IP - IMAGE METADATA CONSTANTS

Constant	Baseline	Max.	Allowed values	Description	Reference in [AD-1]
Nx_GEN	1024	65535	[1:2^{16}-1]	Maximum allowed number of samples in a line.	3.2.2
Ny_GEN	1024	65535	[1:2^{16}-1]	Maximum allowed number of samples in a row.	3.2.2
Nz_GEN	1024	65535	[1:2^{16}-1]	Maximum allowed number of bands.	3.2.2
D_GEN	16	16	[2:16]	Maximum dynamic range of the input samples.	3.3
IS_SIGNED_GEN	0	0	[0,1]	(0) unsigned or (1) signed input samples.	3.2.1
ENDIANESS_GEN	0	0	[0,1]	(0) little endian; (1) big endian.	
DISABLE_HEADER_GEN	0	0	[0,1]	(0) generates header (1) inhibits header generation	
W_BUFFER_GEN	32	64	[8,16,32,64] and > (U_MAX_GEN + D_GEN)< 64	Bit width of the output buffer.	

TABLE 4-4: CCSDS123 IP - PREDICTOR CONSTANTS

Constant	Baseline	Max.	Allowed values	Description	Reference in [AD-1]
P_MAX	3	3	[0:15]	Number of bands used for prediction.	4.2
PREDICTION_GEN	1	0	[0,1]	Full (0) or reduced (1) prediction.	4.5; 4.6
LOCAL_SUM_GEN	1	0	[0,1]	Neighbour (0) or column (1) oriented local sum.	4.4
OMEGA_GEN	13	19	[4:19]	Weight component resolution.	4.6.1.2
R_GEN	32	64	[max(32, D_GEN + OMEGA_GEN+2):64]	Register size.	4.7.1

VMAX_GEN	3	9	[VMIN: 9]	Factor for weight update.	4.8.2 a)
VMIN_GEN	-1	-6	[-6 : VMAX]	Factor for weight update.	4.8.2 a)
T_INC_GEN	6	11	[4:11]	Weight update factor change interval.	4.8.2b)
WEIGHT_INIT_GEN	0	0	(0) Default (1) Custom	Weight initialization mode.	4.6.3
Q_GEN	5	16	[3:16]	Custom weight resolution.	4.6.3.3
CWI_GEN	0	0	(0) different weight vectors for each band; (1) same weight vector for all the bands	Custom weight initialization mode.	

TABLE 4-5: CCSDS123 IP - SAMPLE-ADAPTIVE CODER CONSTANTS

Constant	Baseline	Max.	Allowed values	Description	Reference in [AD-1]
INIT_COUNT_E_GEN	1	8	[1:8]	Initial count exponent.	5.4.3.2.2.2
ACC_INIT_TYPE_GEN	0	0	(0) Constant (1) Table	Accumulator initialization type.	5.4.3.2.2.3
ACC_INIT_CONST_GEN	5	14	[0: D-2]	Accumulator initialization constant.	5.3.2.2.3
RESC_COUNT_SIZE_GEN	6	9	[max(4, INIT_COUNT_E_GEN+1): 9]	Rescaling counter size.	5.4.3.2.2.4
U_MAX_GEN	16	16	[8:32]	Unary length limit.	5.4.3.2.3.1
ACC_GEN_TAB1	0	0	4-bit unsigned integer values	Accumulator initialization table values to be used when ACC_INIT_TYPE_GEN = 1; not used otherwise	5.3.4.2.2 5.4.3.2.2.3
ACC_ GEN_TAB2	0	0			
…	0	0			
ACC_ GEN_TABNz	0	0			

4.2.5.1.3 AMBA AHB Parameters

The constants in Table 4-6 are included for AMBA AHB configuration.

TABLE 4-6: CCSDS123 IP - AHB CONSTANTS

Constant	Allowed values	Description
HSINDEX	0-NAHBSLV-1	AHB slave index.
HSCONFIGADDR	0-16#FFF#	ADDR field of the AHB Slave. Sets the 12 most significant bits in the 32-bit AHB address.
HSADDRMASK	0-16#FF0#	MASK field of the AHB Slave.
HMINDEX	0 - NAHBMST-1	AHB master index.
HMAXBURST	[0:16]	AHB master burst beat limit (0 means unlimited*).

** If 0 value is selected, it is replaced by the maximum burst value (16) due to unlimited transfers are unfeasible in a hardware implementation (logic resources must be reserved).*

4.2.5.2 Runtime configuration

4.2.5.2.1 Registers

The CCSDS123 IP core allows for runtime configuration of image metadata and encoding parameters, when the parameter EN_RUNCFG = 1.

In this case, the configuration values used by the IP core during the compression are the ones in the memory-mapped configuration registers, as shown in Table 4-8, Table 4-9, Table 4-10 and Table 4-11.

The "Max." values displayed in Table 4-3, Table 4-4 and Table 4-5 for the implementation time parameters when the configuration is set at runtime (EN_RUNCFG=1) ensure the possibility for the user to select among all the possible range of values allowed by the CCSDS 123 standard, except for parameter *P*. The user might change the values of the generic parameters at convenience, taking into account that the runtime configuration values allowed might be affected and limited, as it is shown in the "Allowed values" column of Table 4-8, Table 4-9, Table 4-10 and Table 4-11.

TABLE 4-7: CCSDS123 IP - RUNTIME CONFIGURATION EXTERNAL MEMORY ADDRESS

Parameter name	Allowed values	Description	Bits taken
ExtMemAddress**	0-16#FFF#	External memory address. Sets the 12 most significant bits in the 32-bit AHB address.	35

*** Runtime configuration values shall override the values of the constants implementing the same functionality.*

TABLE 4-8: CCSDS123 IP - RUNTIME CONFIGURATION ENCODING TYPE

Parameter name	Allowed values	Description	Bits taken
ENCODER_SELECTION**	0	Disables encoding.	2
	1	Selects sample-adaptive coder*.	
	2	Selects external encoder (block-adaptive).	

**Only effective if generic ENCODING_TYPE = 1 ** Runtime configuration values shall override the values of the constants implementing the same functionality.*

TABLE 4-9: CCSDS123 IP - RUNTIME CONFIGURATION OF THE IMAGE METADATA

Parameter name	Allowed values	Description	Bits taken
Nx	[1: Nx_GEN]	Number of samples in a line.	16
Ny	[1: Ny_GEN]	Number of samples in a row.	16
Nz	[1: Nz_GEN]	Number of bands.	16
D	[1: D_GEN]	Dynamic range of the input samples.	5
IS_SIGNED**	[0,1]	(0) unsigned or (1) signed input samples.	1
ENDIANESS**	[0,1]	(0) little endian; (1) big endian.	1

DISABLE_HEADER**	[0,1]	(0) disables the generation of headers. (1) enables the generation of headers.	1
W_BUFFER	[8,16,32,64] and < W_BUFFER_GEN and > (U_MAX + D_GEN)	Bit width of the output buffer.	7

TABLE 4-10: CCSDS123 IP - RUNTIME CONFIGURATION OF THE PREDICTOR PARAMETERS

Parameter name	Allowed values	Description	Bits taken
P	[0:P_MAX]	Number of bands used for prediction.	4
PREDICTION**	[0,1]	Full (0) or reduced (1) prediction.	1
LOCAL_SUM**	[0,1]	Neighbour (0) or column (1) oriented local sum.	1
OMEGA	[2: OMEGA_GEN]	Weight component resolution.	5
R	[max(32, D + OMEGA+2) : R_GEN]	Register size.	7
VMAX	[VMIN : VMAX_GEN]	Factor for weight update.	4
VMIN	[VMIN_GEN : VMAX]	Factor for weight update.	4
T_INC	[4 : T_INC_GEN]	Weight update factor change interval.	4
WEIGHT_INIT***	[0,1]	Default (0) or custom (1) weight initialization mode.	1
Q**	[0, 3:OMEGA+3]	Custom weight resolution. When default initialization is used, Q must be set to 0. When the CWI is used, Q is set to any value in the range [3:OMEGA+3].	5
WEIGHT_TABLE	[0:2^Q-1] for each weight component	Custom weight table.	N_z·Q·(P+3)
WR	[0,1]	Reset of custom weight vectors.	1

****Current implementation supports only WEIGHT_INIT = 0. However, the necessary parameters to implement the custom weight initialization mode are defined.*

TABLE 4-11: CCSDS123 IP - RUNTIME CONFIGURATION OF THE SAMPLE-ADAPTIVE ENCODER PARAMETERS

Parameter name	Allowed values	Description	Bits taken
INIT_COUNT_E*	[1 : INIT_COUNT_E_GEN]	Initial count exponent.	3
ACC_INIT_TYPE*	[0,1]	(0) accumulator initialization table shall not be used; (1) accumulator initialization table shall be read from the generic values.	1
ACC_INIT_CONST*	[0: ACC_INIT_CONST_GEN]	Accumulator initialization constant.	4
RESC_COUNT_SIZE*	[max(4,INIT_COUNT_E+1): ACC_INIT_CONST_GEN]	Rescaling counter size.	4
U_MAX*	[8: U_MAX_GEN]	Unary length limit.	6

**Only effective if generic ENCODING_TYPE = 1. ** Runtime configuration values shall override the values of the constants implementing the same functionality.*

NOTE: It is assumed EDAC protection for these external memory-mapped configuration registers is managed by an external entity, and therefore the IP Core does not actively check for EDAC errors. It is up to the user to evaluate if protection is necessary considering the short life time of the information. Configuration information is read by the IP Core immediately after it is written in the memory-mapped registers and only once, and then stored in internal registers.

4.2.5.2.2 Memory mapping

The values of the memory-mapped registers are accessed by the IP core using the AMBA AHB Slave interface. The IP core includes the following memory-mapped registers:

TABLE 4-12: CCSDS123 IP - MEMORY MAPPING

AHB Offset	Register
0x00	Control&Status
0x04	ExtMemAddress
0x08	123CFG0
0x0C	123CFG1
0x10	123CFG2
0x14	123CFG3

The Control&Status register is used to enable the IP core. It is also used by the IP core to write information about its status, storing the values in the control interface and an error code.

TABLE 4-13: CCSDS123 IP - CONTROL&STATUS REGISTER

AHB Offset	Bit Number	Mode	Description	Example value	Bits taken
0x00	31	w	Value of the **AwaitingConfig** signal.	0	1
	30	w	Value of the **Ready** signal.	0	1
	29	w	Value of the **FIFO_Full** signal.	0	1
	28	w	Value of **EOP** signal.	0	1
	27	w	Value of **Finished** signal.	0	1
	26	w	Value of **Error** signal.	0	1
	25:22	w	Error code.	0	4
	21:1	r	RESERVED.	0	22
	0	r/w	ENABLE.	0	1

When ENABLE is set to 1, the IP core starts running. The IP core understands, then, that all the necessary configuration values have been sent.

The ExtMemAddress register is used to configure the base address of the external memory which stores the intermediate values of compression using the AMBA Master interface.

TABLE 4-14: CCSDS123 IP - EXTERNAL MEMORY ADDRESS CONFIGURATION REGISTER

AHB Offset	Bit Number	Mode	Parameter name	Example value	Bits taken

0x04	31:0	r	ExtMemAddress	0	32

The memory mapped registers shall be of 32-bits, with the following configuration:

TABLE 4-15: CCSDS123 IP - CONFIGURATION REGISTER 123CFG0

AHB Offset	Bit Number	Mode	Parameter name	Example value	Bits taken
0x08	31:16	r	Nx	512	16
	15:11	r	D	16	5
	10	r	IS_SIGNED	0	1
	9	r	DISABLE_HEADER	0	1
	8:7	r	ENCODER_SELECTION	0	2
	6:3	r	P	3	4
	2:0	r	RESERVED	0	3

TABLE 4-16: CCSDS123 IP - CONFIGURATION REGISTER 123CFG1

AHB Offset	Bit Number	Mode	Parameter name	Example value	Bits taken
0x0C	31:16	r	Ny	512	16
	15	r	PREDICTION	0	1
	14	r	LOCAL_SUM	0	1
	13:9	r	OMEGA	13	5
	8:2	r	R	32	7
	1:0	r	RESERVED	0	2

TABLE 4-17: CCSDS123 IP - CONFIGURATION REGISTER 123CFG2

AHB Offset	Bit Number	Mode	Parameter name	Example value	Bits taken
0x10	31:16	r	Nz	512	16
	15:11	r	VMAX	3	5
	10:6	r	VMIN	-1	5
	5:2	r	T_INC	6	4
	1	r	WEIGHT_INIT	0	1
	0	r	ENDIANESS	0	1

TABLE 4 17: CCSDS123 IP - CONFIGURATION REGISTER 123CFG3

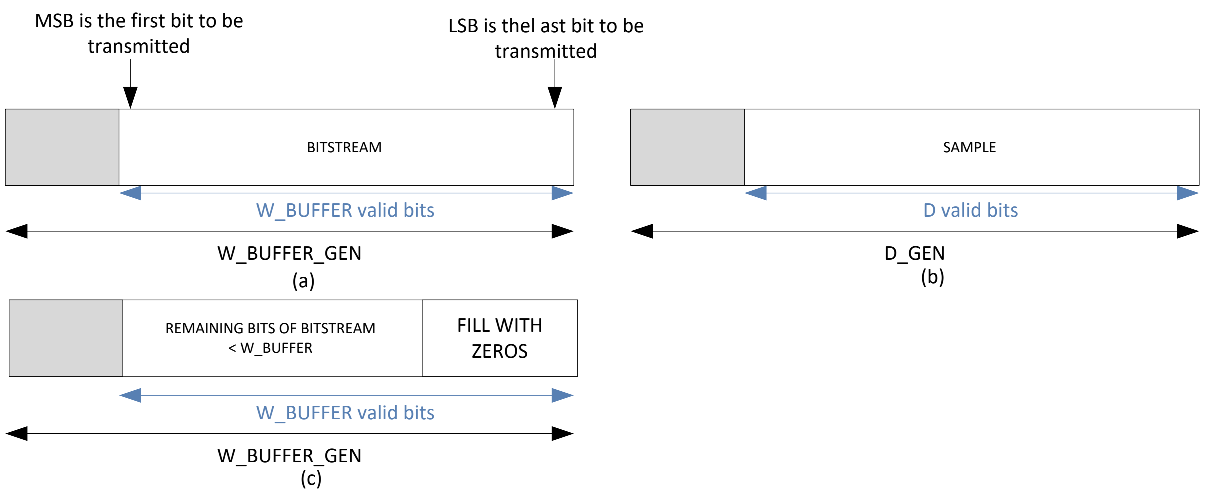
AHB Offset	Bit Number	Mode	Parameter name	Example value	Bits taken
0x14	31:28	r	INIT_COUNT_E	1	4
	27	r	ACC_INIT_TYPE	1	1

	26:23	r	ACC_INIT_CONST	5	4
	22:19	r	RESC_COUNT_SIZE	6	4
	18:13	r	U_MAX	16	6
	12:6	r	W_BUFFER	32	7
	5:1	r	Q	16	5
	0	r	WR	0	1

4.3 Conventions

The following conventions shall be considered to correctly interpret the raw and compressed data values:

- The most significant bits of the compressed bitstream are located to the left and are the first to be transmitted.
- When the number of valid bits in a signal or register is smaller than the bit width, the valid bits are located to the right, i.e. using the least significant bits.
- The bitstream is packed according to the size of the output buffer and the user-defined parameter W_BUFFER. When the compression finishes, the remaining bits of the bitstream shall be flushed. Zeros shall be appended to the right, up to the W_BUFFER boundary.
- The input data samples are interpreted as specified according to the endianness and singedness specified by the user. Once received by the IP, they are always treated as signed values stored in big-endian order.



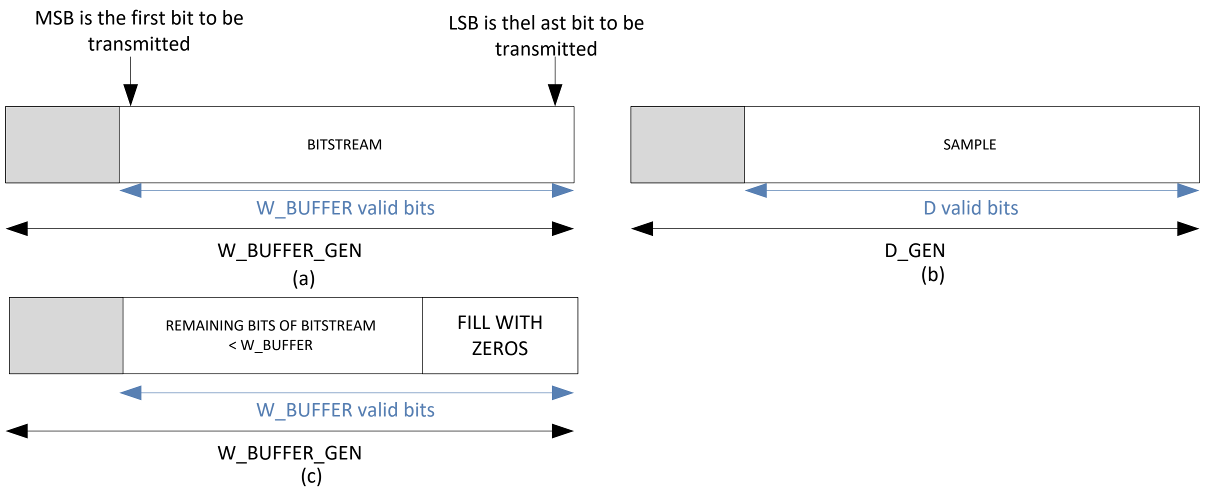


Figure 4-5: Location of the valid bits in the i/o signals. (a) Bitstream (b) Input sample (c) Remaining bits of the bitstream after flushing the output buffer.

4.4 Interfaces and signal description

4.4.1 Interfaces general description

4.4.1.1 System Interface

- **Rst_N**: Active low input signal that resets the IP to its initial state. **Rst_N** might be synchronous or asynchronous according to the RESET_TYPE generic.
- **Clk_S**: System clock signal. The registers internal to the IP are clocked on the rising **Clk_S** edge.

4.4.1.2 Input data interface

The input data interface includes the following signals:

- **DataIn**: Input uncompressed samples, D_GEN bits wide. The actual number of valid bits is determined as follows:
    - D_GEN when EN_RUNCFG = 0.
    - The configurable value D when EN_RUNCFG = 1. If D_GEN > D, the valid bits are placed in the least significant bits of **DataIn**.
- **DataIn_NewValid**: Indicates the presence of new valid values (active high) in **DataIn** for one **Clk_S** cycle.

4.4.1.3 Output data interface

The output data interface shall include the following signals:

- **DataOut:** Output data signal, W_BUFFER_GEN bits wide containing:
    - **DataOut** is used to send the compressed bitstream when the sample-adaptive is implemented (ENCODING_TYPE = 1) and selected (ENCODER_SELECTION = 1). The actual number of valid bits is determined as follows:
        - W_BUFFER_GEN when EN_RUNCFG = 0.
        - The configurable value W_BUFFER when EN_RUNCFG = 1. If W_BUFFER_GEN > W_BUFFER, the valid bits are placed in the least significant bits of **DataOut**.
    - **DataOut** is used to send both the header and the mapped prediction residuals in two situations: when ENCODER_SELECTION = 0 (encoder disabled) or when ENCODER_SELECTION = 2 (external encoder, CCSDS121 block coder).
        - The header is sent packed and byte-aligned. The number of valid bits in the header is determined by the **NbitsOut** signal.
        - Valid header values are accompanied by signals **DataOut_NewValid** and **IsHeaderOut** set to a high value for one **Clk_S** cycle.
        - Once the header has been sent, the **DataOut** signal contains the mapped prediction residuals. The actual number of valid bits is determined as follows:
            - D_GEN when EN_RUNCFG = 0.
            - The configurable value D when EN_RUNCFG = 1. If D_GEN > D, the valid bits are placed in the least significant bits of **DataOut**.
        - Valid residual values are accompanied by the signal **DataOut_NewValid** set to a high value for one **Clk_S** cycle and **IsHeaderOut** set to a low value.
- **DataOut_NewValid:** Indicates the presence of new valid values (active high) in **DataOut** for one **Clk_S** cycle.
- **NbitsOut:** Used to inform an external encoder about the number of valid bits in the **DataOut** signal. **NbitsOut** is only used when the entropy coding is performed by an external IP core.
- **IsHeaderOut:** Indicates the external encoding IP core that the data in **DataIn** corresponds to a header, with **NbitsOut** valid bits.

4.4.1.4 Control interface

The signals in the control interface inform the source about the state of the IP core.

- **ForceStop:** If asserted (high) the IP core shall stop compressing the current image and go back to the IDLE state. The user might then send a new configuration and start the compression of new data.
- **AwaitingConfig**: If asserted (high) the IP core is waiting to be configured. The output is clocked out on the rising **Clk_S** edge.
- **Ready**: If asserted (high) the IP core has been configured and is able to receive new samples for compression. If de-asserted (low) the IP is not ready to receive new samples. If the IP is unavailable to receive new samples, **Ready** is be de-asserted with sufficient time to prevent data losses. The output is clocked out on the rising **Clk_S** edge.
- **FIFO_Full**: If asserted (high) there was an attempt to write in a full input FIFO. The output is clocked out on the rising **Clk_S** edge.
- **EOP**: If asserted (high) the compression of the last sample has started. The output is clocked out on the rising **Clk_S** edge.
- **Finished**: If asserted (high) the IP core has finished processing all samples in the image. The output is clocked out on the rising **Clk_S** edge.
- **Error**: If asserted (high), there has been an error during the compression, for instance, a configuration error. The output is clocked out on the rising **Clk_S** edge.
- **Ready_Ext**: If asserted (high), the output receiver (can be the CCSDS121 block coder) is ready to receive the output samples. If de-asserted the compression IP shall stop the compression and inform the source that it is unavailable to receive more samples using signal **Ready** edge.

4.4.1.5 External encoder control interface

This interface is utilized when an external entropy coder is attached to the CCSDS123 IP. In this case, the CCSDS123 acts as a pre-processor. The following signals are included:

- **ForceStop_Ext:** If asserted (high) the IP core shall stop compressing the current image and go back to the IDLE state. The source might then start the compression of new data.
- **AwaitingConfig_Ext**: If asserted (high) the IP core is waiting to be configured. The output is clocked out on the rising **Clk_S** edge.
- **FIFO_Full_Ext**: If asserted (high) the external IP core is not ready to receive more samples (input FIFO is full). The output is clocked out on the rising **Clk_S** edge.
- **EOP_Ext**: If asserted (high) the compression of the last sample has started. The output is clocked out on the rising **Clk_S** edge.
- **Finished_Ext**: If asserted (high) the external IP core has finished processing all samples in the image. The output is clocked out on the rising **Clk_S** edge.
- **Error_Ext**: If asserted (high), there has been an error during the compression in the external IP core, for instance, a configuration error. The output is clocked out on the rising **Clk_S** edge.

4.4.1.6 AMBA AHB 123 master interface

The AMBA AHB master interface is used to store and retrieve intermediate values of compression in/from an external memory when parameter AHBMEM = 1. The predictor architectures defined as Mem, this is, BIP-Mem and BIL-Mem, include this interface.

The AMBA AHB master interface is able to communicate with slaves that implement the full AHB functionality, i.e. all possible values of HRESP are supported.

The generic HMAXBURST can be used to set the maximum allowed burst size are foreseen.

For detailed information of the record types used for the AMBA AHB interface see [AD-6] and [AD-7].

The width of the AMBA AHB data vectors is the default value (CFG_AHBDW = 32).

4.4.1.7 AMBA AHB 123 slave interface

The AMBA AHB slave interface is used to receive the runtime configuration parameters of the CCSDS 123 compression core. For detailed information of the records used for the AMBA AHB interface see [AD-6] and [AD-7].

The AMBA SLAVE interface responds with either HRESP = “00” (OKAY) or “01” (ERROR). Split and Retry shall not be supported. The width of the AMBA AHB data vectors is specified by the value of the CFG_AHBDW parameter (default value of 32).

4.4.2 Signal overview figure

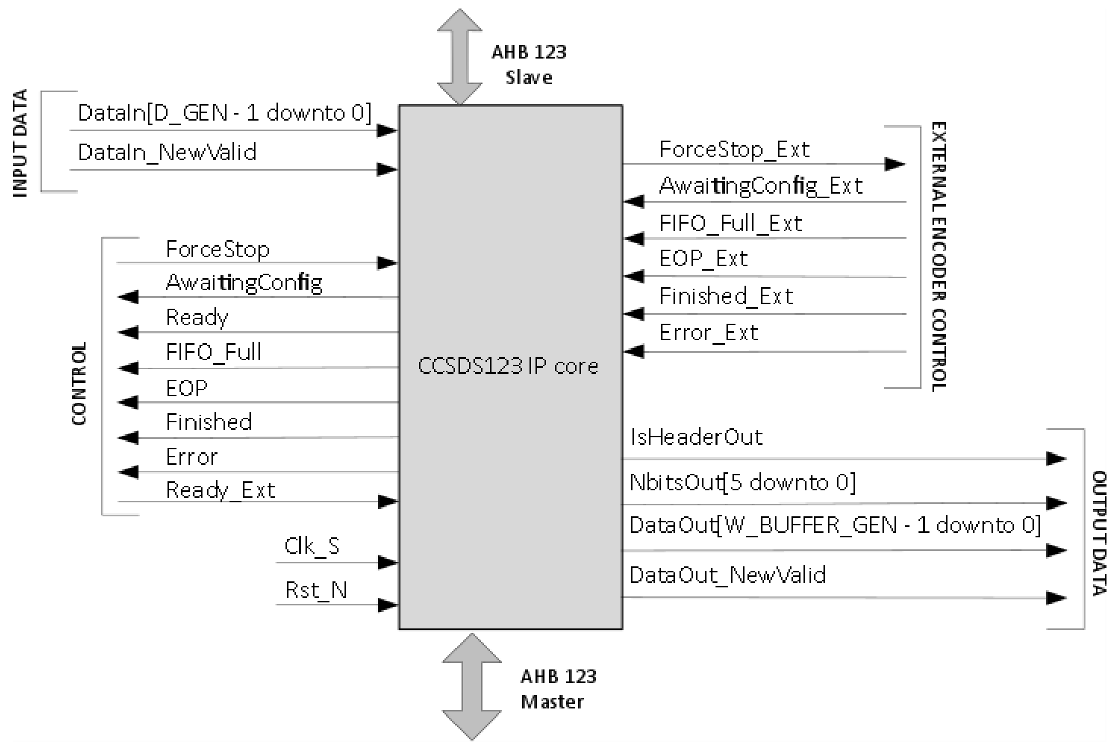


Figure 4-6: CCSDS123 IP - Core signal overview figure.

4.4.3 Signal description

The signal interfaces are described as displayed in Table 4-18.

TABLE 4-18: CCSDS123 IP - CORE INTERFACES

Signal	Type	Mode	Description	Clock domain	Active	Value after reset
System Interface						
Rst_N	Std_Logic	In	Synchronous/Asynchronous reset.	Clk_S/ N/A	Low	N/A
Clk_S	Std_Logic	In	System clock.	Clk_S	N/A	N/A*
Data Input Interface						
DataIn	Std_Logic_Vector (D_GEN - 1 downto 0)	In	Uncompressed samples.	Clk_S	N/A*	N/A
DataIn_NewValid	Std_Logic	In	Flag to validate input signals.	Clk_S	High	N/A
Data Output Interface						
DataOut	Std_Logic_Vector (W_BUFFER_GEN - 1 downto 0)	Out	Compressed samples.	Clk_S	N/A	Low
DataOut_NewValid	Std_Logic	Out	Flag to validate output data.	Clk_S	High	Low
NbitsOut	Std_Logic_Vector(5 downto 0)	Out	Number of valid bits in DataOut signal.	Clk_S	N/A	Low
IsHeaderOut	Std_Logic	Out	DataOut is a header.	Clk_S	High	Low
Control Interface						
ForceStop	Std_Logic	In	Force the stop of the compression.	Clk_S	High	N/A
AwaitingConfig	Std_Logic	Out	The IP core is waiting to receive the configuration through the AMBA AHB Slave bus.	Clk_S	High	Low
Ready	Std_Logic	Out	The IP core has received the configuration and is ready to read input samples. If asserted (high) the IP core has been configured and is able to receive new samples for compression. If de-asserted (low) the IP is not ready to receive new samples.	Clk_S	High	Low
FIFO_Full	Std_Logic	Out	Signals that the input FIFO is full and that the source must wait before sending more input samples. Input data might have been lost, the core returns to IDLE state and gets ready to receive a new configuration.	Clk_S	High	Low

EOP	Std_Logic	Out	Indicates that the compression of the last sample has started.	Clk_S	High	Low
Finished	Std_Logic	Out	The IP has finished compressing all samples in the image.	Clk_S	High	Low
Error	Std_Logic	Out	There was an error during the compression.	Clk_S	High	Low
Ready_Ext	Std_Logic	In	The output is ready to receive more compressed samples.	Clk_S	High	N/A
External encoder control interface						
ForceStop_Ext	Std_Logic	Out	Force the stop of the compression of the external IP core	Clk_S	High	Low
AwaitingConfig_Ext	Std_Logic	In	The external encoder is waiting to receive the configuration through the AMBA AHB Slave bus.	Clk_S	High	N/A
FIFO_Full_Ext	Std_Logic	In	The input FIFO of the external encoder is full.	Clk_S	High	N/A
EOP_Ext	Std_Logic	In	Indicates that the external encoder is starting the compression of the last sample.	Clk_S	High	N/A
Finished_Ext	Std_Logic	In	The IP has finished compressing all samples in the image.	Clk_S	High	N/A
Error_Ext	Std_Logic	In	There was an error during the compression in the external encoder.	Clk_S	High	N/A
AHB Slave (123) Interface						
Clk_AHB	Std_Logic	In	AHB clock.	N/A	N/A	N/A
Reset_AHB	Std_Logic	In	AHB reset.	N/A	Low	N/A
AHBSlave123_In	AHB_Slv_In_Type	In	AHB slave input signals.	Clk_AHB	N/A	N/A
AHBSlave121_Out	AHB_Slv_Out_Type	Out	AHB slave output signals.	Clk_AHB	N/A	Low
AMBA AHB Master Interface						
Clk_AHB	Std_Ulogic	In	AHB clock.	N/A	N/A	N/A
Reset_AHB	Std_Ulogic	In	AHB reset.	N/A	Low	N/A
AHBSMaster123_In	AHB_Slv_In_Type	In	AHB slave input signals.	Clk_AHB	N/A	N/A
AHBSMaster123_Out	AHB_Slv_Out_Type	Out	AHB slave output signals.	Clk_AHB	N/A	Low

**N/A stands for Not Applicable.*

4.4.4 State after reset

The IP core is in IDLE state and asserts signal **AwaitingConfig** after reset, indicating that it is ready to receive the configuration values. It stays in that state until the configuration is received through the AMBA AHB Slave interface. Reset is active low and might be synchronous or asynchronous, according to the user-defined generic parameter RESET_TYPE (0 for asynchronous; 1 for synchronous).

The register values in the CCSDS123 IP core are set to all zeros after reset, except those that explicitly require initialization:

- FSM state register shall be set to IDLE.
- **Weight_Vector**, values is set to the initial values specified in Section 4.6.3.2 of [AD-1].
- **Accumulator_SA,** accumulator value for the sample-adaptive encoder is set to the initial values specified in Section 5.4.3.2.2.3 of [AD-1].
- **Counter_SA,** counter value for the sample-adaptive encoder, is set to the initial values specified in Section 5.4.3.2.2.2 of [AD-1].
- **AwaitingConfig** is set to high.

4.4.5 Initialization

The IP core is in IDLE state and assert signal **AwaitingConfig** after reset, indicating that it is ready to receive the configuration values. The IP receives first the configuration through the AMBA AHB Slave interface. The read configuration registers are stored in internal registers that are shared among the different sub-modules that conform the IP. After the configuration, the IP shall be given the command ENABLE = 1 in the least significant bit of the Control&Status memory-mapped register (see Section 3.2.5.2.2). Once the IP is configured, it asserts signal **Ready**, indicating that it is ready to receive samples. Additionally, it starts issuing the header values, which are packed and sent through the output interface using W_BUFFER bits. If there are header bits remaining in the packer, the IP attaches them to the first bits of the bitstream.

The explained process is shown in the chronogram in Figure 4-7.

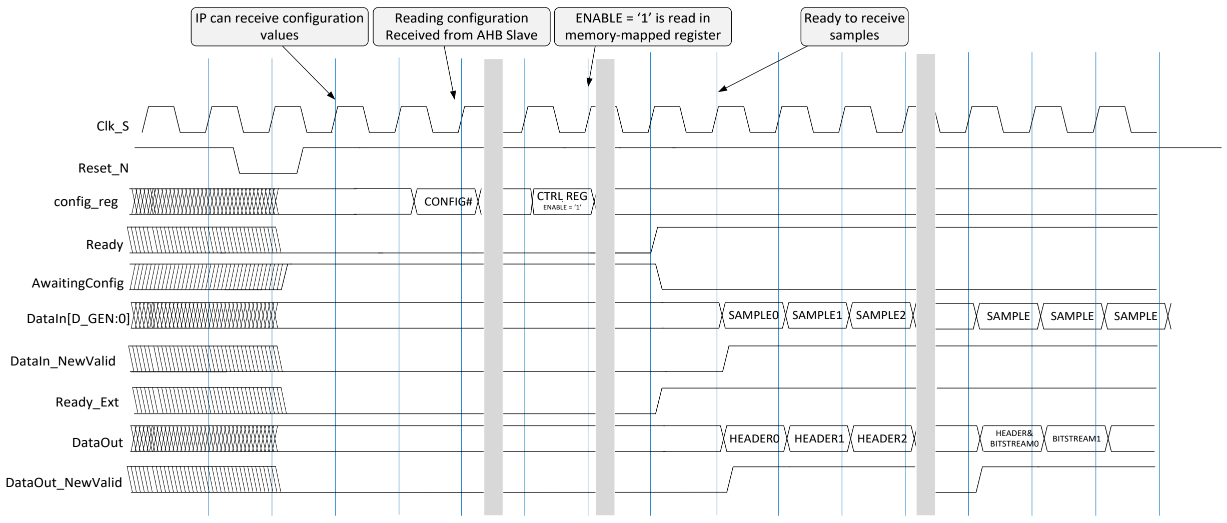


Figure 4-7: Initialization with synchronous reset.

4.4.6 Error handling

The CCSDS 123 IP core detects and inform about the following errors:

- Invalid configuration value is detected:

- Nx, Ny or Nz = 0.
- Values are not in the range allowed by the corresponding constants.
- W_BUFFER < D + U_MAX.
- W_BUFFER_GEN < D_GEN + U_MAX_GEN.
- FSM enters an invalid state.
- EDAC IP detects an uncorrectable error.
- AHB error.
- **Error_Ext** is asserted by the external encoder.

When an error is detected, the IP produces a high value in signal **Error** in the next rising edge of **Clk_S**, and then goes to the IDLE state and assert the **AwaitingConfiguration.** A new configuration can be then sent, as shown in Figure 4-8.

The error code field in the Control&Status memory-mapped register informs about the cause of the error.

TABLE 4-19: CCSDS123 IP - ERROR CODES

Error code	Meaning
0000	No error.
0001	Nx, Ny or Nz = 0.
0010	Values are not in the range allowed by the corresponding constants.
0011	W_BUFFER < D + U_MAX_GEN.
0100	W_BUFFER_GEN < D_GEN + U_MAX_GEN.
0101	Invalid FSM state.
0110	AHB error.

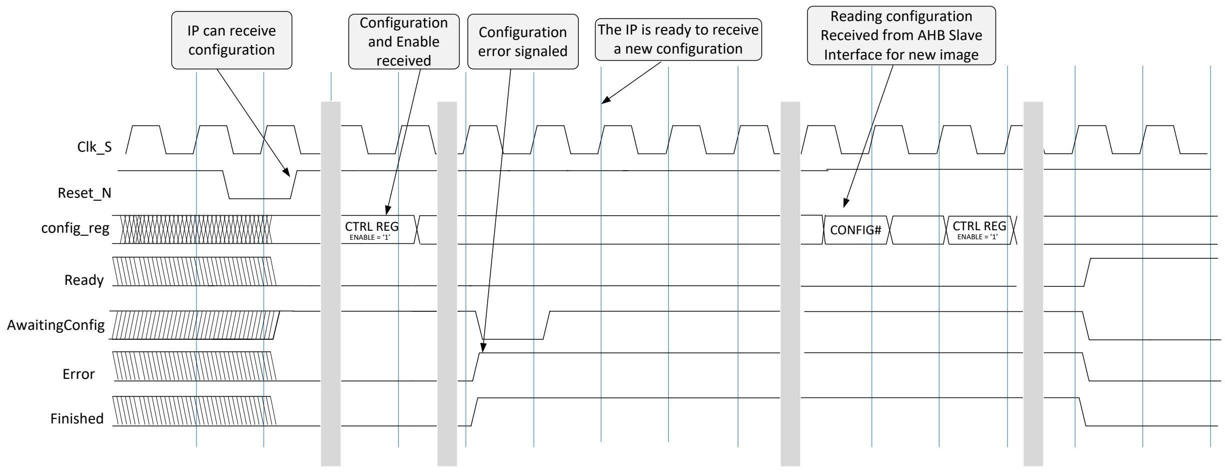

IP can receive configuration
Configuration and Enable received
Configuration error signaled
The IP is ready to receive a new configuration
Reading configuration Received from AHB Slave Interface for new image
Clk_S
Reset_N
config_reg
Ready
AwaitingConfig
Error
Finished
CTRL REG
ENABLE = '1'
CONFIG#
CTRL REG
ENABLE = '1'

Figure 4-8: A configuration error is detected. The IP asserts signals Finished and Error. One cycle after, it is ready to receive a new configuration.

4.4.7 Forcing the IP core to stop

The control signal **ForceStop** can be used to stop the compression at any moment. If a high value is detected in the positive edge of the clock, the IP asserts signal **Finished** in the next clock cycle and then returns to the IDLE state. No more compressed data are produced after **ForceStop**. The explained process is shown in the chronogram in Figure 4-9.

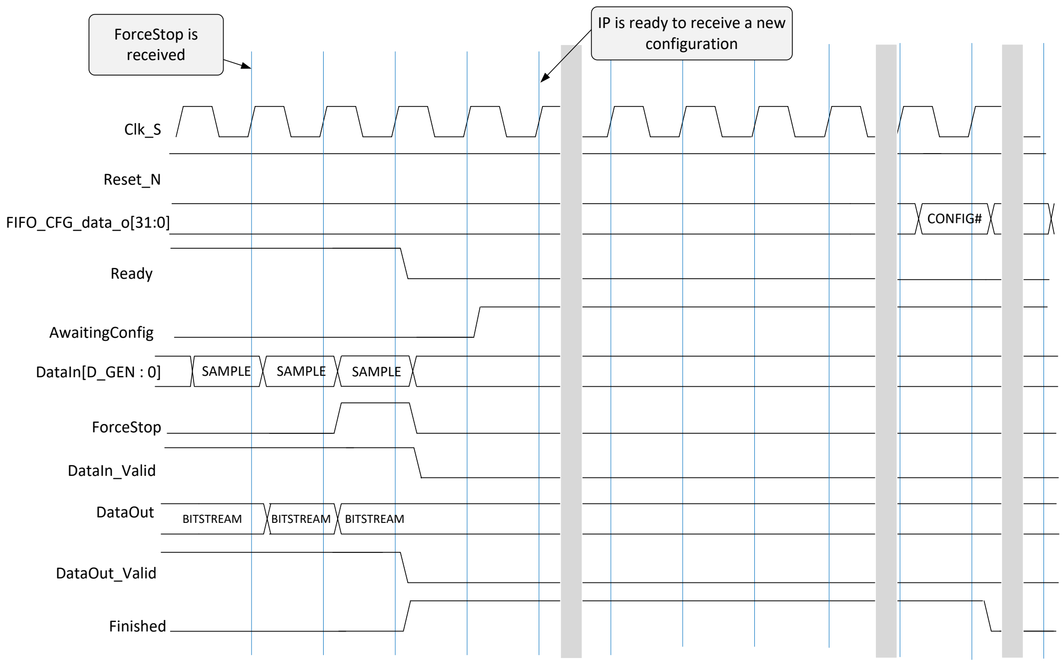


Figure 4-9: ForceStop is detected by the IP. It asserts Finished and one cycle after it is ready to receive a new configuration.

4.4.8 Finishing the compression without error

When the IP has received all the samples to be compressed, **Ready** is de-asserted. When the last sample received starts begin compressed **EOP** is asserted. Later, when such sample has been processed, the IP sends the encoded sample and flushes the output packer buffer, which might contain remaining bits. After flushing the buffer, signal **Finished** is asserted, indicating that no more valid bitstream values are produced at the output. The IP goes then to IDLE state and asserts the **AwaitingConfig** signal to receive a new configuration. The value in signal **Finished** is kept until the new configuration values are received. The chronogram can be seen in Figure 4-10.

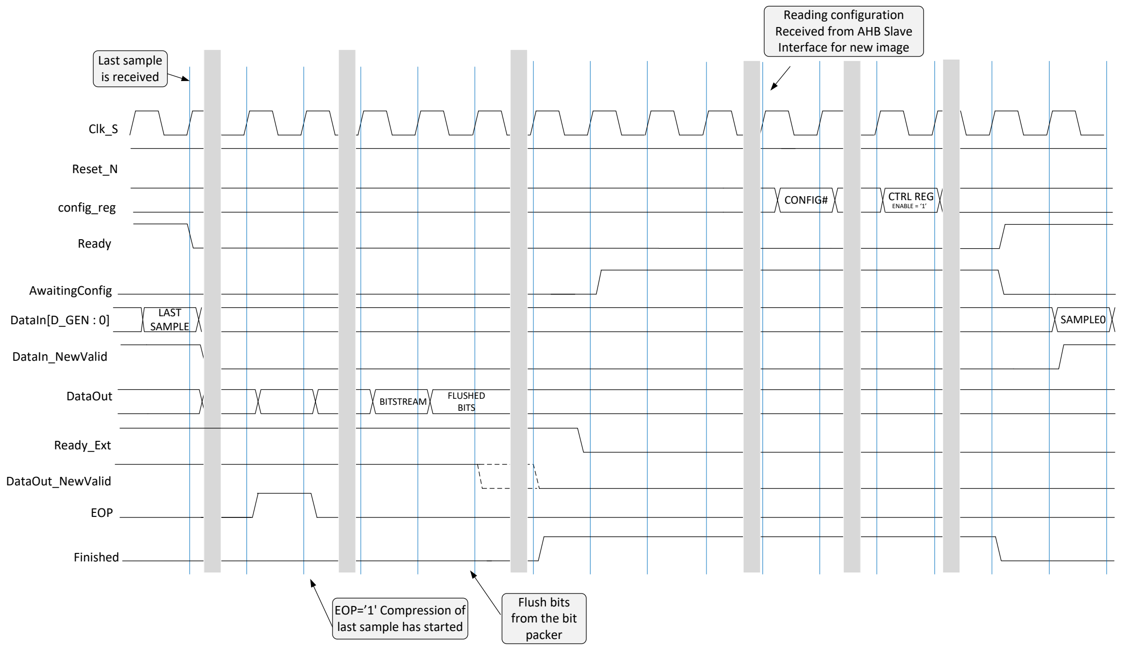


Figure 4-10: Last sample is received.

4.4.9 CCSDS123 IP core as pre-processor of CCSDS121 IP

When the CCSDS123 IP is configured as the pre-processor connected to the CCSDS121 block coder, it makes sure first that the CCSDS121 block coder has been configured before asserting the **Ready** signal (**Ready_Ext** asserted and **AwaitingConfig** de-asserted by the external encoder). The CCSDS123 IP issues first the header, indicating the number of valid bits that it contains with the **NbitsOut** signal. It indicates that the data corresponds to a header asserting **IsHeader**. The last bits of the header are flushed before sending the pre-processed samples to the external encoder. After the header, the mapped residuals are issued at the output.

In order to ensure that the configuration of the two IP cores is compatible, the rules described in Section 5.2 must be followed.

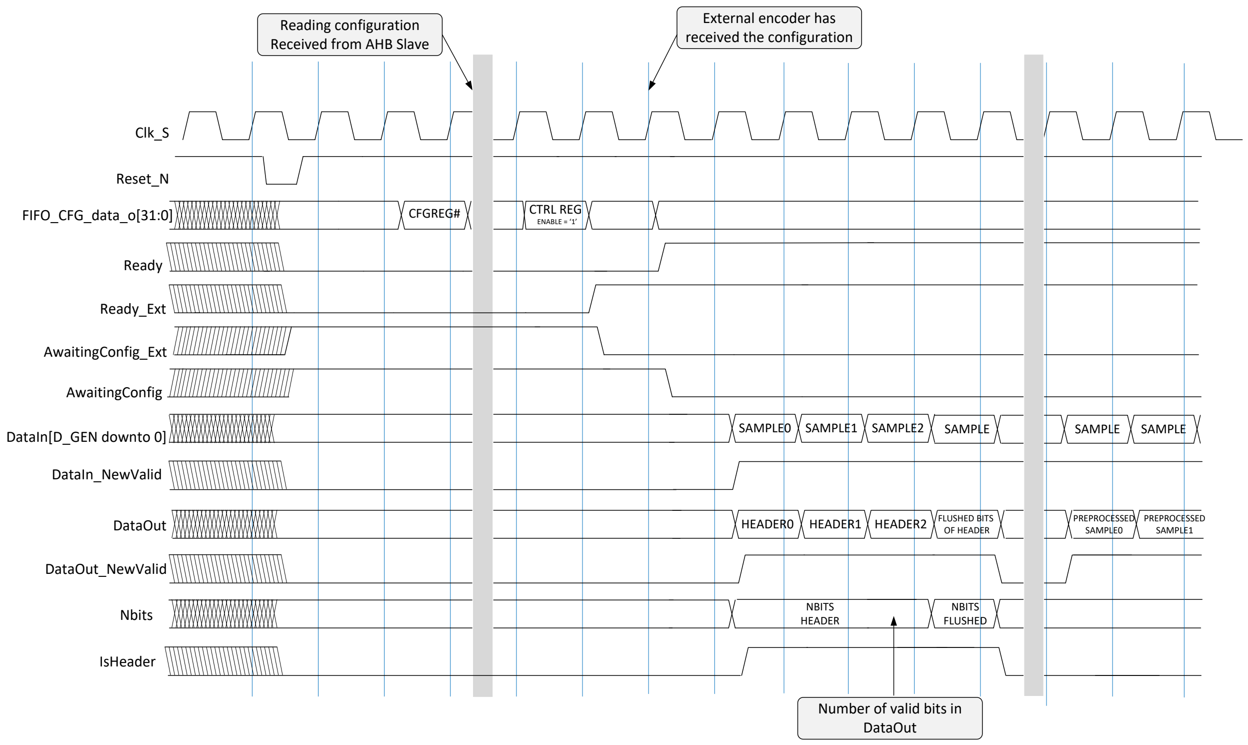


Figure 4-11: Example of the CCSDS123 IP working as a pre-processor with an external encoder and asynchronous reset.

4.5 Design description

This section presents a high-level representation in the form of block diagrams of the VHDL description of the CCSDS123 IP core, in order to facilitate the understanding of the delivered sources and the interpretation of the results when mapping the IP core to different technologies.

4.5.1 Simplified block diagram of the top module

The top module of the CCSDS123 IP core includes the components depicted in Figure 4-12. It includes the necessary components to receive the configuration and the input samples, compress them and send them to the output.

The **config_core** includes several modules that will perform the configuration of the compression. Runtime configuration values are received from the AMBA AHB slave interface by the **ahbslave** module and then adapted to the IP core's clock frequency by the **clk_adapt** module. The **interface_gen** module has a twofold purpose. On one hand, it assigns the configuration values, either from the ones received by the AHB slave interface (input port **config_s**), if runtime configuration is enabled (EN_RUNCFG = 1); or from the user-selected parameters (see Section 4.2.5.1), if the runtime configuration is disabled (EN_RUNCFG = 0). On the other hand, it checks that the selected configuration values are correct, and raises an error otherwise. These values are sent to the rest of the modules and sub-modules in the design.

The compression itself is performed by the **predictor** and **sample adaptive** modules. The **predictor** receives the raw input samples, and might access an external memory using the provided AHB master interface, to store intermediate values during the compression of a sample. The output of the **predictor** are the mapped residuals, which are sent to the output and to the **sample adaptive** module, for it to encode them.

The **control decoder** module generates and interprets control signals to and from I/O ports and ensures that the modules are ready to receive samples to be compressed.

Finally, the **dispatcher** module receives the output from the compression units and organizes them in FIFOs so that they can be sent to the output according to the selected configuration. If the CCSDS123 IP is acting as a pre-processor only with an external encoder, the header values and the mapped residuals are sent. Instead, if the CCSDS123 IP is using the sample-adaptive encoder, the bitstream is produced at the output. This module will monitor the **Ready_Ext** signal and the status of the FIFOs to signal if it is necessary to interrupt the compression, when the receiver is not ready to accept samples.

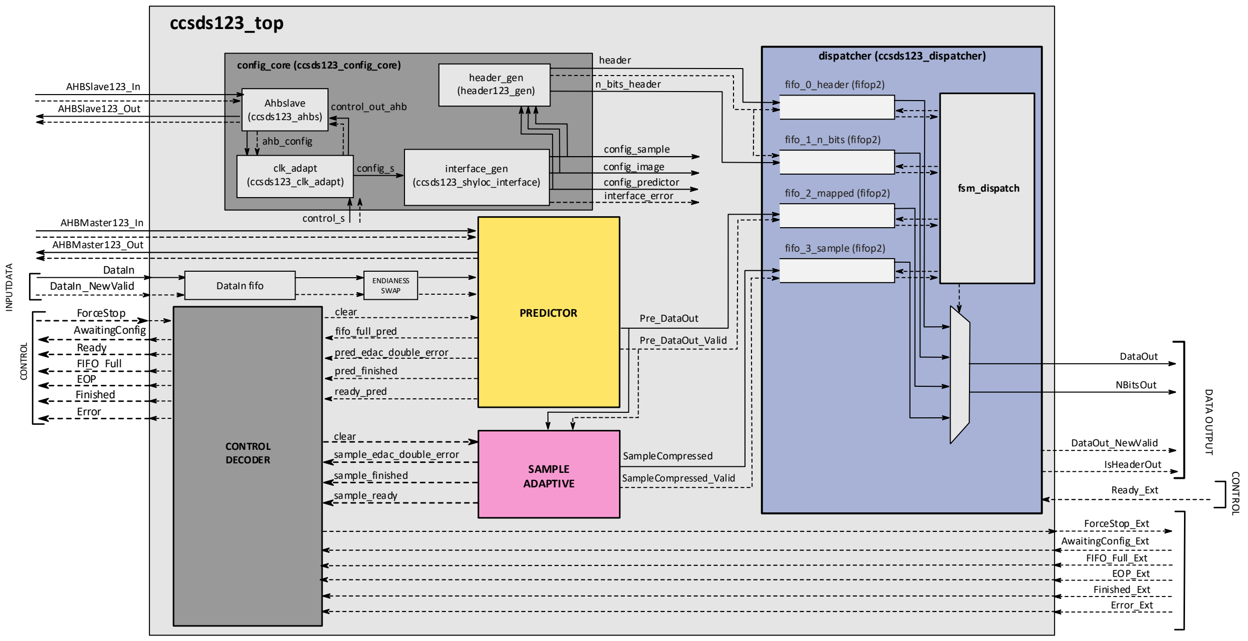


Figure 4-12: CCSDS123 IP – Simplified block diagram of the top module.

4.5.2 Predictor architectures

The predictor module of the CCSDS123 IP core deserves special attention due to its potential complexity. Three different architectures are devised for it, one for each of the possible compression orders (BIP, BSQ and BIL), in order to be able to find the best compromise between complexity and throughput. Additionally, due to the amount of internal storage required by the BIP and BIL architectures, the BIP-MEM and BIL-MEM architectures are designed, which include an AMBA AHB master interface that offers access to an external memory.

Each architecture includes a **comp** and an **fsm** module, as shown in Figure 4-13. The **comp** module instantiates the necessary components to perform the prediction. The scheduling of the operations is performed by the **fsm** module, which contains several finite-state machines in charge of enabling the different compression units in the **components** module.

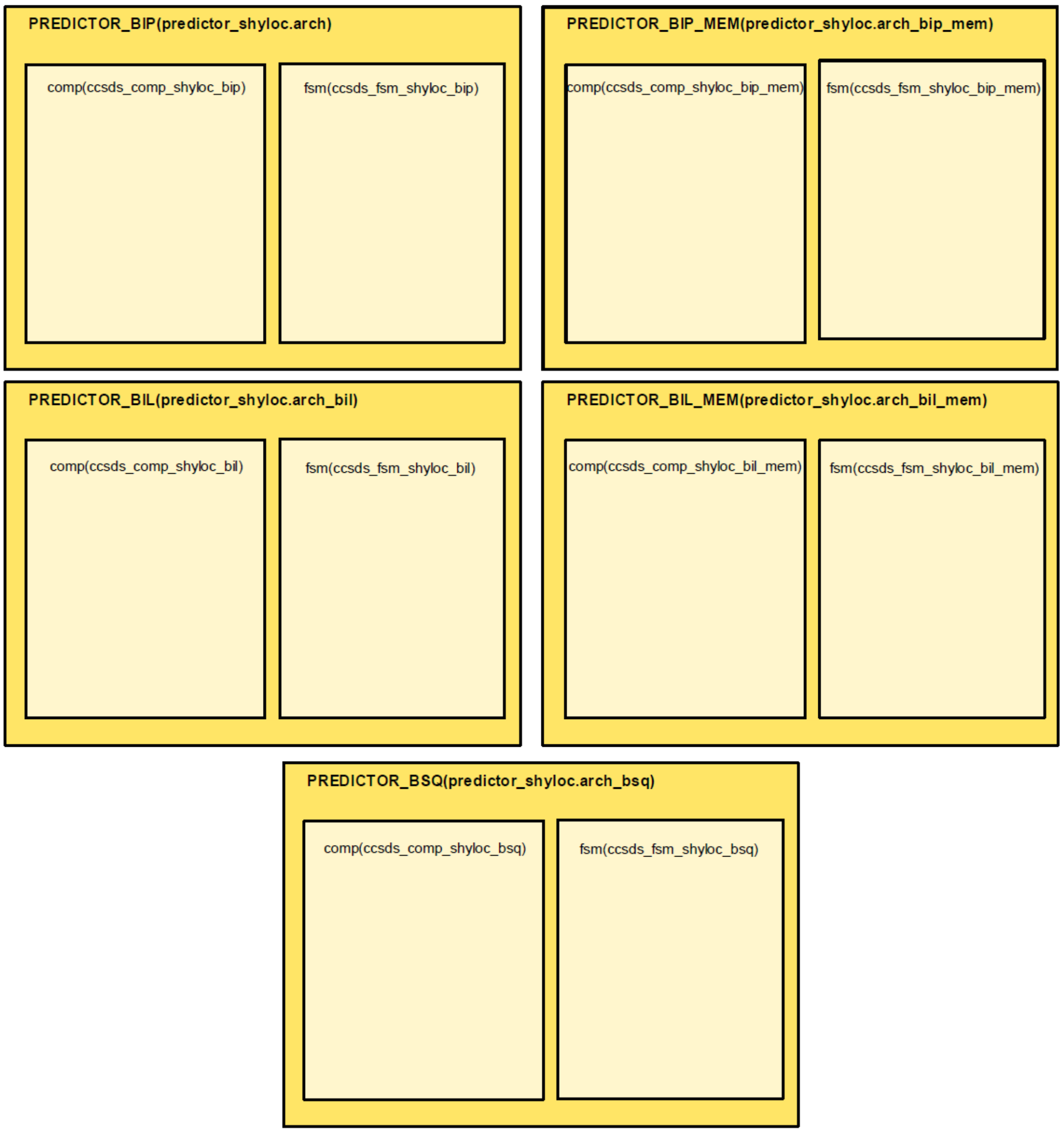

PREDICTOR_BIP(predictor_shyloc.arch)
comp(ccsds_comp_shyloc_bip)
fsm(ccsds_fsm_shyloc_bip)
PREDICTOR_BIP_MEM(predictor_shyloc.arch_bip_mem)
comp(ccsds_comp_shyloc_bip_mem)
fsm(ccsds_fsm_shyloc_bip_mem)
PREDICTOR_BIL(predictor_shyloc.arch_bil)
comp(ccsds_comp_shyloc_bil)
fsm(ccsds_fsm_shyloc_bil)
PREDICTOR_BIL_MEM(predictor_shyloc.arch_bil_mem)
comp(ccsds_comp_shyloc_bil_mem)
fsm(ccsds_fsm_shyloc_bil_mem)
PREDICTOR_BSQ(predictor_shyloc.arch_bsq)
comp(ccsds_comp_shyloc_bsq)
fsm(ccsds_fsm_shyloc_bsq)

Figure 4-13: CCSDS123 IP – Predictor architectures (BIP, BIP-MEM, BIL, BIL-MEM and BSQ)

A basic block diagram of the predictor is shown in Figure 4-14. This basic block diagram is an abstract representation that is common to all the predictor architectures. The input samples to be compressed are first arranged in a set of FIFOs, as determined by the compression order, in such a way that the already compressed samples become the neighbouring samples of the pixels subsequently processed. The amount of samples stored in these FIFOs depends on the compression order and the compile-time parameters, estimations are given in Section 4.6. The storage elements which require the highest amount of memory are depicted in blue, and placed in an external memory depending on the architecture. The modules with the highest mathematical complexity are shown in red.

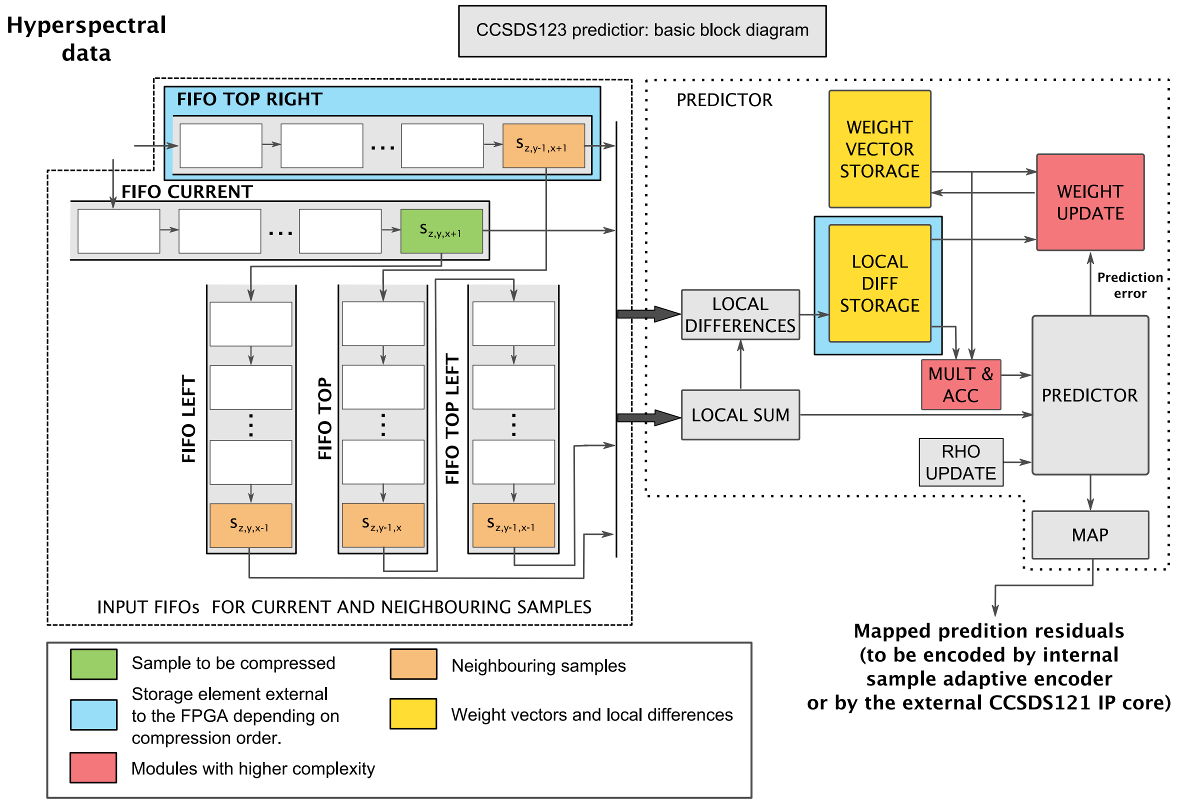

Hyperspectral data
CCSDS123 predictior: basic block diagram
FIFO TOP RIGHT
$s_{z,y-1,x+1}$
FIFO CURRENT
$s_{z,y,x+1}$
FIFO LEFT
$s_{z,y,x-1}$
FIFO TOP
$s_{z,y-1,x}$
FIFO TOP LEFT
$s_{z,y-1,x-1}$
INPUT FIFOs FOR CURRENT AND NEIGHBOURING SAMPLES
PREDICTOR
WEIGHT VECTOR STORAGE
WEIGHT UPDATE
LOCAL DIFF STORAGE
Prediction error
LOCAL DIFFERENCES
MULT & ACC
PREDICTOR
LOCAL SUM
RHO UPDATE
MAP
Mapped predition residuals (to be encoded by internal sample adaptive encoder or by the external CCSDS121 IP core)
Sample to be compressed
Neighbouring samples
Storage element external to the FPGA depending on compression order.
Weight vectors and local differences
Modules with higher complexity

Figure 4-14: CCSDS123 IP – Basic predictor block diagram

4.5.2.1 BIP/BIP-MEM architectures

The architecture described for BIP takes into account that, provided enough samples in the spectral dimension, the predictor has the possibility of accepting one compressed sample per clock cycle. This is hence the compression order with the highest possible throughput. The BIP-MEM architecture stores the FIFO TOP RIGHT in an external memory, whereas the BIP architecture only uses internal memory. The former communicates with the external memory using the AHB master interface. One read and one write operation is needed for the compression of a sample.

The local differences vector (see Table 4-1) is stored in the internal FPGA memory. The multiply and accumulation operations needed for the computation of the dot product of the local differences and weight vectors needed for the prediction are performed using the structure depicted in Figure 4-15. This structure makes it possible to obtain a dot product result every clock cycle, provided there are two input vectors every clock cycle. The weight vectors are updated in parallel using instances of weight update units as shown in Figure 4-16. The amount of instances of multipliers and accumulators used for the dot product and the amount of weight vector units are calculated based on the compile-time parameter P_MAX.

During the compression of a pixel in all the spectral bands $[s_{x,y,0} \,..\, s_{x,y,N_z-1}]$, the obtained weight vectors in each band are stored in a FIFO structure, until needed for the compression of the next pixel $s_{x+1,y,0}$.

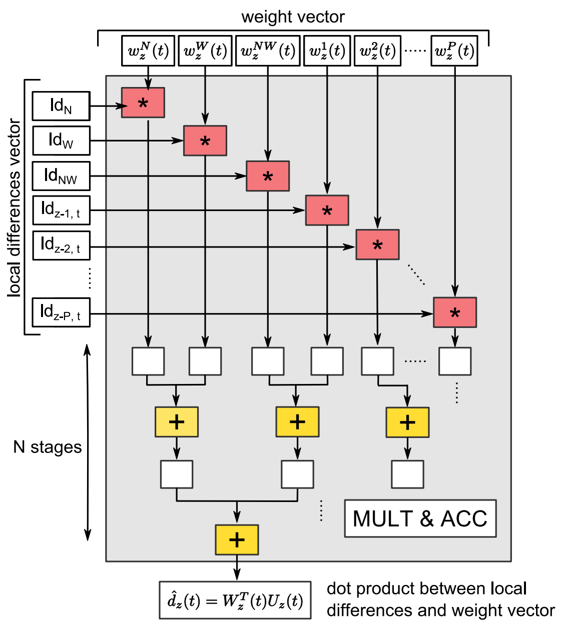


Figure 4-15: CCSDS123 IP – Multiply and accumulate structure to perform the dot product in BIP and BIL.

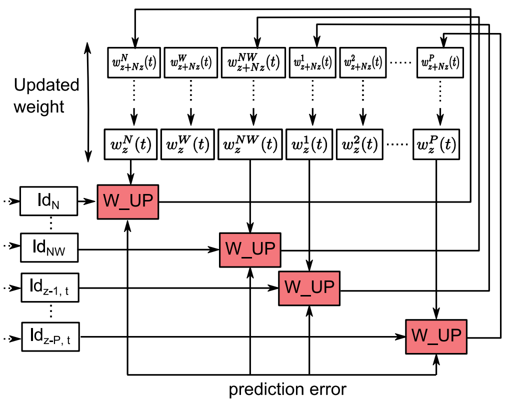


Figure 4-16: CCSDS123 IP – Weight vector update in BIP and BIL.

4.5.2.2 BSQ architecture

The main differences between the BSQ and the BIP architecture lies in the allocation of the local differences vector in an external memory, and the scheduling of the multiply and accumulate operations, which are performed serially.

In BSQ, it is necessary to store a complete vector of local differences per sample during the compression of a band. These local differences vectors (a total of $N_x \times N_y$) are stored in an external memory and sent through the AHB master interface. The memory addresses in which the vectors are stored are calculated by the IP core as shown in Figure 4-17, in such a way that the memory locations are appropriately reused when available. One local difference value needs to be stored per sample, and P need to be read. Within a vector, decreasing write addresses and increasing read addresses are used.

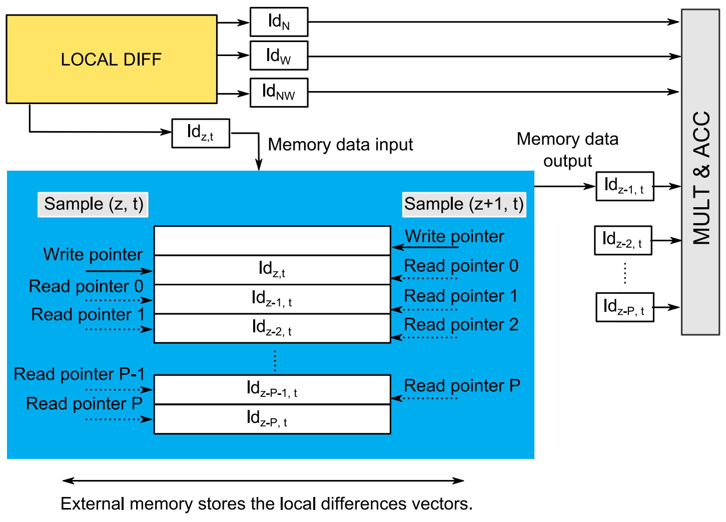


Figure 4-17: CCSDS123 IP – Storage of local differences vector for each sample in a band in an external memory in BSQ order.

In BSQ, data dependencies place an important throughput limitation. Provided that the local differences need to be retrieved from the external memory, we observe that there is no clear advantage in having the multiply and accumulate operations and the weight update performed with the highly parallel structures used for BIP and BIL. Conversely, this operation and the weight vector update are serialized in order to reduce the resource utilization. The rest of the predictor uses the same components as the BIP architecture.

4.5.2.3 BIL/BIL-MEM architectures

BIL inherits most of the components from the BIP architecture. The main difference resides in the local difference vector storage. In BIL, it is necessary to store one vector per sample in a line of pixels. This storage is placed inside the FPGA, in several chained FIFOs, as shown in Figure 4-18.

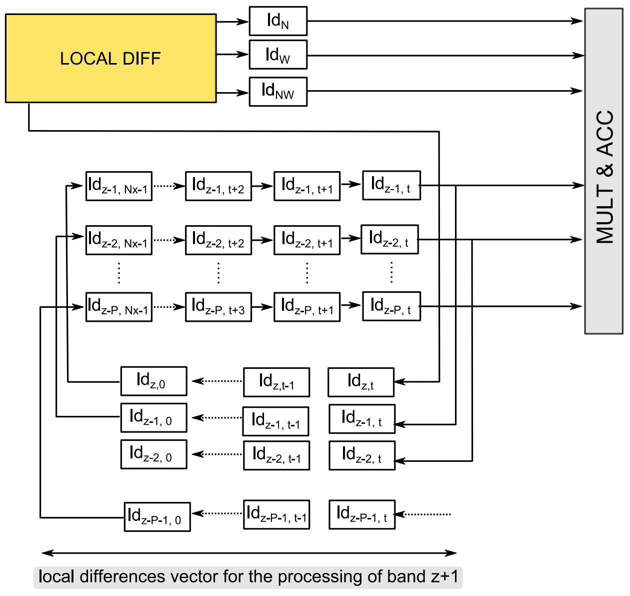


Figure 4-18: CCSDS123 IP – Storage of local differences vector in BIL order.

In the same way that the BIP architectures, the BIL-MEM architecture stores the FIFO TOP RIGHT in an external memory, whereas the BIL architecture only uses internal memory available inside the FPGA.

The structures to compute the dot product and weight update operations are the same as in BIP. A specific scheduling is devised for BIL in order to ensure that the maximum possible throughput is achieved in both situations, when compressing the samples in a line, where we find the same data dependencies as in BSQ, and when compressing the last sample of a line and the first of the next line. In the latter, the data dependencies we find are the same as in BIP.

4.5.3 Sample-adaptive encoder block diagram

A simplified block diagram of the sample-adaptive encoder is shown in Figure 4-19. It contains a **components** module that instantiates the necessary components to perform the encoding, and an **fsm** module that performs the control of the instantiated components, creating a highly-optimized pipeline which allows for encoding a sample every clock cycle.

The sample-adaptive encoder receives the mapped residuals from the CCSDS123 predictor. It stores the input values in a FIFO, and then performs the entropy encoding. The **opcode_update** module generates an operation code depending on the relative location of the sample in the image. This module is tailored to the specific compression order (BIP, BSQ and BIL), including a different architecture for each order. The counters and accumulator values needed to perform the encoding are computed by the **update_counters** module, which also defines a different architecture for each compression order. We note that, when compressing in BIP and BIL, the accumulator values of a sample in all bands are stored in a FIFO, so that they are available when compressing the next sample.

Finally, the module **createcdw** creates the codeword from the input mapped residual, and the computed counter and accumulator values, which are then packed in the **bit_pack** module. This module contains a register buffer of the bit width of the **DataOut** signal where the input values are packed. Once the buffer is full, it the packing module generates a valid flag so that the value can be captured at the output.

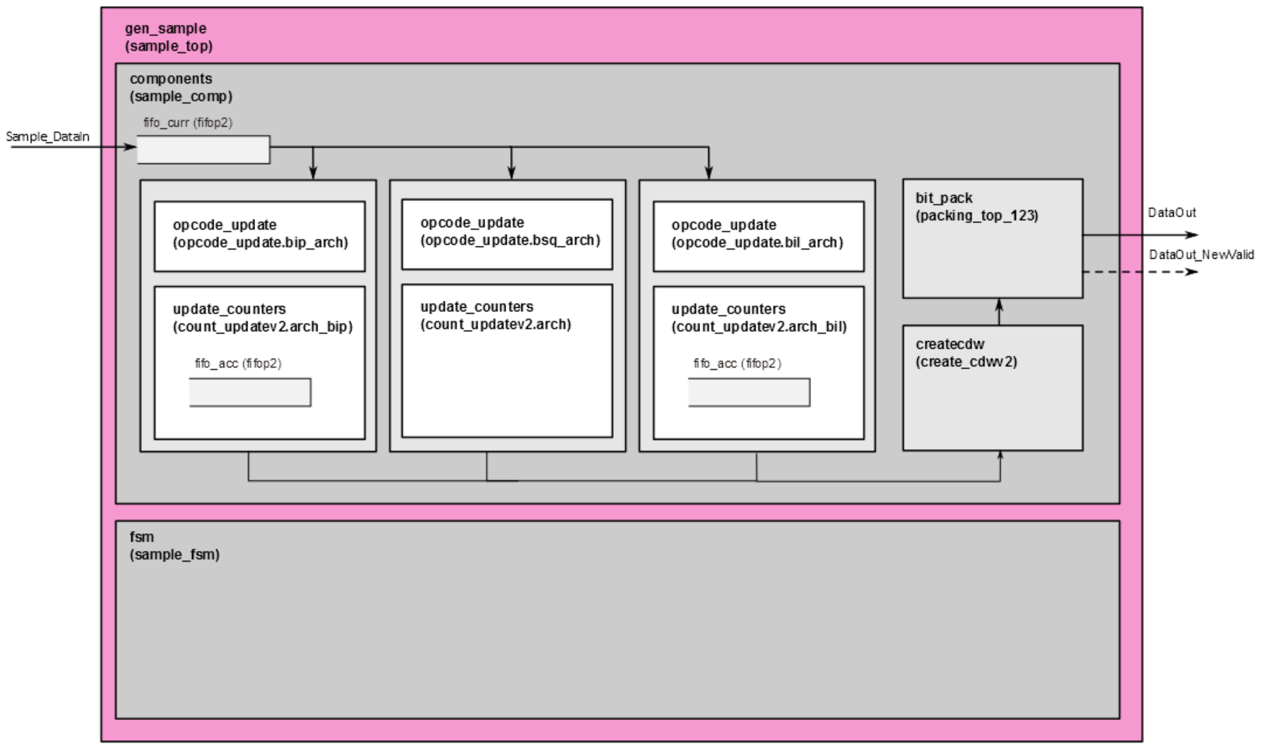


Figure 4-19: CCSDS123 IP – Block diagram of the sample-adaptive encoder.

4.6 Complexity scalability with compile-time parameters

The following sections present the relationship between the complexity and the compile-time parameters in the design in terms of internal and external memory use and computational load. The derivation of the constants that affect complexity from the compile-time parameters is presented. Compile-time parameters are marked in bold.

The following mathematical operators, in order to calculate the constants:

- maximum(*a, b*) → returns the maximum between two unsigned integers.
- log2(*a*) → returns the smallest integer which is greater or equal to the base-2 logarithm of *a*.
- ceil(*a, b*) → returns the smallest integer which is greater or equal to *a*/*b*.
- 2**(*a*) → power-of-two of *a*.
- aling_byte (*a*) → returns the smallest multiple of 8 which is greater or equal to *a*.

4.6.1 BIP

4.6.1.1 Internal storage

4.6.1.1.1 Predictor

TABLE 4-20: CCSDS123 IP – CONSTANTS THAT AFFECT THE AMOUNT OF INTERNAL STORAGE IN BIP (PREDICTOR)

Constant name	Value	Description
Cz_GEN	**P_MAX** + 3 when PREDICTION_GEN = 0 **P_MAX** when PREDICTION_GEN = 1	Number of elements in the local differences and weight vectors.
W_LD	**D_GEN** + 4	Bit width of the local differences values.
W_WEI	**OMEGA_GEN** + 3	Bit width of the weight values.
NE_CURR	16	Number of samples stored in the input FIFO.
W_ADDR_CURR	log2(NE_CURR)	Bit width of the address pointer of the input FIFO.
NE_LEFT_BIP	**Nz_GEN**	Number of elements in the FIFO that stores the LEFT neighbours.
W_ADDR_LEFT_BIP	log2(**Nz_GEN**)	Bit width of the address pointer of the FIFO that stores the LEFT neighbours.
NE_TOP_LEFT_BIP	**Nz_GEN**	Number of elements in the FIFO that stores the TOP LEFT neighbours.
W_ADDR_TOP_LEFT_BIP	**log2(NE_TOP_LEFT_BIP)**	Bit width of the address pointer of the FIFO that stores the TOP LEFT neighbours.
NE_TOP_BIP	**Nz_GEN**	Number of elements in the FIFO that stores the TOP neighbours.
W_ADDR_TOP_BIP	log2(**Nz_GEN**)	Bit width of the address pointer of the FIFO that stores the TOP neighbours.
NE_TOP_RIGHT_BIP	**Nz_GEN*Nx_GEN**	Number of elements in the FIFO that stores the TOP RIGHT neighbours.
W_ADDR_TOP_RIGHT_BIP	log2(NE_TOP_RIGHT_BIP)	Bit width of the address pointer of the FIFO that stores the TOP RIGHT neighbours.
HEIGHT_TREE	log2(ceil(Cz_GEN))	Height of the multipliers and accumulators tree for the dot product.
CYCLES_PRED	2	Number of clock cycles that takes to obtain a predicted value from the dot product result.
W_ADDR_TOP_LEFT_BIP	log2(NE_TOP_LEFT_BIP)	Bit width of the address pointer of the FIFO that stores the TOP LEFT neighbours.
NE_DOT_TO_UPDATE_BIP	HEIGHT_TREE + 1 + CYCLES_PRED	Number of local differences and weight vectors that need to be stored while the dot product operation is performed; to be then used by the weight update module.
W_ADDR_DOT_TO_UPDATE_BIP	log2(NE_DOT_TO_UPDATE_BIP)	Bit width of the address pointer of the FIFO that stores the local differences and weight vectors while the dot product operation is performed; to be then used by the weight update module.
W_ADDR_WEIGHT_UPDATE_BIP	log2(**Nz_GEN**)	Bit width of the address pointer of the FIFO that stores the weight vectors during the compression of a sample in all bands.

TABLE 4-21: CCSDS123 IP – WIDTH AND DEPTH OF THE FIFOS IN BIP (PREDICTOR)

FIFO	Width (no EDAC)*	Depth	Notes
fifo_0_curr	**D_GEN**	2**(W_ADDR_CURR)	
fifo_1_left	**D_GEN**	2**(W_ADDR_LEFT_BIP)	
fifo_2_top	**D_GEN**	2**(W_ADDR_TOP_BIP)	
fifo_3_top_left	**D_GEN**	2**(W_ADDR_TOP_LEFT_BIP)	
fifo_4_top_right	**D_GEN**	2**(W_ADDR_TOP_RIGHT_BIP)	
fifo_5_ld_vector_temp_storage	W_LD	2**(W_ADDR_DOT_TO_UPDATE_BIP) ***Cz_GEN**	Generated only if Cz_GEN > 0.
fifo_6_wei_vector_temp_storage	W_WEI	2**(W_ADDR_DOT_TO_UPDATE_BIP) ***Cz_GEN**	Generated only if Cz_GEN > 0.
fifo_7_wei_update_storage	W_WEI	2**(W_ADDR_WEIGHT_UPDATE_BIP)***Cz_GEN**	Generated only if Cz_GEN > 0. 2-dimensional FIFO, consists of Cz_GEN FIFOs.

* If EDAC is enabled, the width is calculated as: aling_byte (Width) + check_bits.

4.6.1.1.2 Sample-adaptive

TABLE 4-22: CCSDS123 IP – CONSTANTS THAT AFFECT THE AMOUNT OF INTERNAL STORAGE IN BIP (SAMPLE-ADAPTIVE)

Constant name	Value	Description
NE_CURR_SAMPLE	16	Number of samples stored in the input FIFO.
W_ADDR_CURR_SAMPLE	log2(NE_CURR_SAMPLE)	Bit width of the address pointer of the input FIFO.
W_ACC	32	Bit width of the accumulator values.
NE_FIFO	**Nz_GEN**	Number of elements in the FIFO that stores the accumulator values.
W_FIFO	log2(**Nz_GEN**)	Bit width of the address pointer of the FIFO that stores the accumulator values.

TABLE 4-23: CCSDS123 IP – WIDTH AND DEPTH OF THE FIFOS IN BIP (SAMPLE-ADAPTIVE)

FIFO	Width (no EDAC)*	Depth	Notes
fifo_curr	**D_GEN**	2**(W_ADDR_CURR_SAMPLE)	
fifoacc	W_ACC	2**(W_FIFO)	

4.6.1.2 Complexity

TABLE 4-24: CCSDS123 IP – CONSTANTS THAT AFFECT THE COMPUTATIONAL COMPLEXITY IN BIP

Operation	Related constant	Value

Number of parallel multipliers in the dot product operation.	Cz_GEN	**P_MAX** + 3 when PREDICTION_GEN = 0 **P_MAX** when PREDICTION_GEN = 1
Number of instances of weight update modules instantiated.	Cz_GEN	**P_MAX** when PREDICTION_GEN = 0 **P_MAX** when PREDICTION_GEN = 1
Bit width of the input of the multipliers.	W_LD	**D_GEN** + 4
Bit width of the input of the multipliers.	W_WEI	**OMEGA_GEN** + 3
Bit width of the result of the dot product operation.	W_DZ	W_LD + W_WEI + maximum(1, log2(Cz))
Number of parallel adders in the dot product tree.	N_ADDERS_POW2	2**log2(ceil(**Cz_GEN**, 2))

4.6.2 BIP-MEM

In BIP-MEM, the FIFO that stores the top right neighbours is placed in an external memory. Asynchronous FIFOs are used for clock adaptation between the AHB master interface and the IP core. The rest of elements in the architecture are the same as in BIP.

4.6.2.1 Internal storage

4.6.2.1.1 Predictor

TABLE 4-25: CCSDS123 IP – CONSTANTS THAT AFFECT THE AMOUNT OF INTERNAL STORAGE IN BIP-MEM (PREDICTOR)

Constant name	Value	Description
Cz_GEN	**P_MAX** +3 when PREDICTION_GEN = 0 **P_MAX** when PREDICTION_GEN = 1	Number of elements in the local differences and weight vectors.
W_LD	**D_GEN** + 4	Bit width of the local differences values.
W_WEI	**OMEGA_GEN** + 3	Bit width of the weight values.
NE_CURR	16	Number of samples stored in the input FIFO.
W_ADDR_CURR	log2(NE_CURR)	Bit width of the address pointer of the input FIFO.
NE_LEFT_BIP	**Nz_GEN**	Number of elements in the FIFO that stores the LEFT neighbours.
W_ADDR_LEFT_BIP	log2(**Nz_GEN**)	Bit width of the address pointer of the FIFO that stores the LEFT neighbours.
NE_TOP_LEFT_BIP	**Nz_GEN**	Number of elements in the FIFO that stores the TOP LEFT neighbours.
W_ADDR_TOP_LEFT_BIP	**log2(NE_TOP_LEFT_BIP)**	Bit width of the address pointer of the FIFO that stores the TOP LEFT neighbours.
NE_TOP_BIP	**Nz_GEN**	Number of elements in the FIFO that stores the TOP neighbours.
W_ADDR_TOP_BIP	log2(**Nz_GEN**)	Bit width of the address pointer of the FIFO that stores the TOP neighbours.
DIFFERENCE_AHB_BIP	5	Number of spots left in the AHB asynchronous FIFO before raising the half full flag.

NE_AHB_FIFO_BIP	maximum(2*DIFFERENCE_AHB_BIP, 16)	Number of elements in the AHB asynchronous FIFO.
HEIGHT_TREE	log2(ceil(**Cz_GEN**))	Height of the multipliers and accumulators tree for the dot product.
CYCLES_PRED	2	Number of clock cycles that takes to obtain a predicted value from the dot product result.
W_ADDR_TOP_LEFT_BIP	log2(NE_TOP_LEFT_BIP)	Bit width of the address pointer of the FIFO that stores the TOP LEFT neighbours.
NE_DOT_TO_UPDATE_BIP	HEIGHT_TREE + 1 + CYCLES_PRED	Number of local differences and weight vectors that need to be stored while the dot product operation is performed; to be then used by the weight update module.
W_ADDR_DOT_TO_UPDATE_BIP	log2(NE_DOT_TO_UPDATE_BIP)	Bit width of the address pointer of the FIFO that stores the local differences and weight vectors while the dot product operation is performed; to be then used by the weight update module.
W_ADDR_WEIGHT_UPDATE_BIP	log2(Nz_GEN)	Bit width of the address pointer of the FIFO that stores the weight vectors during the compression of a sample in all bands.

TABLE 4-26: CCSDS123 IP – WIDTH AND DEPTH OF THE FIFOS IN BIP-MEM (PREDICTOR)

FIFO	Width (no EDAC)*	Depth	Notes
fifo_0_curr	**D_GEN**	2**(W_ADDR_CURR)	
fifo_1_left	**D_GEN**	2**(W_ADDR_LEFT_BIP)	
fifo_2_top	**D_GEN**	2**(W_ADDR_TOP_BIP)	
fifo_3_top_left	**D_GEN**	2**(W_ADDR_TOP_LEFT_BIP)	
fifo_top_right_to_ahb	32	2**(log2(NE_AHB_FIFO_BIP))	
fifo_top_right_from_ahb	32	2**(log2(NE_AHB_FIFO_BIP))	
fifo_4_ld_vector_temp_storage	W_LD	2**(W_ADDR_DOT_TO_UPDATE_BIP) ***Cz_GEN**	Generated only if Cz_GEN > 0.
fifo_5_wei_vector_temp_storage	W_WEI	2**(W_ADDR_DOT_TO_UPDATE_BIP) ***Cz_GEN**	Generated only if Cz_GEN > 0.
fifo_6_wei_update_storage	W_WEI	2**(W_ADDR_WEIGHT_UPDATE_BIP)***Cz_GEN**	Generated only if Cz_GEN > 0. 2-dimensional FIFO, consists of Cz_GEN FIFOs.

* If EDAC is enabled, the width is calculated as: aling_byte (Width) + check_bits.

4.6.2.1.2 Sample-adaptive

TABLE 4-27: CCSDS123 IP – CONSTANTS THAT AFFECT THE AMOUNT OF INTERNAL STORAGE IN BIP-MEM (SAMPLE-ADAPTIVE)

Constant name	Value	Description

NE_CURR_SAMPLE	16	Number of samples stored in the input FIFO.
W_ADDR_CURR_SAMPLE	log2(NE_CURR_SAMPLE)	Bit width of the address pointer of the input FIFO.
W_ACC	32	Bit width of the accumulator values.
NE_FIFO	**Nz_GEN**	Number of elements in the FIFO that stores the accumulator values.
W_FIFO	log2(**Nz_GEN**)	Bit width of the address pointer of the FIFO that stores the accumulator values.

TABLE 4-28: CCSDS123 IP – WIDTH AND DEPTH OF THE FIFOS IN BIP-MEM (SAMPLE-ADAPTIVE)

FIFO	Width (no EDAC)*	Depth	Notes
fifo_curr	**D_GEN**	2**(W_ADDR_CURR_SAMPLE)	
fifoacc	W_ACC	2**(W_FIFO)	

4.6.2.2 External storage

The number of elements stored in the external memory for BIP-MEM depends on the configured Nx and Nz values. Each element is stored using 4 bytes.

TABLE 4-29: CCSDS123 IP – WIDTH AND DEPTH OF THE FIFOS IN BIP-MEM (SAMPLE-ADAPTIVE)

Number of elements stored in the external memory (bytes)	**Nx*Nz*4**

4.6.2.3 Complexity

TABLE 4-30: CCSDS123 IP – CONSTANTS THAT AFFECT THE COMPUTATIONAL COMPLEXITY IN BIP-MEM

Operation	Related constant	Value
Number of parallel multipliers in the dot product operation.	**Cz_GEN**	**P_MAX** + 3 when PREDICTION_GEN = 0 **P_MAX** when PREDICTION_GEN = 1
Number of instances of weight update modules instantiated.	**Cz_GEN**	**P_MAX** + 3 when PREDICTION_GEN = 0 **P_MAX** when PREDICTION_GEN = 1
Bit width of the input of the multipliers.	W_LD	**D_GEN** + 4
Bit width of the input of the multipliers.	W_WEI	**OMEGA_GEN** + 3
Bit width of the result of the dot product operation.	W_DZ	W_LD + W_WEI + maximum(1, log2(**Cz_GEN**))
Number of parallel adders in the dot product tree.	N_ADDERS_POW2	2**log2(ceil(**Cz_GEN**, 2))

4.6.3 BSQ

In BSQ, the amount of elements stored in the neighbouring FIFOs is considerably smaller. Additionally, the local differences vectors are stored in an external memory. The computation of the dot product for prediction is serialized.

4.6.3.1 Internal storage

4.6.3.1.1 Predictor

TABLE 4-31: CCSDS123 IP – CONSTANTS THAT AFFECT THE AMOUNT OF INTERNAL STORAGE IN BSQ (PREDICTOR)

Constant name	Value	Description
Cz_GEN	**P_MAX** +3 when PREDICTION_GEN = 0 **P_MAX** when PREDICTION_GEN = 1	Number of elements in the local differences and weight vectors.
W_LD	**D_GEN** + 4	Bit width of the local differences values.
W_WEI	**OMEGA_GEN** + 3	Bit width of the weight values.
W_ADDR_IN_IMAGE	16	Bits used to represent the image coordinates values.
W_LS	**D_GEN** + 2	Bit width of the local sum values.
NE_CURR	16	Number of samples stored in the input FIFO.
W_ADDR_CURR	log2(NE_CURR)	Bit width of the address pointer of the input FIFO.
NE_TOP_RIGHT_BSQ	**Nx_GEN**	Number of elements in the FIFO that stores the TOP RIGHT neighbours.
W_ADDR_TOP_RIGHT_BSQ	log2(NE_TOP_RIGHT_BSQ)	Bit width of the address pointer of the FIFO that stores the TOP RIGHT neighbours.
NE_RECORD_BSQ	16	Number of elements in the FIFO that stores values while the local difference vectors are being retrieved from the external memory.
W_ADDR_RECORD_BSQ	4	Bit width of the address pointer of the FIFO that stores values while the local difference vectors are being retrieved from the external memory.
NE_LD_BSQ	**P_MAX**	Number of elements in the FIFO that stores a local difference vector.
W_ADDR_LD_BSQ	log2(**P_MAX**)	Bit width of the address pointer of the FIFO that stores a local difference vector.
NE_WEI_BSQ	**P_MAX**	Number of elements in the FIFO that stores a local weight vector.
W_ADDR_WEI_BSQ	log2(**P_MAX**)	Bit width of the address pointer of the FIFO that stores a local weight vector.
DIFFERENCE_AHB_BSQ	14	Number of spots left in the AHB asynchronous FIFO before raising the half full flag.
NE_AHB_FIFO_BSQ	maximum(2*DIFFERENCE_AHB_BSQ, 16)	Number of elements in the AHB asynchronous FIFO.
NE_LD_DIR_BSQ	HEIGHT_TREE + 1	Number of elements in the FIFO that stores the directional local differences until the weight update operation.
W_ADDR_LD_DIR_BSQ	log2(NE_LD_DIR_BSQ)	Bit width of the address pointer of the FIFO that stores the directional local differences until the weight update operation.

TABLE 4-32: CCSDS123 IP – WIDTH AND DEPTH OF THE FIFOS IN BSQ (PREDICTOR)

FIFO	Width (no EDAC)*	Depth	Notes
fifo_0_curr	**D_GEN**	2**(W_ADDR_CURR)	
fifo_1_top_right	**D_GEN**	2**(W_ADDR_TOP_RIGHT_BSQ)	
record_fifo_2. fifo_0_2d_ld	W_LD	2**(W_ADDR_RECORD_BSQ)***4** when PREDICTION_GEN = 0 2**(W_ADDR_RECORD_BSQ) when PREDICTION_GEN = 0	2-dimensional FIFO.
record_fifo_2. fifo_1_opcode	W_OPC_PREDICT	2**(W_ADDR_RECORD_BSQ)	
record_fifo_2. fifo_2_s_predict	**D_GEN**+1	2**(W_ADDR_RECORD_BSQ)	
record_fifo_2. fifo_3_ls_predict	W_LS	2**(W_ADDR_RECORD_BSQ)	
record_fifo_2. fifo_4_z_predict	W_ADDR_IN_IMAGE	2**(W_ADDR_RECORD_BSQ)	
record_fifo_2. fifo_5_t_predict	2*W_ADDR_IN_IMAGE	2**(W_ADDR_RECORD_BSQ)	
fifo_ld_to_ahb	32	2**(log2(NE_AHB_FIFO_BSQ))	
fifo_ld_from_ahb	32	2**(log2(NE_AHB_FIFO_BSQ))	
fifo_3_ld	W_LD	2**(W_ADDR_LD_BSQ)	Generated only if Cz_GEN > 0.
localdiff_dir_fifo_4	W_LD	2**(W_ADDR_LD_DIR_BSQ)*3	Generated only if PREDICTION_GEN = 0. 2-dimensional FIFO, consists of 3 FIFOs.
fifo_5_weight_update	W_WEI	2**(W_ADDR_WEI_BSQ)	Generated only if P_MAX > 0.

* If EDAC is enabled, the width is calculated as: aling_byte (Width) + check_bits.

4.6.3.1.2 Sample-adaptive

TABLE 4-33: CCSDS123 IP – CONSTANTS THAT AFFECT THE AMOUNT OF INTERNAL STORAGE IN BSQ (SAMPLE-ADAPTIVE)

Constant name	Value	Description
NE_CURR_SAMPLE	16	Number of samples stored in the input FIFO.
W_ADDR_CURR_SAMPLE	log2(NE_CURR_SAMPLE)	Bit width of the address pointer of the input FIFO.
W_ACC	32	Bit width of the accumulator values.
NE_FIFO	**Nz_GEN**	Number of elements in the FIFO that stores the accumulator values.

W_FIFO	log2(**Nz_GEN**)	Bit width of the address pointer of the FIFO that stores the accumulator values.

TABLE 4-34: CCSDS123 IP – WIDTH AND DEPTH OF THE FIFOS IN BSQ (SAMPLE-ADAPTIVE)

FIFO	Width (no EDAC)*	Depth	Notes
fifo_curr	**D_GEN**	2**(W_ADDR_CURR_SAMPLE)	
fifoacc	W_ACC	2**(W_FIFO)	

4.6.3.2 External storage

The number of elements stored in the external memory for BSQ depends on the configured Nx, Ny and P values. Each element is stored using 4 bytes.

TABLE 4-35: CCSDS123 IP – WIDTH AND DEPTH OF THE FIFOS IN BSQ (SAMPLE-ADAPTIVE)

Number of elements stored in the external memory (bytes)	**P*Nx*Ny*4**

4.6.3.3 Complexity

TABLE 4-36: CCSDS123 IP – CONSTANTS THAT AFFECT THE COMPUTATIONAL COMPLEXITY IN BSQ

Operation	Related constant	Value
Number of parallel multipliers in the dot product operation.	PREDICTION_GEN	4 when **PREDICTION_GEN** = 0 1 when **PREDICTION_GEN** = 1
Number of instances of weight update modules instantiated.	PREDICTION_GEN	4 when **PREDICTION_GEN** = 0 1 when **PREDICTION_GEN** = 1
Bit width of the input of the multipliers.	W_LD	**D_GEN** + 4
Bit width of the input of the multipliers.	W_WEI	**OMEGA_GEN** + 3
Bit width of the result of the dot product operation.	W_DZ	W_LD + W_WEI + maximum(1, log2(**Cz_GEN**));
Number of parallel adders in the dot product tree.	PREDICTION_GEN	4 when PREDICTION_GEN = 0 1 when PREDICTION_GEN = 1

4.6.4 BIL

4.6.4.1 Internal storage

4.6.4.1.1 Predictor

TABLE 4-37: CCSDS123 IP – CONSTANTS THAT AFFECT THE AMOUNT OF INTERNAL STORAGE IN BIL (PREDICTOR)

Constant name	Value	Description

Cz_GEN	**P_MAX** + 3 when PREDICTION_GEN = 0 **P_MAX** when PREDICTION_GEN = 1	Number of elements in the local differences and weight vectors.
W_LD	**D_GEN** + 4	Bit width of the local differences values.
W_WEI	**OMEGA_GEN** + 3	Bit width of the weight values.
NE_CURR	16	Number of samples stored in the input FIFO.
W_ADDR_CURR	log2(NE_CURR)	Bit width of the address pointer of the input FIFO.
TOP_RIGHT_BIL	Nz_GEN*Nx_GEN	Number of elements in the FIFO that stores the TOP RIGHT neighbours.
NE_AHB_FIFO_BIL	maximum(2*DIFFERENCE_AHB_BIL, 16)	Number of elements in the AHB asynchronous FIFO.
W_ADDR_TOP_RIGHT_BIL	log2(NE_TOP_RIGHT_BIP)	Bit width of the address pointer of the FIFO that stores the TOP RIGHT neighbours.
NE_RECORD_BIL	16	Number of elements in the FIFO that stores values while the local difference vectors are being retrieved from the external memory.
W_ADDR_RECORD_BIL	4	Bit width of the address pointer of the FIFO that stores values while the local difference vectors are being retrieved from the external memory.
W_ADDR_CENTRAL_BIL	log2(**Nx_GEN**);	Bit width of the address pointer of the FIFO that stores the local differences vector during the compression of a line of pixels in all bands.
HEIGHT_TREE	log2(ceil(**Cz_GEN**))	Height of the multipliers and accumulators tree for the dot product.
CYCLES_PRED	2	Number of clock cycles that takes to obtain a predicted value from the dot product result.
W_ADDR_DOT_TO_UPDATE_BIL	2	Bit width of the address pointer of the FIFO that stores the local differences and weight vectors while the dot product operation is performed; to be then used by the weight update module.
W_ADDR_WEIGHT_UPDATE_BIL	log2(**Nz_GEN**)	Bit width of the address pointer of the FIFO that stores the weight vectors during the compression of a sample in all bands.

TABLE 4-38: CCSDS123 IP – WIDTH AND DEPTH OF THE FIFOS IN BIL (PREDICTOR)

FIFO	**Width (no EDAC)***	**Depth**	**Notes**
fifo_0_curr	**D_GEN**	2**(W_ADDR_CURR)	
fifo_1_top_right	**D_GEN**	2**(W_ADDR_TOP_RIGHT_BIL)	
record_fifo_2. fifo_0_2d_ld	W_LD	2**(W_ADDR_RECORD_BIL)***4** when PREDICTION_GEN = 0 2**(W_ADDR_RECORD_BIL) when PREDICTION_GEN = 0	2-dimensional FIFO.

record_fifo_2.fifo_1_opcode	W_OPC_PREDICT	2**(W_ADDR_RECORD_BIL)	
record_fifo_2.fifo_2_s_predict	**D_GEN**+1	2**(W_ADDR_RECORD_BIL)	
record_fifo_2.fifo_3_ls_predict	W_LS	2**(W_ADDR_RECORD_BIL)	
record_fifo_2.fifo_4_z_predict	W_ADDR_IN_IMAGE	2**(W_ADDR_RECORD_BIL)	
record_fifo_2.fifo_5_t_predict	2*W_ADDR_IN_IMAGE	2**(W_ADDR_RECORD_BIL)	
ld_central_vector_fifo_3	W_LD	2**(W_ADDR_CENTRAL_BIL)***P_MAX**	Only generated if P_MAX > 0. 2-dimensional FIFO, instantiates P_MAX FIFOs.
fifo_4_ld_store_to_update	W_LD	2**(W_ADDR_DOT_TO_UPDATE_BIL)***Cz_GEN**	Only generated if Cz_GEN > 0. 2-dimensional FIFO, instantiates Cz_GEN FIFOs.
fifo_5_wei_storage_from_dot_to_update	W_WEI	2**(W_ADDR_DOT_TO_UPDATE_BIL)***Cz_GEN**	Only generated if Cz_GEN > 0. 2-dimensional FIFO, instantiates Cz_GEN FIFOs.
fifo_6_wei_update_storage	W_WEI	2**(W_ADDR_WEIGHT_UPDATE_BIL)***Cz_GEN**	Only generated if Cz_GEN > 0. 2-dimensional FIFO, instantiates Cz_GEN FIFOs.

* If EDAC is enabled, the width is calculated as: aling_byte (Width) + check_bits.

4.6.4.1.2 Sample-adaptive

TABLE 4-39: CCSDS123 IP – CONSTANTS THAT AFFECT THE AMOUNT OF INTERNAL STORAGE IN BIL (SAMPLE-ADAPTIVE)

Constant name	Value	Description
NE_CURR_SAMPLE	16	Number of samples stored in the input FIFO.
W_ADDR_CURR_SAMPLE	log2(NE_CURR_SAMPLE)	Bit width of the address pointer of the input FIFO.
W_ACC	32	Bit width of the accumulator values.
NE_FIFO	**Nz_GEN**	Number of elements in the FIFO that stores the accumulator values.
W_FIFO	log2(**Nz_GEN**)	Bit width of the address pointer of the FIFO that stores the accumulator values.

TABLE 4-40: CCSDS123 IP – WIDTH AND DEPTH OF THE FIFOS IN BIL (SAMPLE-ADAPTIVE)

FIFO	Width (no EDAC)*	Depth	Notes
fifo_curr	**D_GEN**	2**(W_ADDR_CURR_SAMPLE)	

fifoacc	W_ACC	2**(W_FIFO)	

4.6.4.2 Complexity

TABLE 4-41: CCSDS123 IP – CONSTANTS THAT AFFECT THE COMPUTATIONAL COMPLEXITY IN BIL

Operation	Related constant	Value
Number of parallel multipliers in the dot product operation.	Cz_GEN	**P_MAX** + 3 when PREDICTION_GEN = 0 **P_MAX** when PREDICTION_GEN = 1
Number of instances of weight update modules instantiated.	Cz_GEN	**P_MAX** + 3 when PREDICTION_GEN = 0 **P_MAX** when PREDICTION_GEN = 1
Bit width of the input of the multipliers.	W_LD	**D_GEN** + 4
Bit width of the input of the multipliers.	W_WEI	**OMEGA_GEN** + 3
Bit width of the result of the dot product operation.	W_DZ	W_LD + W_WEI + maximum(1, log2(**Cz_GEN**))
Number of parallel adders in the dot product tree.	N_ADDERS_POW2	2**log2(ceil(**Cz_GEN,** 2))

4.6.5 BIL-MEM

As it has been aforementioned, in BIL-MEM the FIFO that stores the top right neighbours is placed in an external memory, as in the BIP-MEM architecture. Therefore, asynchronous FIFOs are used too for clock adaptation between the AHB master interface and the IP core. The rest of elements in the architecture are the same as in BIL.

4.6.5.1 Internal storage

4.6.5.1.1 Predictor

TABLE 4-42: CCSDS123 IP – CONSTANTS THAT AFFECT THE AMOUNT OF INTERNAL STORAGE IN BIL-MEM (PREDICTOR)

Constant name	Value	Description
Cz_GEN	**P_MAX** + 3 when PREDICTION_GEN = 0 **P_MAX** when PREDICTION_GEN = 1	Number of elements in the local differences and weight vectors.
W_LD	**D_GEN** + 4	Bit width of the local differences values.
W_WEI	**OMEGA_GEN** + 3	Bit width of the weight values.
NE_CURR	16	Number of samples stored in the input FIFO.
W_ADDR_CURR	log2(NE_CURR)	Bit width of the address pointer of the input FIFO.
DIFFERENCE_AHB_BIL	5	Number of spots left in the AHB asynchronous FIFO before raising the half full flag.
W_ADDR_TOP_RIGHT_BIL	log2(NE_TOP_RIGHT_BIP)	Bit width of the address pointer of the FIFO that stores the TOP RIGHT neighbours.

NE_RECORD_BIL	16	Number of elements in the FIFO that stores values while the local difference vectors are being retrieved from the external memory.
W_ADDR_RECORD_BIL	4	Bit width of the address pointer of the FIFO that stores values while the local difference vectors are being retrieved from the external memory.
W_ADDR_CENTRAL_BIL	log2(**Nx_GEN**);	Bit width of the address pointer of the FIFO that stores the local differences vector during the compression of a line of pixels in all bands.
HEIGHT_TREE	log2(ceil(**Cz_GEN**))	Height of the multipliers and accumulators tree for the dot product.
CYCLES_PRED	2	Number of clock cycles that takes to obtain a predicted value from the dot product result.
W_ADDR_DOT_TO_UPDATE_BIL	2	Bit width of the address pointer of the FIFO that stores the local differences and weight vectors while the dot product operation is performed; to be then used by the weight update module.
W_ADDR_WEIGHT_UPDATE_BIL	log2(**Nz_GEN**)	Bit width of the address pointer of the FIFO that stores the weight vectors during the compression of a sample in all bands.

TABLE 4-43: CCSDS123 IP – WIDTH AND DEPTH OF THE FIFOS IN BIL-MEM (PREDICTOR)

FIFO	**Width (no EDAC)***	**Depth**	**Notes**
fifo_0_curr	**D_GEN**	2**(W_ADDR_CURR)	
fifo_top_right_to_ahb	32	2**(log2(NE_AHB_FIFO_BIL))	
fifo_top_right_from_ahb	32	2**(log2(NE_AHB_FIFO_BIL))	
record_fifo_2. fifo_0_2d_ld	W_LD	2**(W_ADDR_RECORD_BIL)***4** when PREDICTION_GEN = 0 2**(W_ADDR_RECORD_BIL) when PREDICTION_GEN = 0	2-dimensional FIFO.
record_fifo_2. fifo_1_opcode	W_OPC_PREDICT	2**(W_ADDR_RECORD_BIL)	
record_fifo_2. fifo_2_s_predict	**D_GEN**+1	2**(W_ADDR_RECORD_BIL)	
record_fifo_2. fifo_3_ls_predict	W_LS	2**(W_ADDR_RECORD_BIL)	
record_fifo_2. fifo_4_z_predict	W_ADDR_IN_IMAGE	2**(W_ADDR_RECORD_BIL)	
record_fifo_2. fifo_5_t_predict	2*W_ADDR_IN_IMAGE	2**(W_ADDR_RECORD_BIL)	
ld_central_vector_fifo_3	W_LD	2**(W_ADDR_CENTRAL_BIL)***P_MAX**	Only generated if P_MAX > 0. 2-dimensional FIFO, instantiates P_MAX FIFOs.

fifo_4_ld_store_to_update	W_LD	2**(W_ADDR_DOT_TO_UPDATE_BIL)***Cz_GEN**	Only generated if Cz_GEN > 0. 2-dimensional FIFO, instantiates Cz_GEN FIFOs.
fifo_5_wei_storage_from_dot_to_update	W_WEI	2**(W_ADDR_DOT_TO_UPDATE_BIL)***Cz_GEN**	Only generated if Cz_GEN > 0. 2-dimensional FIFO, instantiates Cz_GEN FIFOs.
fifo_6_wei_update_storage	W_WEI	2**(W_ADDR_WEIGHT_UPDATE_BIL)***Cz_GEN**	Only generated if Cz_GEN > 0. 2-dimensional FIFO, instantiates Cz_GEN FIFOs.

* If EDAC is enabled, the width is calculated as: aling_byte (Width) + check_bits.

4.6.5.1.2 Sample-adaptive

TABLE 4-44: CCSDS123 IP – CONSTANTS THAT AFFECT THE AMOUNT OF INTERNAL STORAGE IN BIL-MEM (SAMPLE-ADAPTIVE)

Constant name	Value	Description
NE_CURR_SAMPLE	16	Number of samples stored in the input FIFO.
W_ADDR_CURR_SAMPLE	log2(NE_CURR_SAMPLE)	Bit width of the address pointer of the input FIFO.
W_ACC	32	Bit width of the accumulator values.
NE_FIFO	**Nz_GEN**	Number of elements in the FIFO that stores the accumulator values.
W_FIFO	log2(**Nz_GEN**)	Bit width of the address pointer of the FIFO that stores the accumulator values.

TABLE 4-45: CCSDS123 IP – WIDTH AND DEPTH OF THE FIFOS IN BIL-MEM (SAMPLE-ADAPTIVE)

FIFO	Width (no EDAC)*	Depth	Notes
fifo_curr	**D_GEN**	2**(W_ADDR_CURR_SAMPLE)	
fifoacc	W_ACC	2**(W_FIFO)	

4.6.5.2 External storage

As happens with the BIP-MEM architecture, the number of elements stored in the external memory for BIL-MEM depends on the configured Nx and Nz values. Each element is stored using 4 bytes.

TABLE 4-46: CCSDS123 IP – WIDTH AND DEPTH OF THE FIFOS IN BIL-MEM (SAMPLE-ADAPTIVE)

Number of elements stored in the external memory (bytes)	**Nx*Nz*4**

4.6.5.3 Complexity

TABLE 4-47: CCSDS123 IP – CONSTANTS THAT AFFECT THE COMPUTATIONAL COMPLEXITY IN BIL-MEM

Operation	Related constant	Value
Number of parallel multipliers in the dot product operation.	Cz_GEN	**P_MAX** + 3 when PREDICTION_GEN = 0 **P_MAX** when PREDICTION_GEN = 1
Number of instances of weight update modules instantiated.	Cz_GEN	**P_MAX** + 3 when PREDICTION_GEN = 0 **P_MAX** when PREDICTION_GEN = 1
Bit width of the input of the multipliers.	W_LD	**D_GEN** + 4
Bit width of the input of the multipliers.	W_WEI	**OMEGA_GEN** + 3
Bit width of the result of the dot product operation.	W_DZ	W_LD + W_WEI + maximum(1, log2(**Cz_GEN**))
Number of parallel adders in the dot product tree.	N_ADDERS_POW2	2**log2(ceil(**Cz_GEN,** 2))

4.6.6 Summary of differences between architectures

Table 3-1 summarizes the main differences between the BIP, BIP-MEM, BSQ, BIL and BIL-MEM architectures, in terms of memory usage and resources.

TABLE 4-48: CCSDS123 IP – SUMMARY OF DIFFERENCES BETWEEN THE BIP, BIP-MEM, BSQ, BIL AND BIL-MEM ARCHITECTURES

ARCHITECTURE	BIP	BIP-MEM	BSQ	BIL	BIL-MEM
Depth of FIFO to store left neighbouring sample	2**log2(**Nz_GEN**)	2**log2(**Nz_GEN**)	0*	0	0
Depth of FIFO to store top neighbouring sample	2**log2(**Nz_GEN**)	2**log2(**Nz_GEN**)	0	0	0
Depth of FIFO to store top left neighbouring sample	2**log2(**Nz_GEN**)	2**log2(**Nz_GEN**)	0	0	0
Depth of FIFO store top right neighbouring sample	2**log2(**Nz_GEN *Nx_GEN**)	0	2**log2(**Nx_GEN)**	2**log2(**Nz_GEN*Nx_GEN**)	0
Depth of the FIFO to store the local differences vector.	0	0	0	2**log2(**Nx_GEN**)	2**log2(**Nx_GEN**)
AHB master interface to access external memory	NO	YES	YES	NO	YES
Depth of asynchronous AHB output FIFO	0	16	32	0	16
Depth of asynchronous AHB input FIFO	0	16	32	0	16
Depth of record FIFO to store intermediate opcode, z, sample and local differences.	0	0	160/70**	160/70**	160/70**
Depth of FIFO to store the local differences vector.	0	0	2**(log2(**P_MAX**))	2**(log2(**Nx_GEN**))***P_MAX**	2**(log2(**Nx_GEN**))***P_MAX**
Depth of FIFO to store the local differences vector while the dot product is being computed.	2**(log2(ceil(**Cz_GEN**, 2)+3)) ***Cz_GEN**	2**(log2(ceil(**Cz_GEN**,2)+3)) ***Cz_GEN**	2**(log2(ceil(**Cz_GEN**,2)+3))	4***Cz_GEN**	4***Cz_GEN**

Depth of FIFO to store the weight differences vector while the dot product is being computed.	2**(log2(ceil(Cz_GEN,2)+3))*Cz_GEN	2**(log2(ceil(Cz_GEN,2)+3))*Cz_GEN	2**(log2(ceil(Cz_GEN)+3))	4*Cz_GEN	4*Cz_GEN
Depth of FIFO to store the updated weight values.	2**(log2(Nz_GEN))*Cz_GEN	2**(log2(Nz_GEN))*Cz_GEN	2**(log2(P_MAX))	2**(log2(Nz_GEN))*Cz_GEN	2**(log2(Nz_GEN))*Cz_GEN
Depth of FIFO to store accumulator values in the sample-adaptive encoder.	2**(log2(Nz_GEN))	2**(log2(Nz_GEN))	0	2**(log2(Nz_GEN))	2**(log2(Nz_GEN))
Number of parallel multipliers in the dot product operation.	P_MAX + 3/ P_MAX**	P_MAX + 3/ P_MAX**	4/1**	P_MAX + 3/ P_MAX**	P_MAX + 3/ P_MAX**
Number of parallel adders in the dot product.	2**log2(ceil(Cz_GEN, 2))	2**log2(ceil(Cz_GEN, 2))	4/1**	2**log2(ceil(Cz_GEN, 2))	2**log2(ceil(Cz_GEN, 2))

** 0 means the element is not generated in the architecture.*
***When PREDICTION_GEN = 1*

4.7 CCSDS123 IP Implementation performance

This section includes implementation results of the CSCSDS123 IP core in different technologies with different sets of generic parameters. A set of baseline configuration values is common to all the preliminary mapping cases. The parameter EN_RUNCFG and those related to the image configuration are varied in such a way that situations are created that are representative of different acquisition scenarios (multispectral, hyperspectral and ultraspectral), as shown in Table 4-49. The configurations for mapping are selected based in the acquisition scenarios in the four possible CCSDS 123 predictor architectures (BIP; BIP-MEM; BSQ; BIL and BIL-MEM), generating the configurations in Table 4-51 and Table 4-52. Sets of parameters with the prefix –e are defined to verify the BIL-MEM architecture (PREDICTION_TYPE=4).

All these results have been obtained using Synopsys Synplify Premier N-2018.03, except for NanoXplore technology where most recent version of NanoXmap tools was used (2.9.2). Grey cells mean that not enough memory resources are available in that specific technology for those set of parameters.

TABLE 4-49: CCSDS123 IP – ACQUISITION SCENARIOS FOR MAPPING

IMAGE	EN_RUNCFG	Nx_GEN	Ny_GEN	Nz_GEN	D_GEN
MULTISPECTRAL	0	1024	1024	6	8
HYPERSPECTRAL	0	512	680	224	16
ULTRASPECTRAL	0	90	135	1501	14
RUNTIME CONFIG	1	512 (512)*	680 (1024)*	224 (256)*	16

** The depth of all FIFOs in the design is constrained to a power of two.*

The amount of external memory resources used depends on the selected predictor architecture (BIP, BIP-MEM, BSQ, BIL, BIL-MEM) and the run-time configuration selection of the image size. It can be calculated as shown in Table 4-50.

TABLE 4-50: CCSDS123 IP - IMPLEMENTATION PERFORMANCE. EXTERNAL MEMORY RESOURCES

	PREDICTION_TYPE = 0 (BIP)	PREDICTION_TYPE = 1 (BIP-MEM)	PREDICTION_TYPE = 2 (BSQ)	PREDICTION_TYPE = 3 (BIL)	PREDICTION_TYPE = 4 (BIL-MEM)

External memory (bytes)	0	Nx*Nz*4	Nx*Ny*P*4	0	Nx*Nz*4

TABLE 4-51: CCSDS123 IP - SET OF PARAMETERS USED TO STUDY THE IMPLEMENTATION PERFORMANCE (I)

Generic	Set1	Set2	Set3	Set4	Set4-e	Set5	Set6	Set7	Set8	Set8-e
EN_RUNCFG	0	0	0	0	0	0	0	0	0	0
EDAC	0	0	0	0	0	0	0	0	0	0
RESET_TYPE	1	1	1	1	1	1	1	1	1	1
ENCODING_TYPE	1	1	1	1	1	1	1	1	1	1
PREDICTION_TYPE	0	1	2	3	4	0	1	2	3	4
Ny_GEN	1024	1024	1024	1024	1024	680	680	680	680	680
Nx_GEN	1024	1024	1024	1024	1024	512	512	512	512	512
Nz_GEN	6	6	6	6	6	224	224	224	224	224
D_GEN	8	8	8	8	8	16	16	16	16	16
IS_SIGNED_GEN	0	0	0	0	0	0	0	0	0	0
ENDIANESS_GEN	0	0	0	0	0	0	0	0	0	0
DISABLE_HEADER_GEN	0	0	0	0	0	0	0	0	0	0
W_BUFFER_GEN	32	32	32	32	32	32	32	32	32	32
P_MAX	3	3	3	3	3	3	3	3	3	3
PREDICTION_GEN	0	0	0	0	0	0	0	0	0	0
LOCAL_SUM_GEN	0	0	0	0	0	0	0	0	0	0
OMEGA_GEN	13	13	13	13	13	13	13	13	13	13
R_GEN	32	32	32	32	32	32	32	32	32	32
VMAX_GEN	3	3	3	3	3	3	3	3	3	3
VMIN_GEN	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
T_INC_GEN	6	6	6	6	6	6	6	6	6	6
WEIGHT_INIT_GEN	0	0	0	0	0	0	0	0	0	0
Q_GEN	1	1	1	1	1	1	1	1	1	1
INIT_COUNT_E_GEN	1	1	1	1	1	1	1	1	1	1
ACC_INIT_TYPE_GEN	0	0	0	0	0	0	0	0	0	0
ACC_INIT_CONST_GEN	5	5	5	5	5	5	5	5	5	5
RESC_COUNT_SIZE_GEN	6	6	6	6	6	6	6	6	6	6
U_MAX_GEN	16	16	16	16	16	16	16	16	16	16
HMAXBURST	1	1	1	1	0	1	1	1	1	4

TABLE 4-52: CCSDS123 IP - SET OF PARAMETERS USED TO STUDY THE IMPLEMENTATION PERFORMANCE (II)

Generic	Set9	Set10	Set11	Set12	Set12-e	Set13	Set14	Set15	Set16	Set16-e
EN_RUNCFG	0	0	0	0	0	1	1	1	1	1
EDAC	0	0	0	0	0	0	0	0	0	0
RESET_TYPE	1	1	1	1	1	1	1	1	1	1
ENCODING_TYPE	1	1	1	1	1	1	1	1	1	1
PREDICTION_TYPE	0	1	2	3	4	0	1	2	3	4
AHBMEM	1	1	1	1	1	1	1	1	1	1
Ny_GEN	90	90	90	90	90	680	680	680	680	680
Nx_GEN	135	135	135	135	135	512	512	512	512	512
Nz_GEN	1501	1501	1501	1501	1501	224	224	224	224	224
D_GEN	14	14	14	14	14	16	16	16	16	16
IS_SIGNED_GEN	0	0	0	0	0	0	0	0	0	0
ENDIANESS_GEN	0	0	0	0	0	0	0	0	0	0
DISABLE_HEADER_GEN	0	0	0	0	0	0	0	0	0	0
W_BUFFER_GEN	32	32	32	32	32	32	32	32	32	32
P_MAX	3	3	3	3	3	3	3	3	3	3
PREDICTION_GEN	0	0	0	0	0	0	0	0	0	0
LOCAL_SUM_GEN	0	0	0	0	0	0	0	0	0	0
OMEGA_GEN	13	13	13	13	13	13	13	13	13	13
R_GEN	32	32	32	32	32	32	32	32	32	32
VMAX_GEN	3	3	3	3	3	3	3	3	3	3
VMIN_GEN	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
T_INC_GEN	6	6	6	6	6	6	6	6	6	6
WEIGHT_INIT_GEN	0	0	0	0	0	0	0	0	0	0
Q_GEN	1	1	1	1	1	1	1	1	1	1
INIT_COUNT_E_GEN	1	1	1	1	1	1	1	1	1	1
ACC_INIT_TYPE_GEN	0	0	0	0	0	0	0	0	0	0
ACC_INIT_CONST_GEN	5	5	5	5	5	5	5	5	5	5
RESC_COUNT_SIZE_GEN	6	6	6	6	6	6	6	6	6	6
U_MAX_GEN	16	16	16	16	16	16	16	16	16	16
HMAXBURST	1	1	1	1	8	1	1	1	1	16

4.7.1 Implementation performance on one-time programmable FPGAs

Implementation results of the CSCSDS123 IP core in one-time programmable RTAX4000S are shown in Table 4-53.

TABLE 4-53: CCSDS123 IP - IMPLEMENTATION PERFORMANCE ON RTAX4000S.

Parameters	**Total Resources**	**Set 1**	**Set 2**	**Set 3**	**Set 4**	**Set 4-e**
I/O	840	190	204	208	190	204
Combinatorial	40320	11738	13431	13316	12480	14498
Sequential	20160	3039	4285	4875	3336	4636
Core RAM Blocks	120	44	28	24	68	52
Maximum Frequency (Clk_AHB) (MHz)		---	72.9	56.8	---	54.6
Maximum Frequency (Clk_S) (MHz)		46.4	42.0	49.2	41.7	41.9
Parameters	**Total Resources**	**Set 5**	**Set 6**	**Set 7**	**Set 8**	**Set 8-e**
I/O	840		220	224		220
Combinatorial	40320		18055	17286		19424
Sequential	20160		5437	6365		5805
Core RAM Blocks	120		29	24		50
Maximum Frequency (Clk_AHB) (MHz)			77.2	53.9		54.2
Maximum Frequency (Clk_S) (MHz)			46.9	49.3		46.1
Parameters	**Total Resources**	**Set 9**	**Set 10**	**Set 11**	**Set 12**	**Set 12-e**
I/O	840		216	220		216

Combinatorial	40320		17348	16519		18370
Sequential	20160		5276	5982		5591
Core RAM Blocks	120		103	21		94
Maximum Frequency (Clk_AHB) (MHz)			84.1	60.8		49.3
Maximum Frequency (Clk_S) (MHz)			43.8	46.7		40.0
Parameters	**Total Resources**	**Set 13**	**Set 14**	**Set 15**	**Set 16**	**Set 16-e**
I/O	840		272	276		272
Combinatorial	40320		20725	19647		22155
Sequential	20160		6196	6982		6471
Core RAM Blocks	120		30	25		51
Maximum Frequency (Clk_AHB) (MHz)			72.8	56.9		57.7
Maximum Frequency (Clk_S) (MHz)			42.3	42.2		42.3

4.7.2 Implementation performance on programmable FPGAs

Implementation results of the CSCSDS123 IP core in Virtex5FX 130 and Virtex5QR are shown in Table 4-54 and Table 4-55 respectively.

TABLE 4-54: CCSDS123 IP - IMPLEMENTATION PERFORMANCE ON VIRTEX5 (XC5VFX130T).

Parameters	**Total Resources**	**Set 1**	**Set 2**	**Set 3**	**Set 4**	**Set 4-e**
I/O	840	317	317	317	317	317
BUFGs	32	1	2	2	1	2

Block RAMs	298	2	0	1	5	3
DSP48	320	8	7	5	7	7
Registers	81920	2309	3116	2900	2703	3497
LUTs	81920	3651	4269	4634	4218	5185
Maximum Frequency (Clk_AHB) (MHz)		---	156.1	128.3	---	119.2
Maximum Frequency (Clk_S) (MHz)		111.9	136.1	106.0	126.3	113.7
Parameters	**Total Resources**	**Set 5**	**Set 6**	**Set 7**	**Set 8**	**Set 8-e**
I/O	840	325	325	325	325	325
BUFGs	32	1	2	2	1	2
Block RAMs	298	74	10	1	74	10
DSP48	320	10	10	5	10	10
Registers	81920	2902	3326	3277	3049	3902
LUTs	81920	4707	5332	5778	5248	6257
Maximum Frequency (Clk_AHB) (MHz)		---	161.6	129.2	---	125.0
Maximum Frequency (Clk_S) (MHz)		113.7	113.2	112.9	113.7	123.2
Parameters	**Total Resources**	**Set 9**	**Set 10**	**Set 11**	**Set 12**	**Set 12-e**
I/O	840	323	323	323	323	323
BUFGs	32	1	2	2	1	2
Block RAMs	298	123	11	1	123	11
DSP48	320	8	10	6	10	10

Registers	81920	2781	3256	3238	2918	3783
LUTs	81920	4600	4967	5409	4925	5897
Maximum Frequency (Clk_AHB) (MHz)		---	164.8	139.2	---	119.5
Maximum Frequency (Clk_S) (MHz)		110.7	112.7	110.9	114.5	112.7
Parameters	**Total Resources**	**Set 13**	**Set 14**	**Set 15**	**Set 16**	**Set 16-e**
I/O	840	325	325	325	325	325
BUFGs	32	2	2	2	2	2
Block RAMs	298	74	10	1	74	10
DSP48	320	10	14	9	12	15
Registers	81920	3695	4223	4145	3843	4838
LUTs	81920	5918	6325	6768	6350	7330
Maximum Frequency (Clk_AHB) (MHz)		231.4	156.7	128.7	209.0	118.2
Maximum Frequency (Clk_S) (MHz)		112.4	111.6	106.0	104.4	104.7

TABLE 4-55: CCSDS123 IP - IMPLEMENTATION PERFORMANCE ON VIRTEX5QR (XQR5VFX130).

Parameters	**Total Resources**	**Set 1**	**Set 2**	**Set 3**	**Set 4**	**Set 4-e**
I/O	836	317	317	317	317	317
BUFGs	32	1	2	2	1	2
Block RAMs	298	2	0	1	5	3
DSP48	320	8	7	5	7	7

Registers	81920	2313	3009	2900	2703	3493
LUTs	81920	3655	4284	4656	4214	5169
Maximum Frequency (Clk_AHB) (MHz)		---	182.3	127.3	---	122.7
Maximum Frequency (Clk_S) (MHz)		109.7	143.0	106.2	126.3	113.7
Parameters	**Total Resources**	**Set 5**	**Set 6**	**Set 7**	**Set 8**	**Set 8-e**
I/O	836	325	325	325	325	325
BUFGs	32	1	2	2	1	2
Block RAMs	298	74	10	1	74	10
DSP48	320	10	10	5	10	15
Registers	81920	2902	3326	3277	3049	4843
LUTs	81920	4707	5351	5786	5248	7083
Maximum Frequency (Clk_AHB) (MHz)		---	164.4	131.0	---	111.2
Maximum Frequency (Clk_S) (MHz)		113.7	113.2	112.9	113.7	105.7
Parameters	**Total Resources**	**Set 9**	**Set 10**	**Set 11**	**Set 12**	**Set 12-e**
I/O	836	323	323	323	323	323
BUFGs	32	1	2	2	1	2
Block RAMs	298	123	11	1	123	11
DSP48	320	8	10	6	10	10
Registers	81920	2781	3256	3238	2918	3778
LUTs	81920	4600	4972	5385	4925	5858

Maximum Frequency (Clk_AHB) (MHz)		---	158.7	125.8	---	115.0
Maximum Frequency (Clk_S) (MHz)		110.7	112.7	110.9	114.5	112.7
Parameters	**Total Resources**	**Set 13**	**Set 14**	**Set 15**	**Set 16**	**Set 16-e**
I/O	836	325	325	325	325	325
BUFGs	32	2	2	2	2	2
Block RAMs	298	74	10	1	74	10
DSP48	320	10	14	9	12	15
Registers	81920	3694	4223	4093	3845	4834
LUTs	81920	5926	6332	6704	6357	7311
Maximum Frequency (Clk_AHB) (MHz)		231.4	162.0	130.9	209.0	120.5
Maximum Frequency (Clk_S) (MHz)		113.0	111.4	106.0	105.6	105.4

Synthesis results for Microsemi devices have been also considered for ProASIC3E and RTG4 devices specifically, and summarized in Table 4-56 and Table 4-57.

TABLE 4-56: CCSDS123 IP - IMPLEMENTATION PERFORMANCE ON PROASIC3E (A3PE3000).

Parameters	**Total Resources**	**Set 1**	**Set 2**	**Set 3**	**Set 4**	**Set 4-e**
I/O Cells	620	190	204	208	190	204
Core Cells	75264	22337	24957	23610	24473	27740
Block RAMs	112	47	33	29	72	58

Maximum Frequency (Clk_AHB) (MHz)		---	42.8	36.5	---	37.7
Maximum Frequency (Clk_S) (MHz)		28.1	28.4	29.1	26.0	27.2
Parameters	**Total Resources**	**Set 5**	**Set 6**	**Set 7**	**Set 8**	**Set 8-e**
I/O Cells	620		220	224		220
Core Cells	75264		33495	30220		36210
Block RAMs	112		39	40		66
Maximum Frequency (Clk_AHB) (MHz)			40.2	36.8		40.1
Maximum Frequency (Clk_S) (MHz)			26.4	25.6		24.7
Parameters	**Total Resources**	**Set 9**	**Set 10**	**Set 11**	**Set 12**	**Set 12-e**
I/O Cells	620		216	220		216
Core Cells	75264		33572	28229		35644
Block RAMs	112		107	26		99
Maximum Frequency (Clk_AHB) (MHz)			37.4	35.6		38.6
Maximum Frequency (Clk_S) (MHz)			26.9	27.5		25.2
Parameters	**Total Resources**	**Set 13**	**Set 14**	**Set 15**	**Set 16**	**Set 16-e**
I/O Cells	620		272	276		272
Core Cells	75264		40151	36571		42328
Block RAMs	112		41	42		68
Maximum Frequency (Clk_AHB) (MHz)			40.6	37.7		36.2
Maximum Frequency (Clk_S) (MHz)			25.2	25.7		22.0

TABLE 4-57: CCSDS123 IP - IMPLEMENTATION PERFORMANCE ON RTG4 (RTG4 150).

Parameters	**Total Resources**	**Set 1**	**Set 2**	**Set 3**	**Set 4**	**Set 4-e**

IO Cells	720	190	212	212	190	212
Carry Cells	151824	2017	2288	2489	2179	2420
Sequential Cells	151824	2485	3081	2800	2755	3413
Block RAMs (1Kx18 + 64x18)	209 + 210	4+31	0+33	1+25	10+38	6+40
DSP Blocks	462	7	7	7	7	7
LUTs	151824	5317	6198	6276	5639	6983
Maximum Frequency (Clk_AHB) (MHz)		---	101.5	81.7	---	72.0
Maximum Frequency (Clk_S) (MHz)		72.8	83.6	80.4	74.2	69.6
Parameters	**Total Resources**	**Set 5**	**Set 6**	**Set 7**	**Set 8**	**Set 8-e**
IO Cells	720	198	220	236	198	220
Carry Cells	151824	3604	3299	3044	3764	3376
Sequential Cells	151824	3076	3713	3181	3438	4128
Block RAMs (1Kx18 + 64x18)	209 + 210	129+62	1+64	1+36	135+65	7+67
DSP Blocks	462	13	13	11	13	13
LUTs	151824	7935	7985	7382	8175	8743
Maximum Frequency (Clk_AHB) (MHz)		---	95.9	85.2	---	76.3
Maximum Frequency (Clk_S) (MHz)		64.6	79.9	80.8	56.6	76.2
Parameters	**Total Resources**	**Set 9**	**Set 10**	**Set 11**	**Set 12**	**Set 12-e**
IO Cells	720	196	218	234	196	218

Carry Cells	151824	5092	4162	2954	4125	3290
Sequential Cells	151824	2994	3615	3079	3172	3844
Block RAMs (1Kx18 + 64x18)	209 + 210	266+212	10+214	0+30	275+30	19+32
DSP Blocks	462	7	7	6	7	7
LUTs	151824	9661	8878	7182	8774	8278
Maximum Frequency (Clk_AHB) (MHz)		---	84.8	79.5	---	82.5
Maximum Frequency (Clk_S) (MHz)		69.2	69.4	79.5	69.3	76.6

Parameters	**Total Resources**	**Set 13**	**Set 14**	**Set 15**	**Set 16**	**Set 16-e**
IO Cells	720	252	272	288	252	272
Carry Cells	151824	3960	3665	3447	4078	3756
Sequential Cells	151824	3641	4397	3789	3977	4787
Block RAMs (1Kx18 + 64x18)	209 + 210	129+64	1+66	1+38	135+67	7+69
DSP Blocks	462	16	16	14	16	16
LUTs	151824	10027	10268	9648	10293	10877
Maximum Frequency (Clk_AHB) (MHz)		132.8	96.3	85.3	132.8	76.3
Maximum Frequency (Clk_S) (MHz)		66.1	79.6	74.2	73.9	74.2

Finally, implementation results for NanoXplore technologies are obtained for NG-MEDIUM NX1H35S and NG-LARGE NX1H140TSP and shown in Table 4-58 and Table 4-59, respectively.

TABLE 4-58. CCSDS123 IP - IMPLEMENTATION PERFORMANCE ON NG-MEDIUM NX1H35S.

Parameters	Total Resources	Set 1	Set 2	Set 3	Set 4	Set 4-e
Carry Cells	8064	2360	2778	2936	2743	3114
Registers	32256	2687	3417	3041	3083	3816
Block RAMs (48Kb)	56	33	34	27	43	44
DSP Blocks	112	6	6	4	6	6
LUTs	32256	3526	4922	4300	3806	5094
Maximum Frequency (Clk_AHB) (MHz)		---	31.9	39.5	---	31.5
Maximum Frequency (Clk_S) (MHz)		41.1	37.2	40.6	39.1	39.0
Parameters	**Total Resources**	**Set 5**	**Set 6**	**Set 7**	**Set 8**	**Set 8-e**
Carry Cells	8064	3353	3834	3646	3678	4128
Registers	32256	3148	3895	3308	3542	4291
Block RAMs (48Kb)	56	95	34	27	105	44
DSP Blocks	112	6	6	4	6	6
LUTs	32256	4263	5636	4896	4663	5866
Maximum Frequency (Clk_AHB) (MHz)			35.1	50.6		29.2
Maximum Frequency (Clk_S) (MHz)			40.1	38.0		33.3
Parameters	**Total Resources**	**Set 9**	**Set 10**	**Set 11**	**Set 12**	**Set 12-e**
Carry Cells	8064	3333	3857	3562	3634	4126
Registers	32256	3170	3906	3240	3507	4246

Block RAMs (48Kb)	56	143	34	27	153	44
DSP Blocks	112	6	6	4	6	6
LUTs	32256	4118	5564	4815	4381	5735
Maximum Frequency (Clk_AHB) (MHz)			30.8	42.0		25.2
Maximum Frequency (Clk_S) (MHz)			37.3	40.6		31.2
Parameters	**Total Resources**	**Set 13**	**Set 14**	**Set 15**	**Set 16**	**Set 16-e**
Carry Cells	8064	3855	4440	4249	4198	4729
Registers	32256	3665	4518	3897	4023	4888
Block RAMs (48Kb)	56	97	36	29	107	46
DSP Blocks	112	8	8	6	8	8
LUTs	32256	6004	7251	6544	6221	7517
Maximum Frequency (Clk_AHB) (MHz)			35.1	36.5		28.2
Maximum Frequency (Clk_S) (MHz)			31.7	32.8		27.5

TABLE 4-59. CCSDS123 IP- IMPLEMENTATION PERFORMANCE ON NG-LARGE NX1H140TSP.

Parameters	Total Resources	Set 1	Set 2	Set 3	Set 4	Set 4-e
Carry Cells	32256	2360	2778	2936	2743	3114
Registers	129024	2687	3417	3041	3083	3816
Block RAMs (48Kb)	192	33	34	27	43	44
DSP Blocks	384	6	6	4	6	6

LUTs	129024	3526	4922	4300	3806	5094
Maximum Frequency (Clk_AHB) (MHz)		---	33.0	43.9	---	34.0
Maximum Frequency (Clk_S) (MHz)		44.3	34.8	41.4	42.0	35.7
Parameters	**Total Resources**	**Set 5**	**Set 6**	**Set 7**	**Set 8**	**Set 8-e**
Carry Cells	32256	3353	3834	3646	3678	4128
Registers	129024	3148	3895	3308	3542	4291
Block RAMs (48Kb)	192	95	34	27	105	44
DSP Blocks	384	6	6	4	6	6
LUTs	129024	4263	5636	4896	4663	5866
Maximum Frequency (Clk_AHB) (MHz)		---	35.8	40.2	---	26.4
Maximum Frequency (Clk_S) (MHz)		41.9	35.9	43.8	32.0	34.0
Parameters	**Total Resources**	**Set 9**	**Set 10**	**Set 11**	**Set 12**	**Set 12-e**
Carry Cells	32256	3333	3857	3562	3634	4126
Registers	129024	3170	3906	3240	3507	4246
Block RAMs (48Kb)	192	143	34	27	153	44
DSP Blocks	384	6	6	4	6	6
LUTs	129024	4118	5564	4815	4381	5735
Maximum Frequency (Clk_AHB) (MHz)		---	28.9	42.2	---	26.4
Maximum Frequency (Clk_S) (MHz)		34.8	36.0	40.7	31.9	37.9

Parameters	Total Resources	Set 13	Set 14	Set 15	Set 16	Set 16-e
Carry Cells	32256	3855	4440	4249	4198	4729
Registers	129024	3665	4518	3897	4023	4888
Block RAMs (48Kb)	192	97	36	29	107	46
DSP Blocks	384	8	8	6	8	8
LUTs	129024	6004	7251	6544	6221	7517
Maximum Frequency (Clk_AHB) (MHz)		108.5	29.7	38.3	80.1	29.7
Maximum Frequency (Clk_S) (MHz)		35.0	32.9	37.9	28.7	30.8

4.7.3 Implementation performance on ASICs

Table 4-60 shows implementation results for the DARE 180 Standard Cell ASIC Library. These results were obtained for the original SHyLoC implementation; this is, they correspond to the CCSDS123 IP without the updates proposed in this extension. The conditions where these results were obtained are:

- Synopsys Design Compiler M-2016.12-SP4 for 64-bit Linux, with conservative optimizations;
- Clk_AHB is constrained to 10 ns (100 MHz);
- Clk_S is constrained to 2.5 ns (400 MHz);
- Memories smaller than 64 words are mapped to Flip-Flops;
- Memories larger than 64 words are mapped into SRAM macros;
- No DFT nor BIST mechanisms have been synthesized;

- No EDAC circuitry has been included for the memories.

Table 4-60 shows results for synthesis with and without memories. These results are presented in order to give an idea of the complexity of the IP core, and the amount of memory required for each Set. Discrepancies in gate count between the scenarios are due to memories smaller than 64 words being mapped into FFs. Only a few SRAM macros were available for synthesis, therefore all the needed memory aspect ratios had to be built from this limited set. This means there is an area overhead on the area results below, which should be taken as an upper bound, i.e. synthesis with appropriate memory views should yield lower area.

TABLE 4-60: CCSDS123 IP - IMPLEMENTATION AND PERFORMANCE FIGURES ON DARE 180 STANDARD CELL ASIC LIBRARY

Parameters	**Set 1**		**Set 2**			**Set 3**		**Set 4**	
		Without Memories	**With Memories**	**Without Memories**	**With Memories**	**Without Memories**	**With Memories**	**Without Memories**	**With Memories**
	Gate Count (kGates)	69.14	125.55	81.95	155.59	73.50	123.47	78.64	149.09
	Area (mm^2)	2.61	13.31	3.11	5.99	2.79	7.16	2.97	20.57
Clk_S Max. Freq. (MHz)	224.22	160.77	222.22	222.22		202.02	166.11	201.61	121.65
Parameters	**Set 5**		**Set 6**			**Set 7**		**Set 8**	
		Without Memories	**With Memories**	**Without Memories**	**With Memories**	**Without Memories**	**With Memories**	**Without Memories**	**With Memories**
	Gate Count (kGates)	90.11	152.44	103.24	176.75	88.16	162.16	100.01	187.11
	Area (mm^2)	3.41	146.77	3.91	12.09	3.34	7.58	3.77	149.70
Clk_S Max. Freq. (MHz)	204.50	119.19	207.47	119.05		182.82	162.34	182.48	116.96
	Parameters	**Set 9**		**Set 10**		**Set 11**		**Set 12**	
		Without Memories	**With Memories**	**Without Memories**	**With Memories**	**Without Memories**	**With Memories**	**Without Memories**	**With Memories**
	Gate Count (kGates)	88.95	152.21	102.33	172.01	85.43	153.48	97.81	183.02

	Area (mm^2)	3.36	298.29	3.87	27.80	3.23	6.72	3.69	294.73
Clk_S Max. Freq. (MHz)	212.31	116.41	214.59	117.10		188.32	168.63	189.39	114.16
Parameters	**Set 13**		**Set 14**			**Set 15**		**Set 16**	
	Without Memories	**With Memories**	**Without Memories**	**With Memories**		**Without Memories**	**With Memories**	**Without Memories**	**With Memories**
	Gate Count (kGates)	109.13	174.64	122.90	199.69	108.78	186.14	118.15	209.06
	Area (mm^2)	4.13	147.62	4.66	12.96	4.13	8.49	4.47	150.55
Clk_S Max. Freq. (MHz)	152.67	119.19	153.61	119.33		154.08	151.98	151.98	116.55

5 SHYLOC-E

The CCSDS121 block coder and the CCSDS123 IP core have been designed in a way that their interfaces make it possible to easily combine them. In such a scenario, the CCSDS121 block coder is used as external entropy coder of the CCSDS123 IP core. The system that includes the two IP cores is named SHyLoC-e.

5.1 Connecting the two IP cores

The I/O interfaces of the IP cores can be connected as shown in Figure 4-20.

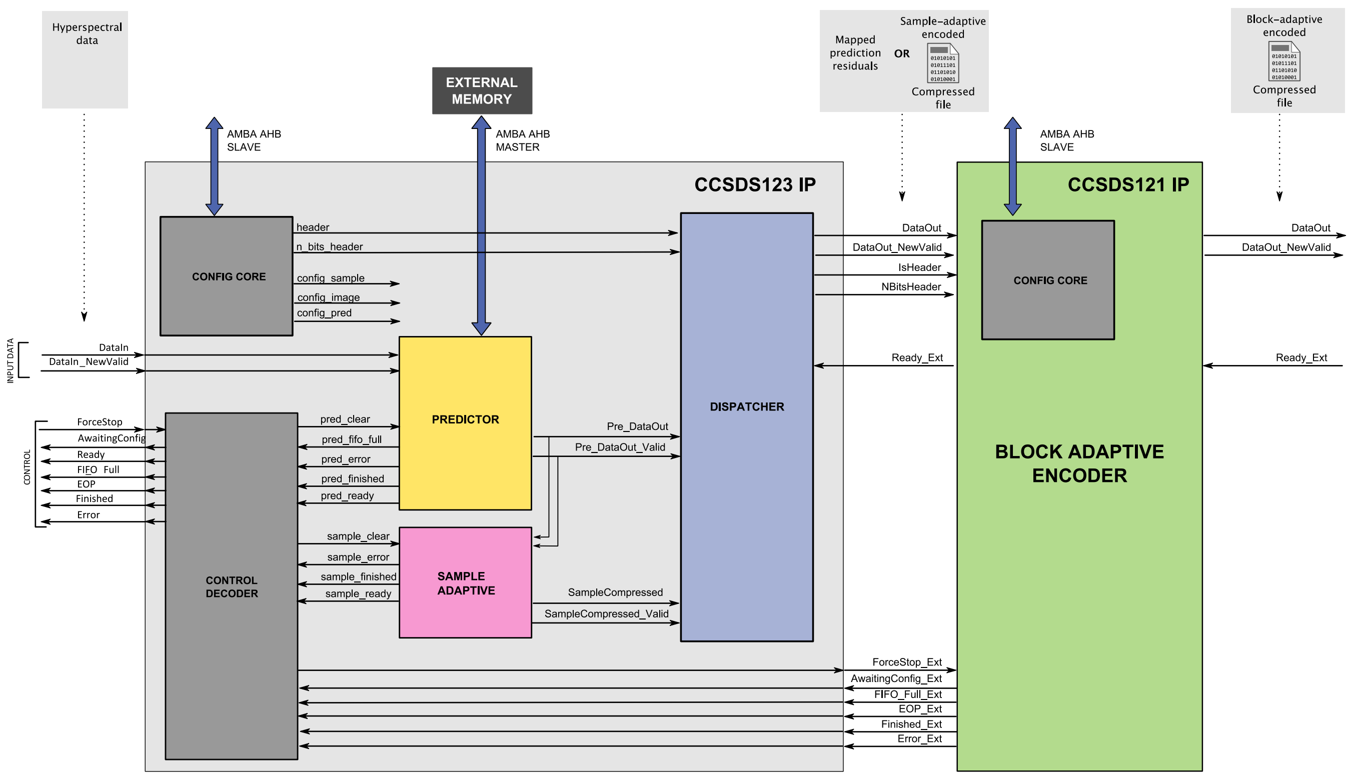


Figure 4-20. Connectivity between CCSDS123 IP and CCSDS121 IP.

5.2 CCSDS121 and CCSDS123 compatibility rules

When configuring the CCSDS123 IP core and the CCSDS121 block coder to work jointly, the compatibility rules described in
Table 4-61 must be followed.

TABLE 4-61: SHYLOC - CCSDS121 AND CCSDS123 COMPATIBILITY RULES

CCSDS123 parameter	CCSDS121 parameter	Rule

EN_RUNCFG	EN_RUNCFG	EN_RUNCFG in 121 = EN_RUNCFG in 123
RESET_TYPE	RESET_TYPE	RESET_TYPE in 121 = RESET_TYPE in 123
Nx, Ny, Nz	Nx, Ny, Nz	Configured image size (Nx, Ny, Nz) must be the same in both IP cores.
D	D	Runtime configuration of dynamic range D in 123 = D in 121.
D_GEN	D_GEN	Implementation time parameter dynamic range in the CCSDS121 block coder, D_GEN must be a multiple of 8 (byte-aligned); and greater or equal to the D_GEN parameter in the CCSDS_123 IP.
N/A	ENDIANESS	Configured ENDIANESS in the CCSDS121 block coder shall be always set to big endian (1).
N/A	IS_SIGNED	Configured IS_SIGNED in the CCSDS121 block coder shall be always set to unsigned (0).