

Next generation earth-to-space telecommand coding and synchronization: ground system design, optimization and software implementation

Original

Next generation earth-to-space telecommand coding and synchronization: ground system design, optimization and software implementation / Abello, R.; Baldi, M.; Carvalho, F.; Chiaraluce, F.; Fernandes, R.; Garelo, R.; Paolini, E.; Prata, R.. - In: EURASIP JOURNAL ON WIRELESS COMMUNICATIONS AND NETWORKING. - ISSN 1687-1472. - ELETTRONICO. - 2021:1(2021). [10.1186/s13638-021-02078-z]

Availability:

This version is available at: 11583/2970408 since: 2022-08-01T09:12:33Z

Publisher:

Springer Science and Business Media Deutschland GmbH

Published

DOI:10.1186/s13638-021-02078-z

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright


(Article begins on next page)

RESEARCH

Open Access



Next generation earth-to-space telecommand coding and synchronization: ground system design, optimization and software implementation

Ricard Abelló^{1†}, Marco Baldi^{2,6†}, Filipe Carvalho^{3†}, Franco Chiaraluce^{2,6*†} , Ricardo Fernandes^{3†}, Roberto Garelo^{4,6†}, Enrico Paolini^{5,6†} and Ricardo Prata^{3†}

*Correspondence:

f.chiaraluce@univpm.it

[†]Ricard Abelló, Marco Baldi, Filipe Carvalho, Franco Chiaraluce, Ricardo Fernandes, Roberto Garelo, Enrico Paolini and Ricardo Prata have contributed equally to this work

⁶ Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), Viale G.P. Usberti 181/A, 43124 Parma, Italy
Full list of author information is available at the end of the article

Abstract

The Consultative Committee for Space Data Systems, followed by all national and international space agencies, has updated the Telecommand Coding and Synchronization sublayer to introduce new powerful low-density parity-check (LDPC) codes. Their large coding gains significantly improve the system performance and allow new Telecommand services and profiles with higher bit rates and volumes. In this paper, we focus on the Telecommand transmitter implementation in the Ground Station baseband segment. First, we discuss the most important blocks and we focus on the most critical one, i.e., the LDPC encoder. We present and analyze two techniques, one based on a Shift Register Adder Accumulator and the other on Winograd convolution both exploiting the block circulant nature of the LDPC matrix. We show that these techniques provide a significant complexity reduction with respect to the usual encoder mapping, thus allowing to obtain high uplink bit rates. We then discuss the choice of a proper hardware or software platform, and we show that a Central Processing Unit-based software solution is able to achieve the high bit rates requested by the new Telecommand applications. Finally, we present the results of a set of tests on the real-time software implementation of the new system, comparing the performance achievable with the different encoding options.

Keywords: Encoding, Low-density parity-check codes, Space communications, Telecommand

1 Introduction

Space Telecommand (TC) systems from ground stations to space vehicles have traditionally been characterized by well-established requirements, which may be summarized as transmission of short messages (mostly for command and emergency), very high reliability (to prevent the execution of wrong commands), and receiver simplicity (to limit the on-board complexity) [1]. However, the uplink systems of new generation space missions shall support applications that are far beyond the transmission of

simple commands to the spacecraft on-board computer, leading to the need of updating the set of requirements [2, 3].

A common feature of these new uplink applications is to be more demanding in terms of bit rates and data volumes than traditional TCs. A notable example in this sense is file upload, intended for reprogrammable spacecraft instruments, with bit rates as high as 1 Mbps and block lengths in the order of 1–4 kbits. Another example is human support, intended for the emerging need to interact with astronauts orbiting the Earth or landing on the Moon or on Mars: voice, video, and Internet traffic, with bit rates as high as 20 Mbps and block lengths larger than 4 kbits. To support new applications and cope with the corresponding more stringent requirements, more sophisticated flight controllers and TC systems must be adopted on ground and more complex implementations must be introduced on-board, thus enabling the adoption of more powerful transmission techniques.

In response to the progressive rise of new applications and the consequent evolution of TC requirements, the Consultative Committee for Space Data Systems (CCSDS) [4] has constantly updated, over the years, its TC Synchronization and Channel Coding Recommendation [5]. For example, a cyclic redundancy check (CRC) for error detection and an automatic repeat request (ARQ) protocol were introduced to improve the reliability for larger message sets. Even more important, in 2017, the CCSDS has added two new powerful error correcting codes as an alternative to the traditional Bose–Chaudhuri–Hocquenghem (BCH) (63, 56) code, the latter representing the only uplink coding option in all previous versions of the CCSDS TC Recommendation.

More precisely, after a careful selection process involving several coding options, among which binary and non-binary Low-Density Parity-Check (LDPC) codes [6–8], two binary LDPC codes with parameters (128, 64) and (512, 256) (following classic notation, the first and the second number between brackets represent the codeword length and the information length, respectively) have been added to the Recommendation [5]. The new codes achieve large coding gains, compared with the one exhibited by the BCH code, heavily improving the TC link budget. This way, the upcoming Near Earth and Deep Space missions will be able to work at lower signal-to-noise ratio (SNR) values, increasing the uplink bit rates and data volumes for the same target performance. Obviously, the new codes require an extra-complexity both at the ground station transmitter and at the on-board receiver. The results of a study on the design and real-time implementation of the on-board receiver were presented in [9, 10].

In the current paper, we instead focus on the ground station transmitter base-band design, optimization and real-time implementation. In particular, once having checked that the most critical module at the ground station transmitter consists of the encoding stage, we propose two alternative methods for realizing it, and compare their performance in terms of achievable output bit rate, the latter being a direct measure of the efficiency of the algorithms: in practice, the higher the bit rate the faster the encoding procedure and, consequently, the lower the complexity which allows real-time implementation. More precisely, the first alternative algorithm is based on a Shift Register Adder Accumulator (SRAA) structure and the second algorithm on the Winograd convolution. Both exploit the block circulant structure of

the code matrix that characterizes the TC LDPC codes (as specified in Sect. 2.4) and allow a significant complexity reduction.

In order to implement the encoding algorithms, as well as the other side functionalities, we have first examined different hardware and software platforms, discussing the pros and cons related to their adoption. Based on the performance goals that are currently fixed for this kind of application (e.g., a target output bit rate of 2.048 Mbps) we have verified that a central processing unit (CPU)-based software solution turns out to be the most appropriate one to fulfill the increased bit rate requirements and to facilitate integration with the other blocks of the existing ground stations. So, we have implemented the encoders on a properly chosen CPU-based platform and we have evaluated the achievable performance.

The organization of the paper is as follows. In Sect. 2, we describe the TC Synchronization and Channel Coding sublayer and its critical modules including, besides LDPC encoding, randomization, BCH encoding and Command Link Transmission Unit (CLTU) generation. In Sect. 3, we focus on efficient LDPC encoding, describing the structure and rationale of the SRAA and the Winograd convolution. In Sect. 4, we present hardware and software platforms suited to Ground Station implementation and show that a CPU-based software solution is compliant with the goals and constraints of the study, corresponding to the bit-rate requirements imposed by the new applications. In Sect. 5, we present the results for the real-time software implementation of the new TC Synchronization and Channel Coding sublayer and we quantify the bit rate improvement achieved by the two encoding methods for the two different LDPC codes. Finally, in Sect. 6, we draw some conclusions.

2 On-ground telecommand synchronization and channel coding sublayer

The CCSDS Recommendation [5] provides all elements to implement the Earth-to-Space TC transmitter. The components of both the Ground and Space elements are summarized in Fig. 1, with reference to the Open Systems Interconnection (OSI) reference model. The TC Frames contain the messages to be delivered to the space vehicles. They are processed in the Channel Coding and Synchronization sublayer to produce the CLTUs that are actually transmitted over the channel according to the physical layer.

A Field-Programmable Gate Array (FPGA) implementation of the in-space TC receiver, working in real time, was presented in [9, 10]. In contrast, in this paper we address the ground segment transmission chain. The main blocks of the Channel Coding and Synchronization sublayer are illustrated in Fig. 2 and consist of:

- **Randomizer**, for bit transition generation;
- **Encoder**, performing either BCH or LDPC coding;
- **CLTU generation**, for frame synchronization and final data unit generation.

As shown in Fig. 2, the order of the randomizer and the encoder is different for BCH and LDPC encoding. In the remainder of the section, we describe the most important features of the input frames and the layer subsystems.

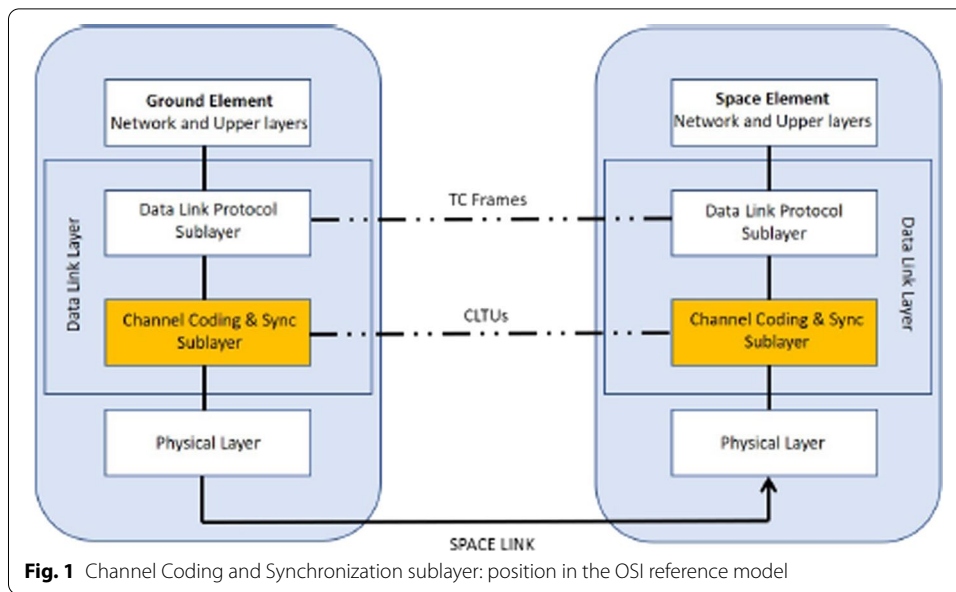


Fig. 1 Channel Coding and Synchronization sublayer: position in the OSI reference model

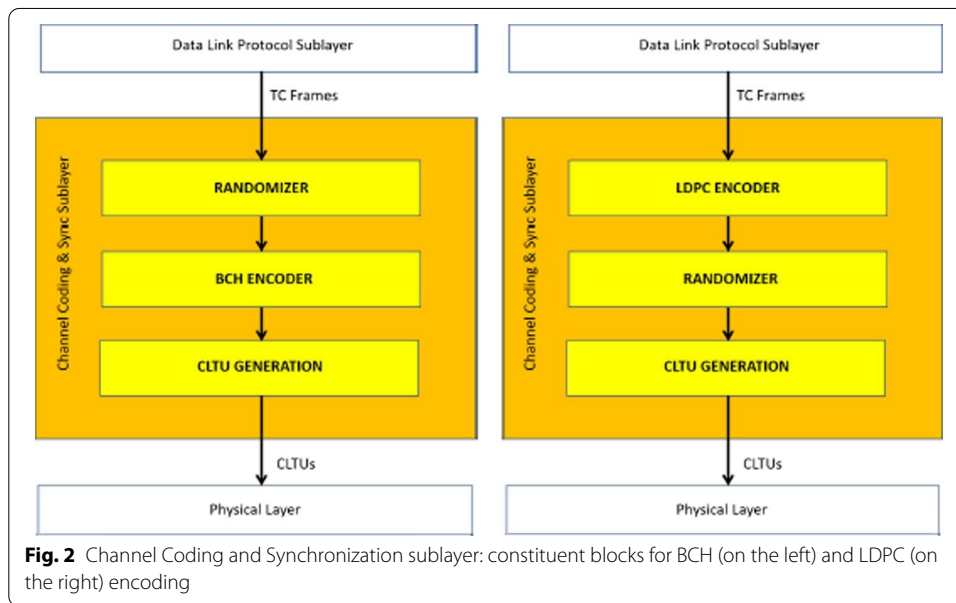


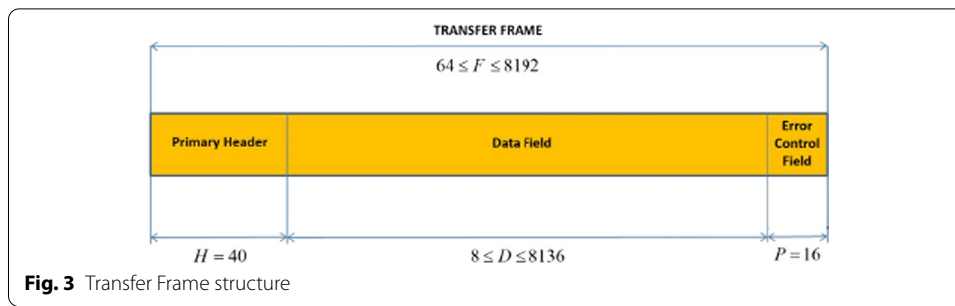
Fig. 2 Channel Coding and Synchronization sublayer: constituent blocks for BCH (on the left) and LDPC (on the right) encoding

2.1 Transfer Frame

The structure of a TC Transfer Frame (TF) generated by the Data Link Protocol sublayer is shown in Fig. 3.

As described in [11], a TF is composed of:

- Transfer Frame Primary Header: $H = 40$ bits. It contains control information like the Spacecraft Identifier, the Frame length and the Frame Sequence Number.
- Transfer Frame Data Field: $8 \leq D \leq 8136$ bits. It contains the message to be delivered to the space vehicle.



- Transfer Frame Error Control Field: $P = 16$ parity bits generated by a CRC code to reduce the risk of undetected errors. This field is optional and is obviously absent when the CRC code is unused. When present, the CRC code accepts in input the $H + D$ bits of the transfer frame and produces $P = 16$ bits at its output, which are appended, as shown in Fig. 3, at the end of the TF. The CRC generator polynomial is: $g_{CRC}(X) = X^{16} + X^{12} + X^5 + 1$.

If the CRC code is used, the total TF length is: $F = H + D + P$ and, therefore, we have $64 \leq F \leq 8192$ bits. Otherwise, when the CRC is not used, we have $48 \leq F \leq 8176$ bits.

When the TF length F is not an integer multiple of the encoder input length, a sequence of alternating '0' and '1' bits, starting with a '0', is appended until an integer multiple is obtained.

2.2 Randomizer

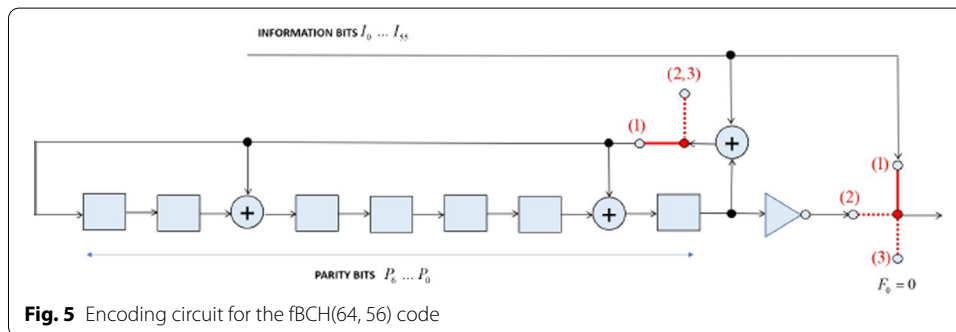
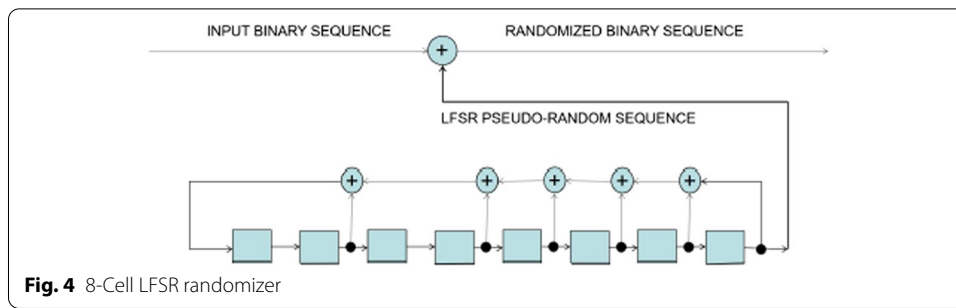
The purpose of the randomizer is to increase the randomness of the transmitted binary sequence. This facilitates on-board synchronization, which requires a sufficient symbol transition density and no long runs. The randomizer generates a pseudorandom binary sequence that is ex-ORed with the data block to be transmitted. Data randomization is optional for BCH coding and mandatory for LDPC coding. As already shown in Fig. 2, the order of application of randomization and channel coding should differ depending on the adopted coding scheme. More specifically, the following sequence should be applied at the transmitter side when using LDPC coding:

- TFs \rightarrow [Fill data if needed] \rightarrow [LDPC encoding] \rightarrow [Randomization] \rightarrow [CLTU generation] \rightarrow CLTUs.

On the other hand, the following sequence should be applied at the transmitter side when using BCH coding and the optional data randomization is employed:

- TFs \rightarrow [Fill data if needed] \rightarrow [Randomization] \rightarrow [BCH encoding] \rightarrow [CLTU generation] \rightarrow CLTUs.

The CCSDS TC randomizer is the 8-cell linear-feedback shift register (LFSR) with polynomial description $p(D) = 1 + D + D^2 + D^3 + D^4 + D^6 + D^8$ depicted in Fig. 4, which generates a maximum length M-sequence with period $N = 2^8 - 1 = 255$ bits. All the LFSR cells are preset to 1 at the beginning of a new block to be randomized. Noting by L the length of the data unit to be randomized (after LDPC encoding or before BCH



encoding), the first L bits generated by the LFSR are ex-ORed to obtain the L randomized bits.

The real-time implementation of the randomizer can be done by using the logic diagram depicted in Fig. 4. As an alternative, we note that, since the LFSR is always preset to the all-one state at the beginning of a new block, the sequence generated by the randomizer is always the same. As a consequence, it is also possible to directly store the 255-bit entire M-sequence period in a memory and sum it when requested. Neither technique is critical for real-time implementation: the first one was used in the presented implementation.

2.3 BCH code

Channel codes are used to detect or correct the errors introduced by noise and other impairments. Given an information frame of k bits, the encoder of a $C(n, k)$ code outputs an n -bit codeword. Usually the encoder is systematic, i.e., the first k bits of each codeword coincide with the corresponding information frame, and linear, i.e., the $r = n - k$ parity bits are obtained as a linear combination of the k information bits.

Before the introduction of LDPC codes, the only channel code available for TC systems was the BCH(63, 56). This is a cyclic code with generator polynomial $g(X) = X^7 + X^6 + X^2 + 1$. The code may be regarded as an expurgated (63, 57) Hamming code, obtained by allowing even-weight codewords only. Since its minimum distance is $d_{\min} = 4$, it can be used in Single Error Correction (SEC) mode, to correct any single error and detect any double error (plus all odd errors), or in triple error detection (TED) mode, to detect any single, double, or triple error (plus all odd errors).

The BCH encoder logic diagram is shown in Fig. 5. The encoder performs systematic encoding of a block $\mathbf{u} = (I_0, I_1, \dots, I_{55})$ of 56 bits. At the beginning of the block, all the

shift register cells are preset to 1 and the two switches are set to position 1. The 56 information bits of \mathbf{u} enters the encoder. They are automatically propagated to the output and processed by the feedback shift register circuit. When all the 56 information bits have been input, the two switches are set to position 2. In the seven subsequent clock cycles the complement of the seven BCH parity-check bits $\mathbf{p} = (P_0, P_1, \dots, P_6)$ are output by the encoder, and concatenated with \mathbf{u} . Finally, the switches are set to position 3, and a filler bit $F_0 = 0$ is appended, in such a way as to have an overall codeword length which is an integer number of octets. The 64-bit codeword is then

$$\mathbf{c} = (I_0, I_1, \dots, I_{55}, P'_0, P'_1, \dots, P'_6, F_0) \quad (1)$$

where $P'_i = \text{XOR}(P_i, 1)$, $i = 0, \dots, 6$, is indeed the complement of bit P_i . For the sake of convenience, we will denote the whole code made by the BCH(63, 56) code and the filler bit by the symbol fBCH(64, 56).

As we will show in Sect. 5, the BCH encoder real-time implementation has not been critical for our study.

2.4 LDPC codes

LDPC codes are state-of-the-art error correcting codes [12, 13] and appear in several international communication standards. In 2017, two new LDPC codes were introduced in the CCSDS TC recommendation [5], with parameters (128, 64) and (512, 256), respectively. These codes were obtained with a protograph construction [14]. Their parity check matrices are block-circulant, i.e., they are composed by elementary square blocks, where each row is a cyclic shift of the row above. The matrices of the two CCSDS TC LDPC codes are described in Fig. 6. The elementary blocks have size $Q \times Q$ bits where $Q = k/4 = n/8$; then, we have $Q = 16$ for the (128, 64) LDPC code and $Q = 64$ for the (512, 256) LDPC code.

The elementary building blocks are the $Q \times Q$ \mathbf{I}_Q and $\mathbf{0}_Q$ identity and zero matrices, respectively, while Φ is the first right circular shift of \mathbf{I}_Q . Explicitly, this means that Φ has a nonzero entry at row i and column j if and only if $j = i + 1 \pmod Q$. Consequently, Φ^2 is the second right circular shift of \mathbf{I}_Q , that is, Φ^2 has a nonzero entry at row i and column j if and only if $j = i + 2 \pmod Q$, and so on. Obviously, $\Phi^0 = \mathbf{I}_Q$. The operator \oplus indicates modulo-2 element-wise matrix addition.

$$\mathbf{H}_{64 \times 128} = \begin{bmatrix} \mathbf{I}_Q \oplus \Phi^7 & \Phi^2 & \Phi^{14} & \Phi^6 & \mathbf{0}_Q & \Phi^0 & \Phi^{13} & \mathbf{I}_Q \\ \Phi^6 & \mathbf{I}_Q \oplus \Phi^{15} & \Phi^0 & \Phi^1 & \mathbf{I}_Q & \mathbf{0}_Q & \Phi^0 & \Phi^7 \\ \Phi^4 & \Phi^1 & \mathbf{I}_Q \oplus \Phi^{15} & \Phi^{14} & \Phi^{11} & \mathbf{I}_Q & \mathbf{0}_Q & \Phi^3 \\ \Phi^0 & \Phi^1 & \Phi^9 & \mathbf{I}_Q \oplus \Phi^{13} & \Phi^{14} & \Phi^1 & \mathbf{I}_Q & \mathbf{0}_Q \end{bmatrix}$$

$$\mathbf{H}_{256 \times 512} = \begin{bmatrix} \mathbf{I}_Q \oplus \Phi^{63} & \Phi^{30} & \Phi^{50} & \Phi^{25} & \mathbf{0}_Q & \Phi^{43} & \Phi^{62} & \mathbf{I}_Q \\ \Phi^{56} & \mathbf{I}_Q \oplus \Phi^{61} & \Phi^{50} & \Phi^{23} & \mathbf{I}_Q & \mathbf{0}_Q & \Phi^{37} & \Phi^{26} \\ \Phi^{16} & \Phi^0 & \mathbf{I}_Q \oplus \Phi^{55} & \Phi^{27} & \Phi^{56} & \mathbf{I}_Q & \mathbf{0}_Q & \Phi^{43} \\ \Phi^{35} & \Phi^{56} & \Phi^{62} & \mathbf{I}_Q \oplus \Phi^{11} & \Phi^{58} & \Phi^3 & \mathbf{I}_Q & \mathbf{0}_Q \end{bmatrix}$$

Fig. 6 Parity check matrices of the LDPC codes

For any code, given the parity check matrix \mathbf{H} we can build a generator matrix \mathbf{G} by using the equation $\mathbf{GH}^T = \mathbf{0}_{k \times r}$ where superscript T denotes transposition, $r = n - k$ is the number of parity check symbols, and $\mathbf{0}_{k \times r}$ is the all-zero matrix with k rows and r columns. For the CCSDS TC codes, a systematic matrix \mathbf{G} can be defined, with the structure reported in Fig. 7. Again, \mathbf{G} is a block-circulant matrix composed by \mathbf{I}_Q and $\mathbf{0}_Q$ identity and null matrices, respectively, plus circulant $Q \times Q$ square matrices \mathbf{W}_{ij} .

Remarkably, even if the initial parity check matrix is sparse, this is not the case for the generator matrix. To better understand this claim, the scatter chart for the parity check and the generator matrices of the LDPC(128, 64) code are depicted in Figs. 8 and 9, respectively, where blue dots represent bits equal to 1 while all remaining bits are 0. We can note that the \mathbf{G} right side is dense (about $kr/2$ elements are equal to 1). As a consequence, in practical implementations the encoder complexity, when

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_Q & \mathbf{0}_Q & \mathbf{0}_Q & \mathbf{0}_Q & \mathbf{W}_{1,1} & \mathbf{W}_{1,2} & \mathbf{W}_{1,3} & \mathbf{W}_{1,4} \\ \mathbf{0}_Q & \mathbf{I}_Q & \mathbf{0}_Q & \mathbf{0}_Q & \mathbf{W}_{2,1} & \mathbf{W}_{2,2} & \mathbf{W}_{2,3} & \mathbf{W}_{2,4} \\ \mathbf{0}_Q & \mathbf{0}_Q & \mathbf{I}_Q & \mathbf{0}_Q & \mathbf{W}_{3,1} & \mathbf{W}_{3,2} & \mathbf{W}_{3,3} & \mathbf{W}_{3,4} \\ \mathbf{0}_Q & \mathbf{0}_Q & \mathbf{0}_Q & \mathbf{I}_Q & \mathbf{W}_{4,1} & \mathbf{W}_{4,2} & \mathbf{W}_{4,3} & \mathbf{W}_{4,4} \end{bmatrix}$$

Fig. 7 Generator matrix structure

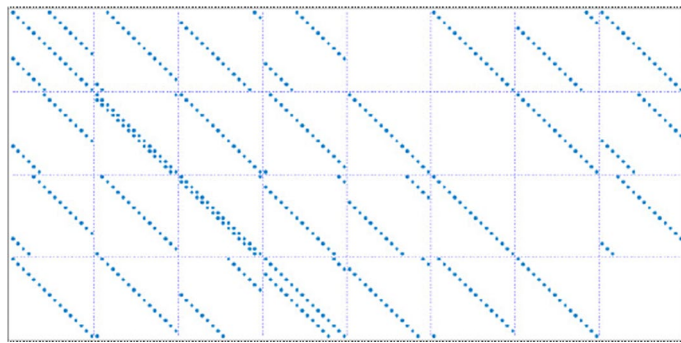


Fig. 8 Scatter chart for the parity check matrix \mathbf{H} of the LDPC(128, 64) code

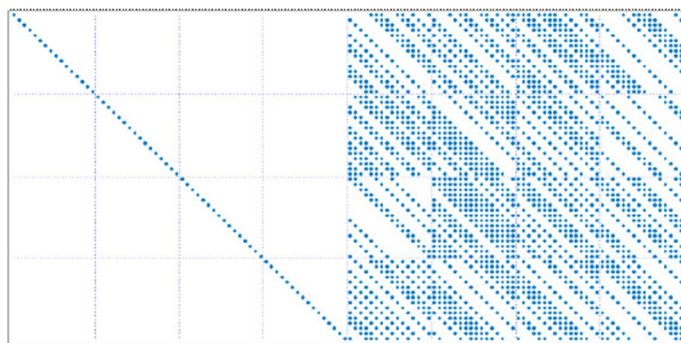


Fig. 9 Scatter chart for the generator matrix \mathbf{G} of the LDPC(128, 64) code

encoding is realized through matrix \mathbf{G} , cannot be neglected and should be carefully investigated. This will be the subject of Sect. 3.

2.5 CLTU generation

The CLTU generation block performs codeword concatenation, prepends a start sequence and (optionally) appends a tail sequence. As discussed above, three coding options are available: the fBCH(64, 56) code, the short LDPC(128, 64) code and the long LDPC(512, 256) code. As mentioned, filler bits may be added in order for the TF length F to be an integer multiple m of the information word length k of the selected error control code (then, respectively, 56 or 64 or 256 bits for the fBCH or short LDPC or long LDPC code). After coding, since $F = mk$, the payload is segmented into m blocks, each of which composed by n bits (respectively, 64 or 128 or 512 bits for the fBCH or short LDPC or long LDPC code). The prepended start sequence length depends on the chosen code, namely:

- For BCH coding, the 16-bits start sequence 1110101110010000 must be used.
- For LDPC coding, the 64-bits start sequence 0347 76C7 2728 95B0_{HEX} (in hexadecimal form) must be used.

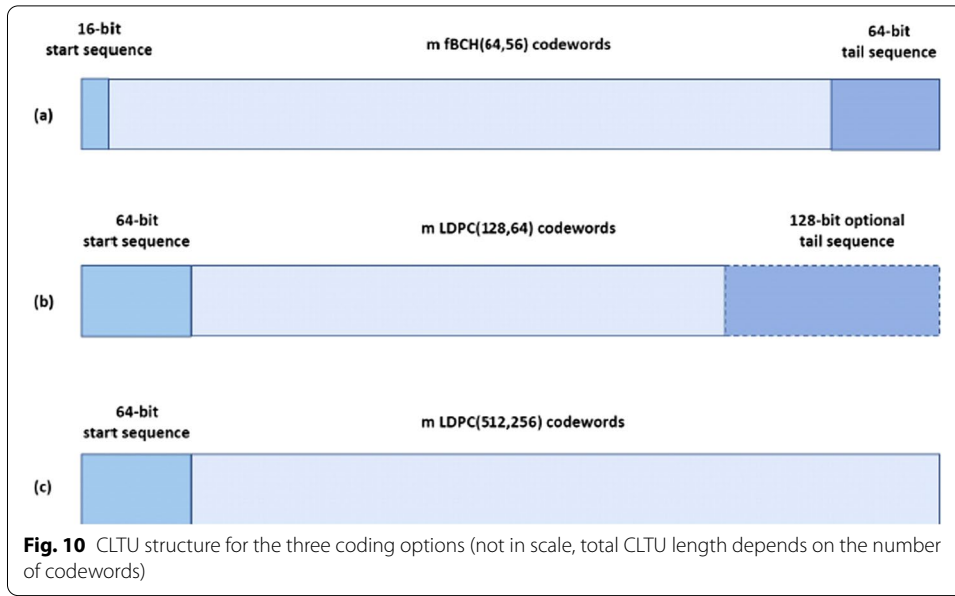
The reason why LDPC codes need longer start sequences is that, thanks to their coding gain, they are able to work at lower SNR. Then, a longer sequence is used to reliably acquire CLTU synchronization [9, 10]. The CLTU generation block may also append a tail sequence for some configurations:

- For BCH coding, the mandatory 64-bit tail sequence C5C5 C5C5 C5C5 C579_{HEX} is used.
- For the LDPC(128, 64) code, the following optional 128-bit tail sequence: 5555 5556 AAAA AAAA 5555 5555 5555 5555_{HEX} may be appended. In particular, this tail sequence is mandatory if we want to apply decoders based on Most Reliable Basis (MRB) [15] for LDPC complete decoding [9, 10].
- For the LDPC(512, 256) code, no tail sequence is appended (in fact, the MRB complexity becomes intractable for this code).

The final CLTU structure for the three coding options is shown in Fig. 10. The CLTU generation block is quite simple and, as shown in Sect. 5, is not critical for real-time implementation.

3 Efficient TC LDPC encoding strategies

Given the information vector \mathbf{u} and the generator matrix \mathbf{G} we can always obtain the codeword as $\mathbf{c} = \mathbf{u}\mathbf{G}$. We will call this method “the $\mathbf{u}\mathbf{G}$ encoder”. As mentioned in the previous section, the generator matrix of the two TC LDPC codes is block-circulant, but not sparse. Since \mathbf{G} is dense, the number of elements equal to 1 is approximately $kr/2 + k$. Note that, since the two LDPC codes have code-rate equal to 1/2, this is equal to $k^2/2 + k$; then, the complexity of LDPC encoding increases quadratically with k . As such, LDPC encoding complexity is not negligible and the encoder turns to be the most



critical block for an implementation of the new TC transmitter baseband system capable to support high data rates. For such a reason, in this section we present and discuss two efficient alternative encoding solutions. Both of them exploit the block circulant structure of the generator matrix \mathbf{G} .

3.1 SRAA-based encoding

The first alternative approach is based on the method described in [16]. Let us consider the structure of the matrix \mathbf{G} reported in Fig. 7. As mentioned above, each $\mathbf{W}_{i,j}$ is a square $Q \times Q$ (generally dense) circulant matrix, where any row is obtained as the right circular shift of the previous row by one position. To exploit the matrix structure, let us write the information block as $\mathbf{u} = (\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4)$, where all \mathbf{u}_i 's, $i = 1, \dots, 4$, have length Q bits. Since the encoder is systematic, the codeword has structure $\mathbf{c} = (\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4)$, where each Q -bit parity vector is given by:

$$\mathbf{p}_j = \mathbf{u}_1 \mathbf{W}_{1,j} + \mathbf{u}_2 \mathbf{W}_{2,j} + \mathbf{u}_3 \mathbf{W}_{3,j} + \mathbf{u}_4 \mathbf{W}_{4,j}. \quad (2)$$

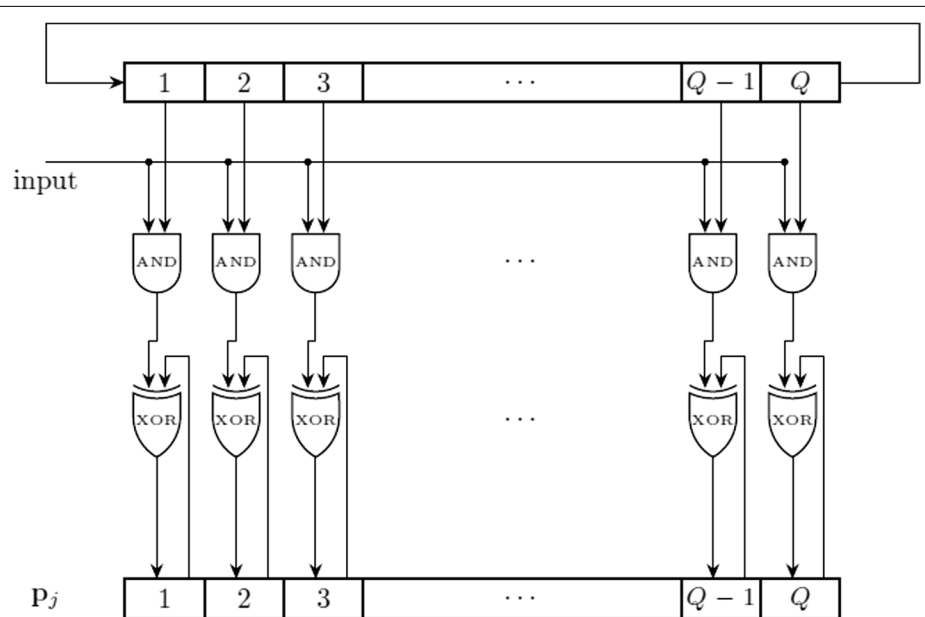
Next, let us focus on one of the elementary terms $\mathbf{u}_i \mathbf{W}_{i,j}$. As mentioned, we can exploit the fact that all $\mathbf{W}_{i,j}$ matrices are $Q \times Q$ circulant blocks. Let us denote by $\mathbf{g}_{i,j}^{(l)}$ the l -th row of $\mathbf{W}_{i,j}$, for $l = 1, \dots, Q$. If we write $\mathbf{u}_i = (u_{(i-1)Q+1}, u_{(i-1)Q+2}, \dots, u_{iQ})$, the elementary term is given by:

$$\mathbf{u}_i \mathbf{W}_{i,j} = u_{(i-1)Q+1} \mathbf{g}_{i,j}^{(1)} + u_{(i-1)Q+2} \mathbf{g}_{i,j}^{(2)} + \dots + u_{iQ} \mathbf{g}_{i,j}^{(Q)}. \quad (3)$$

An example, referred to the (128, 64) code, is shown in Fig. 11. We can see that each bit multiplies a row, but each row is the cyclic shift of the row above. Then we can implement this basic operation by the circuit depicted in Fig. 12.

According to Fig. 12, the logic diagram of the SRAA is made by a Q -bit shift register, Q AND gates and Q XOR gates. The first row of $\mathbf{W}_{i,j}$ is pre-loaded in the register. The Q bits of \mathbf{u}_i are serially input to the circuit. At each clock cycle of phase i , all elements

$u_{i,1}$	1	0	1	1	0	0	1	0	1	0	0	1	1	0	0	1
$u_{i,2}$	1	1	0	1	1	0	0	1	0	1	0	0	1	1	0	0
$u_{i,3}$	0	1	1	0	1	1	0	0	1	0	1	0	0	1	1	0
$u_{i,4}$	0	0	1	1	0	1	1	0	0	1	0	1	0	0	1	1
$u_{i,5}$	1	0	0	1	1	0	1	1	0	0	1	0	1	0	0	1
$u_{i,6}$	1	1	0	0	1	1	0	1	1	0	0	1	0	1	0	0
$u_{i,7}$	0	1	1	0	0	1	1	0	1	1	0	0	1	0	1	0
$u_{i,8}$	0	0	1	1	0	0	1	1	0	1	1	0	0	1	0	1
$u_{i,9}$	1	0	0	1	1	0	0	1	1	0	1	1	0	0	1	0
$u_{i,10}$	0	1	0	0	1	1	0	0	1	1	0	1	1	0	0	1
$u_{i,11}$	1	0	1	0	0	1	1	0	0	1	1	0	1	1	0	0
$u_{i,12}$	0	1	0	1	0	0	1	1	0	0	1	1	0	1	1	0
$u_{i,13}$	0	0	1	0	1	0	0	1	1	0	0	1	1	0	1	1
$u_{i,14}$	1	0	0	1	0	1	0	0	1	1	0	0	1	1	0	1
$u_{i,15}$	1	1	0	0	1	0	1	0	0	1	1	0	0	1	1	0
$u_{i,16}$	0	1	1	0	0	1	0	1	0	0	1	1	0	0	1	1

Fig. 11 Example of \mathbf{uW} multiplication**Fig. 12** SRAA logic diagram

of the shift register are multiplied (logical AND) by the bit $u_{(i-1)Q+t}$, $t = 1, \dots, Q$, and the result is accumulated (logical XOR) in the corresponding position of the second Q -bits register. After Q clock cycles we get $\mathbf{u}_i \mathbf{W}_{i,j}$. The calculation of each \mathbf{p}_j requires

to compute four elementary products $\mathbf{u}_i \mathbf{W}_{i,j}$; then, to perform the above computation four times, one per each $i = 1, \dots, 4$, as shown in Fig. 13.

This encoder scheme, specifically tailored to hardware (e.g., FPGA) implementation, may provide advantages with respect to the classic \mathbf{uG} rule even in the frame of a software implementation, as we will show in Sect. 5.

3.2 Winograd-based encoding

In this section, we present a second alternative encoding technique, based on the Winograd convolution [17]. We consider again the expression of the codeword $\mathbf{c} = (\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4)$, where each Q -bit parity vector is given by (2) and we focus again on a generic elementary term $\mathbf{u}_i \mathbf{W}_{i,j}$. The block matrix $\mathbf{W}_{i,j}$ is a circulant matrix. As it is well known, any circulant matrix is a Toeplitz matrix, meaning that all elements on any descending diagonal are constant. An example for a 4×4 circulant matrix is as follows (on the right the general case of a Toeplitz matrix; on the left the particular case of a circulant matrix):

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & a & b & c \\ f & e & a & b \\ g & f & e & a \end{bmatrix}. \quad (4)$$

On the other hand, any $Q \times Q$ Toeplitz matrix \mathbf{T} with even Q can be decomposed as:

$$\mathbf{T} = \begin{bmatrix} \mathbf{T}_0 & \mathbf{T}_1 \\ \mathbf{T}_2 & \mathbf{T}_0 \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{I} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{T}_1 - \mathbf{T}_0 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{T}_2 - \mathbf{T}_0 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{T}_0 \end{bmatrix} \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{I} & \mathbf{0} \\ \mathbf{I} & \mathbf{I} \end{bmatrix}, \quad (5)$$

where

- $\mathbf{0}$ and \mathbf{I} are $Q/2 \times Q/2$ null and identity matrices, respectively;
- $\mathbf{T}_0, \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_1 - \mathbf{T}_0, \mathbf{T}_2 - \mathbf{T}_0$ are $Q/2 \times Q/2$ Toeplitz matrices.

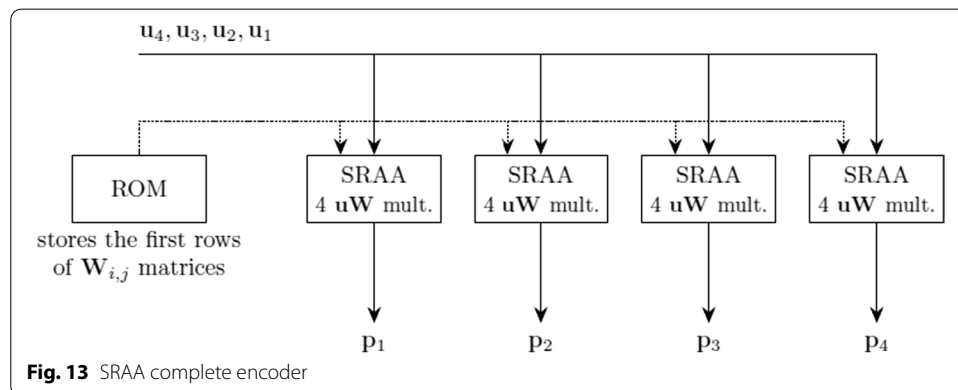


Fig. 13 SRAA complete encoder

If we focus on the $Q \times Q$ binary Toeplitz matrix \mathbf{W} we are considering here, it can hence be decomposed as:

$$\mathbf{W}_{i,j} = \begin{bmatrix} \mathbf{W}_0 & \mathbf{W}_1 \\ \mathbf{W}_2 & \mathbf{W}_0 \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{I} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{W}_1 + \mathbf{W}_0 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_2 + \mathbf{W}_0 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{W}_0 \end{bmatrix} \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{I} & \mathbf{0} \\ \mathbf{I} & \mathbf{I} \end{bmatrix}, \quad (6)$$

where \mathbf{W}_0 , \mathbf{W}_1 and \mathbf{W}_2 are $Q/2 \times Q/2$ binary Toeplitz matrices and operations over the binary field have been taken into account (for binary symbols, plus and minus operations give the same result).

Let us focus again on the computation of the elementary term $\mathbf{u}_i \mathbf{W}_{i,j}$ and let us write the vector \mathbf{u}_i as $\mathbf{u}_i = [\mathbf{u}_0 \ \mathbf{u}_1]$. By exploiting the form (6) for $\mathbf{W}_{i,j}$, we have:

$$\mathbf{u}_i \mathbf{W}_{i,j} = [\mathbf{u}_1(\mathbf{W}_2 + \mathbf{W}_0) + (\mathbf{u}_0 + \mathbf{u}_1)\mathbf{W}_0 \ \mathbf{u}_0(\mathbf{W}_1 + \mathbf{W}_0) + (\mathbf{u}_0 + \mathbf{u}_1)\mathbf{W}_0]. \quad (7)$$

In this way, we have transformed the multiplication of a Q -bit vector \mathbf{u}_i by a $Q \times Q$ matrix $\mathbf{W}_{i,j}$ into three multiplications of $Q/2$ -bit vectors by $Q/2 \times Q/2$ Toeplitz matrices, plus some $Q/2$ -bit vector additions. Since the vector-matrix product gives the most important contribution to the overall complexity and has a quadratic cost in the matrix size, by exploiting (7) we achieve a considerable complexity reduction. Moreover, the process can be iterated, as far as we get matrices of even dimension. As shown in Sect. 5, this approach provides a significant speedup with respect to the classical \mathbf{uG} encoding map and also with respect to SRAA in most operation conditions.

3.3 Complexity considerations

The generator matrix of the CCSDS LDPC codes is systematic and composed by $Q \times Q$ circulant matrices, then we can focus on one of them and for computing the total complexity we must multiply by the number of circulant matrices in G . For the classical \mathbf{uG} encoder, the memory is proportional to the entire size ($Q \times Q$ bits) and the number of operations to the entire size, too ($Q \times Q$). For the SRAA encoder, the memory is proportional to the first row (Q) and the number of operations to the entire size ($Q \times Q$). For the Winograd encoder, the memory is proportional to $3^{\log_2(Q)}$ and the number of operations to $3^{\log_2(Q)+1}$.

4 Methods/experimental

In this section we consider our real-time implementation of the Ground System Telecommand. We focus on a target output bit rate of 2.048 Mbps, which is currently considered realistic by the European Space Agency (ESA) for the new applications described in Sect. 1. A main issue toward an operational system available at Ground Station is represented by the selection of the platform where the TC Synchronization and Channel Coding modules should be implemented, mainly having into account the target output bit rate as well as portability.

4.1 Considered digital platforms

This section briefly describes the ESA platforms currently available at the Telemetry Tracking and Command Processor (TTCP), i.e., the Ground Station. There are three

kinds of platforms available: CPU, ARM-based FPGA, and FPGA; they are integrated in two types of units/devices: Data Processing Unit (DPU) and Signal Processing Modules (SPMs).

TTCP has a DPU that includes an Intel Xeon CPU E5-2637 v4 @ 3.50 GHz performing Man-Machine Interface (MMI) and some level of processing functions with a low overall CPU load ($< 5\%$). Regarding the SPMs, there are several options, and each one has two FPGAs from Altera/Intel inside: the FPGA Stratix which is a powerful pure FPGA and a Cyclone SoC FPGA with dual-core ARM-CortexA9.

The Intel Xeon processor E5 v4 is a multi-core enterprise processor built on 14-nm process technology designed to have low power and high performance. The processor was designed for a platform consisting of a processor and the Platform Controller Hub (PCH) supporting up to 46 bits of physical address space and 48 bits of virtual address space. Table 1 addresses the main features of the CPU platform available at TTCP.

The Stratix device offers up to 48 integrated transceivers with 14.1 Gbps data rate capability. These transceivers also support backplane and optical interface applications. The device features a rich set of high-performance building blocks, including a redesigned adaptive logic module (ALM), 20 kbit (M20K) embedded memory blocks, variable precision digital signal processing (DSP) blocks, and fractional phase-locked loops (PLLs). All these building blocks are interconnected by a multi-track routing architecture and comprehensive fabric clocking network. The main features of the FPGA platform are reported in Table 2.

The Cyclone FPGA built on 28-nm Low-Power (28LP) process provides a low cost and low power system achieving 40 percent lower total power compared with the previous generation, efficient logic integration capabilities, integrated transceiver variants and SoC FPGA with an ARM-based hard processor system (HPS). The capabilities and logic integration were improved thanks to an 8-input ALM, with up to 12 MB of memory and variable precision DSP blocks. Cyclone integrates an HPS that includes processors, peripherals and memory controller with the FPGA fabric using a high-bandwidth interconnect backbone.

Table 1 CPU platform (Intel Xeon CPU E5-2637 v4 @ 3.50 GHz) specification (from ESA)

Features	Values
Threads	16
Threads per core	2
Cores per socket	4
Sockets	2
Model name	Intel® Xeon® CPU E5-2637 v4 @ 3.50 GHz
L1d cache	32K
L1i cache	32K
L2 cache	256K
L3 cache	15360K
NUMA node0 CPU(s)	0–3, 8–11
NUMA node1 CPU(s)	4–7, 12–15

Table 2 FPGA platform (Altera Stratix) specification (from Altera/Intel)

Features	Stratix
Logic elements (K)	952
ALMs	359200
Registers (K)	1437
M20K Memory Blocks	2640
M20K Memory (Mbits)	52
MLAB Memory (Mbits)	10.96
Variable Precision DSP blocks (27 × 27)	352
Variable Precision Multipliers (18 × 18)	704
LVDS channels, 1.4 Gbps (receive/transmit)	210
14.1-Gbps Transceivers	48
Fractional PLLs	28
DDR3 SDRAM × 72 DIMM Interfaces	6

Table 3 Computer used to evaluate the CPU platform

Features	Values
Model name	Intel® Xeon® CPU E5-1620 v2 @ 3.70 GHz
Memory RAM	24 GB
Operating System	Windows 10 Pro 64-bits

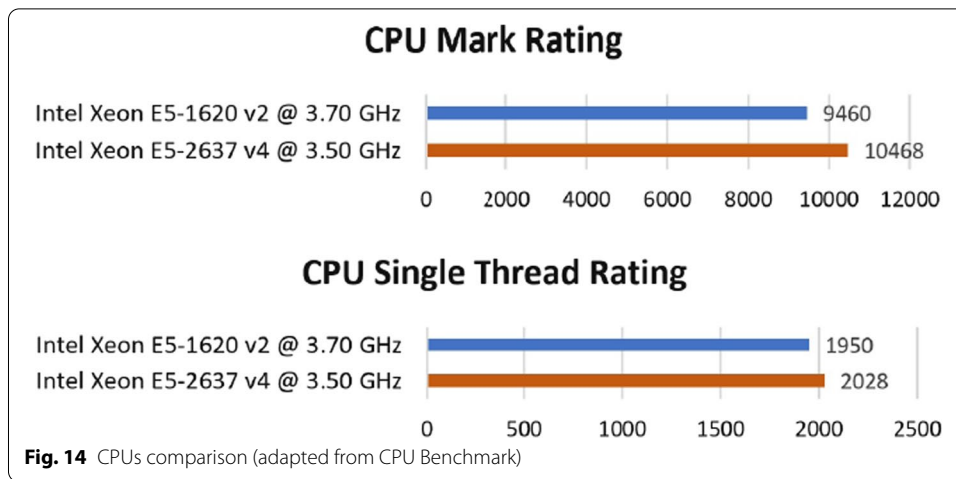
4.2 Platform comparison and preliminary Output Bit Rate results

As from the critical modules description (see Sect. 2), it is clear that channel encoding is the most complex module at the platform. Hence, in this subsection all available platforms are evaluated in terms of bit rate performance with reference to the channel encoding module. Because of the focus on the encoding operations, the filler bit can be ignored and, in presenting the results, we simply refer to the BCH(63, 56) code instead of the fBCH(64, 56) code.

4.2.1 CPU platform

The Intel Xeon CPU E5-1620 v2 @ 3.70 GHz was used, with similar features to the one available at TTCP, as shown in Table 3. Moreover, it was required to develop a preliminary/simple code in C/C++ language where the compiler GCC version 7.3.0 was used and optimization level of “-O3” (maximum performance) was selected.

As observed in Table 3, the CPU used in this evaluation has a clock slightly higher than the target (3.7 GHz vs. 3.5 GHz); however, it is a Xeon processor from first generation while the CPU available at TTCP is from second generation. Therefore, considering these features, we can conclude that a similar performance is expected. This is confirmed in Fig. 14, which reports the comparison between CPUs performance: so, we see that the rating of ESA CPU (orange colour) is only slightly better than the one used in the study (blue colour).

**Table 4** Target performance considering only the encoder

Target output coded bit rate (Mbps)	2.048
CPU clock (GHz)	3.50
Instruction cycle (ns)	0.29
Number of instructions cycles per bit	1709
Max time for channel encoding (μs)	
LDPC(128, 64)	62.5
LDPC(512, 256)	250.0
BCH(63, 56)	30.8

Table 5 Performance achieved by the CPU platform

Channel encoding	Output bit rate (Mbps)
LDPC(128, 64)	≥ 22.1
LDPC(512, 256)	≥ 4.8
BCH(63, 56)	≥ 45.0

In this evaluation, the LDPC encoder based on a simple **uG** product (described in Sect. 3) was used. Table 4 presents the target performance (particularly as regards the desired, minimum output coded bit rate) and timings for the CPU platform. Owing to the high CPU clock, it is observed that the target coded bit rate of 2.048 Mbps allows 1709 instruction cycles per bit resulting in the maximum allowed spent time per codeword presented in Table 4.

In order to assess the CPU performance, a preliminary experiment was developed which consisted in encoding an information word for ten times. The results for both LDPC codes and for the BCH code (included for completeness and encoded by mimicking the circuit in Fig. 5) are presented in Table 5.

It can be seen that the CPU performance complies with the target coded bit rate for both LDPC codes and for the BCH code. However, for the first cycle a lower bit rate is

Table 6 LDPC and BCH encoders complexity and timing (maximum frequency) estimate

Area report	LDPC(128, 64)				LDPC(512, 256)				BCH(63, 56)			
	Cyclone		Stratix		Cyclone		Stratix		Cyclone		Stratix	
	Use	%	Use	%	Use	%	Use	%	Use	%	Use	%
Flip Flops	402	≤ 1	384	≤ 1	1578	≤ 3	1565	≤ 1	18	≤ 1	19	≤ 1
Combinatorial logic elements	234	≤ 1	347	≤ 1	1117	≤ 2	1117	≤ 1	13	≤ 1	12	≤ 1
Timing report												
Max frequency	341 MHz		717 MHz		264 MHz		627 MHz		36 MHz		717 MHz	

Table 7 Output bit rate achieved by hardware platforms

	Cyclone	Stratix
LDPC encoders	≥ 100 Mbps	≥ 250 Mbps
BCH encoder	≥ 100 Mbps	≥ 250 Mbps

achieved, in comparison with the remaining cycles, which can be related with the CPU automatic resources allocation assignment by the operating system. For instance, sometimes prior to the first execution the CPU clock is about 1.5 GHz instead of 3.5 GHz.

It should be noted that these results hold for the classic uG encoder, hence a higher output bit rate is expected for the two other encoders described in Sect. 3, namely, the SRAA-based and the Winograd-based encoders. Furthermore, it should be highlighted that these results do not include the start and tail sequences, which are easy to add to the codewords with little processing time required and consequently increasing the bit rate. More precisely, it is expected to increase the output bit rate by at least 50% for the LDPC(128, 64) code (the increase is larger when the tail sequence is selected), and by 12.5% for the LDPC(512, 256) code.

4.2.2 Hardware platforms

In this subsection both hardware platforms available at TTCP are evaluated with reference to LDPC and BCH encoders. The SRAA-based encoder is known to be particularly efficient for hardware implementation. Its parallel structure, presented in Sect. 3.1, has been evaluated for both Cyclone and Stratix FPGA devices and has been chosen with respect to the iterative architecture due to the few logic operations required to implement the SRAA circuit and to take advantage of the higher bit rate achievable with the parallel solution.

In Table 6 the estimated complexity and timing (maximum frequency) achieved by synthesis and place and route performed with ALTERA QUARTUS® software are reported for both Cyclone and Stratix devices and all encoding methods.

The estimate regarding the maximum frequency is rough, due the unconstrained synthesis and place and route: the FPGA device contains only the tested encoder without other subsystems and without any pinout constraints. Considering this preliminary implementation on FPGA device and the required design margin (a factor of 2.5) to be

adopted according to the authors' experience in this preliminary phase, the minimum bit rates presented in Table 7 can be considered achievable with no particular problem.

4.2.3 Selected platform

Considering the platforms available at TTCP and the main points for choosing one, namely, performance evaluation and future portability toward an operational platform, it is reasonable to conclude that the CPU platform is the best choice for the proposed application. It complies with the target output bit rate and, additionally, it guarantees an easy portability, being a software approach. The start and tail sequences were not included in the bit rate evaluation which meant that it was expected to increase the performance even more.

Moreover, considering that the classic uG encoder for LDPC codes was used in this evaluation, the output bit rate is expected to be even better if an encoder more appropriate to software is implemented such as the encoder based on Winograd or the SRRA methods, described in Sect. 3. Indeed, this has been investigated during the output bit rate tests, whose results are presented in Sect. 5.

As a side remark, we point out that the options considered in this study, agreed with ESA, were the most important solutions currently available for TTC: CPU, ARM and FPGA. Nevertheless, other platforms could be taken into account in the future. One potential solution is represented by hardware platforms relying on a Graphics Processing Unit (GPU), which is a paradigm that is receiving growing interest. To the best of our knowledge, GPU platforms have not been used yet for space applications, but the parallel nature of some of the considered decoding techniques may be well suited for exploiting GPU architectures. The most important feature of GPU-based hardware is in fact in a large number of relatively slow processors that can work in parallel. This may be well suited to the application of techniques such as layered decoding, which enable high degrees of parallelism in the execution of LDPC decoding. The QC structure of the matrices of the considered codes can in fact be suitable for the application of layered decoding approaches [18–20], and therefore their decoding could benefit from the use of GPU-based hardware.

5 Results and discussion

Taking into account the output target bit rate of 2.048 Mbps at the TC Synchronization and Channel Coding sublayer, this section describes the achieved results for the three encoding methods presented in Sect. 3, running on the selected platform.

5.1 Time measurement and Output Bit Rate computation

In order to evaluate the performance of the encoding schemes and, as concerns the LDPC codes, to compare the three options presented in Sect. 3, a breadboard was developed, which allowed measuring two relevant parameters, that is, the processing time and the output bit rate. The breadboard architecture and, in particular, the processing time measurement procedure are shown in Fig. 15. For the sake of brevity, we omit details on the breadboard architecture (further information can be found in [21]) while we focus on the processing time measurement procedure. In short, the tests are based on the data saved in the Local Storage module, which stores several useful information including the

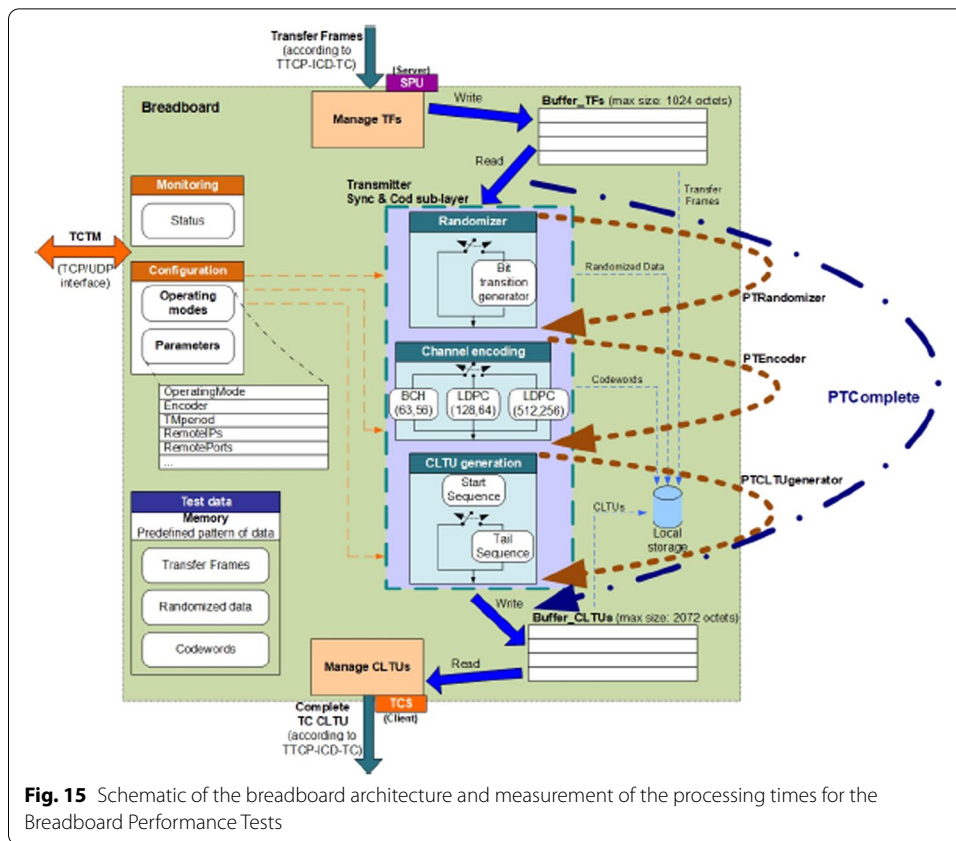


Fig. 15 Schematic of the breadboard architecture and measurement of the processing times for the Breadboard Performance Tests

processing time: of the Randomizer (*PTRandomizer*), of the Encoder (*PTEncoder*), of the CLTU Generator (*PTCLTUgenerator*) and of the whole process (*PTcomplete*). The latter is measured by starting the timer when the first critical module starts being used (either the Randomizer module, when enabled and BCH is selected, or the LDPC encoder module, as depicted in Fig. 2) and stopping the timer when the CLTU is completed. The time required to fill extra bits to the received TF when it is not a multiple of an information word is also measured and added to the *PTcomplete*.

We have generated 10000 TFs and computed the average Output Bit Rate (OBR) as the ratio between the total number of CLTU bits and the overall processing time (that is, the sum of the *PTcomplete* values relative to the single transmissions). Numerical results are shown next.

5.2 Output Bit Rate performance results

This subsection summarizes the most important results of the analysis using the target processor and SUSE Linux Enterprise Server 12 Service Pack 1 as operating system. We remind that the data rates required, on the basis of the current inputs, are comprised between 7.8125 bps and 2.048 Mbps [22].

Table 8, for the BCH code, and Table 9, for the LDPC codes, show the average OBR achieved at the breadboard for all identified critical modules, by considering different TF lengths, where “Mixed” means TFs with length randomly selected within the allowed range.

Table 8 Breadboard OBR performance of all critical modules for the BCH(63,56) code

Breadboard average output bit rate [Mbps]		
TF length (bytes)	W/o Randomizer	With randomizer
6	81.335	61.047
7	85.878	64.419
Mixed	92.220	59.510
1022	93.071	59.709
1024	92.580	59.440

Table 9 Breadboard OBR performance of all critical modules for the LDPC codes (C = Classic, S = SRAA, W = Winograd)

Breadboard average output bit rate (Mbps)									
TF length (bytes)	LDPC(128, 64) w/o tail sequence			LDPC(128, 64) with tail sequence			LDPC(512, 256)		
	C	S	W	C	S	W	C	S	W
6	35.049	31.184	42.781	58.066	51.726	70.379	10.608	13.055	16.920
8	36.067	31.932	44.528	59.879	53.103	73.258			
32							11.035	13.787	18.057
Mixed	31.325	27.669	41.116	31.749	28.108	41.723	10.258	13.074	17.160
1024	31.338	27.659	41.217	31.534	27.922	41.478	10.279	13.091	17.208

We observe that, for all possible different combinations of encoders, encoding methods (for the LDPC case), with or without randomizer (for the BCH encoder) and with or without tail sequence (for the LDPC(128, 64) code), the target (maximum) output bit rate of 2.048 Mbps is achieved. The largest OBR is achieved by using the Winograd method, while the SRAA is better than the classic **uG** encoder for the LDPC(512, 256) code. In absolute terms, as predictable, the minimum OBR is achieved, for all the considered algorithms, with the long LDPC code remaining, in any case, larger than the target OBR of 2.048 Mbps. Actually, even with this selected CPU platform, an average minimum of about 10.2 Mbps is achieved, for the LDPC(512, 256) encoder when the classic encoder is used. Furthermore, it is observed that for any scenario of BCH(63, 56), and for LDPC(128, 64) when Winograd-based method is used, an average OBR higher than 41.1 Mbps is obtained. Moreover, even the worst result, for the LDPC(512, 256) code, is still about five to eight times better (when classic and Winograd method are selected, respectively) than the target OBR.

6 Conclusions

In this paper, the TC Synchronization and Channel Coding sublayer has been analyzed, in particular with reference to the TC LDPC encoders that were implemented following the classic approach and with two efficient encoding methods, namely: the SRAA and Winograd convolution. Both these alternative methods have been tested and we have verified that the Winograd algorithm is actually able to outperform the classic encoding method in all the considered scenarios, while SRAA does the same in specific frameworks, e.g., when using the long LDPC code. Indeed, such a conclusion follows from the

choice to privilege a software implementation, whilst in case of using a hardware implementation SRAA would show its benefits as well, thanks to the possibility of efficiently exploiting parallel computation.

Different platforms available at the TTCP have been evaluated, being selected the CPU platform, since it is compliant with the target output bit rate performance of 2.048 Mbps and, additionally, it guarantees an easy portability owing to its software approach. Subsequently, the breadboard software including all critical modules was developed in C++ language using the CPU platform.

The three different methods identified for LDPC encoding have been successfully implemented on the breadboard. A minimum Output Bit Rate performance higher than 10 Mbps for all critical modules has been achieved, about five times higher than the target performance of 2.048 Mbps. These results show the importance of an optimized implementation of the critical LDPC encoder and the other transmitter blocks for high-rate real-time implementations.

Abbreviations

ALM: Adaptive Logic Module; ARQ: Automatic Repeat Request; BCH: Bose-Chaudhuri-Hocquenghem; CCSDS: Consultative Committee for Space Data Systems; CLTU: Command Link Transmission Unit; CPU: Central Processing Unit; CRC: Cyclic Redundancy Check; DPU: Data Processing Unit; DSP: Digital Signal Processing; ESA: European Space Agency; FPGA: Field-Programmable Gate Array; GPU: Graphics Processing Unit; HPS: Hard Processor System; LDPC: Low-Density Parity-Check; LFSR: Linear Feedback Shift Register; LP: Low Power; MMI: Man-Machine Interface; MRB: Most Reliable Basis; OBR: Output Bit Rate; OSI: Open Systems Interconnection; PCH: Platform Controller Hub; PLL: Phase-Locked Loop; SEC: Single Error Correction; SNR: Signal-to-Noise Ratio; SPM: Signal Processing Module; SRAA: Shift Register Adder Accumulator; TC: Telecommand; TED: Triple Error Detection; TF: Transfer Frame; TTCP: Telemetry Tracking and Command Processor.

Acknowledgements

This work was supported in part by the European Space Agency under contract 4000124933/18/D/MB.

Authors' contributions

All the authors participated in writing the article and revising the manuscript. All authors read and approved the final manuscript.

Availability of data and materials

Not applicable.

Declarations

Competing interests

The authors declare that they have no competing interests.

Author details

¹European Space Agency/ESOC, Robert-Bosch-Straße 5, 64293 Darmstadt, Germany. ²Dipartimento di Ingegneria dell'Informazione, Università Politecnica delle Marche, Via Brecce Bianche 12, 60131 Ancona, Italy. ³Deimos Engenharia, Avenida Dom João II 41, 1998-023 Lisbon, Portugal. ⁴Dipartimento di Elettronica e Telecomunicazioni, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Turin, Italy. ⁵Dipartimento di Ingegneria dell'Energia Elettrica e dell'Informazione "Guglielmo Marconi", Università di Bologna, Via dell'Università 50, 47522 Cesena, Italy. ⁶Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), Viale G.P. Usberti 181/A, 43124 Parma, Italy.

Received: 1 June 2021 Accepted: 9 December 2021

Published online: 24 December 2021

References

1. Consultative Committee for Space Data Systems (CCSDS), Telecommand, CCSDS 200.0-G-6 Green Book (1987)
2. Consultative Committee for Space Data Systems (CCSDS), Next Generation Uplink, CCSDS 230.2-G-1 Green Book (2014)
3. G. Kazz, E. Greenberg, C. Scott, Replacing the CCSDS telecommand protocol with the next generation uplink (NGU), in *Proceedings of the 2012 AIAA International Conference Space Operations, Stockholm, Sweden* (2012)
4. N. Peccia, A brief story of a success: the CCSDS, in *Proceedings of the 2014 AIAA International Conference on Space Operations, Pasadena, CA, USA* (2014)

5. Consultative Committee for Space Data Systems (CCSDS), TC Synchronization and Channel Coding, CCSDS 231.0-B-3 Blue Book (2014)
6. G. Liva, E. Paolini, T. De Cola, M. Chiani, Codes on high-order fields for the CCSDS next generation uplink, in *Proceedings of the 2012 6th Advanced Satellite Multimedia Systems Conference and 12th Signal Processing for Space Communication Workshop, Vigo, Spain* (2012)
7. K. Andrews, D. Divsalar, J. Hamkins, F. Pollara, Error correcting codes for next generation spacecraft telecommand, in *Proceedings of the 2013 IEEE Aerospace Conference, Big Sky, MT, USA* (2013)
8. Á. Álvarez, V. Fernández, B. Matuz, An efficient NB-LDPC decoder architecture for space telecommand links. *IEEE Trans. Circuits Syst. II Express Briefs* **68**(4), 1213–1217 (2021)
9. M. Baldi, M. Bertinelli, F. Chiaraluce, P. Closas, P. Dhakal, R. Garelo, N. Maturo, M. Navarro, J.M. Palomo, E. Paolini, S. Pfletschinger, P.F. Silva, L. Simone, J. Vilà-Valls, State-of-the-art space mission telecommand receivers. *IEEE Aerosp. Electron. Syst. Mag.* **32**(6), 4–15 (2017)
10. M. Baldi, M. Bertinelli, F. Chiaraluce, P. Freire de Silva, R. Garelo, N. Maturo, M. Navarro, J.M. Palomo, E. Paolini, R. Prata, L. Simone, C. Urrutia, Theoretical analysis and implementation of effective receivers for telecommand space links, in *Proceedings of the TTC 2019 International Workshop on Tracking, Telemetry and Command Systems for Space Applications, Darmstadt, Germany* (2019)
11. Consultative Committee for Space Data Systems (CCSDS), TC Space Data Link Protocol, CCSDS 232.0-B-3 Blue Book (2015)
12. Y. Fang, P. Chen, G. Cai, F.C.M. Lau, S.C. Liew, G. Han, Outage-limit-approaching channel coding for future wireless communications: root-protograph low-density parity-check codes. *IEEE Veh. Technol. Mag.* **14**(2), 85–93 (2019)
13. L. Dai, Y. Fang, Z. Yang, P. Chen, Y. Li, Protograph LDPC-coded BICM-ID with irregular CSK mapping in visible light communication systems. *IEEE Trans. Veh. Technol.* **70**(10), 11033–11038 (2021)
14. Y. Fang, G. Bi, Y.L. Guan, F.C.M. Lau, A survey on protograph LDPC codes and their applications. *IEEE Commun. Surveys Tuts.* **17**(4), 1989–2016 (2015)
15. M. Baldi, N. Maturo, E. Paolini, F. Chiaraluce, On the use of ordered statistics decoders for low-density parity-check codes in space telecommand links. *EURASIP J. Wirel. Commun. Netw.* **2016**, 272 (2016)
16. Z. Li, L. Chen, L. Zeng, S. Lin, W.H. Fong, Efficient encoding of quasi-cyclic low-density parity-check codes. *IEEE Trans. Commun.* **54**(1), 71–81 (2006)
17. S. Winograd, Arithmetic Complexity of Computations, CBMS-NSF Regional Conference Series in Mathematics, Book Code: CB33 (1980)
18. G. Wang, M. Wu, Y. Sun, J.R. Cavallaro, A massively parallel implementation of QC-LDPC decoder on GPU, in *Proceedings of the 2011 IEEE 9th Symposium on Application Specific Processors (SASP)*, pp. 82–85 (2011)
19. Y. Lin, W. Niu, High throughput LDPC decoder on GPU. *IEEE Commun. Lett.* **18**(2), 344–347 (2014)
20. R. Li, X. Zhou, H. Pan, H. Su, Y. Dou, A high-throughput LDPC decoder based on GPUs for 5G new radio, in *Proceedings of the IEEE Symposium on Computers and Communications (ISCC) 2020*, pp. 1–7 (2020)
21. R. Abelló, M. Baldi, F. Chiaraluce, R. Fernandes, P. Freire da Silva, R. Garelo, D. Gelfusa, J.M. Palomo, E. Paolini, R. Prata, L. Santos Ugarte, L. Simone, NEXTRACK—Next Generation ESTRACK Uplink Services, in *Proceedings of the TTC 2019 International Workshop on Tracking, Telemetry and Command Systems for Space Applications, Darmstadt, Germany* (2019)
22. Consultative Committee for Space Data Systems (CCSDS), Radio Frequency and Modulation Systems, CCSDS 401.0-B-31 Blue Book (2021)

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen[®] journal and benefit from:

- Convenient online submission
- Rigorous peer review
- Open access: articles freely available online
- High visibility within the field
- Retaining the copyright to your article

Submit your next manuscript at ► [springeropen.com](https://www.springeropen.com)