

IP Core for Medium Data Rate PDT

Final Report

ESA Contract
4000117901/16/NL/FE/eg

Revision 1.0
Date 10/07/2018

Preparato/Prepared	Approvato/Approved	Autorizzato/Authorized
R. Cassettari, D. Davalle	S. Saponara, L. Fanucci	L. Fanucci

Roberto Cassettari

Davalle

S. Saponara

L. Fanucci

Change Record

Rev	Date	Change description	Edited
1.0	10/07/2018	First issue	R. Cassettari, D. Davalle

Contents

1	Introduction	6
1.1	Scope	6
1.2	Documents.....	6
1.2.1	Applicable documents	6
1.3	Definitions and Acronyms	6
1.3.1	Definitions	6
1.3.2	Acronyms	6
2	Overview.....	8
3	Architecture Description	11
3.1	ccsds_tm_top.....	11
3.1.1	Data-path optimisation.....	16
3.1.2	Buffering strategy and throughput.....	23
3.1.3	Generics.....	25
3.1.4	Ports.....	28
3.2	Graphical User Interface	34
4	Technology mapping results	39
4.1	BER Performance	41
5	Validation set-up	43

List of Figures

Figure 2-1 IP Core High-level functional block diagram	8
Figure 3-1 Input FIFO data alignment	12
Figure 3-2 IP core architecture.....	15
Figure 3-3 Data-rates within the SCCC encoder	18
Figure 3-4 Double buffering architecture timing diagram	23
Figure 3-5 Single buffering architecture timing diagram	23
Figure 3-6 CCSDS Telemetry Transmitter IP Core top-level interface.....	28
Figure 3-7 Input data interface timing	29
Figure 3-8 Output data interface timing	31
Figure 3-9 Configuration and status interface timing	32
Figure 3-10 LUT loader interface timing	33
Figure 3-11 IP Core Graphical User Interface	35
Figure 3-12 Example of ModCods ad output symbol rate setting	36
Figure 3-13 Architecture panel options.....	36
Figure 3-14 PL Framing and Filtering setting example	36
Figure 3-15 Predistortion file selection example	37
Figure 3-16 Predistortion file format	37
Figure 3-17 Target technology selection	38
Figure 4-1 IP core BER performance for ACM1 and ACM7	41
Figure 4-2 IP core BER performance for ACM13 and ACM18.....	41
Figure 4-3 IP core BER performance for ACM23	42
• Figure 5-1 HW set-up.....	44
Figure 5-2 Validation System, FPGA Block Diagram.....	44

List of Tables

Table 1-1 Definitions	6
Table 1-2 Acronyms	7
Table 2-1 Encoder parameters for the 27 supported ModCods	10
Table 3-1 Theoretical and supported data rates into the SCCC encoder.....	18
Table 3-2 Architecture parallelism parameters for $P_s = 1$	20
Table 3-3 Architecture parallelism parameters for $P_s = 2$	21
Table 3-4 Architecture parallelism parameters for $P_s = 4$	22
Table 3-5 Top-level Generic Map	27
Table 3-6 Input data interface	29
Table 3-7 Output data interface	30
Table 3-8 Configuration and status interface	31
Table 3-9 LUT loader interface	33
Table 3-10 Clock, reset and enable interface	34
Table 4-1 ModCod support on the different technologies	39
Table 4-2 Multiple-ModCod instantiation synthesis results on the different technologies.....	40
Table 4-3 IP core post place and route performance	41

1 Introduction

1.1 Scope

This document represents the Final Report for the activity “IP Core for Medium Data Rate PDT” initiated by the European Space Agency under ESTEC contract 400117901/16/NL/FE/eg

1.2 Documents

1.2.1 Applicable documents

AD	Doc. No.	Issue/ Rev.	Title
[AD01]	CCSDS 131.2-B-1	1	Flexible advanced coding and modulation scheme for high rate telemetry applications
[AD02]	TEC-ETC/2015.172	2	Statement of Work - IP Core for Medium Data Rate PDT

1.3 Definitions and Acronyms

1.3.1 Definitions

Recurring definitions are reported in the following table.

Definition	Description
Hard configuration	Configuration item selected during IP core generation or through code parameter, which cannot change at run-time. Hard configuration parameters are typically used to optimize the specific implementation of the IP core, while maintaining a more general purpose IP core
ModCod	Modulation and Coding scheme
Soft configuration	Configuration item which can be modified at run-time, e.g., through configuration registers.

Table 1-1 Definitions

1.3.2 Acronyms

Acronyms used in the present document are listed in the following table.

Acronym	Description
ACM	Adaptive Coding and Modulations
CDC	Clock Domain Crossing
FFT	Fast Fourier Transform
FIFO	First In First Out
FIR	Finite Impulse Response
GUI	Graphical User Interface
HPA	High Power Amplifier
LUT	Look-Up Table
PL	Physical Layer
SCCC	Serially Concatenated Convolutional Coding

Acronym	Description
SEE	Single Event Effect
SRRC	Square Root Raised Cosine
VHDL	VHSIC Hardware Description Language

Table 1-2 Acronyms

2 Overview

The IP Core for Medium Data Rate PDT implements a Serially Concatenated Convolutional Coding (SCCC) scheme for telemetry application, which functionality is compliant with CCSDS 131.2-B-1 standard. The transmitter makes use of a large variety of modulation schemes (including QPSK, 8-PSK, 16-APSK, 32-APSK and 64-APSK) and a wide range of coding rates.

The high-level functional block diagram of the CCSDS telemetry transmitter IP core is presented in Figure 2-1.

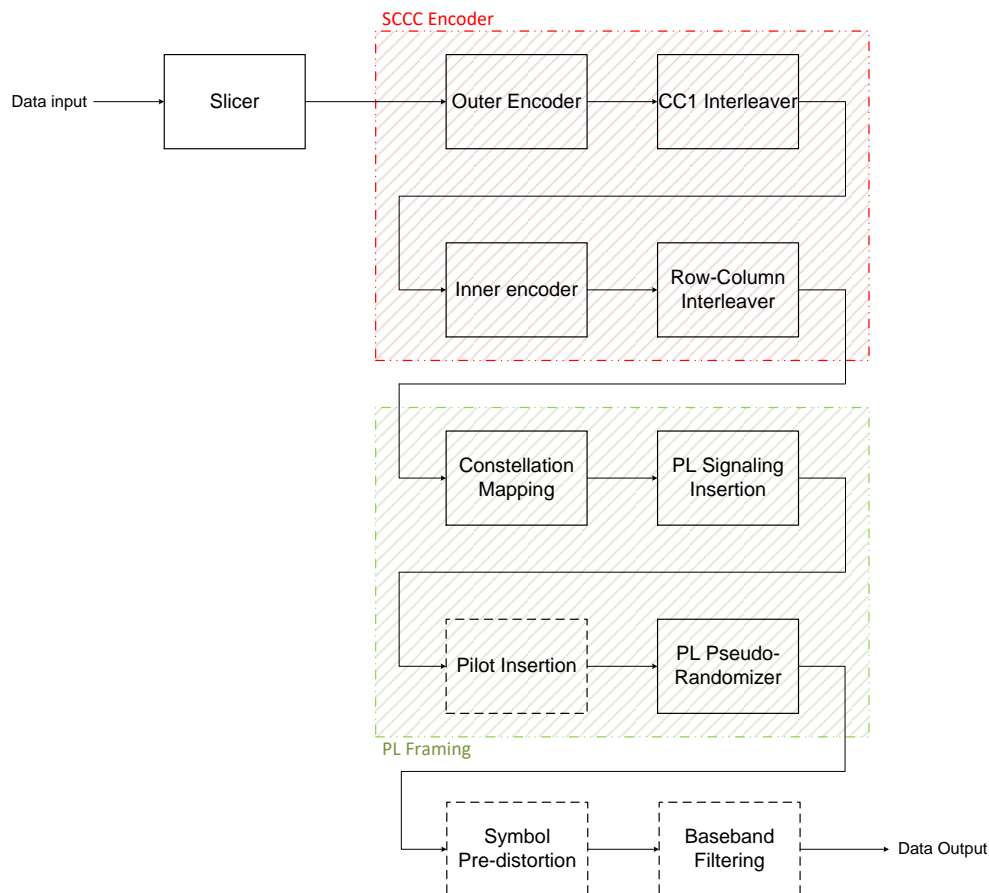


Figure 2-1 IP Core High-level functional block diagram

The functionality of each block in the processing chain is described in the following. The algorithm consists of two main functional blocks: 1) a Serially Concatenated Convolutional Coding (SCCC) Encoder and 2) a Physical Layer (PL) framing block. Other functional blocks, which take place before the Slicer block in the processing chain, are defined in [AD01], but they are not implemented in this IP Core and then will not be described in this document. More details about these functional blocks and the algorithms used in the Telemetry Transmitter can be found in the CCSDS 131.2-B standard. Table 2-1 contains the algorithm parameters for the encoder processing, as defined in [AD01] and which are widely used in the rest of the document for the description of the architecture.

In detail, the processing chain is composed by the following blocks:

- **Slicer:** this block has to split the input CADU stream into a sequence of information blocks of length K , corresponding to the information block size of the selected ModCod.
- **Outer encoder (CC1):** performs the first convolutional encoding and the output bit-stream is punctured by decimating the parity bits by half for an overall coding rate of $2/3$. The input data block has length K bits and the output data block has length I bits.
- **CC1 interleaver:** this block implements the ad-hoc interleaving law described in the annex B of the CCSDS 131.2-B standard, which makes extensive use of Look-Up Tables. The input/output data block has length I bits.
- **Inner Encoder (CC2):** carries out the second convolutional encoding, with the same algorithm as the outer encoder. Puncturing adjusts the data size of the block in order to fit the 8100 output symbols. Differently from outer puncturing, the puncturing after inner encoder is much more complex as it is composed by different algorithms for puncturing systematic bits and parity bits. The input data block has length I bits, and the output data block is punctured to have 8100 output symbols for all the ModCods, therefore, the number of output bits for each ModCod is $8100 \cdot m$, where m is the modulation index relative to the ModCod.
- **Row-Column Interleaver:** is used to interleave the bits that are assigned to symbols. The input to the Row-Column interleaver are the systematic bits followed by the parity bits from the punctured output of the inner encoder.
- **Constellation Mapping:** is used to map the SCCC encoded bit-stream to symbols, according to the modulation index defined in each ModCod.
- **PL Signalling Insertion:** is used to frame the sequence of codewords sections, each composed of 8100 symbols. A frame header segment is added every 16 codeword sections (codeword segment). The frame header together with the codeword segment constitutes the Physical Layer Frame.
- **Pilot Insertion:** is used to include a pilot sequence to facilitate carrier and phase synchronization. The pilot insertion is an optional functional block and it is possible to bypass it at IP Core instantiation.
- **PL Pseudo-Randomiser:** is used to have a sufficient number of bit transitions in order to allow proper synchronisation of the decoder.
- **Symbol pre-distortion:** is used to correct HPA non-linearities, both in phase and amplitude, by multiplication with a constant complex pre-distortion coefficient on the output symbols, right before baseband filtering. The symbol pre-distortion is an optional functional block and it is possible to bypass it at IP Core instantiation.
- **Baseband filtering:** performs Square Root Raised Cosine (SRRC) pulse-shaping on I and Q arms, with the roll-off options given in the CCSDS 131.2-B standard. The baseband filtering is an optional functional block and it is possible to bypass it at IP Core instantiation.

ModCod	Information block size (bits) (K)	Information W (Wi)	outer CC output after puncturing (bits) (I)	Interleaver W	Systematic bits after inner CC puncturing (S)	Parity bits after inner CC puncturing (P)	inner CC output (bits)	Modula tion index (m)
1	5758	48	8640	72	8642	7558	16200	2
2	6958	58	10440	87	10442	5758	16200	2
3	8398	70	12600	105	11510	4690	16200	2
4	9838	82	14760	123	12351	3849	16200	2
5	11278	94	16920	141	13200	3000	16200	2
6	13198	110	19800	165	14390	1810	16200	2
7	11278	94	16920	141	16470	7830	24300	3
8	13198	110	19800	165	15842	8458	24300	3
9	14878	124	22320	186	18602	5698	24300	3
10	17038	142	25560	213	19939	4361	24300	3
11	19198	160	28800	240	21218	3082	24300	3
12	21358	178	32040	267	22857	1443	24300	3
13	19198	160	28800	240	24482	7918	32400	4
14	21358	178	32040	267	25741	6659	32400	4
15	23518	196	35280	294	27051	5349	32400	4
16	25918	216	38880	324	28515	3885	32400	4
17	28318	236	42480	354	29880	2520	32400	4
18	25918	216	38880	324	31755	8745	40500	5
19	28318	236	42480	354	33137	7363	40500	5
20	30958	258	46440	387	34677	5823	40500	5
21	33358	278	50040	417	36197	4303	40500	5
22	35998	300	54000	450	37802	2698	40500	5
23	33358	278	50040	417	39366	9234	48600	6
24	35998	300	54000	450	41042	7558	48600	6
25	38638	322	57960	483	42507	6093	48600	6
26	41038	342	61560	513	43915	4685	48600	6
27	43678	364	65520	546	45429	3171	48600	6

Table 2-1 Encoder parameters for the 27 supported ModCods

3 Architecture Description

The following sections contain the description of the top-level architecture of the CCSDS Telemetry Transmitter IP-core.

3.1 ccsds_tm_top

This represents the top-level entity of CCSDS Telemetry transmitter IP core.

In general, a fully synchronous approach has been adopted, avoiding internal clock-domain crossings. In order to deal with host system integration, some interfaces are driven by a different clock with respect to the IP core clock, provided by the host. The resulting clock-domain crossings (CDC) are limited to those interfaces and are treated with well-known methods such as asynchronous FIFO, which totally eliminate the problems involved with CDC.

The global reset signal will be connected to all flip-flops without combinational logic. The reset behavior is configurable by hard configuration: active-high/active low; synchronous/asynchronous.

Figure 3-2 shows the IP-core architecture, which consists of the following blocks:

- The **Slicer**, which has the task of reorganizing the input data stream for the next processing. Figure 3-1 shows how the blocks of data are written and organised into the input FIFO. In general the block size K (variable with the ModCod selected) is not a multiple of the FIFO word length, $L = 32$, therefore it is necessary to realign the data for the subsequent processing. The asynchronous FIFO is used by the external host to write data to the IP-core for processing. The FIFO is read internally by the word align block which organises data for the outer convolutional encoder (CC1 encoder). The slicer is composed by:
 - **input_fifo**: the input FIFO buffer represents the external host interface of the IP Core and provides a simple FIFO interface composed of a 32-bit input data bus, the write-enable and the fifo-full flag. Writing into FIFO is managed using an external clock domain, which is given as input to the IP Core. The input FIFO is asynchronous to allow easy integration in the IP core host system.
 - **word_align_block**: this block is used to extract the P_{cc1} bits word which shall be processed in parallel by the outer convolutional encoder, keeping alignment to the input data block of P_{cc1} is a generic parameter of the IP Core.
- The **SCCC encoder**, which performs in cascade:
 - **CC1**, the outer convolutional encoder, features an innovative data processing architecture, which allows to optimize resource optimization.
 - **word_align_row** this block and the following row2col block have the task of reshaping the data to be provided to the CC1 Interleaver block. the CC1 Interleaver works on data organised as a matrix of 120 rows and W columns, where W is a specific parameter of the transmitter defined in [AD01] and depends on the ModCod

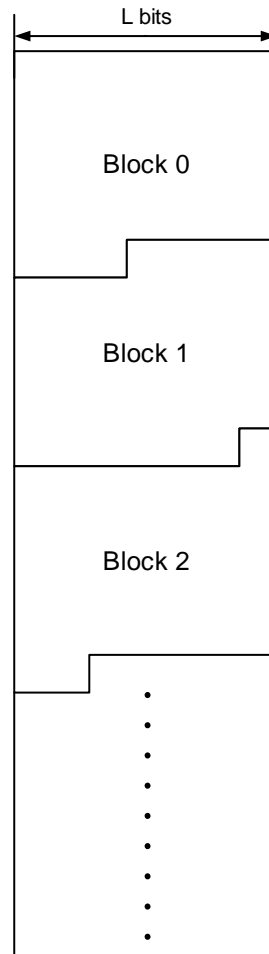


Figure 3-1 Input FIFO data alignment

selected (see Table 2-1). The word align block in particular provides 36-bit data words aligned to the row's length W , i.e., after alignment, each row starts on the lsb of the 36-bit output word and each 36-bit word will contain data belonging to one row only. This aspect is fundamental as in the following processing contiguous rows shall be accessed simultaneously and each row is stored in a different memory block to achieve this.

- **row2col**, this block transposes data of the input data matrix, which is then read column by column by the CC1 interleaver, P_{ILV} bits per clock cycle. P_{ILV} is a generic parameter of the IP Core.
- **cc1_interleaver**, this block implements the interleaver law described in [AD01]. The input data matrix is read column by column to implement the algorithm efficiently. The input data matrix is segmented, i.e., instead of processing whole columns, P_{ILV} -bit column segments are processed, in order to reduce the complexity. The output is stored in a memory matrix, implemented with RAM blocks. The CC1 interleaver processes P_{ILV} bits per clock cycle. P_{ILV} is in general less than 120 (column length),

thus, a column in the memory matrix is partially written $120/P_{ILV}$ times during the interleaving processing.

- **memory_matrix**, this block stores the output data of the Interleaver processing in a RAM of W words of size 120. The maximum W depends on the ModCods supported in a certain instantiation of the IP core (see Table 2-1 for the correspondence between W and ModCod). Data for the following processing block is valid only after the completion of the Interleaver processing thus, in order to avoid interruptions in the data path processing, two memory matrices working in ping-pong are expected at this stage, i.e., while the Interleaver is writing in one memory, the Inner Encoder can read and process data from the second memory and vice-versa. The IP Core generic BUFFERING_TYPE allows an implementation of the memory matrix either as ping-pong memory, or as a single memory to reduce hardware complexity at the price of a lower data rate achievable.
- **cc2**, this block implements the same encoding algorithm used in the Outer Encoder, however with a different architecture. Where the outer encoder processes data row-wise, the inner encoder processes data column-wise, to have an efficient integration with the CC1 interleaver. In order to reduce hardware complexity, CC2 is able to elaborate column segments other than full columns, i.e., P_{cc2} rows of the memory matrix at a time. The elaboration of each P_{cc2} rows segment requires two passes, which means $2 \cdot W$ clock cycles. In the first pass, the P_{cc2} rows of the memory matrix are read to calculate the initial states for each sub-trellis. During the first pass, the output of CC2 are the systematic bits. During the second pass, the P_{cc2} rows are read again to produce the related parity bits. The encoding of the whole data matrix lasts $2 \cdot W \cdot \frac{120}{P_{cc2}}$ clock cycles, where P_{cc2} is a generic parameter of the IP Core.
- **cc2_puncturing**, this block performs the puncturing of both the systematic and parity bits produced by the Inner Encoder. In particular, the systematic bits are punctured during CC2 first pass, while the parity bits are punctured during CC2 second pass of each segment. Systematic and parity bits are punctured with different algorithms, as described in [AD01]. As for the Inner Encoder, this block processes P_{cc2} bits in parallel. Output data is stored into the Row-Column memory.
- **row_column_memory**. This memory consists of m RAM blocks of 8100 bits each, to facilitate the next row-column interleaver processing. Each RAM block corresponds to a column of the row-column interleaver. The Row-Column Interleaver cannot start the processing of the Row-Column memory until the Inner Puncturing block has terminated its processing. For this reason, in order to guarantee the continuous transmission of symbols, two memory buffers must be implemented in ping-pong.
- **row-column interleaver**, this block reads m columns of the Row-Column memory in parallel, where m is the modulation index and depends on the selected ModCod. Each set of m bits represents a *symbol*. The Row-Column interleaver outputs P_s symbols in parallel. The Row-Column interleaver produces 8100 symbols every input block of K bits. Data produced by the row-column interleaver constitutes the input for Physical Layer (PL) framing.

- The **PL framing**, which performs the constellation mapping, the optional pilot insertion, symbol pseudo-randomization.
 - **frame_header_insertion**, this block generates the Frame Marker and Frame Descriptor fields to be inserted at the beginning of each PL frame. The architecture is able to produce P_S symbols per clock cycle. The modulation scheme used for these symbols is a $\pi/2 - BPSK$.
 - **constellation_mapper**, which translates the data symbols coming from the row-column interleaver in constellation symbols, depending on the modulation set. 10 different constellation maps are supported. The architecture is able to produce P_S symbols per clock cycle.
 - **pilot_insertion**, this block inserts 16 Pilot symbols every 540 data symbols for a total of 240 Pilot symbols per Codeword section ($240 \cdot 16 = 3840$ pilot symbols in a whole PL frame). The pilot insertion can be excluded in a given IP Core instantiation through the generic parameter PILOT_INSERTION.
 - **pl_randomizer**, which generates the pseudo-random sequence used to randomize the data and pilots symbols. The frame Header symbols are not randomized. The pseudo-random sequence is reset at the end of each PL frame. The architecture of this block is able to randomize P_S symbols per clock cycle.
- **predistortion**, this block is not specified in [AD01], symbol pre-distortion is provided in the IP Core to allow a first compensation of the amplitude/phase distortion introduced by the HPA. It implements a complex multiplication between the data symbols (as well as the pilot symbols when expected) and a specific set of pre-distortion parameters. It is possible to implement the pre-distortion parameters internally to the IP Core (as a LUT), or load them through the Configuration & status interface. In the first case, pre-distortion parameters can be set by the user through dedicated generics (see Table 3-5). The PREDISTORTION_TYPE generic allows to select the desired pre-distortion parameters implementation.
- **srrc_filter**, baseband SRRC filtering on both I and Q branches. The filter order is 47, supporting roll-off factor of 0.20, 0.25, 0.30, 0.35. P_S filtered symbols are provided in parallel, where P_S is a generic parameter of the IP Core, and each filtered symbols is represented in its real and imaginary part with 4 samples (4 is the designed samples per symbol, SRRC_OVERSAMPLING). Output samples are provided in parallel $4P_S$ per clock cycle, sample serialisation is up to the IP core user. The SRRC filter can be excluded in a given IP Core instantiation through the generic parameter SRRC_ENABLE. In this case, the IP Core output will coincide with the output of the pre-distortion block.
- **config_status**, responsible of managing the IP core soft configuration process, providing the IP core status to the host system and loading the algorithm LUTs in case they are loaded externally.

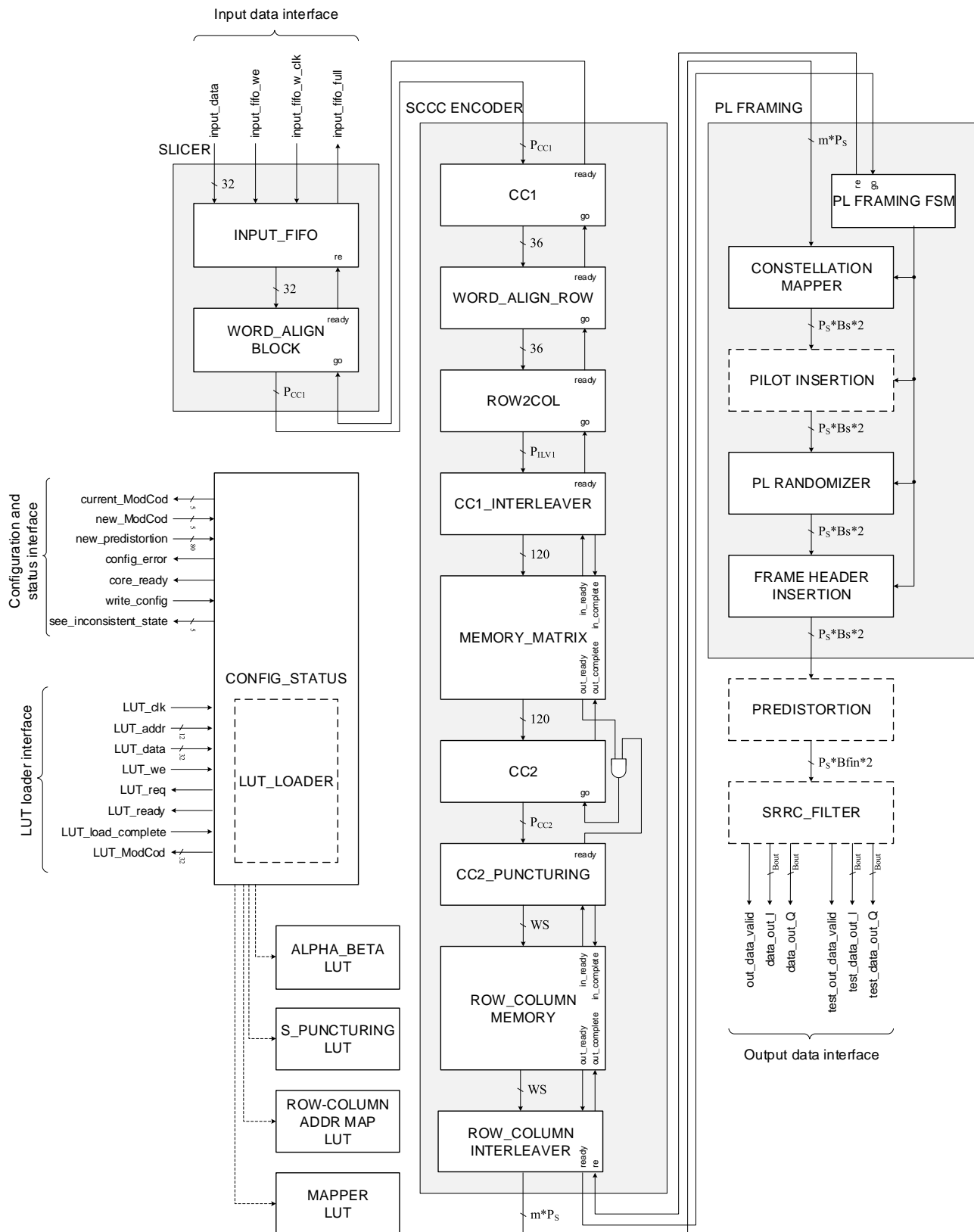


Figure 3-2 IP core architecture

The architecture was conceived with flexibility in mind. In order to easily adapt the IP core to many different use cases, it is provided with a complete set of hard configuration parameters, which allow enabling/disabling features of the processing, as well as optimizing the architecture in terms of performance/complexity trade-off. The hard configuration parameters are all accessible via the top-level generics and are described in Table 3-5.

Block RAM data buses were set to 36 bits wherever convenient. This is seen as an optimal number in terms of memory management, as every reference target technology has a block RAM cut with word size 36 and the complexity given by this data-path size is acceptable.

The data flow was designed to have continuous transmission of symbols at the output, provided that continuous data is provided at the IP core input, i.e., the IP core input buffer never gets empty. To this aim, the producer blocks in the chain are designed to be faster than the final consumer block so that the output symbol buffer never gets empty, by design. However, due to this design choice, the buffers in the chain will unavoidably get full, therefore the producer blocks need to support processing stop-and-go. Each block in the processing chain is then provided with a `ready` output signal, indicating that the specific block is ready to take data for processing, and a `go` signal, enabling the processing of the specific block.

At the interleaver buffers, i.e., the `memory_matrix` for CC1 interleaver and row-column memory for the row-column interleaver, the data flow control is managed with additional signals to efficiently handle both single and double buffering architectures. The data flow is regulated by the signals `in_ready`, `in_complete`, `out_ready`, `out_complete`. When the input write operation is complete, the producer asserts `in_complete`, so that the buffer asserts `out_ready` to inform the consumer that a data block is fully stored in the memory. In case single buffering is selected, `in_ready` is de-asserted by the buffer, so that the producer knows that it cannot write data to the buffer. When the consumer has read all the data block, it asserts `out_complete`, so that the buffer is informed that a memory block can be freed, and the producer is informed through the assertion of `in_ready`. When double-buffering is used, the buffer manages two memory blocks as a ping-pong memory, so that both the input and output interfaces are always ready, provided that the producer and the consumer operate steadily.

3.1.1 Data-path optimisation

This section describes all the architectural (i.e., parallelism) parameters which are managed by the IP Core generator and not visible to the final user. Once the “user-level” parameters are set through the GUI, all the linked parallelisation parameters are derived automatically by the software engine.

As a matter of fact, parallelisation parameters at each stage of the architecture might seem independent values at first sight, but in reality they are all linked to the output symbol rate. The optimum choice is the minimum parallelism that satisfies the required throughput at each stage. Therefore, once the output symbol rate is set, a certain data-rate is imposed to all the other stages of the processing; consequently all the other parameters are automatically extracted in order to sustain such a data-rate. In this way we obtain “classes” of optimised architecture parallelism and not all the possible combinations of the parallelisation parameters, which might be very inefficient.

The main architecture parallelisation parameters, i.e., how many bits are processed per clock cycle, are four:

1. P_{CC1} which represents the parallelism at CC1. P_{CC1} must be a number such that $P_{CC1} \cdot \frac{3}{2}$ is submultiple of 36 in order to simplify puncturing and reshaping for the subsequent memory write. Therefore it holds $P_{CC1} \in \{2, 4, 6, 8, 12, 24\}$
2. P_{ILV} , i.e. the parallelism at CC1 interleaver. P_{ILV} must be submultiple of 120 in order to effectively achieve the read-modify-write operation at CC1 interleaver. Therefore it holds $P_{ILV} \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60, 120\}$
3. P_{CC2} is the parallelism at CC2. P_{CC2} must be as well submultiple of 120. Therefore it holds $P_{CC2} \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60, 120\}$
4. P_S represents how many symbols are produced in parallel by the IP Core. P_S , together with the clock frequency defines the output symbol rate. P_S must be a power of two in order to reduce the complexity of the interface between the Row-Column interleaver and the Constellation mapper. Moreover too high values will result in a not-feasible implementation, since the corresponding Symbol data rate could not be sustained by the maximum input bit-rate which is fixed by the 32-bit input bus. Therefore $P_S \in \{1, 2, 4\}$

The parallelism parameters for the rest of the architecture are derived from these.

As mentioned, once the symbol rate parameter is set, a data-rate constraint is imposed to all the other stages. In this way all the other architecture parallelism parameters will be selected as the minimum value fulfilling the data-rate requirement for each stage (see Figure 3-3). In order to guarantee a continuous transmission of data, the “producer” blocks must always be faster than the “consumer” blocks in the processing chain, in this way the buffers will never get empty by design. Therefore, the processing chain will be designed to allow the interruption of processing as buffers will unavoidably get full.

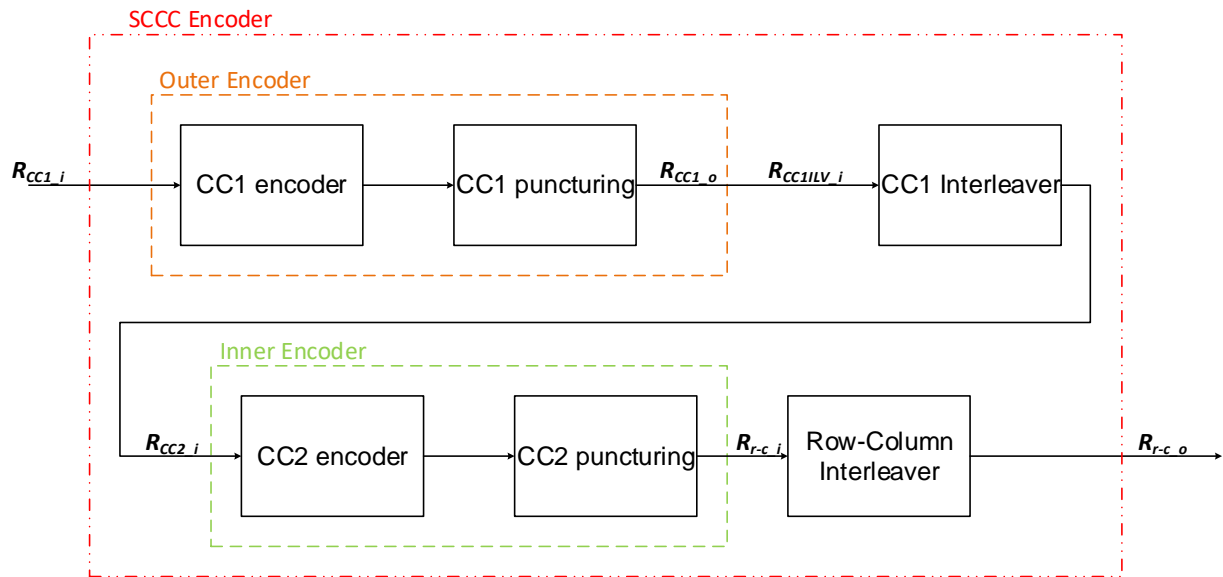


Figure 3-3 Data-rates within the SCCC encoder

The minimum required and the implemented bit-rates (the latter depending on the P_{CC1} , P_{ILV} and P_{CC2} parameters previously described) at the various stages of the SCCC encoder can be computed with the equations in Table 3-1.

MINIMUM REQUIRED data rate	IMPLEMENTED data rate
R_{sout}	$R_{sout} = f_{clk} \cdot P_S \text{ (MS/s)}$
$R_{r-c_o} = f_{clk} \cdot P_S \cdot m = R_{r-c_i} \text{ (Mb/s)}$	$R_{r-c_o} = f_{clk} \cdot P_S \cdot m \text{ (Mb/s)}$
$R_{CC2_i} = R_{r-c_i} \cdot \frac{I}{8100 \cdot m} \text{ (Mb/s)}$	$R_{CC2_i} = \frac{P_{CC2} \cdot f_{clk}}{2} \text{ (Mb/s)}$
$R_{CC1ILV_i} = R_{CC2_i} \text{ (Mb/s)}$	$R_{CC1ILV_i} = P_{ILV1} \cdot f_{clk} \text{ (Mb/s)}$
$R_{CC1_o} = R_{CC1ILV_i} \text{ (Mb/s)}$	$R_{CC1_o} = R_{CC1_i} \cdot \frac{3}{2} \text{ (Mb/s)}$
$R_{CC1_i} = R_{CC1_o} \cdot \frac{2}{3} \text{ (Mb/s)}$	$R_{CC1_i} = P_{CC1} \cdot f_{clk} \text{ (Mb/s)}$

Table 3-1 Theoretical and supported data rates into the SCCC encoder

P_{CC1} , P_{ILV1} and P_{CC2} parameters must be set in order to get the implemented data rates equal or greater than the theoretical ones. With this constraint we can re-elaborate the equations in Table 3-1 to get the following relationships between the P_S parameter and P_{CC1} , P_{ILV} and P_{CC2} parameters:

$$P_{CC2} \geq 2 \cdot P_S \cdot \frac{I}{8100}$$

$$P_{ILV1} \geq P_S \cdot \frac{I}{8100}$$

$$P_{CC1} \geq \frac{2}{3} \cdot P_S \cdot \frac{I}{8100}$$

The parameters must be then set considering that:

$$P_S \in \{ 1, 2, 4 \}$$

$$P_{CC1} \in \{ 2, 4, 6, 8, 12, 24 \}$$

$$P_{ILV1} \in \{ 1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60, 120 \}$$

$$P_{CC2} \in \{ 1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60, 120 \}$$

Table 3-2, Table 3-3 and Table 3-4 show the selected P_{CC1} , P_{ILV} , P_{CC2} values for each ModCod, and the input/output data rates considering a reference clock frequency of 100 MHz, for $P_S = 1, 2, 4$, respectively.

ModCod	P_{CC1}	P_{ILV}	P_{CC2}	$R_{b,in}$ Mb/s	$R_{s,out}$ Ms/s
1	2	2	3	71.11	100.00
2	2	2	3	85.93	100.00
3	2	2	4	103.70	100.00
4	2	2	4	121.48	100.00
5	2	3	5	139.26	100.00
6	2	3	5	162.96	100.00
7	2	3	5	139.26	100.00
8	2	3	5	162.96	100.00
9	2	3	6	183.70	100.00
10	4	4	8	210.37	100.00
11	4	4	8	237.04	100.00
12	4	4	8	263.70	100.00
13	4	4	8	237.04	100.00
14	4	4	8	263.70	100.00
15	4	5	10	290.37	100.00
16	4	5	10	320.00	100.00
17	4	6	12	349.63	100.00
18	4	5	10	320.00	100.00
19	4	6	12	349.63	100.00
20	4	6	12	382.22	100.00
21	6	8	15	411.85	100.00
22	6	8	15	444.44	100.00
23	6	8	15	411.85	100.00
24	6	8	15	444.44	100.00
25	6	8	15	477.04	100.00
26	6	8	20	506.67	100.00
27	6	10	20	539.26	100.00

Table 3-2 Architecture parallelism parameters for $P_s = 1$

ModCod	P_{CC1}	P_{ILV}	P_{CC2}	$R_{b,in}$ Mb/s	$R_{s,out}$ Ms/s
1	2	3	5	142.22	200.00
2	2	3	6	171.85	200.00
3	4	4	8	207.41	200.00
4	4	4	8	242.96	200.00
5	4	5	10	278.52	200.00
6	4	5	10	325.93	200.00
7	4	5	10	278.52	200.00
8	4	5	10	325.93	200.00
9	4	6	12	367.41	200.00
10	6	8	15	420.74	200.00
11	6	8	15	474.07	200.00
12	6	8	20	527.41	200.00
13	6	8	15	474.07	200.00
14	6	8	20	527.41	200.00
15	6	10	20	580.74	200.00
16	8	10	20	640.00	200.00
17	8	12	24	699.26	200.00
18	8	10	20	640.00	200.00
19	8	12	24	699.26	200.00
20	8	12	24	764.44	200.00
21	12	15	30	823.70	200.00
22	12	15	30	888.89	200.00
23	12	15	30	823.70	200.00
24	12	15	30	888.89	200.00
25	12	15	30	954.07	200.00
26	12	20	40	1013.33	200.00
27	12	20	40	1078.52	200.00

Table 3-3 Architecture parallelism parameters for $P_s = 2$

ModCod	P_{CC1}	P_{ILV}	P_{CC2}	$R_{b,in}$ Mb/s	$R_{s,out}$ Ms/s
1	4	5	10	284.44	400.00
2	4	6	12	343.70	400.00
3	6	8	15	414.81	400.00
4	6	8	15	485.93	400.00
5	6	10	20	557.04	400.00
6	8	10	24	651.85	400.00
7	6	10	20	557.04	400.00
8	8	10	20	651.85	400.00
9	8	12	24	734.81	400.00
10	12	15	30	841.48	400.00
11	12	15	30	948.15	400.00
12	12	20	40	1054.81	400.00
13	12	15	30	948.15	400.00
14	12	20	40	1054.81	400.00
15	12	20	40	1161.48	400.00
16	24	20	40	1280.00	400.00
17	24	24	60	1398.52	400.00
18	24	20	40	1280.00	400.00
19	24	24	60	1398.52	400.00
20	24	24	60	1528.89	400.00
21	24	30	60	1647.41	400.00
22	24	30	60	1777.78	400.00
23	24	30	60	1647.41	400.00
24	24	30	60	1777.78	400.00
25	24	30	120	1908.15	400.00
26	24	40	120	2026.67	400.00
27	24	40	120	2157.04	400.00

Table 3-4 Architecture parallelism parameters for $P_s = 4$

3.1.2 Buffering strategy and throughput

In order to optimize the performance/complexity trade-off, the IP core allows the selection of the buffering strategy at CC1 interleaver, which can be either double buffering or single buffering.

Figure 3-4 shows the timing diagram for the double buffering architecture. With the relations found among the parallelisation parameters in Section 3.1.1, the output of the row-column interleaver is continuous and therefore there is no risk of row-column buffer underrun, provided that the IP core input buffer never gets empty. This ensures that there will be no holes in the transmission as the symbol buffer is never empty by design.

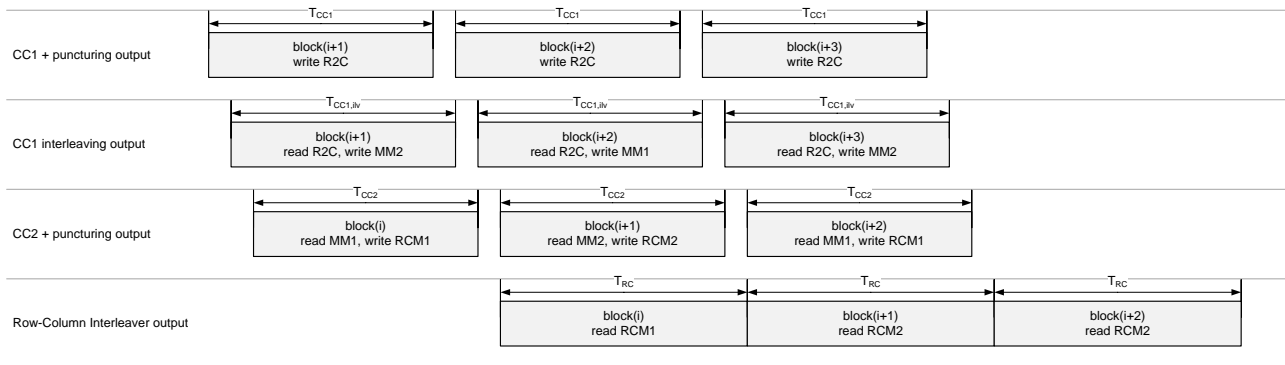


Figure 3-4 Double buffering architecture timing diagram

Figure 3-5 shows the timing diagram for the single buffering architecture. In this scenario, the memory matrix contains only one data block, and does not work as a ping-pong memory. However, the row-column memory is forced to be double, otherwise it is not possible to achieve a continuous communication. In fact, considering that an entire block must be written to the row-column memory before being read by the row-column interleaver, it is not possible to read two blocks consecutively without interruption.

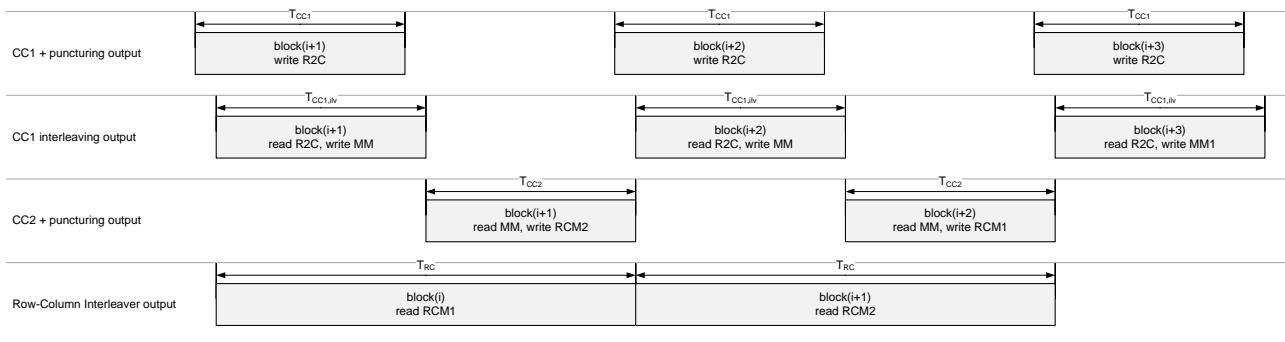


Figure 3-5 Single buffering architecture timing diagram

In order to guarantee the continuous transmission in case of single buffering architecture, the following condition must hold:

$$T_{RC} \geq T_{CC1,ilv} + T_{CC2}$$

Where T_{RC} is the time required to produce the row-column interleaver output, T_{CC2} is the time required to produce the CC2 output $T_{CC1,ilv}$ is the time required to produce the interleaver output. The relation ensures that when the row-column interleaver has finished processing a block, it has the next block available for processing in the row-column memory.

Given that: $T_{RC} = \frac{8100}{R_{s,out}}$, $T_{CC1,ilv} = \frac{I}{R_{CC1,ilv,o}} = \frac{I}{P_{ILV} f_{ck}}$ and $T_{CC2} = \frac{I}{R_{CC2,i}} = \frac{2I}{P_{CC2} f_{ck}}$, it follows that

$$R_{s,out} \leq \frac{8100}{I} \frac{P_{ILV} P_{CC2}}{2P_{ILV} + P_{CC2}} f_{ck}$$

Which represents a limit on the output symbol rate, in order to slow down the reading of the row-column memory. With the values set for P_{ILV} and P_{CC2} in Table 3-2, Table 3-3 and Table 3-4, it holds:

$$\frac{8100}{I} \frac{P_{ILV} P_{CC2}}{2P_{ILV} + P_{CC2}} \geq \frac{P_S}{2}$$

Then, the output symbol rate with the single buffering architecture can be set to $R_{s,out,sb} = \frac{P_S f_{ck}}{2}$ to have continuous transmission, which is half the data rate in the double buffering mode. The reduced data rate can be achieved by reducing the symbol parallelism to half the original parallelism.

3.1.3 Generics

Generic name	Type	Allowed values	Description
SUPPORTED_MODCODS	std_logic_vector(26 downto 0)	[0x0000001, 0x7FFFFFFF]	Specifies the ModCods supported by the specific instantiation of the IP core. For each bit, if the bit i is set at '1', the ModCod $i+1$ is supported.
DEFAULT_MODCOD	integer	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27}	Set the default ModCod after reset. The corresponding bit in the SUPPORTED_MODCODS generic must be set at '1'
BUFFERING_TYPE	integer	{0, 1}	Specifies the buffering strategy at the CC1 interleaver stages. Set at '1' to select a double buffering strategy, set at '0' to select a single buffering strategy. Double buffering allows higher output data-rate with respect to single buffering, but makes use of more RAM resources.
INPUT_FIFO_SIZE	integer	Integer power of 2's	Specifies the size of the input FIFO in memory locations (32-bit words).
P_S	integer	{1, 2, 4}	Specifies the number of symbols produced in parallel at IP-core output. Note that the higher the P_S, the higher the size of the output data signal size
P_CC1	integer	{2,4,6,8,12,24}	Specifies the parallelization degree at the outer encoder (CC1). This parameter depends on the setting of P_S
P_CC2	integer	{3,4,5,6,8,10,12,15,20,24,30,40,60,120}	Specifies the parallelization degree at the inner encoder (CC2). This parameter depends on the setting of P_S
P_ILV	integer	{2,3,4,5,6,8,10,12,15,20,24,30,40,60}	Specifies the parallelization degree at CC1 interleaver. This parameter depends on the setting of P_S
LOOKUP_TABLES	integer	{0, 1}	Specifies whether the look-up tables for the implementation of the interleaving and puncturing functions is stored internally or must be loaded from an external non-volatile memory. Set at '0' to implement lookup tables for interleaver and inner puncturing algorithms within the IP design. Set at '1' to store these values onto an external non-volatile memory and load them during the configuration phase of the telemetry transmitter.
SEE_MITIGATION	integer	{0, 1}	Enables/disables the SEE mitigation techniques in the IP Core. Set at '1' to implement SEE mitigation
PILOT_INSERTION	integer	{0, 1}	Enables/disables the pilot insertion in the PL framing. Set at '1' to implement the Pilot symbol insertion during the transmission of a Physical Frame.

Generic name	Type	Allowed values	Description
PREDISTORTION_TYPE	integer	{0, 1}	Selects the pre-distortion type, which can be OFF (0) or ON (1). Set at 0 to turn off symbol pre-distortion. Set at 1 to load pre-distortion parameters in dedicated registers. This solution allows changing the pre-distortion parameters at run-time. Predistortion registers can be initialized at reset through dedicated generics.
PREDIST_RING_0_REAL	real	[-1.2, 1.2]	Pre-distortion parameter reset value for ModCods from 1 to 12, real part (QPSK and 8PSK modulations). It is used also for Pilot symbols when expected.
PREDIST_RING_0_IM	real	[-1.2, 1.2]	Pre-distortion parameter reset value for ModCods from 1 to 12, imaginary part (QPSK and 8PSK modulations). It is used also for Pilot symbols when expected.
PREDIST_RING_1_REAL	real	[-1.2, 1.2]	Pre-distortion parameter reset value for ModCods from 13 to 27, first ring, real part
PREDIST_RING_1_IM	real	[-1.2, 1.2]	Pre-distortion parameter reset value for ModCods from 13 to 27, first ring, imaginary part
PREDIST_RING_2_REAL	real	[-1.2, 1.2]	Pre-distortion parameter reset value for ModCods from 13 to 27, second ring, real part
PREDIST_RING_2_IM	real	[-1.2, 1.2]	Pre-distortion parameter reset value for ModCods from 13 to 27, second ring, imaginary part
PREDIST_RING_3_REAL	real	[-1.2, 1.2]	Pre-distortion parameter reset value for ModCods from 18 to 27 third ring, real part
PREDIST_RING_3_IM	real	[-1.2, 1.2]	Pre-distortion parameter reset value for ModCods from 18 to 27 third ring, imaginary part
PREDIST_RING_4_REAL	real	[-1.2, 1.2]	Pre-distortion parameter reset value for ModCods from 23 to 27, fourth ring, real part
PREDIST_RING_4_IM	real	[-1.2, 1.2]	Pre-distortion parameter reset value for ModCods 23 to 27, fourth ring, imaginary part
SRRC_ENABLE	integer	{0, 1}	Enables/disables the SRRC filtering. Set at '1' to implement the SRRC filter
SRRC_ROLLOFF	integer	{0, 1, 2, 3}	Specifies the roll-off for the SRRC filter. '0' means a roll-off of 0.20, '1' means a roll-off of 0.25, '2' means a roll-off of 0.30, and '3' means a roll-off of 0.35
RESET_POLARITY	std_logic	{'0', '1'}	Specifies the polarity of the reset signal. '0': Active-low reset '1': Active-high reset
SCRAMBLING_NUMBER	integer	[0, 262142]	Specifies the scrambling number used for the pseudo-randomizer block.
CC1_PIPE	integer	{0, 1}	Specifies the presence of the outer encoder output pipeline. Set at '1' to enable the pipeline instantiation.

Generic name	Type	Allowed values	Description
ROW2COL_PIPE	integer	{0, 1}	Specifies the presence of the row2col output pipeline. Set at '1' to enable the pipeline instantiation.
CC2_PIPE	integer	{0, 1}	Specifies the presence of the inner encoder output pipeline. Set at '1' to enable the pipeline instantiation.
PREDISTORTION PIPE	integer	{0, 1, 2, 3}	Specifies the presence of the pre-distortion input/output pipeline registers: '0' : no pipeline stages present '1' : output pipeline stage present '2' : input pipeline stage present '3' : both input and output pipeline stages present
SRRC_PIPE	std_logic_vector(4 downto 0)	["00000", "11111"]	Specifies the presence of SRRC pipelines. The available pipelines are 5

Table 3-5 Top-level Generic Map

3.1.4 Ports

The CCSDS Telemetry Transmitter IP core ports can be divided into the following interfaces, based on the functionality they are related to:

- Input data interface
- Output data interface
- Configuration and status interface
- LUT loader interface
- Clock, reset and enable

Figure 3-6 shows the top-level interfaces of the IP core. The interfaces are detailed in the following sub-sections.

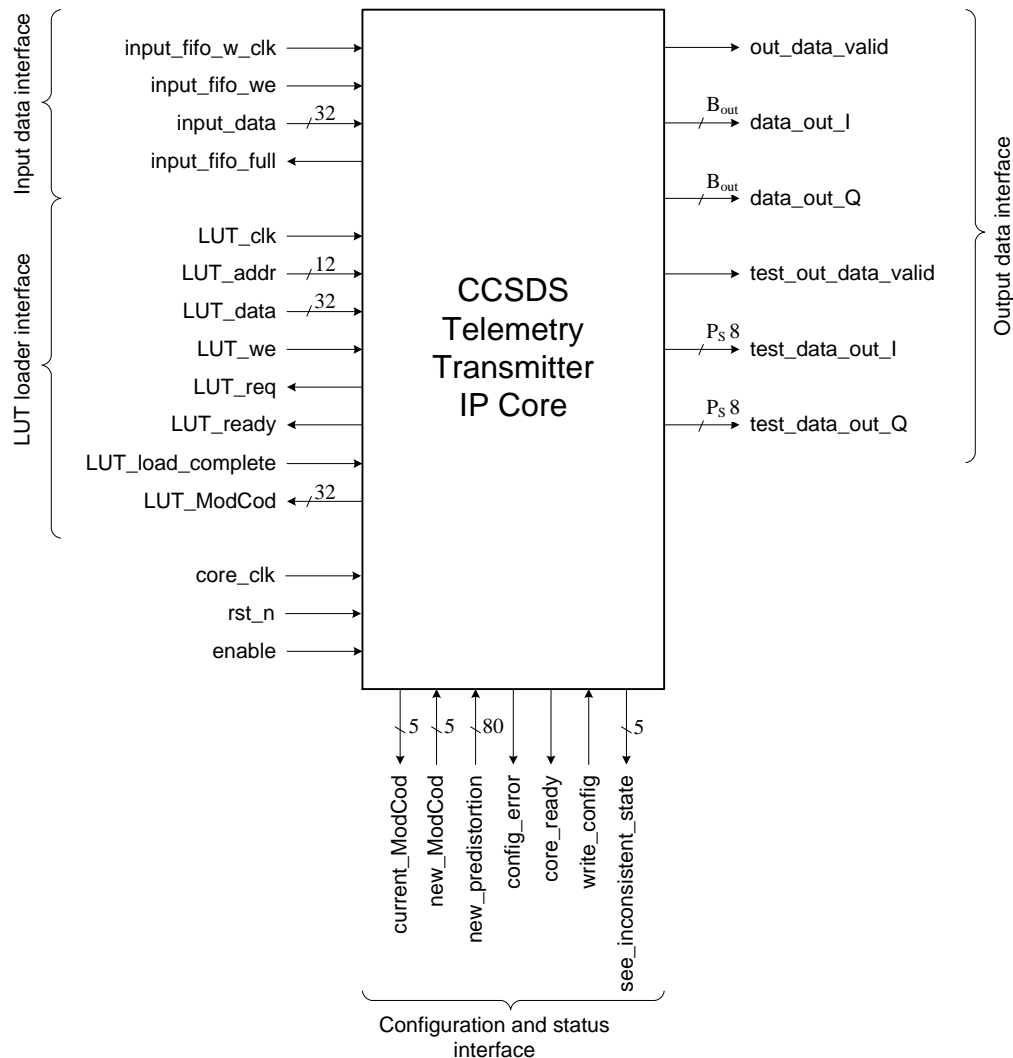


Figure 3-6 CCSDS Telemetry Transmitter IP Core top-level interface

3.1.4.1 Input data interface

Port name	I/O	bits	Clock domain	Description
input_fifo_w_clk	I	1	input_fifo_w_clk	Write clock for the input FIFO buffer.
input_fifo_we	I	1	input_fifo_w_clk	Write enable for the input FIFO buffer. When this signal is asserted on the rising-edge of input_fifo_w_clk, input_data is written to the input FIFO buffer.
input_data	I	32	input_fifo_w_clk	Data bits to be written to the input FIFO buffer.
input_fifo_full	O	1	input_fifo_w_clk	Indicates that the input FIFO buffer is full and every write operation is ignored.

Table 3-6 Input data interface

The input data interface provides the means to feed data to the IP core for processing. The IP core has an asynchronous input buffer to overcome problems due to clock domain crossing when integrating the IP core in the system. The input data interface is made of a 32-bit data bus `input_data`, with simple write_enable/full control signals (`input_fifo_we`, `input_fifo_full`). The interface is synchronous to `input_fifo_w_clk`. When `input_fifo_full` is asserted, write operations are ignored, to avoid overwriting FIFO contents.

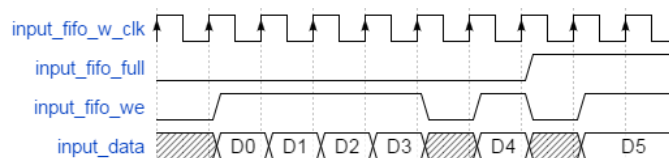


Figure 3-7 Input data interface timing

3.1.4.2 Output data interface

Port name	I/O	bits	Clock domain	Description
out_data_valid	O	1	core_clk	Indicates that data_out_I and data_out_Q outputs are valid.
data_out_I	O	Bout	core_clk	<p>Output signal on I-arm. Bout depends on:</p> <ul style="list-style-type: none"> - Ps the output symbol parallelism - whether the SRRC filter is enabled - the oversampling of the SRRC filter - The number of bits used to represent the symbols/the filtered output <p>When the SRRC filter is disabled: $B_{out} = P_s * B_{OUT_UNFILTERED}$, where $B_{OUT_UNFILTERED} = 8$</p> <p>When the SRRC filter is enabled $B_{out} = P_s * SRRC_OVERSAMPLING * B_{OUT_FILTERED}$, where $B_{OUT_FILTERED} = 12$</p>
data_out_Q	O	Bout	core_clk	Output signal on Q-arm. Bout is the same as data_out_I.
test_out_data_val id	O	1	core_clk	Indicates that test_data_out_I and test_data_out_Q outputs are valid. This port is used for test only and should be left unconnected in normal operating conditions.
test_data_out_I	O	$8 * P_s$	core_clk	Unfiltered I-arm output. This port is used for test only and should be left unconnected in normal operating conditions.
test_data_out_Q	O	$8 * P_s$	core_clk	Unfiltered Q-arm output. This port is used for test only and should be left unconnected in normal operating conditions.

Table 3-7 Output data interface

The modulated and encoded signal is produced through the output data interface. The in-phase component and quadrature component are presented on `data_out_I` and `data_out_Q`, respectively. In order to maximize the IP core throughput while keeping core clock frequency low, output samples are produced in parallel on B_{out} bits. The value of B_{out} depends on whether the SRRC filter is enabled or not. Each sample is represented on a different number of bits B_s depending on the presence of the SRRC filter: $B_s = 8$ bits in case the filter is not present and $B_s = 12$ bits in case the filter is present. In addition, when the filter is present, it interpolates the symbols by a factor of 4 (SRRC_OVERSAMPLING, ρ). Therefore, when the filter is not present, $B_{out} = P_s B_s$ (P_s is the number of symbols to be presented in parallel) and when the filter is present, $B_{out} = \rho P_s B_s$. `data_out_I[Ps* ρ *Bs-1:(Ps* ρ -1)*Bs]` and `data_out_Q[Ps* ρ *Bs-1:(Ps* ρ -1)*Bs]` represent the complex sample to be transmitted first, then `data_out_I[(Ps* ρ -1)*Bs-1:(Ps* ρ -2)*Bs]` and `data_out_Q[(Ps* ρ -1)*Bs-1:(Ps* ρ -2)*Bs]` is transmitted, and so on, the last complex sample is `data_out_I[Bs-1:0]` and `data_out_Q[Bs-1:0]`. `out_data_valid` indicates that the output complex sample is valid. Each sample is represented in 2's complement binary format.

The output data interface is synchronous to the core clock, `core_clk`.

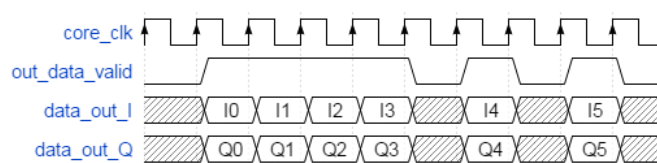


Figure 3-8 Output data interface timing

3.1.4.3 Configuration and status interface

Port name	I/O	bits	Clock domain	Description
core_ready	O	1	core_clk	When asserted, it indicates that the IP core is operating and is ready to receive the configuration.
config_error	O	1	core_clk	Asserted when the configuration written is not valid. In this case the IP core runs with the previous valid configuration. De-asserted when the configuration process is initialised.
write_config	I	1	core_clk	When asserted, the IP core enters the configuration process, if core_ready is asserted.
new_ModCod	I	5	core_clk	ModCod to be set in the next configuration process. This input must be stable on the rising edge of core_clk, when write config is asserted.
new_predistortion	I	80	core_clk	Predistortion values to be set in the next configuration process. This input must be stable on the rising edge of core_clk, when write config is asserted. This input is neglected when static predistortion is set as hard configuration parameter
current_ModCod	O	5	core_clk	Currently set ModCod.
see_inconsistent_state	O	2	core_clk	Asserted when an internal inconsistency due to SEE is detected. This could be due to EDAC on memories or invalid state in FSM.

Table 3-8 Configuration and status interface

Soft-configuration and information on the IP core status is provided through the configuration and status interface. The operational status is indicated by `core_ready` and `current_ModCod`. When `core_ready` is asserted, it indicates that the IP core is operating properly with the selected `current_ModCod`. When `core_ready` is asserted, it also indicates that the IP core is ready to receive a new configuration. The new configuration can be provided by the host system by asserting `write_config` together with the desired `new_ModCod` and `new_predistortion` when predistortion is set as soft configuration. The assertion of `write_config` initiates the configuration process: `core_ready` is de-asserted, the IP core finishes the transmission of the buffered data, then the configuration is applied. External loading of the look-up tables may be required depending on the IP core hard configuration settings. The configuration process terminates when `core_ready` is asserted. If `config_error` is not asserted, then the configuration process succeeded. If `config_error` is also asserted, then an invalid ModCod was selected for configuration and the configuration process did not succeed. In this case, the previous ModCod is kept as current ModCod and the `config_error` flag remains asserted until the next configuration.

When SEE mitigation is enabled, internal inconsistencies due to SEE is reported on `see_inconsistent_state` port. In particular, unrecoverable errors on EDAC or invalid state detected at FSMs is reported.

The configuration and status interface is synchronous to the core clock, `core_clk`.

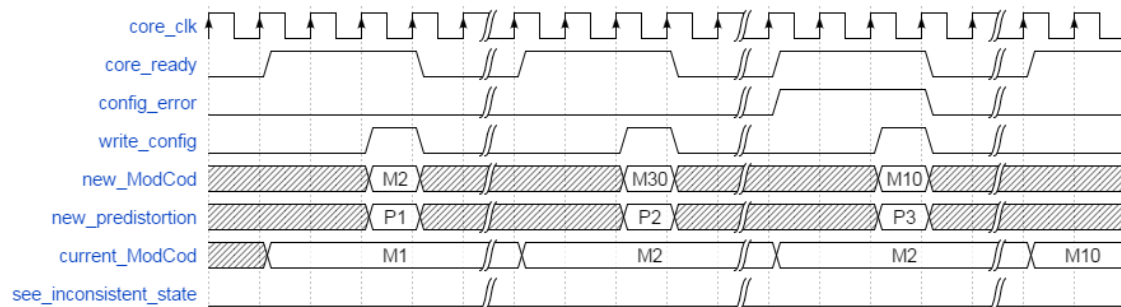


Figure 3-9 Configuration and status interface timing

This interface is handled by the configuration and status block

3.1.4.4 LUT loader interface

Port name	I/O	bits	Clock domain	Description
LUT_clk	I	1	LUT_clk	Clock for LUT loader section.
LUT_addr	I	16	LUT_clk	Address for writing the LUTs to the IP-core.
LUT_data	I	32	LUT_clk	Data to write the LUTs to the IP-core
LUT_ready	O	1	LUT_clk	Indicates that the LUT loader section is ready to receive new data. If LUT_we is asserted while LUT_ready is asserted, then the write operation succeeds. Otherwise, if LUT_we is asserted while LUT_ready is de-asserted, then the write operation fails and the host system has to repeat it. LUT_ready allows the integration in systems where LUT_clk frequency is higher than core_clk frequency.
LUT_we	I	1	LUT_clk	When asserted on the rising edge of core_clk, LUT_data is written to LUT_addr memory location.
LUT_req	O	1	LUT_clk	When asserted, the IP core requests the LUTs to be written to its internal memory, for the requested LUT_ModCod.
LUT_load_complete	I	1	LUT_clk	Asserted when the external LUT loader has finished writing data to the IP core, therefore the IP core exits the configuration state.
LUT_ModCod	O	5	LUT_clk	Indicates the ModCod related to the LUTs to be loaded to the IP core.

Table 3-9 LUT loader interface

The LUT loader interface allows the IP core to load LUT data from an external memory, so that hardware complexity can be considerably reduced. When a soft configuration is initiated and LUTs are not stored internally, the IP core requests new LUT values by asserting `LUT_req` signal, specifying the ModCod required on the `LUT_ModCod` port. Then, the IP core expects that data is written onto its internal RAM with the signals `LUT_addr`, `LUT_data` and `LUT_we`, according to a specific memory map. When the LUT loading is complete, the IP core expects `LUT_load_complete` asserted to continue its operations.

The LUT loader interface is synchronous to `LUT_clk` clock signal, in order to ease the integration with external memory.

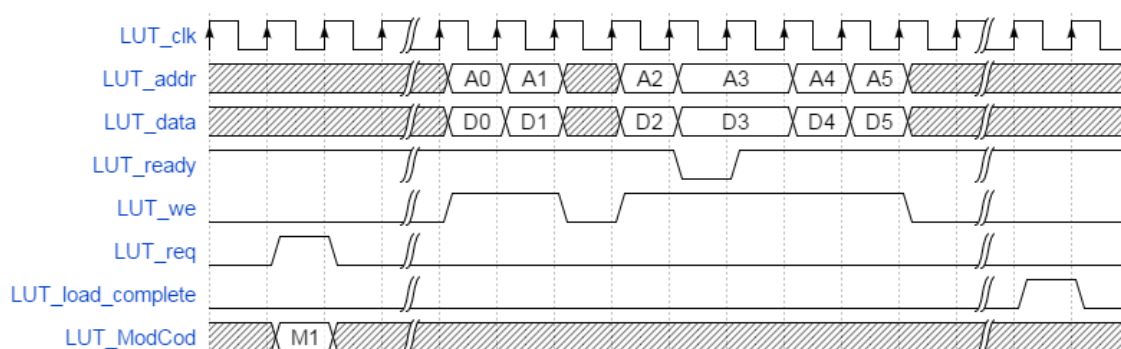


Figure 3-10 LUT loader interface timing

This interface is handled by the LUT loader block.

3.1.4.5 Clock, reset and enable

Port name	I/O	bits	Clock domain	Description
core_clk	I	1	core_clk	IP core clock
rst_n	I	1	core_clk	Global active-low asynchronous reset (synchronous/asynchronous, active-low/active-high, are based on hard configuration parameters)
enable	I	1	core_clk	Active-high enable signal

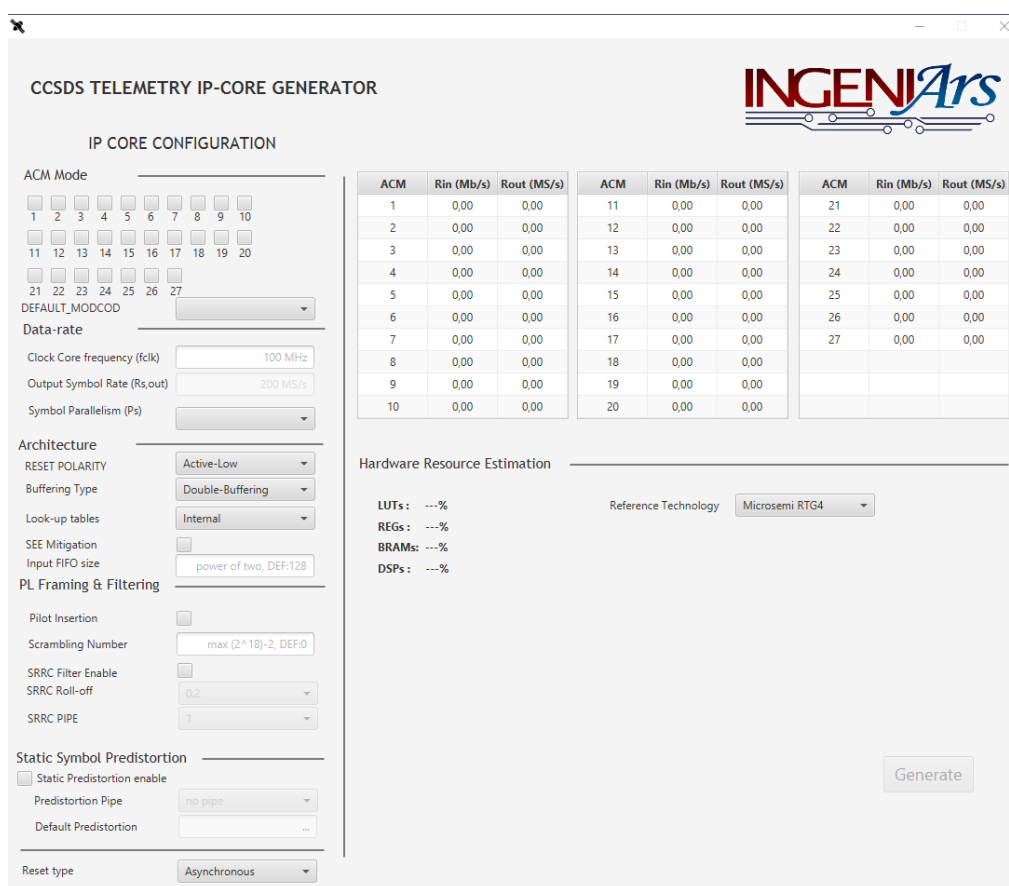
Table 3-10 Clock, reset and enable interface

3.2 Graphical User Interface

Besides the IP Core database, an IP Core generator software with a simple Graphical User Interface (see Figure 3-11) is provided to guide the end-user to the IP configuration process. Within the tool, the architectural parameters are translated at system-level, so that the user can exploit the most advanced features of the IP core without the deep knowledge required at architectural level. This will of course reduce IP core integration time while at the same time minimizing the risk of mistakes in the IP core configuration.

The end-user can then trade-off hardware resource occupation vs performance/functionality and set of ModCods supported in order to have an efficient implementation for the specific application and fit the IP-core onto the target technology (even on relatively small FPGAs such as Microsemi RTAX2000).

The IP-core generation tool guides the end-user towards the generation of a fully functional IP-core, giving estimations on the performance and complexity of the fitted instantiation. Note that the configuration system parameters are not all independent and not all the combinations generate a valid implementable IP Core. However the software takes care of this aspect automatically so the end-user can start configuring the IP-core by setting any of the parameters and while the configuration goes on, the non-compatible options will be grayed-out by the software. For example, if the user selects the Microsemi RTAX2000S as target technology with high ModCods selected, then the 200 Ms/s output baud-rate will be grayed-out.



CCSDS TELEMETRY IP-CORE GENERATOR

IP CORE CONFIGURATION

ACM Mode

1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27

DEFAULT_MODCOD

Data-rate

Clock Core frequency (fclk) 100 MHz

Output Symbol Rate (Rs,out) 200 MS/s

Symbol Parallelism (Ps)

Architecture

RESET POLARITY Active-Low

Buffering Type Double-Buffering

Look-up tables Internal

SEE Mitigation

Input FIFO size power of two, DEF:128

PL Framing & Filtering

Pilot Insertion

Scrambling Number max (2^18)-2, DEF:0

SRRC Filter Enable

SRRC Roll-off 0.2

SRRC PIPE 1

Static Symbol Predistortion

Static Predistortion enable

Predistortion Pipe no pipe

Default Predistortion

Reset type Asynchronous

Hardware Resource Estimation

LUTs: ---%
REGs: ---%
BRAMs: ---%
DSPs: ---%

Reference Technology Microsemi RTC4

Generate

Figure 3-11 IP Core Graphical User Interface

The IP core user can configure the IP-core through the panels on the left side of the GUI as described in the following.

ACM mode: by acting on the check-boxes on the left side of the GUI (in the *ACM mode* panel), the user can enable the desired ModCods.

Data rate: in this panel, the user can set the desired output data rate by acting on the two of the three parameters available: the core clock frequency, the output symbol rate and the output symbol parallelism. The GUI will, automatically set the third parameter not set by the user. An example is shown in Figure 3-12, where is possible note that the GUI highlights the maximum ModCod selected and furthermore it provides information about the minimum input data rate, for each ModCod, that the user has to respect in order to obtain the desired output symbol rate without holes in the transmission.

CCSDS TELEMETRY TRANSMITTER IP-CORE GENERATOR

IP CORE CONFIGURATION

ACM Mode

Core Clock frequency (fclk)

Output Symbol Rate (Rs,out)

Symbol Parallelism (Ps)

ACM	Rin (Mb/s)	Rout (Ms/s)
1	142.22	200
2	171.85	200
3	207.41	200
4	242.96	200
5	278.52	200
6	325.93	200
7	278.52	200
8	325.93	200
9	367.41	200
10	420.74	200

ACM	Rin (Mb/s)	Rout (Ms/s)
11	474.07	200
12	527.41	200
13	474.07	200
14	527.41	200
15	580.74	200
16	640.0	200
17	699.26	200
18	640.0	200
19	699.26	200
20	764.44	200

ACM	Rin (Mb/s)	Rout (Ms/s)
21	823.7	200
22	888.89	200
23	823.7	200
24	888.89	200
25	954.07	200
26	1013.33	200
27	1078.52	200

Hardware Resource estimation

Figure 3-12 Example of ModCodes ad output symbol rate setting

Architecture: through this panel is possible set the buffering strategy (single or double), if implement the algorithm LUTs internally or externally and finally if implements or not the SEE mitigation techniques within the IP Core (see Figure 3-13).

Architecture

Buffering Type: Double-buffering

Look-up tables: Internal

SEE Mitigation: External

Architecture

Buffering Type: Double-buffering

Look-up tables: Single-buffering

SEE Mitigation: ☐

Figure 3-13 Architecture panel options

PL Framing & Filtering: in this panel, the user can set the parameters related to the Physical Layer framing section. Figure 3-14 shows an example where Pilot insertion is set, the Scrambling number is 0, the SRRC filter is implemented with a Roll-off factor of 0.2.

PL Framing & Filtering

Pilot Insertion ☒

Scrambling Number

SRRC filter enable ☒

SRRC Roll-off

Static Symbol Predistortion

☐ Static Predistortion enable

Default Predistortion

Figure 3-14 PL Framing and Filtering setting example

Static Symbol Predistortion: a dedicated panel is provided to set the static predistortion parameters. The user can enable/disable static predistortion, and select the set of parameters used after IP core reset. The predistortion parameters can be provided to the GUI by file as shown in Figure 3-15. The file format is shown in Figure 3-16. Please note that the default predistortion parameters must be set according to the default modcod, which will be set after IP core reset as well.

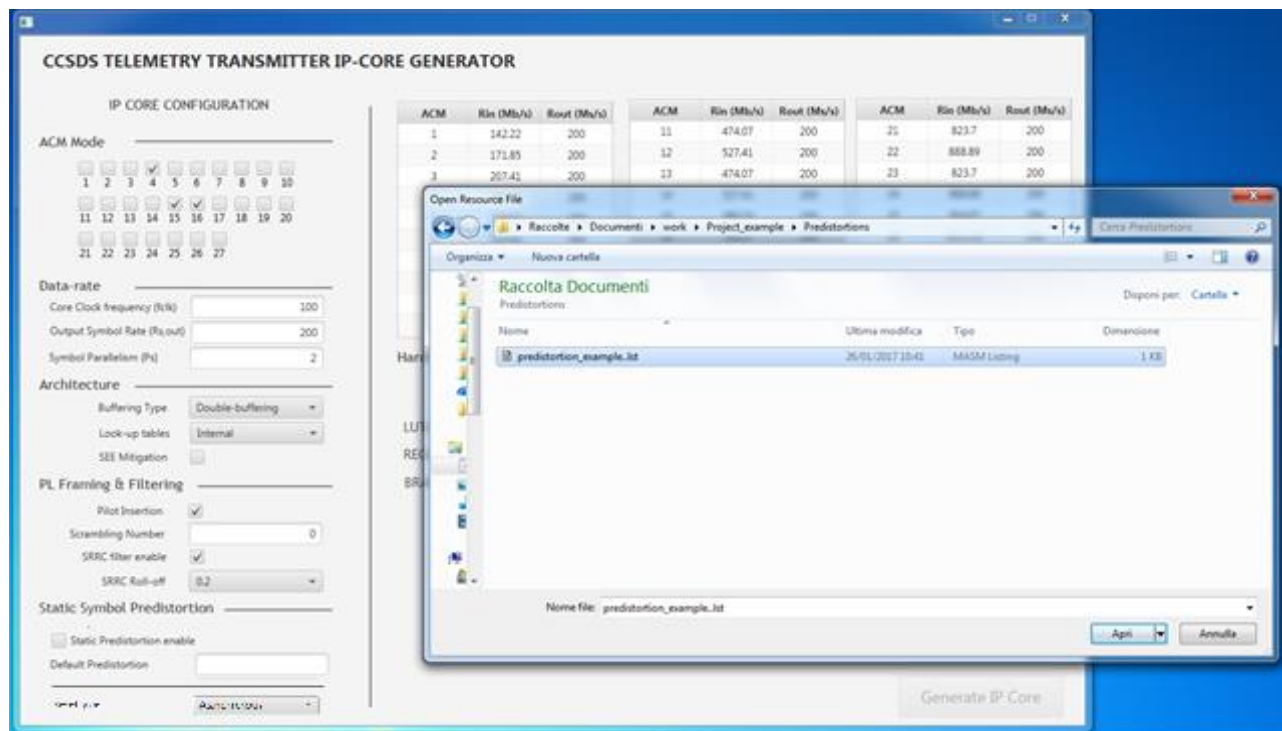


Figure 3-15 Predistortion file selection example

```

1  # RING 0
2  0.2 + 0.2j
3  # RING 1
4  0.2 + 0.2j
5  # RING 2
6  0.2 + 0.2j
7  # RING 3
8  0.2 + 0.2j
9  # RING 4
10 0.2 + 0.2j
11

```

Figure 3-16 Predistortion file format

Reference Technology: the user can select the target technology among one of those shown in Figure 3-17.

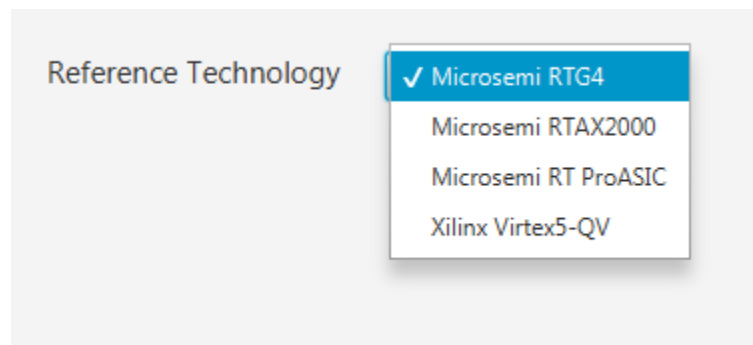


Figure 3-17 Target technology selection

Additionally, in case the device is fixed and cannot be changed for a certain application, the user can select the target technology as the first parameter, so that only the compatible options with this choice will be allowed by the GUI (feasible clock frequency, ModCods supported, etc.)

Hardware Resource estimation: Once all parameters are set, the GUI provides a hardware resource estimation of the IP Core being generating, and the *Generate IP Core* button becomes available. The resulting Database, will include the configured VHDL data base and all the scripts needed to perform the synthesis on the selected target technology.

4 Technology mapping results

The IP Core has been characterized on four different target technologies:

- Microsemi RTAX2000
- Microsemi ProASIC RT 3000L
- Microsemi RTG4
- Xilinx Virtex5-QV

Table 4-1 shows an overview of the ModCods that can be implemented for each technology considering single ModCod support. Table 4-1 is based on synthesis results for the IP Core.

ModCod	Microsemi RTAX2000		Microsemi ProASIC		Microsemi RTG4			Xilinx Virtex 5 QV		
	Ps = 1	Ps = 2	Ps = 1	Ps = 2	Ps = 1	Ps = 2	Ps = 4	Ps = 1	Ps = 2	Ps = 4
1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
4	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
6	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
8	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
9	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
10	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
11	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
12	✓		✓	✓	✓	✓	✓	✓	✓	✓
13	✓		✓	✓	✓	✓	✓	✓	✓	✓
14			✓	✓	✓	✓	✓	✓	✓	✓
15			✓	✓	✓	✓	✓	✓	✓	✓
16			✓	✓	✓	✓	✓	✓	✓	✓
17			✓	✓	✓	✓	✓	✓	✓	✓
18			✓	✓	✓	✓	✓	✓	✓	✓
19			✓	✓	✓	✓	✓	✓	✓	✓
20			✓	✓	✓	✓	✓	✓	✓	✓
21			✓	✓	✓	✓	✓	✓	✓	✓
22			✓	✓	✓	✓	✓	✓	✓	✓
23			✓	✓	✓	✓	✓	✓	✓	✓
24			✓	✓	✓	✓	✓	✓	✓	✓
25			✓	✓	✓	✓	✓	✓	✓	✓
26			✓		✓	✓	✓	✓	✓	✓
27			✓		✓	✓	✓	✓	✓	✓

Table 4-1 ModCod support on the different technologies

Regarding multiple-ModCod instantiations, the IP core was characterized for incremental configurations starting from ModCod 1 ([1], [1,2], [1,2,3] ... [1,2,3...,26, 27]) to find the maximum number of multiple ModCods supported in a single instantiation. Of course, the results presented here do not exclude multiple-ModCod instantiation with higher order ModCods by removing support for some lower order ModCods. Again, this assumption was made to limit the configuration space of the IP core. Table 4-2 shows multi-ModCod IP core instantiation synthesis results on the target technologies.

Technology	ModCods	Ps	LUT	Buffering	SRRC	Comb / Tiles	Regs	BRAM	DSP
Virtex 5 QV	ALL	2	INT	DOUBLE	YES	46%	24%	19%	3%
Virtex 5 QV	ALL	2	EXT	DOUBLE	YES	33%	23%	9%	3%
Virtex 5 QV	1 to 25	4	INT	DOUBLE	YES	72%	55%	26%	5%
Virtex 5 QV	ALL	4	EXT	DOUBLE	YES	68%	55%	13%	5%
RTG4	ALL	2	INT	DOUBLE	YES	78%	21%	21%	24%
RTG4	ALL	2	EXT	DOUBLE	YES	28%	17%	24%	24%
RTG4	1 to 24	4	INT	DOUBLE	YES	80%	31%	29%	37%
RTG4	ALL	4	EXT	DOUBLE	YES	55%	39%	34%	50%
ProASIC RT	1 to 8	1	INT	DOUBLE	NO	58%		31%	
ProASIC RT	1 to 20	1	EXT	DOUBLE	NO	44%		75%	
ProASIC RT	1 to 9	2	INT	DOUBLE	NO	44%		51%	
ProASIC RT	1 to 12	2	EXT	DOUBLE	NO	52%		76%	
RTAX2000	1 to 4	1	INT	DOUBLE	NO	69%	37%	31%	
RTAX2000	1, 2, 3, 7, 8, 9, 10, 13	1	EXT	DOUBLE	NO	60%	50%	78%	

Table 4-2 Multiple-ModCod instantiation synthesis results on the different technologies

Table 4-3 shows some IP core post place & route results. These results were obtained by targeting both a relevant number of supported ModCods within a single instantiation and 200 Mbaud for Virtex 5 and RTG4 and 100 Mbaud for RTAX2000 and ProASIC. Of course, a higher symbol rate can be achieved by reducing the number of supported ModCods and, conversely, a higher number of ModCods in a single instantiation can be achieved by targeting a lower symbol rate. Finally, results on commercial Xilinx Kintex 7 and Ultrascale devices are presented.

Technology	ModCods	Ps	LUT	Buffering	SRRC	Max clock freq (MHz)	Max input data rate (Mb/s)	Max output Symbol Rate (Mbaud)
RTAX2000	1 to 3	4	INT	DOUBLE	NO	30	124	120
ProASIC RT	1 to 2	4	INT	DOUBLE	YES	30	103	120
Virtex 5 QV	1 to 24	4	INT	DOUBLE	YES	50	888	200
Virtex 5 QV	25 to 27	4	INT	DOUBLE	YES	50	1078	200
RTG4	1 to 18	4	INT	DOUBLE	YES	56	716	224
Virtex 5 QV	1, 7, 13, 18, 23	2	INT	DOUBLE	YES	76	626	152
RTG4	1, 7, 13, 18, 23	2	INT	DOUBLE	YES	78	642	156
Kintex 7 Ultrascale xcku15pffva1156-2	ALL	4	INT	DOUBLE	YES	138	2980	553

Kintex 7 xc7k325t-ffg900	ALL	4	INT	DOUBLE	YES	117	2538	471
--------------------------	-----	---	-----	--------	-----	-----	------	-----

Table 4-3 IP core post place and route performance

4.1 BER Performance

Figure 4-1, Figure 4-2, and Figure 4-3 show the coded BER performance of the CCSDS 131.2-B Telemetry Transmitter IP core. The results were obtained by means of simulation with the bit-true model of the IP core, with SRRC filter enabled and ideal receiver. The floating point reference curve is also reported to evaluate the implementation loss, which overall is below 0.2 dB.

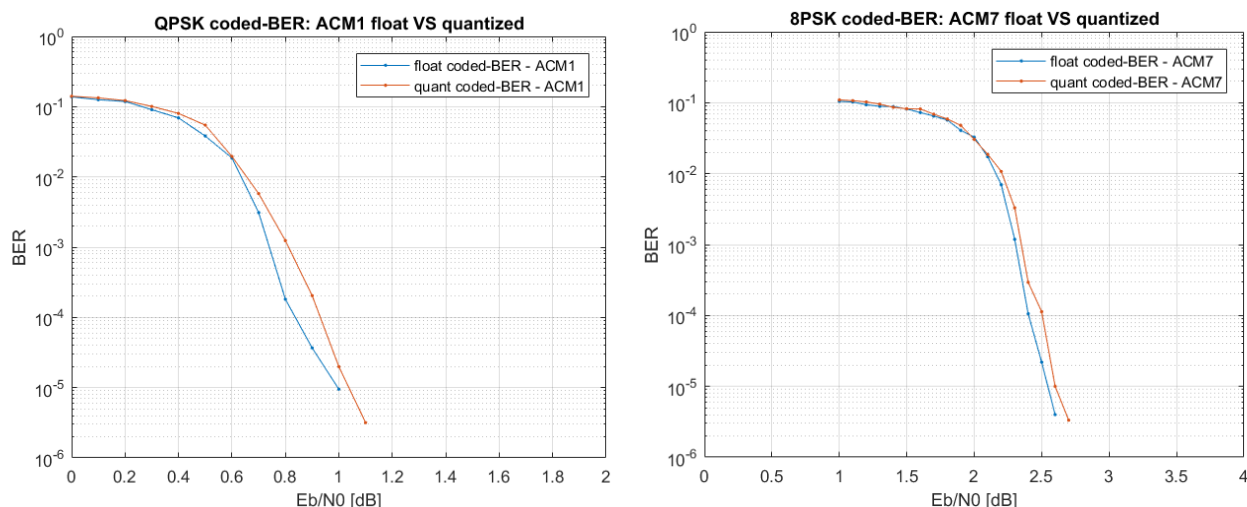


Figure 4-1 IP core BER performance for ACM1 and ACM7

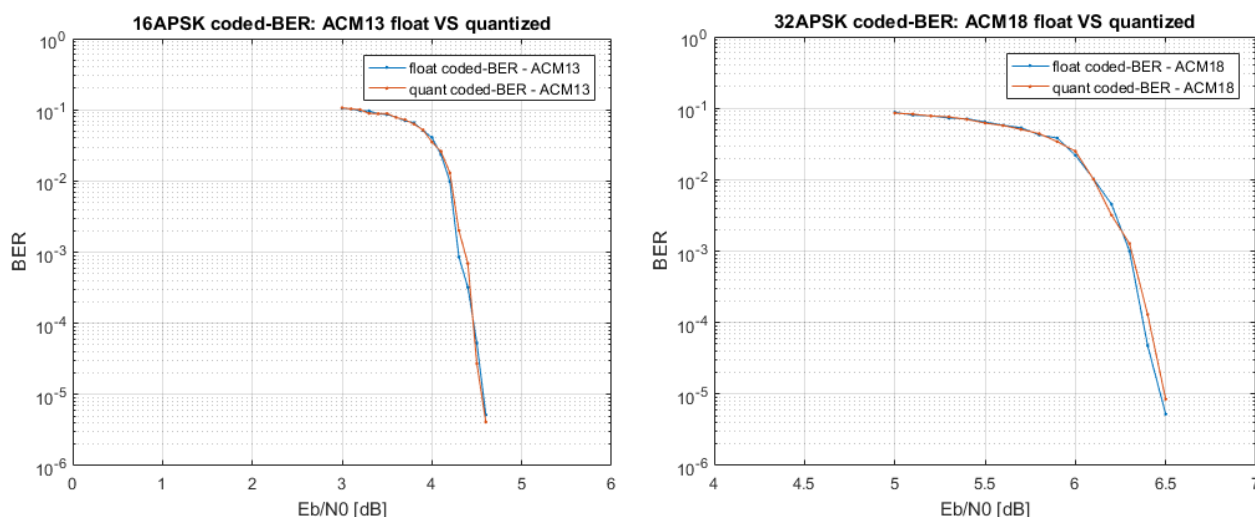


Figure 4-2 IP core BER performance for ACM13 and ACM18

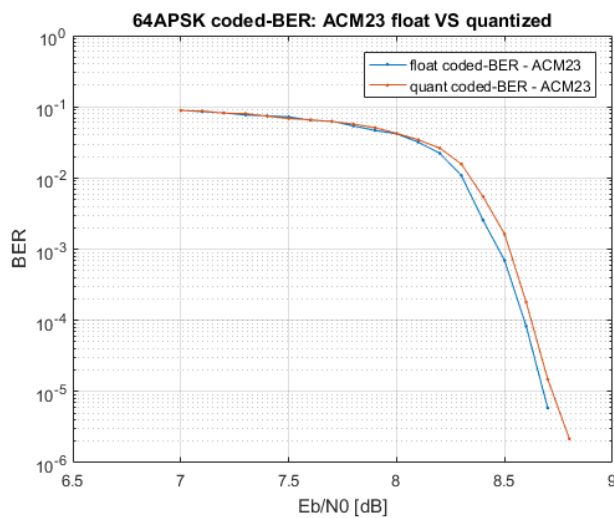


Figure 4-3 IP core BER performance for ACM23

5 Validation set-up

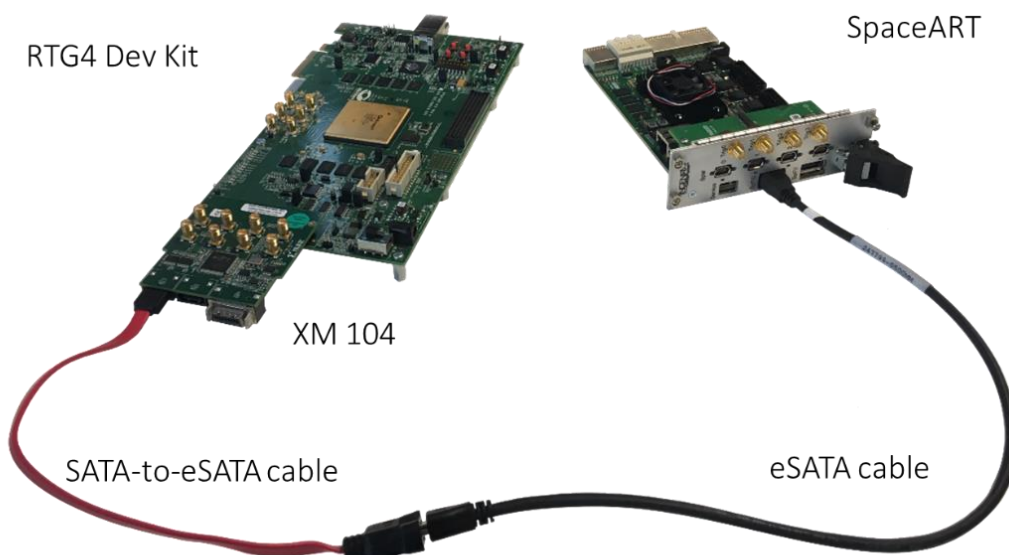
After the IP core verification, a hardware test has been performed for the validation purpose. The RTG4 Development Kit board has been used in the validation environment.

Because of the limited number of times that it is possible to program the RTG4 FPGA, only the most significant Hard Configuration has been tested. On the other hand, on real hardware it is possible to test much more Soft Configurations (all ModCodS are supported in this particular hardware implementation of the IP Core) for a longer period.

The validation set-up consists of:

- RTG4 development kit board. The following items are implemented on RTG4 FPGA:
 - The CCSDS IP Core configured to supports all Modcods, with external LUT configuration, double buffering, static symbol predistortion enabled, SRRC filter enabled, P_S = 4 selected.
 - IngeniArs SpaceFibre codec IP, used together with the IngeniArs SpaceArt EGSE, to implement the communication between RTG4 and the host PC;
 - Custom logic that elaborates the received packets from the host PC and consequently drives the CCSDS IP Core input and configuration interfaces;
 - Custom logic that format the CCSDS IP Core output and send it to the host PC through the SpaceFibre codec
- Xilinx XM104: this daughter board provides the SATA connection required for SpaceFibre communication;
- IngeniArs SpaceArt EGSE, configured with the CompactPCIe interface, used to communicate with the RTG4 development kit through a SpaceFibre link;
- Host PC with a Java application that uses the SpaceArt Java API to send/receive data to/from the CCSDS IP Core through the SpaceFibre link, and checks test results.

Figure 5-1 shows the HW-set-up implemented for the validation phase.



• *Figure 5-1 HW set-up*

An overview of the system block diagram in the FPGA is depicted in Figure 5-2

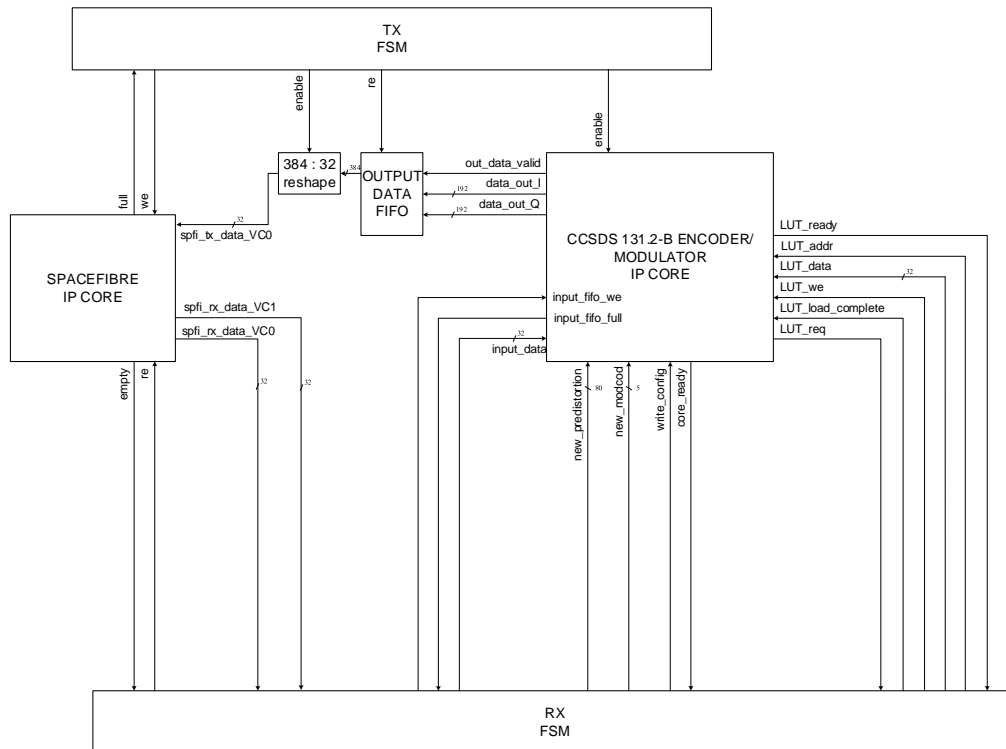


Figure 5-2 Validation System, FPGA Block Diagram

The SpaceFibre IP Core is instantiated with two virtual channels VC0 and VC1. VC0 is used to receive from the host PC data packet and configuration packets and send back data to the host PC. VC1 is used to receive the LUTs for the next configuration change from the host PC. The SpaceFibre virtual channel interface consists of a TX FIFO and a RX FIFO both with 32-bit word size.

The SpaceFibre link rate has been set to 2.5GB/s.

Two main FSMs handle the TX and RX data flows to/from the CCSDS IP Core:

- The RX FSM elaborates the received packets and drives the CCSDS input and configuration interfaces. When a data packet is recognized, the RX FSM feeds the CCSDS input interface with data coming from SpaceFibre
- The TX FSM has the main task of re-formatting the CCSDS output data in order to fit the SpaceFibre interface i.e. each 384-bit word coming from the CCSDS output interface is split in twelve 32-bit words that are written into the SpaceFibre VC0 TX FIFO.

The validation process involves the following steps:

1. Use the transmitter software model to generate a big set of stimuli/expected outputs files, in order to execute one of the validation tests.
2. Connect the RTG4 development board to SpaceART using an eSATA cable and turn on the board.
3. Run TestApp (Java application) on the Host PC.
4. From the TestApp Graphical User Interface, select the root folder containing all the files generated at point 1.
5. TestApp will pass stimuli to the board and will wait for the results.
6. When TestApp receives the results from the board, it generates a PASS/FAIL file.