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

IUMA/1410/AO8032

SHyLoC 2.0 IP User Manual

By

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

23 October 2020

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

DISCLAIMER

The work associated with this report has been carried out in accordance with the highest technical standards and TRPAO8032 partners have endeavoured to achieve the degree of accuracy and reliability appropriate to the work in question. However, since the partners have no control over the use to which the information contained within the report is to be put by any other party, any other such party shall be deemed to have satisfied itself as to the suitability and reliability of the information in relation to any particular use, purpose or application.

Under no circumstances will any of the partners, their servants, employees or agents accept any liability whatsoever arising out of any error or inaccuracy contained in this report (or any further consolidation, summary, publication or dissemination of the information contained within this report) and/or the connected work and disclaim all liability for any loss, damage, expenses, claims or infringement of third party rights.

DOCUMENT HISTORY

Date	Version	Author	Description	Status
13-10-2017	1.0	Lucana Santos	SHyLoC IP User Manual, public version created 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

1 TABLE OF CONTENT

Disclaimer 2
Document History 3
List of Authors 4
Figures 7
Tables 8
Glossary 10
1 Introduction 11
1.1 Document scope 11
1.2 Applicable documents 11
1.3 Reference documents 11
1.4 Document description 11
1.5 Cross-reference 11
2 Overall functionality of the CCSDS121 and CCSDS123 compression IP cores 12
3 General requirements 13
4 CCSDS121 IP core user manual 14
4.1 System overview 14
4.2 Installation 14
4.3 CCSDS121 IP database directory structure 14
4.4 Makefile 15
4.5 CCSDS121 IP VHDL sources and compile order 16
4.6 CCSDS121 IP VHDL operational description 17
4.6.1 Compile-time configuration 17
4.6.2 Run-time configuration 18
4.7 CCSDS121 IP Testbench 18
4.7.1 Testbench behavioural description 20
4.7.2 Testbench assertions 22
4.8 CCSDS121 IP simulation and synthesis scripts 23
4.8.1 Writing the desired parameters in the *.csv file 23
4.8.2 Simulation 25
4.8.3 Synthesis 28
4.8.4 Post-synthesis simulations 33
4.8.5 Post-PAR simulations 34

5 CCSDS123 IP core user manual 36
5.1 System overview 36
5.2 Installation 36
5.3 CCSDS123 IP database directory structure 36
5.4 Makefile 37
5.5 CSDS123 IP VHDL sources and compile order 38
5.6 CSDS123 IP VHDL operational description 40
5.6.1 Compile-time configuration 40
5.6.2 Run-time configuration 42
5.7 CCSDS123 IP Testbench 42
5.7.1 Testbench behavioural description 44
5.7.2 Testbench assertions 47
5.8 CCSDS123 IP simulation and synthesis scripts 48
5.8.1 Writing the desired parameters in the *.csv file 49
5.8.2 Simulation 54
5.8.3 Synthesis 57
5.8.4 Post-synthesis simulations 62
5.8.5 Post-PAR simulations 63
6 SHyloC (CCSDS123 IP + CCSDS121 IP) user manual 65
6.1 Connecting the two IP cores 65
6.2 SHyLoC testbench 67
6.2.1 SHyLoC testbench behavioural description 67
6.2.2 SHyLoC testbench assertions 67
6.3 SHyLoC simulation scripts 68
6.3.1 CCSDS121 and CCSDS123 compatibility rules 68

FIGURES

Figure 2-1: Designed IP cores and connectivity between them. 12

TABLES

Table 4-1: CCSDS121 IP – VHDL sources and compile order 16

Table 4-2: CCSDS121 IP – Compile-time configuration constants in the ccsds121_parameters.vhd file 17

Table 4-3: CCSDS121 IP – Testbench VHDL sources 18

Table 4-4: CCSDS121 IP – Testbench GRLIB components 19

Table 4-5: CCSDS121 IP – run-time configuration values in the ccsds121_tb_paramters.vhd file 19

Table 4-6: CCSDS121 IP Testbench – Test sequences 20

Table 4-7: CCSDS121 IP Testbench – assertions 22

Table 4-8: CCSDS121 IP Content of the *.csv configuration file. 24

Table 4-9: CCSDS121 IP - Rules to fill the *.csv file when en_runcfg = 0 25

Table 4-10: CCSDS121 IP – Arguments of the run_vhdl_tests_121.py script for simulation 26

Table 4-11: CCSDS121 IP – List of files generated by the python script 26

Table 4-12: CCSDS121 IP – List of files generated after completion of the all_tests.do file 27

Table 4-13: CCSDS121 IP – Arguments of the run_vhdl_tests_121.py script for synthesis 28

Table 4-14: CCSDS121 IP– List of files generated after completion of the all_ises.tcl, all_synplify.tcl and all_nanoxplore.py scripts. 29

Table 4-15: CCSDS121 IP – List of files generated after completion of the all_ise.tcl, all_synplify.tcl and all_nanoxplore.py file. 31

Table 4-16: CCSDS121 IP – List of files pertaining to ASIC Synthesis 32

Table 4-17: CCSDS121 IP – ASIC Standard Cell Library Setup for Logical Synthesis 32

Table 4-18: CCSDS121 IP – List of files Generated by ASIC Synthesis 32

Table 5-1: CCSDS123 IP – VHDL sources and compile order 38

Table 5-2: CCSDS123 IP – Compile-time configuration constants in the ccsds123_parameters.vhd file 40

Table 5-3: CCSDS123 IP – Testbench VHDL sources 42

Table 5-4: CCSDS123 IP – Testbench GRLIB components 42

Table 5-5: CCSDS123 IP – run-time configuration values in the ccsds123_tb_paramters.vhd file 43

Table 5-6: CCSDS123 IP Testbench – Test sequences 44

Table 5-7: CCSDS123 IP Testbench – assertions 47

Table 5-8: CCSDS123 IP Content of the *.csv configuration file. 49

Table 5-9: CCSDS121 IP - Rules to fill the *.csv file when en_runcfg = 0 53

Table 5-10: CCSDS123 IP – Arguments of the run_vhdl_tests_123.py script for simulation 54

Table 5-11: CCSDS123 IP – List of files generated by the python script 54

Table 5-12: CCSDS123 IP – List of files generated after completion of the all_tests.do file 56

Table 5-13: CCSDS123 IP – Arguments of the run_vhdl_tests_123.py script for synthesis. 57

Table 5-14: CCSDS123 IP– List of files generated after running the python script with $option = synplify, ise, brave or dc. 58

Table 5-15: CCSDS123 IP – List of files generated after completion of the all_ise.tcl, all_synplify.tcl and all_nanoxplore.py file. 59

Table 5-16: CCSDS123 IP – List of files pertaining to ASIC Synthesis 60

Table 5-17: CCSDS123 IP – ASIC Standard Cell Library Setup for Logical Synthesis 61

Table 5-18: CCSDS123 IP – List of files Generated by ASIC Synthesis 61

Table 6-1: SHyLoC – Testbench VHDL sources 67

Table 6-2: CCSDS123 IP Testbench – Test sequences 67

Table 6-3: SHyLoC Testbench – assertions 67

Table 6-4: ShyLoC - CCSDS121 and CCSDS123 compatibility rules 68

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 deliverable D12 IP User Manual 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] TRP AO8032 – Deliverable D6, Verification and Validation Plan, IUMA, May 2015

[AD-4] CCSDS 123 and 121 software implementation from ESA, http://www.esa.int/TEC/OBDP/SEM069KOXDG_0.html

1.3 Reference documents

[RD-1] Extended SHyLoC IP Datasheet, IUMA, October 2018.

1.4 Document description

This document presents the user manual of the CCSDS121 and CCSDS123 IP cores.

1.5 Cross-reference

This deliverable D4 (Extended SHyLoC IP User Manual) has been created based on the document TRP-AO8032_D12_IP_User_Manual_v2.5.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 user manual of two different IPs 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.

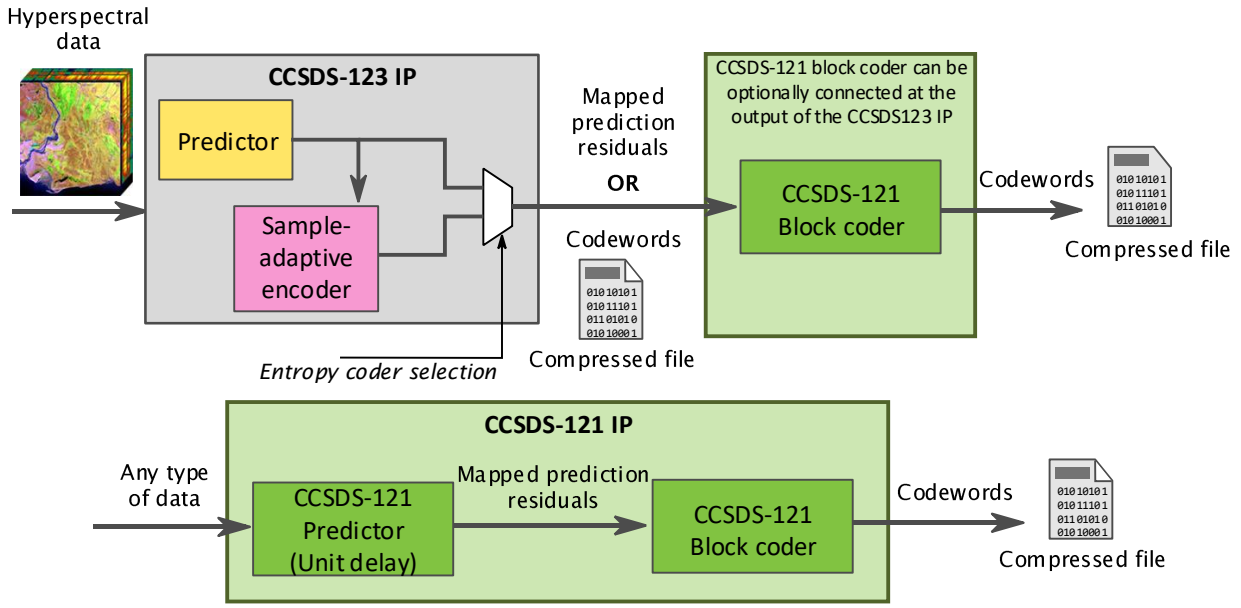

Figure 2-1: Designed IP cores and connectivity between them.

3 GENERAL REQUIREMENTS

To compile and run the provided sources, the following libraries and tools must be present:

Tool	Version	Notes
Python	2.7.10	
QuestaSim	10.4c	
Git	GitLab 11.6.3	Any git client
Synplify	2018.03	
Xilinx ISE	14.7	
NanoXmap	2.9.2	

Additionally, the following environment variables must be set:

Variable	Value	Example value
$GRLIB	Path to the folder where the grlib library is installed.	C:\Lib\grlib-gpl-1.5.0-b4164
$MODEL_TECH	Path to the folder that contains the QuestaSim.	C:\Herramientas\questasim_10.4c\win32
$PATH	Path variable including the paths that contains the ISE Project Navigator and the Synplify premier with DP.	… C:\Herramientas\Xilinx\14.7\ISE_DS\ISE\bin\nt64 C:\Herramientas\Synopsys\fpga_N-2018.03\bin …

4 CCSDS121 IP CORE USER MANUAL

4.1 System overview

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

4.2 Installation

The Git repository is located in: https://git.iuma.ulpgc.es:8300/TRP-AO8032/SHyLoC-e.git

The sources can be cloned with the command:

```
$ git clone https://git.iuma.ulpgc.es:8300/TRP-AO8032/SHyLoC-e.git
```

This will create a copy of the database in the user's local directory. The user might access the root folder of the database and run the provided makefile (see Section 4.4).

```
$ cd CCSDS121IP-VHDL
```

Throughout this document, all the paths starting with $121DB are related to the root of the database directory.

4.3 CCSDS121 IP database directory structure

```
CCSDS121IP-VHDL ($121DB)
├──images
│   ├──compressed
│   ├──raw
│   └──reference
├──modelsim
│   ├──cover
│   ├──tb_scripts
│   └──tb_stimuli
├──src
│   ├──post_syn
│   ├──shyloc_121
│   ├──shyloc_utils
│   └──tb
├──synthesis
│   ├──premap_parameters
│   └──syn_scripts
│       ├──dc
│       │   ├──base
│       │   ├──report
│       │   └──results
│       ├──ise
│       └──synplify
```

```
            └──report
└──verification_scripts
```

4.4 Makefile

The makefile can be called from the database root directory, and provides the following options:

- *make ccsds121*: generates the configuration files that conform the simulation environment and runs the simulations in QuestaSim. The set of test cases to be simulated is determined by the information contained in the comma-separated file: $121DB/verification_ scripts/testcases_121_e_all.csv. Such file comes already pre-filled with the test cases contemplated in the Verification and Validation plan [AD-3]. A pass/fail report is generated and saved in $121DB/modelsim/tb_scripts/verification_report.txt. Coverage results are additionally generated and merged after completion of all tests.

- *make synplify:* generates the configuration files that conform the synthesis environment and runs the synthesis in Synplify Premier with DP. The set of synthesis cases to be implemented is determined by the information contained in the comma-separated file: $121DB/verification_ scripts /synthesis_params_121_e.csv. Such file comes already pre-filled with synthesis cases. Synthesis results in terms of timing and area are saved in the folder $121DB/synthesis/syn_scripts/synplify/report.

- *make ise:* generates the configuration files that conform the synthesis environment and runs the synthesis in ISE Project Navigator. The set of synthesis cases to be implemented is determined by the information contained in the comma-separated file: $121DB/verification_scripts/synthesis_ params_121_e.csv. Such file comes already pre-filled with synthesis cases.

- *make brave:* generates the Python scripts that conform the synthesis environment and run the synthesis in NanoXplore NanoXmap, using the Python API named as NanoXpython. The set of synthesis cases to be implemented is determined by the information contained in the comma-separated file: $121DB/verification_scripts/synthesis_params_121_e.csv. Such file comes already pre-filled with synthesis cases. Synthesis results in terms of timing and area are saved in the folder $121DB/synthesis/syn_scripts/brave/report.

- *make dc:* generates the configuration files that conform the synthesis environment and runs the synthesis in Synopsys Design Compiler. The set of synthesis cases to be implemented is determined by the information contained in the comma-separated file: $121DB/verification_scripts/synthesis_params_121.csv. Such file comes already pre-filled with synthesis cases. Synthesis results in terms of timing and area are saved in folders contained in $121DB/synthesis/syn_scripts/dc/report, and resulting netlist and associated files are saved in folders contained in $121DB/synthesis/syn_scripts/dc/results.

IMPORTANT NOTE: Successful execution of the simulations relies on the presence of a set of raw images and compressed reference files. Such images are not included as part of the IP core's database due to their excessive volume. They can be downloaded from the NAS repository:

http://nasdsi.iuma.ulpgc.es/

in the folder:

/TRPAO8032--IUMA/SHyLoC-e/Test-Images/121

They must be placed in the available $121DB/images/raw and $121DB/images/reference folders of the IP core's database.

4.5 CCSDS121 IP VHDL sources and compile order

TABLE 4-1: CCSDS121 IP – VHDL SOURCES AND COMPILE ORDER

Folder	Compile order	Filename	Library
$121DB/src/shyloc_121	1	ccsds121_parameters.vhd	shyloc_121
$121DB/src/shyloc_utils	2	toggle_sync.vhd	shyloc_utils
	3	reset_sync.vhd	shyloc_utils
	4	amba.vhd	shyloc_utils
	5	shyloc_functions.vhd	shyloc_utils
	6	fixed_shifter.vhd	shyloc_utils
	7	barrel_shifter.vhd	shyloc_utils
	8	reg_bank_inf.vhd	shyloc_utils
	9	reg_bank_tech.vhd	shyloc_utils
	10	reg_bank.vhd	shyloc_utils
	11	fifop2_base.vhd	shyloc_utils
$121DB/src/shyloc_utils/edac-0-7-src	12	edac-decl-0-7.vhd	shyloc_utils
	13	edac-body-0-7.vhd	shyloc_utils
	14	edac-rtl.vhd	shyloc_utils
$121DB/src/shyloc_utils	15	fifop2_edac.vhd	shyloc_utils
	16	fifop2.vhd	shyloc_utils
	17	bitpackv2.vhd	shyloc_utils
$121DB/src/shyloc_121	18	ccsds121_constants.vhd	shyloc_121
	19	config121_package.vhd	shyloc_121
	20	splitter.vhd	shyloc_121
	21	optcoder.vhd	shyloc_121
	22	lkcomp.vhd	shyloc_121
	23	header121_shyloc.vhd	shyloc_121
	24	sndextension.vhd	shyloc_121
	25	packing_top.vhd	shyloc_121
	26	fscoderv2.vhd	shyloc_121
	27	lkoptions.vhd	shyloc_121
	28	ccsds121_shyloc_interface.vhd	shyloc_121
	29	ccsds121_clk_adapt.vhd	shyloc_121
	30	ccsds121_ahbs.vhd	shyloc_121

	31	ccsds121_shyloc_comp.vhd	shyloc_121
	32	ccsds121_shyloc_fsm.vhd	shyloc_121
	33	ccsds121_blockcoder_top.vhd	shyloc_121
	34	ccsds121_predictor_comp.vhd	shyloc_121
	35	ccsds121_predictor_fsm.vhd	shyloc_121
	36	ccsds121_predictor_top.vhd	shyloc_121
	37	ccsds121_shyloc_top.vhd	shyloc_121
$121DB/src/post_syn	38*	ccsds121_top_wrapper.vhd	shyloc_121

(*) This file is just a wrapper that flattens the records present in the I/O ports and instantiates the top module ccsds121_shyloc_top.vhd. It is not actually part of the design, but it remains in the database, because it aids the preparation of post-synthesis or post-PAR simulations.

4.6 CCSDS121 IP VHDL operational description

4.6.1 Compile-time configuration

The compile-time options are set by editing the file "ccsds121_parameters.vhd", and assigning the desired values to the constants included. This file can be automatically generated by editing an appropriately formatted *.csv file and running a python script as described in Section 4.8. More information about the meaning of the compile-time configuration values can be found in [RD-1].

TABLE 4-2: CCSDS121 IP – COMPILE-TIME CONFIGURATION CONSTANTS IN THE CCSDS121_PARAMETERS.VHD FILE

Constant	Allowed values	Description
EN_RUNCFG	[0,1]	(0) Disables runtime configuration. (1) Enables runtime configuration.
RESET_TYPE	[0,1]	(0) Asynchronous reset. (1) Synchronous reset.
HSINDEX_121	[0 - NAHBSLV-1]	AHB slave index.
HSCONFIGADDR_121	[0 - 16#FFF#]	ADDR field of the AHB slave. Sets the 12 most significant bits in the 32-bit AHB address.
HSADDRMASK_121	[0 - 16#FF0]	MASK field of the AHB slave.
EDAC	[0,1]	(0) Inhibits EDAC implementation. (1) EDAC is implemented. *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.*
Nx_GEN	[1-2^{16}-1]	Maximum allowed number of samples in a line.
Ny_GEN	[1-2^{16}-1]	Maximum allowed number of samples in a row.
Nz_GEN	[1-2^{16}-1]	Maximum allowed number of bands.
D_GEN	[2-32]	Maximum dynamic range of the input samples.
ENDIANESS_GEN	[0,1]	(0) Little-Endian.

		(1) Big-Endian.
IS_SIGNED_GEN*	[0,1]	(0) Unsigned samples. (1) Signed samples.
J_GEN	[8,16,32,64]	Block Size.
REF_SAMPLE_GEN	≤ 4096	Reference Sample Interval.
CODESET_GEN	[0,1]	Code Option.
W_BUFFER_GEN	[8,16,24,32,40,48,56,64]	Bit width of the output buffer.
PREPROCESSOR_GEN	[0,1,2,3]	(0) Pre-processor is not present; (1) CCSDS123 pre-processor is present; (2) CCSDS121 pre-processor (unit-delay predictor) is present ;(3) Any-other pre-processor is present.
DISABLE_HEADER_GEN	[0,1]	Selects whether to disable (1) or not (0) the header generation.

(*) IS_SIGNED_GEN configuration parameter is taken into account just when the unit-delay predictor is present (PREPROCESSOR_GEN = 2). Otherwise, it is assumed that input samples are always unsigned.

4.6.2 Run-time configuration

Runtime configuration is performed by writing in the memory-mapped registers. The allowed values depend on the selected compile-time configuration and the standard, as specified in the [RD-1].

In the provided simulation environment, the run-time configuration values are selected by setting the testbench constants provided in the "ccsds121_tb_parameters.vhd", as detailed in Section 4.7. The "ccsds121_tb_parameters.vhd" file can be automatically generated by using the provided *.csv file and the scripts described in Section 4.8.

4.7 CCSDS121 IP Testbench

The provided testbench is designed following the Verification and Validation plan [AD-3]. The testbench performs the compression of an image and checks if the results are correct by comparing with a reference image. The VHDL sources listed in Table 4-3 are used as part of the testbench. In the table, {*TestId*} is a unique test identifier that defines the folder in which the testbench configuration file (ccsds121_tb_parameters.vhd) and the compile-time configuration file (ccsds121_parameters.vhd) is stored.

TABLE 4-3: CCSDS121 IP – TESTBENCH VHDL SOURCES

Folder	Compile order	Filename	Library
$121DB/modelsim/tb_stimuli/{*TestId*}	36	ccsds121_tb_parameters.vhd	work
$121DB/tb	37	ahbtbp.vhd	work
	38	ccsds_ahbtbp.vhd	work
	39	ccsds121_tb_v3.vhd	work

$121DB/modelsim/tb_stimuli/{*11-n_Test*}	40*	ccsds121_tb_v3.vhd	work

** This special test is used to perform compression inserting input samples at a non-constant rate.*

Additionally, the following components from the GRLIB library are instantiated by the testbench.

TABLE 4-4: CCSDS121 IP – TESTBENCH GRLIB COMPONENTS

GRLIB component	Description
ahbctrl	AHB controller
ahbtbm	AHB master

The testbench will first write the configuration values in the memory-mapped registers. The "ccsds121_tb_parameters.vhd" is used to configure those values, as detailed in Table 4-5, and it can be automatically generated by using the python configuration script described in Section 4.8.

TABLE 4-5: CCSDS121 IP – RUN-TIME CONFIGURATION VALUES IN THE CCSDS121_TB_PARAMTERS.VHD FILE

Constant	Type	Allowed values	Description
Nx_tb	integer	[1 - Nx_GEN]	Number of columns .
Ny_tb	integer	[1 - Ny_GEN]	Number of lines.
Nz_tb	integer	[1 - Nz_GEN]	Number of bands.
D_tb	integer	[2 – D_GEN]	Dynamic range of the input samples.
ENDIANESS_tb	integer	[0,1]	Endianess of the input samples: (0) Little-Endian (1) Big-Endian
IS_SIGNED_tb*	integer	[0,1]	Sign of the input samples: (0) Unsigned samples (1) Signed samples
J_tb	integer	[8,16,32,64] ≤ J_GEN	Block Size.
REF_SAMPLE_tb	integer	≤ REF_SAMPLE_GEN	Reference Sample Interval.
CODESET_tb	integer	[0,1]	Code Option.
W_BUFFER_tb	integer	[8,16,24,32,40,48,56,64] ≤ W_BUFFER_GEN	Output word size.
BYPASS_tb	integer	[0,1]	(0) Compression. (1) Bypass Compression.
PREPROCESSOR_tb	integer	[0,1,2,3]	(0) Pre-processor is not present. (1) CCSDS123 pre-processor is present.

			(2) CCSDS121 pre-processor (unit-delay predictor) is present (3) Any-other pre-processor is present.
DISABLE_HEADER_tb	integer	[0,1]	Selects whether to disable (1) or not (0) the header generation.
test_id	Integer	[0,2,4,5,7,40]	Selects the behaviour of the testbench.
stim_file	string	*Stimuli_file*	Stimuli file with the samples to compress.
ref_file	string	*Reference file*	Reference file with samples already compressed.
out_file	string	*Output file*	Compressed file obtained with the CCSDS-121 IP Core.

(*) IS_SIGNED_tb configuration parameter is taken into account just when the unit-delay predictor is present (PREPROCESSOR_tb = 2). Otherwise, it is assumed that input samples are always unsigned.

4.7.1 Testbench behavioural description

The behaviour of the testbench can be determined by setting the constant "test_id". Table 4-6 shows the possible "test_id" values and the behaviour of the testbench for each case.

TABLE 4-6: CCSDS121 IP TESTBENCH – TEST SEQUENCES

test_id	Description
0	1. Write run-time configuration values in memory-mapped registers if required. 2. Send raw samples to the CCSDS121 IP core when it is ready. 3. Wait for CCSDS121 IP to signal that it has finished the compression and is ready to receive a new configuration. 4. Compare the output of the CCSDS121 IP core with reference file. 5. Repeat procedures from step 1 to 4.
2	1. Write run-time configuration values in memory-mapped registers if required. 2. Send raw samples to the CCSDS121 IP core. 3. Send configuration while the CCSDS121 IP core is compressing. 4. Wait for CCSDS121 IP to signal that it has finished the compression and is ready to receive a new configuration (configuration sent during compression has been ignored). 5. Compare the output of the CCSDS121 IP core with reference file. 6. Repeat procedures from step 1 to 5.

4	1. Write run-time configuration values in memory-mapped registers if appropriate. 2. Send some raw samples to the CCSDS121 IP core. 3. Activate ForceStop signal. 4. Wait for CCSDS121 IP to signal that it has finished the compression (because of the ForceStop) and is ready to receive a new configuration. 5. Write run-time configuration values in memory-mapped registers. 6. Send raw samples to the CCSDS121 IP core. 7. Wait for CCSDS121 IP to signal that it has finished the compression and is ready to receive a new configuration. 8. Compare the output of the CCSDS121 IP core with reference file.
5	1. Write run-time configuration values in memory-mapped registers if required. 2. Send raw samples to the CCSDS121 IP core. 3. Wait for CCSDS121 IP to signal that it has finished the compression and is ready to receive a new configuration. 4. Compare the output of the CCSDS121 IP core with reference file. 5. Repeat procedures from step 1 to 4 with different configuration and different stimuli and reference files.
7	1. Write invalid run-time configuration values in memory-mapped registers. 2. Wait for CCSDS121 IP to signal that there has been a configuration error, and it has finished the compression and is ready to receive a new configuration. 3. Write valid run-time configuration values in memory-mapped registers for the second time. 4. Send raw samples to the CCSDS121 IP core. 5. Wait for CCSDS121 IP to signal that it has finished the compression and is ready to receive a new configuration. 6. Compare the output of the CCSDS121 IP core with reference file.
40	1. Write run-time configuration values in memory-mapped registers if appropriate. 2. Send some raw samples to the CCSDS121 IP core. 3. Deactivate Ready_Ext signal for a few clock cycles during compression in order to halt the process. 4. Reactivate Ready_Ext to resume execution. 5. Wait for CCSDS121 IP to signal that it has finished the compression and is ready to receive a new configuration.

	6. Compare the output of the CCSDS121 IP core with reference file.

4.7.2 Testbench assertions

The following assertions are included in the testbench:

TABLE 4-7: CCSDS121 IP TESTBENCH – ASSERTIONS

Report	Severity	Description
Finished started with a high value	warning	Checks correct value of control signals after reset of the IP core.
Ready started with a high value	warning	
AwaitingConfig started with a low value	warning	
Sending a new configuration	note	Configuration registers are sent to the AHB bus.
AwaitingConfig lowered correctly when configuration was received	note	Configuration has been correctly received by the IP core.
Ready asserted correctly when IP core is ready to receive new samples	note	IP core is ready to receive samples.
Ready not asserted correctly when IP core has been configured	warning	IP core has been configured, but has not signalled that it is ready to receive samples.
Unexpected IP core error during compression	error	Unexpected error during compression.
Unexpected IP core error after compression	error	Unexpected error after compression.
Finished correctly activated when compression finished	note	IP core has finished the compression.
AwaitingConfig correctly activated after compression finished	note	IP core is waiting to be sent a new configuration after a previous configuration operation finished.
Error between sequential compressions, value of Finished shall be kept high	error	Incorrect Finished signal value after a configuration operation has finished.
Error for sequential compressions, Finished shall be de-asserted with AwaitingConfig	error	Incorrect Finished signal value after the configuration is received between sequential compressions.
Two sequential compressions test performed	note	Test finished correctly when test_id = 0
Two sequential compressions (attempting to reconfigure) test performed	note	Test finished correctly when test_id = 2
ForceStop assertion	note	ForceStop signal asserted by the testbench.
Finished correctly activated after ForceStop	note	Correct value in Finished and
AwaitingConfig asserted correctly after ForceStop	note	AwaitingConfig signals after ForceStop assertion.
ForceStop and one compression test performed	note	Test finished correctly when test_id = 4
Two different compressions test performed	note	Test finished correctly when test_id = 5
Sending invalid configuration...	note	The testbench is sending configuration registers with invalid values through the AHB bus.
AwaitingConfig lowered correctly when configuration was received (even with error)	note	Correct value in AwaitingConfig signal after the configuration was received.
Error has been correctly asserted	note	Correct value in Error signal after invalid configuration values were sent.

Finished has been correctly asserted after an error	note	Correct values in Finished and AwaitingConfig after a configuration error was signalled by the IP.
AwaitingConfig asserted correctly after an error	note	
Error has not been correctly asserted when error	error	Incorrect values in Error and Finished signals after a configuration error was signalled by the IP.
Finished has not been correctly asserted after an error	error	
Configuration error and one compression test performed	note	Test finished correctly when test_id = 7
Comparison was successful!	note	Compressed output file and reference file are identical.
Comparison not possible because there has been a ForceStop assertion	note	Compressed output file and reference file cannot be compared.
Comparison not possible because there has not been compression performed (configuration error)	note	
Problems in final stream	error	Comparison between reference and compressed file was not successful (IP core output file and reference file are not identical).
Reference file has more samples	error	
Output file has more samples	error	
****** CCSDS-121 Testbench done ******	note	Testbench has reached the end.

4.8 CCSDS121 IP simulation and synthesis scripts

The information in this section is provided to allow a user to understand how the configuration of the IP core, the testbench or the synthesis can be performed. All the procedures explained below are automated in the provided makefile described in Section 4.4.

A python script is provided in order to ease the generation of the necessary configuration files for synthesis or simulation. Such script reads the configuration values from a *.csv file with a specific format (see Section 4.8.1) and generates the necessary *.vhd files that are used to configure the IP core and the testbench.

4.8.1 Writing the desired parameters in the *.csv file

The configuration *.csv file is used to generate the necessary configuration files for simulation or synthesis. It is structured as follows:

- Line1: Comma-separated row identifiers.
- Line2: Comma-separated parameter names.
- All other lines: Comma-separated configuration values (one set of configuration values per line).

The meaning of each row in the *.csv file is explained in Table 4-8, where the “Mandatory” field has the following meaning:

- Y: parameter is mandatory.
- SM: parameter is mandatory for simulation.
- SMR: parameter is mandatory for simulation only if runtime configuration is enabled (EN_RUNCFG =1).
- N: parameter is not mandatory.

TABLE 4-8: CCSDS121 IP CONTENT OF THE *.CSV CONFIGURATION FILE.

Row Identifier	Parameter name	Description	Mandatory	Written in
0	TestId	Unique test identifier.	Y	ccsds121_tb_parameters.vhd
1	Image ID	Image identifier.	N	N/A*
2	Input Image	Filename of the input image to compress.	SM	ccsds121_tb_parameters.vhd
3	Ny	Number of lines.	SM	ccsds121_tb_parameters.vhd
4	Nx	Number of columns.	SM	ccsds121_tb_parameters.vhd
5	Nz	Number of bands.	SM	ccsds121_tb_parameters.vhd
6	D	Dynamic range of the input samples.	SM	ccsds121_tb_parameters.vhd
7	IS_SIGNED***	(0) Unsigned samples. (1) Signed samples.	SM	ccsds121_tb_parameters.vhd
8	ENDIANESS	(0) Little Endian. (1) Big Endian.	SM	ccsds121_tb_parameters.vhd
9	Set	Identifier of the selected set of parameters.	SM	N/A
10	EN_RUNCFG	(1) Enable run-time configuration (0) Disable run-time configuration	Y	ccsds121_parameters.vhd
11	RESET_TYPE	(1) Synchronous reset. (0) Asynchronous reset.	Y	ccsds121_parameters.vhd
12	PREPROCESSOR	(0) Pre-processor is not present. (1) CCSDS123 pre-processor is present. (2) CCSDS121 pre-processor (unit-delay predictor) is present. (3) Any-other pre-processor is present.	Y	ccsds121_tb_parameters.vhd and ccsds121 _parameters.vhd
13	J_GEN	Maximum allowed Block Size.	Y	ccsds121_parameters.vhd
14	CODESET_GEN	Generic Code Option.	Y	ccsds121_parameters.vhd
15	REF_SAMPLE_GEN	Maximum allowed reference sample interval.	Y	ccsds121_parameters.vhd
16	W_BUFFER_GEN	Maximum allowed number of bits in the output buffer.	Y	ccsds121_parameters.vhd
17	DISABLE_HEADER_GEN	Generic to select whether to disable (1) or not (0) the header generation.	Y	ccsds121_parameters.vhd
18	RESERVED	Reserved	N	N/A

19	DISABLE_HEADER	Selects whether to disable (1) or not (0) the header generation.	SMR	ccsds121_tb_parameters.vhd
20	J	Block Size	SMR	ccsds121_tb_parameters.vhd
21	CODESET	Code Option	SMR	ccsds121_tb_parameters.vhd
22	REF_SAMPLE	Reference Sample Interval	SMR	ccsds121_tb_parameters.vhd
23	W_BUFFER	Number of bits in the output buffer	SMR	ccsds121_tb_parameters.vhd
24	BYPASS	Selects whether to bypass residuals (1) or compress them (0)	SMR	ccsds121_tb_parameters.vhd
25	output image	Output file name to store the compressed image	SM	ccsds121_tb_parameters.vhd
26	Ny_GEN	Maximum number of lines	Y**	ccsds121_parameters.vhd
27	Nx_GEN	Maximum number of columns	Y**	ccsds121_parameters.vhd
28	Nz_GEN	Maximum number of bands	Y**	ccsds121_parameters.vhd
29	D_GEN	Maximum dynamic range	Y**	ccsds121_parameters.vhd
30	IS_SIGNED_GEN ***	(0) Unsigned samples (1) Signed samples	Y**	ccsds121_parameters.vhd
31	ENDIANESS_GEN	(0) Little Endian (1) Big Endian	Y**	ccsds121_parameters.vhd

** N/A means not applicable.*
*** When EN_RUNCFG = 0 (runtime configuration disabled), the compile-time parameters used to set the image size (Nx_GEN, Ny_GEN, Nz_GEN), dynamic range (D_GEN), sign (IS_SIGNED_GEN) and endianness (ENDIANESS_GEN) have to be set according to the actual size, dynamic range, sign and endianness of the image to be compressed. The python script will check the rules in* Table 4-9 *and raise an exception if the rules are not met.*
**** IS_SIGNED and IS_SIGNED_GEN configuration parameters are taken into account just when the unit-delay predictor is present (PREPROCESSOR = 2). Otherwise, it is assumed that input samples are always unsigned.*

TABLE 4-9: CCSDS121 IP - RULES TO FILL THE *.CSV FILE WHEN EN_RUNCFG = 0

Configuration rule	Correspondence in *.csv file
Ny_GEN = Ny	row[26] = row[3]
Nx_GEN = Nx	row[27] = row[4]
Nz_GEN = Nz	row[28] = row[5]
D_GEN = D	row[29] = row[6]
IS_SIGNED_GEN = IS_SIGNED	row[30] = row[7]
ENDIANESS_GEN = ENDIANESS	row[31] = row[8]

4.8.2 Simulation

Simulations can be executed by first generating the necessary configuration files and simulation scripts and then running the aforementioned scripts in Questa/ModelSim.

4.8.2.1 Generating parameters files and scripts for simulation

Follow these steps in order to generate the parameter files and scripts for simulation:

1. Fill the *.csv file with the desired configuration values and names of the raw image to be compressed and reference image for verification as explained in Section 4.8.1.

2. Ensure that the raw images to be compressed and reference images for comparison are located respectively in the $121DB/images/raw folders and $121DB/images/reference (the script will notice missing files and will not include test cases with missing files for simulation).

3. Run the script $121DB/verification_scripts/run_vhdl_tests_121.py from a terminal, with the following arguments:

```
$ 121DB/verification_scripts/run_vhdl_tests_121.py $CSV_FILE $RAW_FOLDER $COMPRESSED_FOLDER $R
                                                   EFERENCE_FOLDER $DATABASE_DIRECTORY $OPTION
```

TABLE 4-10: CCSDS121 IP – ARGUMENTS OF THE RUN_VHDL_TESTS_121.PY SCRIPT FOR SIMULATION

Argument	Meaning	Example
$CSV_FILE	*.csv file used to generate the configuration files.	$121DB/verification_scripts/testcases_121_e_all.csv
$RAW_FOLDER	Path to the folder that contains the raw images to be compressed.	$121DB/images/raw
$COMPRESSED_FOLDER	Path to the folder where the compressed images will be stored.	$121DB/images/compressed
$REFERENCE_FOLDER	Path to the folder that contains the reference images for comparison.	$121DB/images/reference
$DATABASE_DIRECTORY	Path to the database root directory.	$121DB
$OPTION	If $OPTION = modelsim, it generates the necessary scripts for simulation.	N/A

The script will generate the files listed in Table 4-11.

TABLE 4-11: CCSDS121 IP – LIST OF FILES GENERATED BY THE PYTHON SCRIPT

Folder	Filename	Description
$121DB/modelsim/tb_stimuli/{*TestId*}	ccsds121_parameters.vhd	File containing generic parameters. {TestId} is the unique test identifier (row[0] of the *.csv file).
	ccsds121_tb_parameters.vhd	File containing run-time configuration values used by the testbench. {TestId} is the unique test identifier (row[0] of the *.csv file).

$121DB/modelsim/tb_scripts	testbench.do	Script to compile the testbench and necessary files for the test.
	ip_core.do	Script to compile VHDL sources of the CCSDS121 IP core. This file includes corresponding flags to enable code coverage in the compilation command.
	{*TestId*}.do	Script to perform each simulation test case. This script covers the elimination of existing libraries and its creation, compilation of the corresponding parameters file, calling previous scripts and finally starting simulation. {*TestId*} is the unique test identifier (row[0] of the *.csv file).
	all_tests.do	Script to run the simulations of all the individual simulation scripts. This script manages the coverage merge from the successful simulations performed.
	verification_report_not_performed.txt	File containing test cases impossible to perform, since the input raw file to be compressed or the reference compressed file have not been found. This file will not be generated if all the test cases are successful.

4.8.2.2 Running the testbench

After creating the necessary configuration scripts and *.do files, the test cases can be simulated in QuestaSim/Modelsim by following these steps:

1. Open Modelsim/Questasim.
2. Open the project $121DB/modelsim/shyloc121.mpf.
3. Run:
    a. A single test:
        i. Create variable to the location of the database directory:

        set SRC $121DB

        ii. do $121DB/modelsim/tb_scripts/{TestId}.do.
    b. All tests cases in the *.csv file: do $121DB/modelsim/tb_scripts/all_tests.do.

{TestId} is the unique test identifier (row[0] of the *.csv file).

If the file all_tests.do is run, the following files are generated after completion of all test cases:

TABLE 4-12: CCSDS121 IP – LIST OF FILES GENERATED AFTER COMPLETION OF THE ALL_TESTS.DO FILE

Folder	Filename	Description

$121DB/modelsil/tb_scripts	verification_report.txt	File containing performed test cases and their result, in terms of: PASSED or FAILED.
$121DB/modelsim/tb_stimuli/{*TestId*}	report_coverage.txt	If test passed, this file contains the report coverage for the specific simulation.
	report_coverage_details.txt	If test passed, this file contains the report coverage in detail for the specific simulation.
$121DB/modelsim/cover	{*TestId*}_cover.ucdb	If test passed, this file contains the coverage database for the specific simulation.
	merged_result.ucdb	This file contains the merged coverage database from the {*TestId*}_cover.ucdb files.
	merged_result.txt	This file contains the merged coverage report.

4.8.3 Synthesis

Synthesis can be performed by first generating the necessary configuration files and synthesis scripts and then running the aforementioned scripts in ISE Project Navigator, Synplify or NanoXmap for FPGA targets, or Synopsys Design Compiler for ASIC targets.

4.8.3.1 Generating parameters files and scripts for synthesis

Follow these steps in order to generate the parameter files and scripts for synthesis:

1. Fill the *.csv file with the desired configuration values.
2. Run the script $121DB/verification_scripts/run_vhdl_tests_121.py from a terminal, with the following arguments:

```
$ $  $121DB/verification_scripts/run_vhdl_tests_121.py $CSV_FILE $RAW_FOLDER $COMPRESSED_FOLDE
R $REFERENCE_FOLDER $DATABASE_DIRECTORY $OPTION
```

TABLE 4-13: CCSDS121 IP – ARGUMENTS OF THE RUN_VHDL_TESTS_121.PY SCRIPT FOR SYNTHESIS

Argument	Meaning	Example
$CSV_FILE	*.csv file used to generate the configuration files.	$121DB/verification_scripts/synthesis_params_121_e.csv
$RAW_FOLDER	Path to the folder that contains the raw images to be compressed.*	$121DB/images/raw

$COMPRESSED_FOLDER	Path to the folder where the compressed images will be stored. *	$121DB/images/compressed
$REFERENCE_FOLDER	Path to the folder that contains the reference images for comparison. *	$121DB/images/reference
$DATABASE_DIRECTORY	Path to the database root directory.	$121DB
$OPTION	If ($OPTION = ise or $OPTION = synplify or $OPTION = brave or $OPTION = dc), it generates the necessary scripts for synthesis with the selected tool.	N/A

** The script requires the paths to be present, however when $OPTION = ise or $OPTION = synplify or $OPTION = brave or $OPTION = dc the folders are not used.*

The script will generate the files listed in Table 4-14.

TABLE 4-14: CCSDS121 IP– LIST OF FILES GENERATED AFTER COMPLETION OF THE ALL_ISES.TCL, ALL_SYNPLIFY.TCL AND ALL_NANOXPLORE.PY SCRIPTS.

Folder	Filename	Description
$121DB/synthesis/premap_parameters/{*TestId*}	ccsds121_parameters.vhd	File containing generic parameters used for each synthesis case. {*TestId*} is the unique synthesis identifier (row[0] of the *.csv file).
$121DB/synthesis/syn_scripts/ise **when $OPTION = ise** **$121DB/synthesis/syn_scripts/synplify** **when $OPTION = synplify** **$121DB/synthesis/syn_scripts/brave** **when $OPTION = brave** **$121DB/synthesis/syn_scripts/dc** **when $OPTION = dc**	{*TestId* }.tcl (for synthesis in ISE Navigator or Synplify) {*TestId* }.py (for synthesis in NanoXmap)	Script to perform each synthesis test case. This script covers the elimination of existing sources its addition, addition of the corresponding parameters file, calling other necessary scripts and finally launching the synthesis. {*TestId*} is the unique synthesis identifier (row[0] of the *.csv file).
	add_ip_core.tcl (for synthesis in ISE Navigator or Synplify) add_ip_core.py (for synthesis in NanoXmap)	Script to include and compile the necessary VHDL sources of the CCSDS121 IP core linked to the proper library.
	all_ise.tcl (when $OPTION = ise) all_synplify.tcl (when $OPTION = synplify) all_nanoxplore.py (when $OPTION = brave) all_dc.tcl (when $OPTION = dc)	Script to create or just open the synthesis project and perform each individual synthesis script. This script manages the storage process of the synthesis results in the proper folder.
Only **$121DB/synthesis/syn_scripts/dc**	setupLibs.do	Setup the libraries, for Design Compiler elaboration.

when $OPTION = dc		

4.8.3.2 Running the synthesis for FPGA targets

After creating the necessary configuration scripts and *.tcl or *.py files, the synthesis can be performed in ISE, Synplify or NanoXmap by following these steps:

1. Open the synthesis tool.
2. Change directory to:
    a. In ISE:

       cd $121DB/synthesis/syn_scripts/ise

    b. In Synplify:

       cd $121DB/synthesis/syn_scripts/synplify

    c. In NanoXmap:

       cd $121DB/synthesis/syn_scripts/brave

3. Run:
    a. A single synthesis process:
        i. Create variable to the location of the database directory:

           set SRC $121DB

        ii. In ISE:

            source $SRC/synthesis/syn_scripts/ise/{*TestId*}.tcl.

            In Synplify:

            run_tcl $SRC/synthesis/syn_scripts/synplify/{*TestId*}.tcl.

            In NanoXmap:

            nanoxpython $SRC/synthesis/syn_scripts/brave/{*TestId*}.py.

    b. All synthesis cases in the *.csv file:

       In ISE:

       source $SRC/synthesis/syn_scripts/ise/all_ise.tcl.

       In Synplify:

run_tcl $SRC/synthesis/syn_scripts/synplify/all_synplify.tcl.

In NanoXmap:

nanoxpython $SRC/synthesis/syn_scripts/brave/all_nanoxplore.py.

{*TestId*} is the unique synthesis identifier (row[0] of the *.csv file).

If the file all_{ise,synplify}.tcl or all_nanoxplore.py is run, the following files are generated after completion of all synthesis:

TABLE 4-15: CCSDS121 IP – LIST OF FILES GENERATED AFTER COMPLETION OF THE ALL_ISE.TCL, ALL_SYNPLIFY.TCL AND ALL_NANOXPLORE.PY FILE.

Folder	Filename	Description
$121DB/synthesis/syn_scripts/synplify/report when ($OPTION = synplify)	{*TestId*}_synplify.srr	Log file containing implementation results.
	{*TestId*}_synplify _fpga_mapper_timing_report .xml	File containing synthesis results in terms of timing.
	{*TestId*}_synplify _fpga_mapper_area_report.x ml	File containing synthesis results in terms of area.
$121DB/synthesis/syn_scripts/brave/report when ($OPTION = brave)	general_{*TestId*}_{*Device*}.log	Log file containing implementation results, including timing and area.
$121DB/synthesis/syn_scripts/ise/report when ($OPTION = ise)	N/A*	N/A*

**Current implementation does not create copies of implementation results for ISE. The user can find them in their local ISE project directory.*

4.8.3.3 Running the synthesis for ASIC targets

The IP core can also be synthesized targeting ASIC standard cell libraries, through a set of scripts. The available flow is based on Synopsys Design Compiler. The result of the synthesis process is a set of files containing the netlist, and reports for area, timing, power and quality of results (QoR).

The execution of ASIC synthesis requires setting up the Standard Cell Libraries to be used, and may also require setting up technology-specific memories, for the FIFOs' SRAM memories.

4.8.3.3.1 Base Scripts and Files

The scripts are based on Synopsys Reference Methodology scripts, which can be used both for logical and/or physical synthesis. The default corner considered is Worst-Case Military (WCMIL).

The following table describes the available files, both the provided scripts and files generated by the synthesis process.

TABLE 4-16: CCSDS121 IP – LIST OF FILES PERTAINING TO ASIC SYNTHESIS

Folder	Filename	Description
$121DB/synthesis/syn_scripts/base	ccsds121_shyloc_top.sdc	Constraint file, with the clock signals
	dc_setup.tcl	Set-up script, where the user has to specifiy the location of the Standard Cell libraries
	dc.tcl	Main Design Compiler script

In order to perform synthesis with a standard cell library, the dc_setup.tcl file has to be edited. The following table describes the variables which must be set in order to be able to perform logical synthesis.

TABLE 4-17: CCSDS121 IP – ASIC STANDARD CELL LIBRARY SETUP FOR LOGICAL SYNTHESIS

File	Variable	Description
$121DB/synthesis/syn_scripts/base/dc_setup.tcl	ADDITIONAL_SEARCH_PATH	Path to be added to the search path, e.g . Std. Cell Library based folder
	TARGET_LIBRARY_FILES	Target technology logical libraries, e.g. core cells
	ADDITIONAL_LINK_LIB_FILES	Extra link logical libraries not included in TARGET_LIBRARY_FILES, e.g. corners

Additionally, Milkyway libraries may be defined, in order to perform physical synthesis. These must be set in the same file, and the variables are below the ones for logical synthesis.

4.8.3.3.2 Synthesis Execution

The execution of the synthesis scripts is performed through the Makefile available at the IP base folder ($121DB). In order to run, the user has to first edit the dc_setup.tcl in order to setup the standard cell library environment. When set, the user can issue the command “make dc” and the targets defined in the CSV file will be synthesised.

4.8.3.3.3 Generated Output Files

The synthesis process generates several files, with reports and design descriptions which can be used in simulation or analysis, e.g. Verilog netlist. The following table describes the generated files.

TABLE 4-18: CCSDS121 IP – LIST OF FILES GENERATED BY ASIC SYNTHESIS

Folder	Filename	Description
	ccsds121_shyloc_top.check_design.rpt	Design rule check, with internal lint

$121DB/synthesis/syn_scripts/dc/report/{*Id*}	ccsds121_shyloc_top.mapped.area.rpt	Area usage report
	ccsds121_shyloc_top.mapped.clock_gating.rpt	Clock gating report, not applicable but part of the reference methodology
	ccsds121_shyloc_top.mapped.power.rpt	Power consumption report
	ccsds121_shyloc_top.mapped.qor.rpt	Quality of results, and overall summary, including slack from static timing analysis.
	ccsds121_shyloc_top.mapped.timing.rpt	Short report, with one path per clock
$121DB/synthesis/syn_scripts/dc/results/{*Id*}	ccsds121_shyloc_top.elab.ddc	Elaborated (Generic Cells) design database, to be used with Design Compiler
	ccsds121_shyloc_top.mapped.ddc	Mapped (Standard Library Cells) design database, to be used with Design Compiler
	ccsds121_shyloc_top.mapped.sdc	Mapped design constraint file
	ccsds121_shyloc_top.mapped.sdf	Mapped design delay file, for timing analysis and simulation
	ccsds121_shyloc_top.mapped.svf	Mapped design information for formal equivalence checking
	ccsds121_shyloc_top.mapped.v	Mapped design Verilog netlist
	elab	Temporary folder with the elaborated design units
	ICC2_files	Files for IC-Compiler II back-end

4.8.4 Post-synthesis simulations

Running the post-synthesis simulations has not been automated in the provided *makefile*. However, the python script offers the possibility to configure the IP and generate the necessary configuration scripts that enable the generation of the post-synthesis model with Synplify. Instructions are provided below.

4.8.4.1 Generating the post-synthesis model

Follow the procedures in Section 4.8.3 with the following exceptions:

- When running the python script, using the command line:

```
$ $  $121DB/verification_scripts/run_vhdl_tests_121.py $CSV_FILE $RAW_FOLDER $COMPRESSED_FOLDE
R $REFERENCE_FOLDER $DATABASE_DIRECTORY $OPTION $TECHNOLOGY
```

  use $OPTION = synplify-ps.

- The user is offered the possibility of selecting the target technology with the argument $TECHNOLOGY. The following values are accepted {XC5VFX130T, XQR5VFX130, A3PE3000, RTAX4000S, RT4G4150}. If the user does not specify a target technology, the synthesis will be run for all the possible target technologies.

The python script will generate the *.tcl scripts for Synplify listed in Table 4-14.

With the *.tcl scripts, the synthesis can be run with Synplify as explained in Section 4.8.3.2. The *.tcl scripts will use the provided wrapper, located in $121DB/src/post_syn/ccsds121_top_wrapper.vhd, as top module. This wrapper flattens the records used as I/O ports and instantiates the CCSDS121 IP core.

After the synthesis is completed, the generated post-syntesis model (ccsds121_top_wrapper.vhm) is stored in $121DB/src/post_syn/$TECHNOLOGY/{TestId}/ccsds121_top_wrapper.vhm.

4.8.4.2 Generating the scripts for running the post-synthesis simulations

Follow the same procedure used for behavioural simulations, going along the instructions in Section 4.8.2.1 with the following exception:

- When running the script $121DB/verification_scripts/run_vhdl_tests_121.py, use $OPTION = modelsim-ps and select a technology with the argument $TECHNOLOGY.

```
$ $121DB/verification_scripts/run_vhdl_tests_121.py $CSV_FILE $RAW_FOLDER $COMPRESSED_FOLDER $
REFERENCE_FOLDER $DATABASE_DIRECTORY $OPTION $TECHNOLOGY
```

- The script requires the user to select a $TECHNOLOGY among the possible values { XC5VFX130T, XQR5VFX130, A3PE3000, RTAX4000S, RT4G4150}. This argument is mandatory.

The python script will generate the files listed in Table 4-11, plus the scripts for post-synthesis simulations for each test case, which are located in $121DB/modelsim/tb_scripts/{TestId}_ps.do. The all_tests.do file will be filled with the sentences to execute the post-synthesis simulation scripts for all test cases $121DB/modelsim/tb_scripts/{TestId}_ps.do.

4.8.4.3 Running the testbench for post-synthesis simulations

The testbench can be executed after generating all the necessary *.do files by following the procedures in Section 4.8.2.2.

Note that it is up to the user to pre-compile and map the technology vendor libraries (axcelerator, unisim …) in QuestaSim.

4.8.5 Post-PAR simulations

Post-PAR simulations can be run for Virtex5 only, and for the testcases provided in the file $121DB/verification_scripts/testcases121.csv. The following steps are necessary:

1. Run the python configuration script for post-synthesis simulations on Virtex5 with the following arguments: ($CSV_FILE = $121DB/verification_scripts/testcases121.csv ; $OPTION = synplify-ps and $TECHNOLOGY = XC5VFX130T):

```
$ $121DB/verification_scripts/run_vhdl_tests_121.py $121DB/verification_scripts/testcases121.c
sv $RAW_FOLDER $COMPRESSED_FOLDER $REFERENCE_FOLDER $DATABASE_DIRECTORY synplify-ps XC5VFX130T
```

2. Run the synthesis with Synplify with the script $121DB/synthesis/syn_scripts/synplify/all_synplify.do

3. Once finished, run the script $121DB/synthesis/syn_scripts/synplify/all_synplify_par.do to run the PAR for all the testcases.

4. Run the command:

   "netgen -intstyle ise -s 1 -pcf ccsds123_top_wrapper.pcf -rpw 100 -tpw 0 -ar Structure -tm ccsds123_top_wrapper -insert_pp_buffers true -w -dir netgen/par -ofmt vhdl -sim ccsds123_top_wrapper.ncd ccsds123_top_wrapper_timesim.vhd"

   for all the tests from its par folder (e.g. ./synthesis/syn_scripts/synplify/XC5VFX130T_03_Test/par_1). This Xilinx command generates the vhdl model for PAR simulations and the sdf timing file.

5. Use the provided python script for post-PAR simulations $121DB/verification_scripts/run_vhdl_tests_121_V5_PAR.py with $OPTION = modelsim.

```
$ $121DB/verification_scripts/run_vhdl_tests_121_V5_PAR.py $121DB/verification_scripts/tes
tcases121.csv $RAW_FOLDER $COMPRESSED_FOLDER $REFERENCE_FOLDER $DATABASE_DIRECTORY modelsi
m
```

```
run_vhdl_tests_121_V5_PAR.py    testcases_121.csv    ../images/raw    ../images/compressed
../images/reference ../ modelsim
```

6. The script generates a set of test scripts for all the test cases. The glbl.v file is added for compilation (vlog $XILINX/verilog/src/glbl.v). The user needs to set the $XILINX folder.

7. In order to run the simulations the libraries below are also needed: Simprim & Vital2000 libs.

8. The scripts might be executed from QuestaSim with the command:

   do $121DB/modelsim/tb_scripts/all_tests.do

5 CCSDS123 IP CORE USER MANUAL

5.1 System overview

The CCSDS123 IP Core performs the compression of a set of input samples according to the specifications of the CCSDS123 standard [AD-2].

5.2 Installation

The Git repository is located in: https://git.iuma.ulpgc.es:8300/TRP-AO8032/SHyLoC-e.git

The sources can be cloned with the command:

```
$ git clone https://git.iuma.ulpgc.es:8300/TRP-AO8032/SHyLoC-e.git
```

This will create a copy of the database in the user's local directory. The user might access the root folder of the database and run the provided makefile (see Section 4.4).

```
$ cd CCSDS123IP-VHDL
```

Throughout this document, all the paths starting with $123DB are related to the root of the database directory.

Some simulation test cases instantiate the CCSDS121 IP core as external encoder. Therefore, the IP core database of the CCSDS121 IP core has to be present.

5.3 CCSDS123 IP database directory structure

```
CCSDS123IP-VHDL ($123DB)
├──images
│   ├──compressed
│   ├──raw
│   └──reference
├──modelsim
│   ├──cover
│   ├──tb_scripts
│   └──tb_stimuli
├──src
│   ├──post-syn
│   ├──shyloc_123
│   ├──shyloc_utils
│   └──tb
├──synthesis
│   ├──premap_parameters
│   └──syn_scripts
│       ├──dc
│       │   ├──base
```

```
│           │   ├──report
│           │   └──results
│           └──synplify
│               └──report
└──verification_scripts
```

5.4 Makefile

The makefile can be called from the database root directory, and provides the following options:

- *make ccsds123:* generates the configuration files that conform the simulation environment and runs the simulations in QuestaSim. The set of test cases to be simulated is determined by the information contained in the comma-separated file: $123DB/verification_scripts/testcases_123_e_all.csv. Such file comes already pre-filled with the test cases contemplated in the Verification and Validation plan [AD-3]. A pass/fail report is generated and saved in $123DB/modelsim/tb_scripts/verification_report.txt. Coverage results are additionally generated and merged after completion of all tests.

- *make synplify:* generates the configuration files that conform the synthesis environment and runs the synthesis in Synplify Premier with DP. The set of synthesis cases to be implemented is determined by the information contained in the comma-separated file: $123DB/verification_scripts/synthesis_params_123_e.csv. Such file comes already pre-filled with synthesis cases. Synthesis results in terms of timing and area are saved in the folder $123DB/synthesis/syn_scripts/synplify/report.

- *make ise:* generates the configuration files that conform the synthesis environment and runs the synthesis in ISE Project Navigator. The set of synthesis cases to be implemented is determined by the information contained in the comma-separated file: $123DB/verification_scripts/synthesis_params_123_e.csv. Such file comes already pre-filled with synthesis cases.

- *make brave:* generates the Python scripts that conform the synthesis environment and run the synthesis in NanoXplore NanoXmap, using the Python API named as NanoXpython. The set of synthesis cases to be implemented is determined by the information contained in the comma-separated file: $123DB/verification_scripts/synthesis_params_123_e.csv. Such file comes already pre-filled with synthesis cases. Synthesis results in terms of timing and area are saved in the folder $123DB/synthesis/syn_scripts/brave/report.

- *make dc:* generates the configuration files that conform the synthesis environment and runs the synthesis in Synopsys Design Compiler. The set of synthesis cases to be implemented is determined by the information contained in the comma-separated file: $123DB/verification_scripts/synthesis_params_123.csv. Such file comes already pre-filled with synthesis cases. Synthesis results in terms of timing and area are saved in folders contained in $123DB/synthesis/syn_scripts/dc/report, and resulting netlist and associated files are saved in folders contained in $123DB/synthesis/syn_scripts/dc/results.

IMPORTANTE NOTE: Successful execution of the simulations relies on the presence of a set of raw images and compressed reference files. Such images are not included as part of the IP core's database due to their excessive volume. They can be downloaded from the NAS repository:

http://nasdsi.iuma.ulpgc.es/

in the folder:

/TRPAO8032-IUMA/SHyLoC-e/Test-Images/123

They must be placed in the available $123DB/images/raw and $123DB/images/reference folders of the IP core's database.

5.5 CSDS123 IP VHDL sources and compile order

TABLE 5-1: CCSDS123 IP – VHDL SOURCES AND COMPILE ORDER

Folder	Compile order	Filename	Library
$123DB/src/shyloc_123	1	ccsds123_parameters.vhd	shyloc_123
$123DB/src/shyloc_utils	2	amba.vhd	shyloc_utils
	3	shyloc_functions.vhd	shyloc_utils
	4	reg_bank_inf.vhd	shyloc_utils
	5	reg_bank_tech.vhd	shyloc_utils
	6	reg_bank.vhd	shyloc_utils
$123DB/src/shyloc_utils/edac-0-7-src	5	edac-decl-0-7.vhd	shyloc_utils
	6	edac-body-0-7.vhd	shyloc_utils
	7	edac-rtl.vhd	shyloc_utils
$123DB/src/shyloc_utils	8	fixed_shifter.vhd	shyloc_utils
	9	barrel_shifter.vhd	shyloc_utils
	10	bitpackv2.vhd	shyloc_utils
	11	toggle_sync.vhd	shyloc_utils
	12	reset_sync.vhd	shyloc_utils
	13	fifop2_base.vhd	shyloc_utils
	14	fifop2_edac.vhd	shyloc_utils
	15	fifop2.vhd	shyloc_utils
$123DB/src/shyloc_123	16	ccsds123_constants.vhd	shyloc_123
	17	config123_package.vhd	shyloc_123
	18	ccsds123_shyloc_interface.vhd	shyloc_123
	19	clip.vhd	shyloc_123
	20	create_cdwv2.vhd	shyloc_123
	21	ff.vhd	shyloc_123
	22	fifo_ctr_funcs.vhd	shyloc_123
	23	finished_gen.vhd	shyloc_123
	24	header123_gen.vhd	shyloc_123
	25	ld_2d_fifo.vhd	shyloc_123
	26	ld_2d_fifo_bil.vhd	shyloc_123
	27	localdiff_shift.vhd	shyloc_123
	28	localdiffv3.vhd	shyloc_123
	29	localsumv2.vhd	shyloc_123
	30	map2stagesv2.vhd	shyloc_123
	31	mult.vhd	shyloc_123
	32	mult_acc2stagesv2.vhd	shyloc_123

	33	opcode_update.vhd	shyloc_123
	34	packing_top_123.vhd	shyloc_123
	35	predictor2stagesv2.vhd	shyloc_123
	36	record_2d_fifo.vhd	shyloc_123
	37	ro_update_mathv3_diff.vhd	shyloc_123
	38	count_updatev2.vhd	shyloc_123
	39	sample_comp.vhd	shyloc_123
	40	sample_fsm.vhd	shyloc_123
	41	sample_top.vhd	shyloc_123
	42	wei_2d_fifo.vhd	shyloc_123
	43	weight_update_shyloc.vhd	shyloc_123
	44	weight_update_shyloc_top.vhd	shyloc_123
	45	adder.vhd	shyloc_123
	46	n_adders.vhd	shyloc_123
	47	n_adders_top.vhd	shyloc_123
	48	ccsds_ahb_types.vhd	shyloc_123
	49	ahb_utils.vhd	shyloc_123
	50	ccsds123_ahbs.vhd	shyloc_123
	51	ahbtbm_ctrl_bi.vhd	shyloc_123
	52	ahbtbm_ctrl_bsq.vhd	shyloc_123
	53	ccsds123_ahb_mst.vhd	shyloc_123
	54	async_fifo_write_ctrl.vhd	shyloc_123
	55	async_fifo_read_ctrl.vhd	shyloc_123
	56	async_fifo_synchronizer_g.vhd	shyloc_123
	57	async_fifo_ctrl.vhd	shyloc_123
	58	async_fifo.vhd	shyloc_123
	59	ccsds_fsm_shyloc_bip.vhd	shyloc_123
	60	ccsds_fsm_shyloc_bip_mem.vhd	shyloc_123
	61	ccsds_fsm_shyloc_bil.vhd	shyloc_123
	62	ccsds_fsm_shyloc_bil_mem.vhd	shyloc_123
	63	ccsds_fsm_shyloc_bsq.vhd	shyloc_123
	64	ccsds123_clk_adapt.vhd	shyloc_123
	65	ccsds123_config_core.vhd	shyloc_123
	66	ccsds123_dispatcher.vhd	shyloc_123
	67	mult_acc_shyloc.vhd	shyloc_123
	68	ccsds_comp_shyloc_bip.vhd	shyloc_123
	69	ccsds_comp_shyloc_bip_mem.vhd	shyloc_123
	70	ccsds_comp_shyloc_bil.vhd	shyloc_123
	71	ccsds_comp_shyloc_bil_mem.vhd	shyloc_123
	72	ccsds_comp_shyloc_bsq.vhd	shyloc_123
	73	predictor_shyloc.vhd	shyloc_123
	74	ccsds123_top.vhd	shyloc_123
$121DB/src/post_syn	75*	ccsds121_top_wrapper.vhd	post_syn_lib

(*) This file is just a wrapper that flattens the records present in the I/O ports and instantiates the top module ccsds121_shyloc_top.vhd. It is not actually part of the design, but it remains in the database, because it aids the preparation of post-synthesis or post-PAR simulations.

5.6 CSDS123 IP VHDL operational description

5.6.1 Compile-time configuration

The compile-time options are selected by editing the file "ccsds123_parameters.vhd", and assigning the desired values to the constants included. This file can be automatically generated by editing an appropriately formatted *.csv file and running a python script as described in Section 5.8, and the selected constants are propagated to all the components in the design. More information about the meaning of the compile-time configuration values can be found in [RD-1].

TABLE 5-2: CCSDS123 IP – COMPILE-TIME CONFIGURATION CONSTANTS IN THE CCSDS123_PARAMETERS.VHD FILE

Constant	Allowed values	Description
EN_RUNCFG	[0,1]	(0) Disables runtime configuration. (1) Enables runtime configuration.
RESET_TYPE	[0,1]	(0) Asynchronous reset. (1) Synchronous reset.
EDAC	[0,1, 2, 3]	(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.
PREDICTION_TYPE	[0,1,2,3,4]	(0) BIP-base architecture. (1) BIP-mem architecture. (2) BSQ architecture. (3) BIL-base architecture. (4) BIL-mem architecture.
ENCODING_TYPE	[0,1]	(0) Only pre-processor is implemented (external encoder can be attached). (1) Sample-adaptive encoder implemented.
HSINDEX_123	[0 – NAHBSLV- 1]	AHB slave index.
HSCONFIGADDR_123	[16#FFF#]	ADDR field of the AHB slave. Sets the 12 most significant bits in the 32-bit AHB address.
HSADDRMASK_123	[16#FF0#]	MASK field of the AHB slave.
HMINDEX_123	[0 - NAHBMST-1]	AHB master index.

HMAXBURST_123	[0:16]	AHB master burst beat limit.
ExtMemAddress_GEN	[0-16#FFF#]	External memory address. Sets the 12 most significant bits in the 32-bit AHB address.
Nx_GEN	[1 - 2^{16}-1]	Maximum allowed number of samples in a line.
Ny_GEN	[1 - 2^{16}-1]	Maximum allowed number of samples in a row.
Nz_GEN	[1 - 2^{16}-1]	Maximum allowed number of bands.
D_GEN	[2 - 16]	Maximum dynamic range of the input samples.
IS_SIGNED_GEN	[0,1]	(0) Unsigned samples. (1) Signed samples.
ENDIANESS_GEN	[0,1]	(0) Little-Endian. (1) Big-Endian.
DISABLE_HEADER_GEN	[0,1]	Selects whether to disable (1) or not (0) the header.
P_MAX	[0 - 15]	Number of bands used for prediction.
PREDICTION_GEN	[0,1]	Full (0) or reduced (1) prediction.
LOCAL_SUM_GEN	[0,1]	Neighbour (0) or column (1) oriented local sum.
OMEGA_GEN	[4 - 19]	Weight component resolution.
R_GEN	[max(32, D_GEN + OMEGA_GEN + 2) - 64]	Register size.
VMAX_GEN	[VMIN - 9]	Factor for weight update.
VMIN_GEN	[-6 - VMAX]	Factor for weight update.
T_INC_GEN	[4 - 11]	Weight update factor change interval.
WEIGHT_INIT_GEN	0	Weight initialization mode.
ENCODER_SELECTION_GEN	[0,1,2]	(0) Disables encoding. (1) Selects sample-adaptive coder. (2) Selects external encoder (Block-Adaptive).
INIT_COUNT_E_GEN	[1-8]	Initial count exponent.
ACC_INIT_TYPE_GEN	[0,1]	Accumulator initialization type.
ACC_INIT_CONST_GEN	[0 – (D_GEN – 2)]	Accumulator initialization constant.
RESC_COUNT_SIZE_GEN	[max(4, INIT_COUNT_E_GEN + 1) - 9]	Rescaling counter size.
U_MAX_GEN	[8 - 32]	Unary length limit.
W_BUFFER_GEN	[8,16,24,32,40,48,56,64] ≥ (U_MAX_GEN + D_GEN)	Bit width of the output buffer.
Q_GEN	[3:16]	Weight initialization resolution.

CWI_GEN	(0) different weight vectors for each band; (1) same weight vector for all the bands	Custom weight initialization mode.

5.6.2 Run-time configuration

Runtime configuration is performed by writing in the memory-mapped registers. The allowed values depend on the selected compile-time configuration and the standard, as specified in the [RD-1].

In the provided simulation environment, the run-time configuration values are selected by setting the testbench constants provided in the "ccsds123_tb_parameters.vhd", as detailed in Section 5.7. The "ccsds123_tb_parameters.vhd" file can be automatically generated by using the provided *.csv file and the scripts described in Section 5.8.

5.7 CCSDS123 IP Testbench

The provided testbench is designed following the Verification and Validation plan [AD-3]. The testbench performs the compression of an image and checks if the results are correct by comparing with a reference image. The VHDL sources listed in Table 5-3 are used as part of the testbench. In that table, {*TestId*} is a unique test identifier that defines the folder in which the testbench configuration file (ccsds123_tb_parameters.vhd) and the compile-time configuration file (ccsds123_parameters.vhd) is stored.

TABLE 5-3: CCSDS123 IP – TESTBENCH VHDL SOURCES

Folder	Compile order	Filename	Library
$123DB/modelsim/tb_stimuli/{*TestId*}	75	ccsds123_tb_parameters.vhd	work
$123DB/tb	76	ahbtbp.vhd	work
	77	ahbctrl.vhd	work
	78	ahbtbs.vhd	work
	79	ahbtbm.vhd	work
	80	ccsds_ahbtbp.vhd	work
	81	ccsds_shyloc_tb.vhd	work

Additionally, the following components from the GRLIB library are instantiated by the testbench:

TABLE 5-4: CCSDS123 IP – TESTBENCH GRLIB COMPONENTS

GRLIB component	Description

ahbctrl	AHB controller
ahbtbm	AHB master
ahbtbs	Memory slave

The testbench will first write the configuration values in the memory-mapped registers. The "ccsds123_tb_parameters.vhd" is used to configure those values, as detailed in Table 5.5, and it might be automatically generated by using the configuration script described in Section 5.8.

TABLE 5-5: CCSDS123 IP – RUN-TIME CONFIGURATION VALUES IN THE CCSDS123_TB_PARAMTERS.VHD FILE

Constant	Type	Allowed values	Description
stim_file	string	*Stimuli_file*	Stimuli file with the samples to compress.
ref_file	string	*Reference file*	Reference file with samples already compressed.
out_file	string	*Output file*	Compressed file obtained with the CCSDS-121 IP Core.
test_id	integer	[0,2,4,5,9,10,62,63,67,80,83]	Select the behaviour of the testbench.
ExtMemAddress_G_tb	integer	0-16#FFF#	External memory address. Sets the 12 most significant bits in the 32-bit AHB address.
Nx_tb	integer	[1 - Nx_GEN]	Number of columns.
Ny_tb	integer	[1 - Ny_GEN]	Number of lines.
Nz_tb	integer	[1 - Nz_GEN]	Number of bands.
DISABLE_HEADER_tb	integer	[0,1]	Selects whether to disable (1) or not (0) the header generation.
ENCODER_SELECTION_tb	integer	[0,1,2]	(0) Disables encoding. (1) Selects sample-adaptive coder. (2) Selects external encoder (block-adaptive).
D_tb	integer	[2 – D_GEN]	Dynamic range of the input samples.
IS_SIGNED_tb	integer	[0,1]	(0) Unsigned samples. (1) Signed samples.
ENDIANESS_tb	integer	[0,1]	Endianness of the input samples: (0) Little-Endian. (1) Big-Endian.
BYPASS_tb	integer	[0,1]	(0) Compression. (1) Bypass Compression.
P_tb	integer	[0 - P_MAX]	Number of bands used for prediction.

PREDICTION_tb	integer	[0,1]	Full (0) or reduced (1) mode.
LOCAL_SUM_tb	integer	[0,1]	Neighbour (0) or column (1) oriented local sum.
OMEGA_tb	integer	[4 – OMEGA_GEN]	Weight component resolution.
R_tb	integer	[max(32, D_GEN + OMEGA_GEN + 2) – R_GEN]	Register size.
VMAX_tb	integer	[VMIN – VMAX_GEN]	Factor for weight update.
VMIN_tb	integer	[VMIN_GEN - VMAX]	Factor for weight update.
TINC_tb	integer	[4 – T_INC_GEN]	Weight update factor change interval.
WEIGHT_INIT_tb	integer	0	Weight initialization mode.
INIT_COUNT_E_tb	integer	[1 – INIT_COUNT_E_GEN]	Initial count exponent.
ACC_INIT_TYPE_tb	integer	[0,1]	Accumulator initialization type.
ACC_INIT_CONST_tb	integer	[0 – ACC_INIT_CONST_GEN]	Accumulator initialization constant.
RESC_COUNT_SIZE_tb	integer	[max(4, INIT_COUNT_E_GEN + 1) – ACC_INIT_CONST_GEN]	Rescaling counter size.
U_MAX_tb	integer	[8 – U_MAX_GEN]	Unary length limit.
W_BUFFER_tb	integer	[8,16,24,32,40,48,56,64] ≤ W_BUFFER_GEN	Bit width of the output buffer.
Q_tb	integer	[0, 3:OMEGA_tb+3]	Weight initialization resolution.
WR_tb	integer	[0,1]	Reset of custom weight vectors.

5.7.1 Testbench behavioural description

The behaviour of the testbench can be determined by setting the constant “test_id”. Table 5-6 shows the possible “test_id” values and the behaviour of the testbench for each case.

TABLE 5-6: CCSDS123 IP TESTBENCH – TEST SEQUENCES

TestId	Description
0	1. Write run-time configuration values in memory-mapped registers if required. 2. Send raw samples to the CCSDS123 IP core. 3. Monitor output of the CCSDS123 IP core and compare with reference file when valid compressed data are produced. 4. Wait for CCSDS123 IP to signal that it has finished the compression and is ready to receive a new configuration. 5. Repeat procedures from step 1 to 4.
2	1. Write run-time configuration values in memory-mapped registers if required. 2. Send raw samples to the CCSDS123 IP core.

	3. Monitor output of the CCSDS123 IP core and compare with reference file when valid compressed data are produced. 4. Send configuration while the CCSDS123 IP core is compressing. 5. Wait for CCSDS123 IP to signal that it has finished the compression and is ready to receive a new configuration (configuration sent during compression has been ignored). 6. Repeat procedures from step 1 to 5.
4	1. Write run-time configuration values in memory-mapped registers if required. 2. Send some raw samples to the CCSDS123 IP core. 3. Activate ForceStop signal. 4. Wait for CCSDS123 IP to signal that it has finished the compression (because of the ForceStop) and is ready to receive a new configuration. 5. Write run-time configuration values in memory-mapped registers for the second time. 6. Send raw samples to the CCSDS123 IP core. 7. Monitor output of the CCSDS123 IP core and compare with reference file when valid compressed data are produced. 8. Wait for CCSDS123 IP to signal that it has finished the compression and is ready to receive a new configuration.
5	1. Write run-time configuration values in memory-mapped registers if required. 2. Send raw samples to the CCSDS123 IP core. 3. Monitor output of the CCSDS123 IP core and compare with reference file when valid compressed data are produced. 4. Wait for CCSDS123 IP to signal that it has finished the compression and is ready to receive a new configuration. 5. Repeat procedures from step 1 to 4 with different configuration and different stimuli and reference files.
9	1. Write invalid run-time configuration values in memory-mapped registers if required. 2. Wait for CCSDS123 IP to signal that there has been a configuration error, and it has finished the compression and is ready to receive a new configuration. 3. Write valid run-time configuration values in memory-mapped registers for the second time. 4. Send raw samples to the CCSDS123 IP core. 5. Monitor output of the CCSDS123 IP core and compare with reference file when valid compressed data are produced. 6. Wait for CCSDS123 IP to signal that it has finished the compression and is ready to receive a new configuration.

10	1. Run test_id = 4 2. Run test_id = 9 3. Run test_id = 0
62	1. Write run-time configuration values in memory-mapped registers if required. 2. Send some raw samples to the CCSDS123 IP core. 3. Monitor output of the CCSDS123 IP core and compare with reference file when valid compressed data are produced. 4. Wait for CCSDS123 IP to signal that there was an error during the compression. 5. Repeat procedures from 1 to 4.
63	1. Write run-time configuration values in memory-mapped registers if required. 2. Send raw samples to the CCSDS123 IP even if the IP core is not ready to receive them. 3. Monitor FIFO_Full signal, wait for its assertion, and finish.
67	1. Write run-time configuration values in memory-mapped registers if required. 2. Send raw samples to the CCSDS123 IP core every 4 cycles. 3. Monitor output of the CCSDS123 IP core and compare with reference file when valid compressed data are produced. 4. Wait for CCSDS123 IP to signal that it has finished the compression and is ready to receive a new configuration. 5. Repeat procedures from step 1 to 4.
80	1. Write run-time configuration values in memory-mapped registers if required. 2. Send some raw samples to the CCSDS123 IP core. 3. Deactivate Ready_Ext signal for a few clock cycles during compression in order to halt the process. 4. Reactivate Ready_Ext to resume execution. 5. Wait for CCSDS123 IP to signal that it has finished the compression and is ready to receive a new configuration. 6. Compare the output of the CCSDS123 IP core with reference file.
83	1. Write run-time configuration values in memory-mapped registers if required. 2. Send some raw samples to the CCSDS123 IP core. 3. Block transactions through the AHB bus (by deactivating the HGRANT signal) for several clock cycles.

	4. Resume execution. 5. Repeat steps 3 and 4. 6. Wait for CCSDS123 IP to signal that it has finished the compression and is ready to receive a new configuration. 7. Compare the output of the CCSDS123 IP core with reference file.

5.7.2 Testbench assertions

The following assertions are included in the testbench:

TABLE 5-7: CCSDS123 IP TESTBENCH – ASSERTIONS

Report	Severity	Description
Finished started with a high value	warning	Checks correct value of control signals after reset of the IP core.
Ready started with a high value	warning	
AwaitingConfig started with a low value	warning	
Sending a new configuration	note	Configuration registers are sent to the AHB bus.
AwaitingConfig lowered correctly when configuration was received	note	Configuration has been correctly received by the IP core.
Ready asserted correctly when IP core is ready to receive new samples	note	IP core is ready to receive samples.
Ready not asserted correctly when IP core has been configured	warning	IP core has been configured, but has not signalled that it is ready to receive samples.
Unexpected IP core error during compression	error	Unexpected error during compression.
Unexpected IP core error after compression	error	Unexpected error after compression.
Finished correctly activated when compression finished	note	IP core has finished the compression.
AwaitingConfig correctly activated after compression finished	note	IP core is waiting to be sent a new configuration after a previous configuration operation finished.
Error between sequential compressions, value of Finished shall be kept high	error	Incorrect Finished signal value after a configuration operation has finished.
Error for sequential compressions, finished shall be de-asserted with AwaitingConfig	error	Incorrect Finished signal value after the configuration is received between sequential compressions.
Two sequential compressions test performed	note	Test finished correctly when test_id = 0
One compression test performed	note	Test finished correctly when test_id = 0 and only one compression is performed.
Attempt to enable compressor without sending the necessary configuration	warning	IP core has not received necessary configuration values.

Attempt to send new configuration during compression	note	Attempt to send new configuration during compression.
Expected IP core AHB error during compression" severity note	note	Informs that the AHB error was correctly produced when test_id = 62.
FIFO is full, as expected	note	Informs that the FIFO full was correctly asserted when test_id = 63.
One compression (attempting to reconfigure) test performed	note	Test finished correctly when test_id = 2.
ForceStop assertion	note	ForceStop signal asserted by the testbench.
Finished correctly activated after ForceStop	note	Correct value in Finished and AwaitingConfig signals after ForceStop assertion.
AwaitingConfig asserted correctly after ForceStop	note	
ForceStop and one compression test performed	note	Test finished correctly when test_id = 4.
Two different compressions test performed	note	Test finished correctly when test_id = 5.
Sending invalid configuration...	note	The testbench is sending configuration registers with invalid values through the AHB bus.
AwaitingConfig lowered correctly when configuration was received (even with error)	note	Correct value in AwaitingConfig signal after the configuration was received.
Error has been correctly asserted	note	Correct value in Error signal after invalid configuration values were sent.
Finished has been correctly asserted after an error	note	Correct values in Finished and AwaitingConfig after a configuration error was signalled by the IP.
AwaitingConfig asserted correctly after an error	note	
Error has not been correctly asserted when error	error	Incorrect values in Error and Finished signals after a configuration error was signalled by the IP.
Finished has not been correctly asserted after an error	error	
Configuration error and one compression test performed	note	Test finished correctly when test_id =7.
Comparison was successfull!	note	Compressed output file and reference file are identical.
Comparison not possible because there has been a ForceStop assertion	note	Compressed output file and reference file cannot be compared.
Comparison not possible because there has not been compression performed (configuration error)	note	
Problems in final stream	error	Comparison between reference and compressed file was not successful (IP core output file and reference file are not identical)
Reference file has more samples	error	
Output file has more samples	error	
****** CCSDS-123 Testbench done ******	note	Testbench has reached the end.

5.8 CCSDS123 IP simulation and synthesis scripts

The information in this section is provided to allow a user to understand how the configuration of the IP core, the testbench or the synthesis can be performed. All the procedures explained below are automated in the provided makefile described in Section 5.4.

A python script is provided in order to ease the generation of the necessary configuration files for synthesis or simulation. Such script reads the configuration values from a *.csv file with a specific format (see Section 5.8.1) and generates the necessary *.vhd files that are used to configure the IP core and the testbench.

5.8.1 Writing the desired parameters in the *.csv file

The configuration *.csv file is used to generate the necessary configuration files for simulation or synthesis. It is structured as follows:

- Line1: Comma-separated row identifiers.
- Line2: Comma-separated parameter names.
- All other lines: Comma-separated configuration values (one set of configuration values per line).

The *.csv file is used to set the necessary parameters of the CCSDS123-IP and the parameters of the CCSDS121 IP when it is instantiated as external entropy coder.

The meaning of each row in the *.csv file is explained in Table 4-8, where the "Mandatory" field has the following meaning:

- Y: parameter is mandatory.
- SMR: parameter is mandatory for simulation only if runtime configuration is enabled (EN_RUNCFG =1).
- SM121: parameter is mandatory for simulation only if the CCSDS121 IP is instantiated as external encoder.
- SM: parameter is mandatory for simulation.
- N: parameter is not mandatory.

TABLE 5-8: CCSDS123 IP CONTENT OF THE *.CSV CONFIGURATION FILE.

Row Identifier	Parameter name	Description	Mandatory	Written in
0	TestId	Unique test identifier.	Y	ccsds123_tb_parameters.vhd
1	ImageId	Image identifier.	N	N/A*
2	Input Image	Filename of the input image to compress.	SM	N/A
3	Ny	Number of lines.	SM	ccsds123_tb_parameters.vhd and ccsds121_tb_parameters.vhd (when using CCSDS121 IP as external encoder, i.e. ENCODER_SELECTION = 2)

4	Nx	Number of columns.	SM	ccsds123_tb_parameters.vhd and ccsds121_tb_parameters.vhd (when using CCSDS121 IP as external encoder, i.e. ENCODER_SELECTION = 2)
5	Nz	Number of bands.	SM	ccsds123_tb_parameters.vhd and ccsds121_tb_parameters.vhd (when using CCSDS121 IP as external encoder, i.e. ENCODER_SELECTION = 2)
6	D	Dynamic range of the input samples.	SM	ccsds123_tb_parameters.vhd and ccsds121_tb_parameters.vhd (when using CCSDS121 IP as external encoder, i.e. ENCODER_SELECTION = 2)
7	IS_SIGNED	(1) Signed samples. (0) Unsigned samples.	SM	ccsds123_tb_parameters.vhd
8	ENDIANESS	(0) Little Endian. (1) Big Endian.	SM	ccsds123_tb_parameters.vhd
9	PREDICTION_TYPE	Selects which architecture to implement according to the sample arrangement: (0) BIP (1) BIP-MEM (2) BSQ (3) BIL (4) BIL-MEM	Y	ccsds123_parameters.vhd and ccsds123_tb_parameters.vhd
10	Parameter set	Identifier of the selected set of parameters.	N	N/A
11	EN_RUNCFG	(1) Enables run-time configuration. (0) Disables run-time configuration.	Y	ccsds123_parameters.vhd
12	RESET_TYPE	(1) Synchronous reset. (0) Asynchronous reset.	Y	ccsds123_parameters.vhd
13	EDAC	(0) Inhibits EDAC implementation. (1) EDAC is implemented.	Y	ccsds123_parameters.vhd
14	RESERVED	Reserved	N	N/A
15	ENCODING_TYPE	(0) Only pre-processor is implemented (external encoder can be attached). (1) Sample-adaptive encoder implemented.	Y	ccsds123_parameters.vhd
16	DISABLE_HEADER_GEN	Selects whether to disable (1) or not (0) the header.	Y	ccsds123_parameters.vhd
17	W_BUFFER_GEN	Bit width of the output buffer.	Y	ccsds123_parameters.vhd

18	P_MAX	Number of bands used for prediction.	Y	ccsds123_parameters.vhd
19	PREDICTION_GEN	Full (0) or reduced (1) prediction.	Y	ccsds123_parameters.vhd
20	LOCAL_SUM_GEN	Neighbour (0) or column (1) oriented local sum.	Y	ccsds123_parameters.vhd
21	OMEGA_GEN	Weight component resolution.	Y	ccsds123_parameters.vhd
22	R_GEN	Register size.	Y	ccsds123_parameters.vhd
23	VMAX_GEN	Factor for weight update.	Y	ccsds123_parameters.vhd
24	VMIN_GEN	Factor for weight update.	Y	ccsds123_parameters.vhd
25	T_INC_GEN	Weight update factor change interval.	Y	ccsds123_parameters.vhd
26	RESERVED	Reserved.	N	N/A
27	RESERVED	Reserved.	N	N/A
28	INIT_COUNT_E_GEN	Initial count exponent.	Y	ccsds123_parameters.vhd
29	ACC_INIT_TYPE_GEN	Accumulator initialization type.	Y	ccsds123_parameters.vhd
30	ACC_INIT_CONST_GEN	Accumulator initialization constant.	Y	ccsds123_parameters.vhd
31	RESC_COUNT_SIZE_GEN	Rescaling counter size.	Y	ccsds123_parameters.vhd
32	U_MAX_GEN	Unary length limit.	Y	ccsds123_parameters.vhd
33	RESERVED	Reserved.	N	N/A
34	DISABLE_HEADER	Selects whether to disable (1) or not (0) the header generation.	SMR	ccsds123_tb_parameters.vhd
35	ENCODER_SELECTION	(0) Disables encoding. (1) Selects sample-adaptive coder. (2) Selects external encoder (block-adaptive).	SMR	ccsds123_tb_parameters.vhd
36	BYPASS	(0) Compression. (1) Bypass Compression.	SMR	ccsds123_tb_parameters.vhd
37	P	Number of bands used for prediction.	SMR	ccsds123_tb_parameters.vhd
38	PREDICTION	Full (0) or reduced (1) mode.	SMR	ccsds123_tb_parameters.vhd
39	LOCAL_SUM	Neighbour (0) or column (1) oriented local sum.	SMR	ccsds123_tb_parameters.vhd

40	OMEGA	Weight component resolution.	SMR	ccsds123_tb_parameters.vhd
41	R	Register size.	SMR	ccsds123_tb_parameters.vhd
42	VMAX	Factor for weight update.	SMR	ccsds123_tb_parameters.vhd
43	VMIN	Factor for weight update.	SMR	ccsds123_tb_parameters.vhd
44	T_INC	Weight update factor change interval.	SMR	ccsds123_tb_parameters.vhd
45	RESERVED	Reserved.	N	N/A
46	RESERVED	Reserved.	N	N/A
47	INIT_COUNT_E	Initial count exponent.	SMR	ccsds123_tb_parameters.vhd
48	ACC_INIT_TYPE	Accumulator initialization type.	SMR	ccsds123_tb_parameters.vhd
49	ACC_INIT_CONST	Accumulator initialization constant.	SMR	ccsds123_tb_parameters.vhd
50	RESC_COUNT_SIZE	Rescaling counter size.	SMR	ccsds123_tb_parameters.vhd
51	U_MAX	Unary length limit.	SMR	ccsds123_tb_parameters.vhd
52	RESERVED	Reserved.	N	N/A
53	W_BUFFER	Bit width of the output buffer.	SMR	ccsds123_tb_parameters.vhd
54	Ny_GEN	Maximum number of lines.	Y**	ccsds123_parameters.vhd
55	Nx_GEN	Maximum number of columns.	Y**	ccsds123_parameters.vhd
56	Nz_GEN	Maximum number of bands.	Y**	ccsds123_parameters.vhd
57	D_GEN	Maximum dynamic range.	Y**	ccsds123_parameters.vhd
58	IS_SIGNED_GEN	(1) Signed samples. (0) unsigned samples.	Y**	ccsds123_parameters.vhd
59	ENDIANESS_GEN	(0) Little Endian. (1) Big Endian.	Y**	ccsds123_parameters.vhd
60	RESERVED	Reserved.	N	N/A
61	ENCODER_SELECTION_GEN	(0) Disables encoding. (1) Selects sample-adaptive coder. (2) Selects external encoder (block-adaptive).	Y	ccsds123_parameters.vhd

62	output	Output file name to obtain the reference image name.	SM	ccsds123_tb_parameters.vhd
63	EXT_MEM_ADDRESS	External memory address.	Y	ccsds123_parameters.vhd and ccsds123_tb_parameters.vhd
64	Ny_GEN121	Ny_GEN in the CCSDS121 IP core.	SM121	ccsds121_parameters.vhd
65	Nx_GEN121	Nx_GEN in the CCSDS121 IP core.	SM121	ccsds121_parameters.vhd
66	Nz_GEN121	Nz_GEN in the CCSDS121 IP core.	SM121	ccsds121_parameters.vhd
67	J_GEN	J_GEN in the CCSDS121 IP core.	SM121	ccsds121_parameters.vhd
68	CODESET_GEN	CODESET_GEN in the CCSDS121 IP core.	SM121	ccsds121_parameters.vhd
69	REF_SAMPLE_GEN	REF_SAMPLE_GEN in the CCSDS121 IP core.	SM121	ccsds121_parameters.vhd
70	W_BUFFER_GEN121	W_BUFFER_GEN in the CCSDS121 IP core.	SM121	ccsds121_parameters.vhd
71	J	Block size configuration (J) in the CCSDS121 IP core.	SM121	ccsds121_tb_parameters.vhd
72	CODESET	Codeset configuration in the CCSDS121 IP core.	SM121	ccsds121_tb_parameters.vhd
73	REF_SAMPLE	Reference sample configuration in the CCSDS121 IP core.	SM121	ccsds121_tb_parameters.vhd
74	W_BUFFER121	Output buffer size in the CCSDS121 IP core.	SM121	ccsds121_tb_parameters.vhd
75	RESERVED	Reserved.	N	N/A
76	HMAXBURST	Burst size when PREDICITION_TYPE = 1, 4.	SM	ccsds123_tb_parameters.vhd

** N/A means not applicable.*
*** When EN_RUNCFG = 0 (runtime configuration disabled), the compile-time parameters used to set the image size (Nx_GEN, Ny_GEN, Nz_GEN), dynamic range (D_GEN), endianness (ENDIANESS_GEN) and sign (IS_SIGNED_GEN) have to be set according to the actual size, dynamic range, endianness and sign of the image to be compressed. The python script will check the rules in* Table 4-9 *and raise an exception if the rules are not met.*

TABLE 5-9: CCSDS121 IP - RULES TO FILL THE *.CSV FILE WHEN EN_RUNCFG = 0

Configuration rule	Correspondence in *.csv file
Ny_GEN = Ny	row[54] = row[3]
Nx_GEN = Nx	row[55] = row[4]
Nz_GEN = Nz	row[56] = row[5]
D_GEN = D	row[57] = row[6]
IS_SIGNED_GEN = IS_SIGNED	row[58] = row[7]
ENDIANESS_GEN = ENDIANESS	row[59] = row[8]

5.8.2 Simulation

5.8.2.1 Generating parameters files and scripts for simulation

Follow these steps in order to generate the parameter files and scripts for simulation:

1. Fill the *.csv file with the desired configuration values and names of the raw image to be compressed and reference image for verification as explained in Section 5.8.1.

2. Ensure that the raw images to be compressed and reference images for comparison are located in the $123DB/images/raw folders and $123DB/images/reference (the script will notice missing files and will not include test cases with missing files for simulation).

3. Run the script $123DB/verification_scripts/run_vhdl_tests_123.py from a terminal, with the following arguments:

```
$ $123DB/verification_scripts/run_vhdl_tests_123.py $CSV_FILE $RAW_FOLDER $COMPRESSED_FOLDER $
REFERENCE_FOLDER $DATABASE123_DIRECTORY $DATABASE121_DIRECTORY $OPTION
```

TABLE 5-10: CCSDS123 IP – ARGUMENTS OF THE RUN_VHDL_TESTS_123.PY SCRIPT FOR SIMULATION

Argument	Meaning	Example
$CSV_FILE	*.csv file used to generate the configuration files.	$123DB/verification_scripts/testcases_123.csv
$RAW_FOLDER	Path to the folder that contains the raw images to be compressed	$123DB/images/raw
$COMPRESSED_FOLDER	Path to the folder where the compressed images will be stored	$123DB/images/compressed
$REFERENCE_FOLDER	Path to the folder that contains the reference images for comparison.	$123DB/images/reference
$DATABASE123_DIRECTORY	Path to the database root directory of the CCSDS123 IP core.	$123DB/
$DATABASE121_DIRECTORY	Path to the database root directory of the CCSDS123 IP core.	$121DB/
$OPTION	If ($OPTION = modelsim), it generates the necessary scripts for simulation.	N/A

The script will generate the files listed in Table 5-11.

TABLE 5-11: CCSDS123 IP – LIST OF FILES GENERATED BY THE PYTHON SCRIPT

Folder	Filename	Description

$123DB/modelsim/tb_stimuli/{*TestId*}	ccsds123_parameters.vhd	File containing generic parameters. {TestId} is the unique test identifier (row[0] of the *.csv file).
	ccsds123_tb_parameters.vhd	File containing run-time configuration values used by the testbench.
	ccsds121_parameters.vhd	File containing generic parameters for the CCSDS121 IP core (only generated when using CCSDS121 IP as external encoder, i.e. ENCODER_SELECTION = 2)
	ccsds121_tb_parameters.vhd	File containing run-time configuration values used by the testbench for the CCSDS121 IP core (only generated when using CCSDS121 IP as external encoder, i.e. ENCODER_SELECTION = 2)
$123DB/modelsim/tb_scripts	testbench.do	Script file to compile the testbench and necessary amba files for the test.
	ip_core.do	Script to compile VHDL sources of the CCSDS123 IP core. This file includes corresponding flags to enable code coverage in the compilation command.
	ip_core_block.do	Script to compile VHDL sources of the CCSDS121 IP core (the file is only generated when using CCSDS121 IP as external encoder, i.e. ENCODER_SELECTION = 2). This file includes corresponding flags to enable code coverage in the compilation command.
	{*Test_Id*}.do	Script to perform each simulation test case. This script covers the elimination of existing libraries and its creation, compilation of the corresponding parameters file, calling previous scripts and finally starting simulation.
	all_tests.do	Script to manage simulation of all the individual simulation scripts. This script manages the coverage merge from the successful simulations performed.
	verification_report_not_performed.txt	File containing test cases impossible to perform, since the input raw file to be compressed or the reference compressed file have not been found. This file will not be generated if all the test cases are successful.

5.8.2.2 Running the testbench

After creating the necessary configuration scripts and *.do files, the test cases can be simulated in QuestaSim/Modelsim by following these steps:

1. Open Modelsim/Questasim.
2. Open the project $123DB/modelsim/shyloc123.mpf.
3. Run:
    a. A single test:
        i. Create variable to the location of the database directory of the CCSDS123 IP core:

        set SRC $123DB

        ii. If the CCSDS121 IP is used, then create variable to the location of the database directory of the CCSDS121 IP core:

        set SRC121 $121DB

        iii. do $123DB/modelsim/tb_scripts/{Test_Id}.do.
    b. All tests cases in the *.csv file: do $123DB/modelsim/tb_scripts/all_tests.do.

If the file all_tests.do is run, the following files are generated after completion of all test cases:

TABLE 5-12: CCSDS123 IP – LIST OF FILES GENERATED AFTER COMPLETION OF THE ALL_TESTS.DO FILE

Folder	Filename	Description
$123DB/modelsil/tb_scripts	verification_report.txt	File containing performed test cases and their result, in terms of PASSED or FAILED.
$123DB/modelsim/tb_stimuli/{TestId}	report_coverage.txt	If test passed, this file contains the report coverage for the specific simulation.
	report_coverage_details.txt	If test passed, this file contains the report coverage in detail for the specific simulation.
$123DB/modelsim/cover	*{test_id}*_Test_cover.ucdb	If test passed, this file contains the coverage dataset for the specific simulation.
	merged_result.ucdb	This file contains the merged coverage database from all the *{TestId}*_cover.ucdb files.
	merged_result.txt	This file contains the merged coverage report.

5.8.3 Synthesis

Synthesis can be performed by first generating the necessary configuration files and synthesis scripts and then running the aforementioned scripts in ISE Project Navigator, Synplify or NanoXmap for FPGA targets, or Synopsys Design Compiler for ASIC targets.

5.8.3.1 Generating parameters files and scripts for synthesis

Follow these steps in order to generate the parameter files and scripts for synthesis:

1. Fill the *.csv file with the desired configuration values.
2. Run the script $123DB/verification_scripts/run_vhdl_tests_123.py from a terminal, with the following arguments:

```
$ $  $123DB/verification_scripts/run_vhdl_tests_123.py $CSV_FILE $RAW_FOLDER $COMPRESSED_FOLDE
R $REFERENCE_FOLDER $DATABASE123_DIRECTORY $DATABASE121_DIRECTORY $OPTION
```

TABLE 5-13: CCSDS123 IP – ARGUMENTS OF THE RUN_VHDL_TESTS_123.PY SCRIPT FOR SYNTHESIS.

Argument	Meaning	Example
$CSV_FILE	.csv file used to generate the configuration files.	$123DB/verification_scripts/ synthesis_params_ 123_e.csv
$RAW_FOLDER	Path to the folder that contains the raw images to be compressed.*	$123DB/images/raw
$COMPRESSED_FOLDER	Path to the folder where the compressed images will be stored.*	$123DB/images/compressed
$REFERENCE_FOLDER	Path to the folder that contains the reference images for comparison. *	$123DB/images/reference
$DATABASE123_DIRECTORY	Path to the database root directory of the CCSDS123 IP core.	$123DB
$DATABASE121_DIRECTORY	Path to the database root directory of the CCSDS123 IP core.	$121DB (Not used)
$OPTION	If ($OPTION = ise or $OPTION = synplify or $OPTION = brave or $OPTION = dc), it generates the necessary scripts for synthesis with the selected tool.	N/A

* *The script requires the paths to be present, however when $OPTION = ise or $OPTION = synplify or $OPTION = brave or $OPTION = dc, the folders are not used.*

The script will generate the files listed in Table 4-14.

TABLE 5-14: CCSDS123 IP– LIST OF FILES GENERATED AFTER RUNNING THE PYTHON SCRIPT WITH $OPTION = SYNPLIFY, ISE, BRAVE OR DC.

Folder	Filename	Description
$123DB/synthesis/premap_parameters /{*TestId*}	ccsds123_parameters.vhd	File containing generic parameters.
$123DB/synthesis/syn_scripts/ise **when $OPTION = ise** **$123DB/synthesis/syn_scripts/synplify** **when $OPTION = synplify** **$123DB/synthesis/syn_scripts/brave** **when $OPTION = brave** **$123DB/synthesis/syn_scripts/dc** **when $OPTION = dc**	{*TestId* }.tcl (for synthesis in ISE Navigator or Synplify) {*TestId* }.py (for synthesis in NanoXmap)	Script to perform each synthesis test case. This script covers the elimination of existing sources its addition, addition of the corresponding parameters file, calling other necessary scripts and finally launching the synthesis. {*TestId*} is the unique synthesis identifier (row[0] of the *.csv file).
	add_ip_core.tcl (for synthesis in ISE Navigator or Synplify) add_ip_core.py (for synthesis in NanoXmap)	Script to include VHDL sources of the CCSDS123 IP core linked to the proper library.
	all_ise.tcl (when $OPTION = ise) all_synplify.tcl (when $OPTION = synplify) all_nanoxplore.py (when $OPTION = brave) all_dc.tcl (when $OPTION = dc)	Script to create or just open the synthesis project and perform each individual synthesis script. This script saves the synthesis results in the proper folder.
Only **$121DB/synthesis/syn_scripts/dc** **when $OPTION = dc**	setupLibs.do	Setup the libraries, for Design Compiler elaboration

5.8.3.2 Running the synthesis for FPGA targets

After creating the necessary configuration scripts and *.tcl or *.py files, the synthesis can be performed in ISE, Synplify or NanoXmap by following these steps:

1. Open the synthesis tool.
2. Change directory to:
    a. In ISE:

        cd $123DB/synthesis/syn_scripts/ise

b. In Synplify:

cd $123DB/synthesis/syn_scripts/synplify

c. In NanoXmap:

cd $123DB/synthesis/syn_scripts/brave

3. Run:

a. A single synthesis process:

i. Create variable to the location of the database directory:

set SRC $123DB

ii. In ISE:

source $SRC/synthesis/syn_scripts/ise /{*TestId*}.tcl.

In Synplify:

run_tcl $SRC/synthesis/syn_scripts/synplify /{*TestId*}.tcl.

In NanoXmap:

nanoxpython $SRC/synthesis/syn_scripts/brave/{*TestId*}.py.

b. All synthesis cases in the *.csv file:

In ISE:

source $SRC/synthesis/syn_scripts/ise /all_ise.tcl.

In Synplify:

run_tcl $SRC/synthesis/syn_scripts/synplify /all_synplify.tcl.

In NanoXmap:

nanoxpython $SRC/synthesis/syn_scripts/brave/all_nanoxplore.py.

{*TestId*} is the unique synthesis identifier (row[0] of the *.csv file).

If the file all_{ise,synplify}.tcl or all_nanoxplore.py is run, the following files are generated after completion of all synthesis:

TABLE 5-15: CCSDS123 IP – LIST OF FILES GENERATED AFTER COMPLETION OF THE ALL_ISE.TCL, ALL_SYNPLIFY.TCL AND ALL_NANOXPLORE.PY FILE.

Folder	Filename	Description

$123DB/synthesis/syn_scripts/synplify/report **when ($OPTION = synplify)**	{*TestId*}_synplify.srr	Log file containing implementation results.
	{*TestId*}_synplify _fpga_mapper_timing_report.xml	File containing synthesis results in terms of timing.
	{*TestId*}_synplify _fpga_mapper_area_report.xml	File containing synthesis results in terms of area.
$123DB/synthesis/syn_scripts/brave/report **when ($OPTION = brave)**	general_{*TestId*}_{*Device*}.log	Log file containing implementation results, including timing and area.
$123DB/synthesis/syn_scripts/ise/report **when ($OPTION = ise)**	N/A*	N/A*

**Current implementation does not create copies of implementation results for ISE. The user can find them in their local ISE project directory.*

5.8.3.3 Running the synthesis for ASIC targets

The IP core can also be synthesized targeting ASIC standard cell libraries, through a set of scripts. The available flow is based on Synopsys Design Compiler. The result of the synthesis process is a set of files containing the netlist, and reports for area, timing, power and quality of results (QoR).

The execution of ASIC synthesis requires setting up the Standard Cell Libraries to be used, and may also require setting up technology-specific memories, for the FIFOs' SRAM memories.

5.8.3.3.1 Base Scripts and Files

The scripts are based on Synopsys Reference Methodology scripts, which can be used both for logical and/or physical synthesis. The default corner considered is Worst-Case Military (WCMIL).

The following table describes the available files, both the provided scripts and files generated by the synthesis process.

TABLE 5-16: CCSDS123 IP – LIST OF FILES PERTAINING TO ASIC SYNTHESIS

Folder	Filename	Description
$123DB/synthesis/syn_scripts/base	ccsds123_top.sdc	Constraint file, with the clock signals
	dc_setup.tcl	Set-up script, where the user has to specifiy the location of the Standard Cell libraries
	dc.tcl	Main Design Compiler script

In order to perform synthesis with a standard cell library, the dc_setup.tcl file has to be edited. The following table describes the variables which must be set in order to be able to perform logical synthesis.

TABLE 5-17: CCSDS123 IP – ASIC STANDARD CELL LIBRARY SETUP FOR LOGICAL SYNTHESIS

File	Variable	Description
$123DB/synthesis/syn_scripts/base/dc_setup.tcl	ADDITIONAL_SEARCH_PATH	Path to be added to the search path, e.g . Std. Cell Library based folder
	TARGET_LIBRARY_FILES	Target technology logical libraries, e.g. core cells
	ADDITIONAL_LINK_LIB_FILES	Extra link logical libraries not included in TARGET_LIBRARY_FILES, e.g. corners

Additionally, Milkyway libraries may be defined in order to perform physical synthesis. These must be set in the same file, and the variables are below the ones for logical synthesis.

5.8.3.3.2 Synthesis Execution

The execution of the synthesis scripts is performed through the Makefile available at the IP base folder ($123DB). In order to run, the user has to first edit the dc_setup.tcl in order to setup the standard cell library environment. When set, the user can issue the command “make dc” and the targets defined in the CSV file will be synthesised.

5.8.3.3.3 Generated Output Files

The synthesis process generates several files, with reports and design descriptions which can be used in simulation or analysis, e.g. Verilog netlist. The following table describes the generated files.

TABLE 5-18: CCSDS123 IP – LIST OF FILES GENERATED BY ASIC SYNTHESIS

Folder	Filename	Description
$123DB/synthesis/syn_scripts/dc/report/{*Id*}	ccsds123_top.check_design.rpt	Design rule check, with internal lint
	ccsds123_top.mapped.area.rpt	Area usage report
	ccsds123_top.mapped.clock_gating.rpt	Clock gating report, not applicable but part of the reference methodology
	ccsds123_top.mapped.power.rpt	Power consumption report
	ccsds123_top.mapped.qor.rpt	Quality of results, and overall summary, including slack from static timing analysis.
	ccsds123_top.mapped.timing.rpt	Short report, with one path per clock
$123DB/synthesis/syn_scripts/dc/results/{*Id*}	ccsds123_top.elab.ddc	Elaborated (Generic Cells) design database, to be used with Design Compiler

	ccsds123_top.mapped.ddc	Mapped (Standard Library Cells) design database, to be used with Design Compiler
	ccsds123_top.mapped.sdc	Mapped design constraint file
	ccsds123_top.mapped.sdf	Mapped design delay file, for timing analysis and simulation
	ccsds123_top.mapped.svf	Mapped design information for formal equivalence checking
	ccsds123_top.mapped.v	Mapped design Verilog netlist
	elab	Temporary folder with the elaborated design units
	ICC2_files	Files for IC-Compiler II back-end

5.8.4 Post-synthesis simulations

Running the post-synthesis simulations has not been automated in the provided *makefile.* However, the python script offers the possibility to configure the IP and generate the necessary configuration scripts that enable the generation of the post-synthesis model with Synplify. Instructions are provided below.

5.8.4.1 Generating the post-synthesis model

Follow the procedures in Section 5.8.3 with the following exceptions:

When running the python script, using the command line:

```
$ $123DB/verification_scripts/run_vhdl_tests_123.py $CSV_FILE $RAW_FOLDER $COMPRESSED_FOLDER $
REFERENCE_FOLDER $DATABASE123_DIRECTORY $DATABASE121_DIRECTORY $OPTION $TECHNOLOGY
```

- use $OPTION = synplify-ps.
- The user is offered the possibility of selecting the target technology with the argument $TECHNOLOGY. The following values are accepted {XC5VFX130T, XQR5VFX130, A3PE3000, RTAX4000S, RT4G4150}. If the user does not specify a target technology, the synthesis will be run for all the possible target technologies.

The python script will generate the *.tcl scripts for Synplify listed in Table 5-14

With the *.tcl scripts, the synthesis can be run with Synplify as explained in Section 5.8.3.2. The *.tcl scripts will use the provided wrapper, located in $123DB/src/post_syn/ccsds123_top_wrapper.vhd, as top module. This wrapper flattens the records used as I/O ports and instantiates the CCSDS123 IP core.

After the synthesis is completed, the generated post-syntesis model (ccsds123_top_wrapper.vhm) is stored in $123DB/src/post_syn/$TECHNOLOGY/{TestId}/ccsds123_top_wrapper.vhm.

5.8.4.2 Generating the scripts for running the post-synthesis simulations

Follow the same procedure used for behavioural simulations, going along the instructions in Section 5.8.2.1, with the following exceptions.

- When running the script $123DB/verification_scripts/run_vhdl_tests_123.py, use $OPTION = modelsim-ps and select a technology with the argument $TECHNOLOGY.

```
$ $123DB/verification_scripts/run_vhdl_tests_123.py $CSV_FILE $RAW_FOLDER $COMPRESSED_FOLDER $
REFERENCE_FOLDER $DATABASE123_DIRECTORY $DATABASE121_DIRECTORY $OPTION $TECHNOLOGY
```

- The script requires the user to select a $TECHNOLOGY among the possible values { XC5VFX130T, XQR5VFX130, A3PE3000, RTAX4000S, RT4G4150}. This argument is mandatory.

The python script will generate the files listed in Table 5-11, plus the scripts for post-synthesis simulations for each test case, which are located in $123DB/modelsim/tb_scripts/{TestId}_ps.do. The all_tests.do file will be filled with the sentences to execute the post-synthesis simulation scripts for all test cases $123DB/modelsim/tb_scripts/{TestId}_ps.do.

5.8.4.3 Running the testbench for post-synthesis simulations

The testbench can be executed after generating all the necessary *.do files by following the procedures in Section 4.8.2.2

Note that it is up to the user to pre-compile and map the technology vendor libraries (axcelerator, unisim, …) in QuestaSim.

5.8.5 Post-PAR simulations

Post-PAR simulations can be run for Virtex5 only, and for the testcases provided in the file $123DB/verification_scripts/testcases123_post_syn.csv. The following steps are necessary:

9. Run the python configuration script for post-synthesis simulations on Virtex5 with the following arguments: ($CSV_FILE = $123DB/verification_scripts/testcases121.csv ; $OPTION = synplify-ps and $TECHNOLOGY = XC5VFX130T):

```
$ $123DB/verification_scripts/run_vhdl_tests_123.py $123DB/verification_scripts/testcases123.c
sv $RAW_FOLDER $COMPRESSED_FOLDER $REFERENCE_FOLDER $DATABASE123_DIRECTORY $DATABASE121_DIRECT
ORY synplify-ps XC5VFX130T
```

10. Run the synthesis with Synplify with the script $123DB/synthesis/syn_scripts/synplify/all_synplify.do

11. Once finished, run the script $123DB/synthesis/syn_scripts/synplify/all_synplify_par.do to run the PAR for all the testcases.

12. Run the command:

"netgen -intstyle ise -s 1 -pcf ccsds123_top_wrapper.pcf -rpw 100 -tpw 0 -ar Structure -tm ccsds123_top_wrapper -insert_pp_buffers true -w -dir netgen/par -ofmt vhdl -sim ccsds123_top_wrapper.ncd ccsds123_top_wrapper_timesim.vhd"

for all the tests from its par folder (e.g. ./synthesis/syn_scripts/synplify/XC5VFX130T_03_Test/par_1). This Xilinx command generates the vhdl model for PAR simulations and the sdf timing file.

13. Use the provided python script for post-PAR simulations $123DB/verification_scripts/run_vhdl_tests_123_V5_PAR.py with $OPTION = modelsim.

```
$ $123DB/verification_scripts/run_vhdl_tests_121_V5_PAR.py $123DB/verification_scripts/tes
tcases121.csv $RAW_FOLDER $COMPRESSED_FOLDER $REFERENCE_FOLDER $DATABASE123_DIRECTORY $DAT
ABASE121_DIRECTORY modelsim
```

```
run_vhdl_tests_121_V5_PAR.py testcases_123_post_syn.csv ../images/raw ../images/compressed
../images/reference ../ ../../CCSDS121IP-VHDL modelsim
```

14. The script generates a set of test scripts for all the test cases. The glbl.v file is added for compilation (vlog $XILINX/verilog/src/glbl.v). The user needs to set the $XILINX folder.

15. In order to run the simulations the libraries below are also needed: Simprim & Vital2000 libs.

16. The scripts might be executed from QuestaSim with the command:

    do $123DB/modelsim/tb_scripts/all_tests.do

6 SHYLOC (CCSDS123 IP + CCSDS121 IP) USER MANUAL

6.1 Connecting the two IP cores

The following VHDL code snippet shows an example architecture in which signals are created to connect the two IP cores. The data output of the CCSDS123 IP core is connected to the data input of the CCSDS121 IP core. The external control signals of the CCSDS123 IP core are connected to the control signals of the CCSDS123 IP core.

```
architecture arch of shyloc is

   signal clk, rst_n, clk_ahb, rst_ahb: std_logic;
   signal DataIn: std_logic_vector (shyloc_123.ccsds123_parameters.D_GEN-1 downto 0);
   signal DataIn_NewValid: std_logic;

   signal AHBSlave123_In: shyloc_utils.amba.ahb_slv_in_type;
   signal AHBSlave123_Out: shyloc_utils.amba.ahb_slv_out_type;


   signal AHBSlave121_In: shyloc_utils.amba.ahb_slv_in_type;
   signal AHBSlave121_Out: shyloc_utils.amba.ahb_slv_out_type;

   signal AHBMaster123_In: shyloc_utils.amba.ahb_mst_in_type;
   signal AHBMaster123_Out: shyloc_utils.amba.ahb_mst_out_type;


   signal AwaitingConfig: Std_Logic;
   signal Ready: Std_Logic;
   signal FIFO_Full: Std_Logic;
   signal EOP: Std_Logic;
   signal Finished: Std_Logic;
   signal Error_s: Std_Logic;
   signal ForceStop: Std_Logic;
   signal DataOut: Std_Logic_Vector (shyloc_121.ccsds121_parameters.W_BUFFER_GEN-1 downto 0);
   signal DataOut_Valid: Std_Logic;
   signal IsHeaderOut: Std_Logic;
   signal NbitsOut: Std_Logic_Vector (6 downto 0);


   signal AwaitingConfig_Ext: Std_Logic;
   signal Ready_Ext: Std_Logic;
   signal FIFO_Full_Ext: Std_Logic;
   signal EOP_Ext: Std_Logic;
   signal Finished Ext: Std Logic;
   signal Error_Ext: Std_Logic;
   signal ForceStop_Ext: Std_Logic;


   signal Block_DataIn_Valid:  Std_Logic;
   signal Block Ready Ext: Std Logic;
   signal Block_IsHeaderIn: Std_Logic;
   signal Block_DataIn: Std_Logic_Vector(shyloc_123.ccsds123_parameters.W_BUFFER_GEN-1 downto
   0);
   signal Block_NBitsIn: Std_Logic_Vector(6 downto 0);
   signal ErrorCode_Ext: Std_Logic_Vector(3 downto 0);

begin

----------------------------
--!@brief CCSDS-123 IP Core
----------------------------
ccsds123: entity shyloc_123.ccsds123_top(arch)
   port map(
   clk_s => clk,
   rst_n => rst_n,
   clk_ahb => clk_ahb,
```

```
    rst_ahb => rst_ahb,

    DataIn => DataIn,
    DataIn_NewValid => DataIn_NewValid,
    AwaitingConfig => AwaitingConfig,

    Ready => Ready,
    FIFO_Full => FIFO_Full,
    EOP => EOP,
    Finished => Finished,
    ForceStop => ForceStop,
    Error => Error,

    AHBSlave123_In => AHBSlave123_In,
    AHBSlave123_Out => AHBSlave123_Out,
    AHBMaster123_In => AHBMaster123_In,
    AHBMaster123_Out => AHBMaster123_Out,

    DataOut => Block_DataIn,
    DataOut_NewValid => Block_DataIn_Valid,
    IsHeaderOut => Block_IsHeaderIn,
    NbitsOut => Block_NbitsIn,

    AwaitingConfig_Ext => AwaitingConfig_Ext,
    ForceStop_Ext => ForceStop_Ext,
    Ready_Ext => Block_Ready_Ext,
    FIFO_Full_Ext => FIFO_Full_Ext,
    EOP_Ext => EOP_Ext,
    Finished_Ext => Finished_Ext,
    Error_Ext => Error_Ext
);

----------------------------
--!@brief CCSDS-121 IP Core
----------------------------
Ccsds121: entity shyloc_121.ccsds121_shyloc_top(arch)
    port map (
    Clk_S => clk,
    Rst_N => rst_n,
    Clk_AHB => clk_ahb,
    Reset_AHB => rst_ahb,

    AHBSlave121_In => AHBSlave121_In,
    AHBSlave121_Out => AHBSlave121_Out,

    DataIn_NewValid => Block_DataIn_Valid,
    DataIn => Block_DataIn(shyloc_121.ccsds121_parameters.D_GEN-1 downto 0),
    NBitsIn => Block_NBitsIn(5 downto 0),

    DataOut => DataOut,
    DataOut_NewValid => DataOut_Valid,

    ForceStop => ForceStop_Ext,
    IsHeaderIn => Block_IsHeaderIn,
    AwaitingConfig => AwaitingConfig_Ext,
    Ready => Block_Ready_Ext,
    FIFO_Full => FIFO_Full_Ext,
    EOP => EOP_Ext,
    Finished => Finished_Ext,
    Error => Error_Ext,
    ErrorCode => ErrorCode_Ext,
    Ready_Ext => Ready_Ext
    );
end arch;
```

6.2 SHyLoC testbench

In order to run a simulation in which the CCSDS121 IP core is instantiated as an external entropy coder, a specific testbench file is provided.

TABLE 6-1: SHYLOC – TESTBENCH VHDL SOURCES

Folder	Compile order	Filename	Library
$123DB/tb/shyloc	--	ccsds_shyloc_tb.vhd	work

6.2.1 SHyLoC testbench behavioural description

The behaviour of the testbench is described in Table 6-2.

TABLE 6-2: CCSDS123 IP TESTBENCH – TEST SEQUENCES

Description
1. Write run-time configuration values in memory-mapped registers for the CCSDS123 IP core. 2. Write run-time configuration values in memory-mapped registers for the CCSDS121 IP core. 3. Send raw samples to the CCSDS123 IP core. 4. Monitor output of the CCSDS121 IP core and compare with reference file when valid compressed data are produced. 5. Wait for CCSDS123 IP to signal that it has finished.

6.2.2 SHyLoC testbench assertions

The following assertions are included in the testbench:

TABLE 6-3: SHYLOC TESTBENCH – ASSERTIONS

Report	Severity	Description
Finished started with a high value	warning	Checks correct value of control signals after reset of the IP core.
Ready started with a high value	warning	
AwaitingConfig started with a low value	warning	
Sending 123 configuration...	note	Configuration registers are sent to the AHB bus for the CCSDS123 IP core.
Sending 121 configuration...	note	Configuration registers are sent to the AHB bus for the CCSDS121 IP core.
Wrong testbench configuration: configured Nx for CCSDS123 differs from CCSDS121	failure	Incompatible parameters between CCSDS123 and CCSDS121 IP core.
Wrong testbench configuration: configured Ny for CCSDS123 differs from CCSDS121	failure	Incompatible parameters between CCSDS123 and CCSDS121 IP core.

Wrong testbench configuration: configured Nz for CCSDS123 differs from CCSDS121	failure	Incompatible parameters between CCSDS123 and CCSDS121 IP core.
Wrong parameters configuration: selected EN_RUNCFG value for CCSDS123 differs from EN_RUNCFG value in CCSDS121 check your ccsds121_parameters.vhd and ccsds123_parameters.vhd files	failure	Incompatible parameters between CCSDS123 and CCSDS121 IP core.
Wrong parameters configuration: selected RESET_TYPE value for CCSDS123 differs from RESET_TYPE value in CCSDS121 check your ccsds121_parameters.vhd and ccsds123_parameters.vhd files	failure	Incompatible parameters between CCSDS123 and CCSDS121 IP core.
One compressions test performed	note	Test finished.

6.3 SHyLoC simulation scripts

In order to configure a simulation which includes the CCSDS123 IP core with the CCSDS121 IP core acting as external encoder, the *.csv file and python script provided in the CCSDS123 IP core database can be used. Follow the procedures detailed in Section 5.8, taking into account the compatibility rules described next.

6.3.1 CCSDS121 and CCSDS123 compatibility rules

When configuring the CCSDS123 IP core and the CCSDS121 IP core to work jointly, the compatibility rules described in Table 6-4 must be followed.

TABLE 6-4: 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	Compile-time parameter dynamic range in the CCSDS121 IP core 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 IP core shall be always set to big endian (1).