IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 February 2024, accepted 9 March 2024, date of publication 13 March 2024, date of current version 19 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3376630

RESEARCH ARTICLE

Design and Implementation of a Configurable Fully Compliant DVB-S2 LDPC Encoder for High Data-Rate Downlink Payload

PIETRO NANNIPIERI[1], (Member, IEEE), GIACOMO BARTOLACCI[1], MATTEO BERTOLUCCI[2], AND LUCA FANUCCI[1], (Fellow, IEEE)

[1]Department of Information Engineering, University of Pisa, 56122 Pisa, Italy
[2]IngeniArs S.r.l., 56121 Pisa, Italy

Corresponding author: Pietro Nannipieri (pietro.nannipieri@unipi.it)

This work was supported in part by Italian Ministry of Education and Research (MUR) in the Framework of the FoReLab Project (Departments of Excellence), and in part by IngeniArs s.r.l.

ABSTRACT This work centres on designing and implementing a Low-Density Parity-Check (LDPC) Encoder on a Xilinx Field Programmable Gate Array (FPGA). The encoder will be part of the Digital Video Broadcasting Satellite 2nd generation (DVB-S2) Transmitter Intellectual Property (IP) for a High Data-Rate Downlink Telemetry System in the context of Earth Exploration Satellite Service. The objective is to design an LDPC encoder with three main features. First, the design will prioritize maximizing data processing speed to ensure the efficient transmission and reception of the payload. Second, the encoder will comply with the DVB-S2 Standard for all possible data rates. This will optimize transmission efficiency by adapting to varying channel conditions, utilizing Adaptive Coding and Modulation (ACM) and Variable Coding and Modulation (VCM) techniques. Lastly, the input and output interfaces of the LDPC Encoder will be designed for high reconfigurability, allowing easy adaptation to different operational requirements and facilitating seamless integration into diverse systems. AXI Stream Pipelined Architecture: The LDPC Encoder will utilize an AXI Stream Pipelined architecture. This architecture choice will enhance data transfer efficiency between different functional blocks within the FPGA design, minimizing latency and maximizing overall system performance.

INDEX TERMS DVB-S2, LDPC, low-density parity check, satellite, telemetry transmitter, high throughput, FPGA, parallel processing, PDT, reconfigurable datapath.

I. INTRODUCTION

A. SATELLITE TELECOMMUNICATION SYSTEMS AND EARTH EXPLORATION SATELLITES

In the field of Satellite communication systems, Earth Exploration Satellite Services (EESS) play a significant role in enabling the monitoring, understanding, and management of our planet's environment and resources. One of the most challenging tasks for a satellite is the transmission of science data, also called Payload Data Telemetry (PDT). Considering the limited bandwidth, this involves downlinking sensor data collected by the satellite's instruments to Earth stations containing valuable information about the Earth's environment, such as images, measurements, and other scientific observations. Currently, Earth Observation (EO) systems in Europe typically utilise the S-band (which spans from 2 to 4 Ghz) and the X-band (which spans from 8 - to 12 GHz) for downloading satellite data to ground stations. However, due to increased PDT data rates and increased demands and congestion in lower bands, agencies worldwide have been exploring higher bands, as they (generally) can accommodate a higher bandwidth. The K band, with a bandwidth of 1.5 GHz available between 25.5 and 27 GHz, is considered the following option for higher throughput. Extensive research has been conducted on using the K band for Low Earth Orbit (LEO) satellites,

The associate editor coordinating the review of this manuscript and approving it for publication was Kashif Saleem.

as mentioned in [1]. Notably, the EUMETSAT MetOp Second Generation (MetOp-SG) and the Meteosat Third Generation (MTG) are expected to leverage K-band frequencies for their communication links [2]. High-order digital modulation schemes are required for better spectral efficiency to reach higher data rates. They need a higher signal-to-noise ratio (SNR) and power to maintain the same Bit Error Rate (BER) as lower-order modulation schemes. Commonly used modulation schemes include QPSK, 8-PSK, 16-APSK, 32-APSK and 64-APSK. In DVB-S2X (the extension of the DVB-S2 standard), modulation order can go as high as 256-APSK [3]. As downlink data rates and the use of higher frequency bands increase, some environmental effects that influence the propagation of radio signals in the telecommunication links between earth stations and EESS observation satellites become more demanding. Above 10 GHz, precipitation, especially rain, causes absorption and scattering of radio waves that can lead to severe signal attenuation [4]. Two methods usually cope with the challenges of High Data Rate Payload Data Transmission. The first uses Forward Error Correction (FEC) Codes, accomplished by adding redundancy to the transmitted information with a Channel FEC Encoder using a predetermined algorithm, allowing the receiver to correct errors without retransmission or feedback to the sender. The second technique is Adaptive Coding and Modulation: when the satellite-to-ground distance decreases as the satellite elevation angle increases, the free-space loss, which represents the attenuation of the signal during propagation, also decreases. This improvement in the link distance results in a more favourable link budget, stemming from the reduced free-space loss, and provides an opportunity to utilise more spectrally efficient modulation schemes and higher coding rates. The distance change can be predicted for each pass, and the transmission can be planned to change the modulation and the coding rate when the link budget is more favourable to increase the helpful bit rate. If the distance can be easily anticipated, some tropospheric propagation events may be more difficult to predict. The impact of these events on the link budget may be very significant when using the highest frequencies (such as Ka-band EESS). The ACM mode can cope with these events at the highest frequencies. This ACM mode consists of updating the modulation and the coding rate to the best operating tuning in quasi-real time based on the received signal-to-noise ratio measurement by the receiver.

B. EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE (ETSI) DVB-S2 STANDARD

The DVB-S2 [5] system is a versatile toolkit and can accommodate various application areas without significantly increasing the complexity of the single-chip decoder. Consultative Committee for Space Data System (CCSDS) set out to develop a standard [6] mainly to provide a formalisation of the interface between CCSDS and DVB-S2 for interoperability issues, taking full advantage of the fact that DVB-S2 is defined as a flexible ''toolkit'' system.

The system exhibits the following characteristics:

- An adaptable input stream adapter that can handle single or multiple input streams in various formats.
- A robust FEC system that combines Low-Density Parity Check (LPDC) and Bose-Chaudhuri-Hocquenghem (BCH) codes.
- A wide range of code rates, ranging from 1/4 to 9/10, providing flexibility in data rate and error correction capabilities.
- Four constellations available: QPSK, 8PSK, 16APSK, and 32APSK, offering different levels of spectrum efficiency, ranging from 2 bit/s/Hz for QPSK to 5 bit/s/Hz for 32APSK.
- Three spectrum shapes with roll-off factors of 0.35, 0.25, and 0.20, allowing adaptation to different bandwidth and interference scenarios.
- ACM functionality.

Signal generation is based on four framing structure levels:

- BBFRAME at base-band (BB) level, carrying a variety of signalling bits, to configure the receiver flexibly according to the application scenario;
- FECFRAME, comprised of BBFRAME with the addition of redundancy from the FEC Encoding for a total length of N_{ldpc} bits (64800 for standard frames, 16200 for short frames)
- XFECFRAME, or complex FECFRAME, is the output of the constellation mapper, which identifies complex symbols in the (I, Q) plane, with the modulation efficiency
- PLFRAME at the physical layer (PL) level, carrying a few highly protected signalling bits to provide robust synchronisation and signalling at the physical layer.

The FEC, together with the modulation, is the critical subsystem that achieves excellent performance by satellite in the presence of high noise and interference levels. The effectiveness of the FEC Encoder is primarily attributed to the performance of the inner LDPC codes. However, LDPC codes suffer from error floors at low error rates. This phenomenon can be characterised as an abrupt decrease in the slope of a code's performance curve from the moderate-SNR waterfall region to the high-SNR floor region [7]. Since error floor rates are challenging to measure accurately and may lead to decoding failures, concatenated BCH outer codes are introduced. These BCH codes have the same block length as the LDPC code and offer an error correction capability of 8 to 12 bits, ideal for covering the error floor region in LDPC. The combined BCH and LDPC FEC Encoder output is called FECFRAME and comprises the BBFRAME, to which the BCHFEC and LDPCFEC fields are appended. Finally, bit interleaving is applied to FEC-coded bits in 8PSK, 16APSK, and 32APSK. The row-column interleaver helps to improve signal robustness against burst errors that occur in transmission: if, for example, an impulsive disturb corrupts the symbols at the top of the frame and no interleaving has

been adopted, only the check bits symbols are corrupted, and no FEC would be possible at the receiver.

C. STATE-OF-THE-ART DVB-S2 LDPC ENCODERS

The developers are generally focused on the decoder architectures and implementation, as it is the most complicated transceiver digital part. The encoding operation may appear not only on the transmitter but also on the receiver. The LDPC encoder may be needed in the receiver recovery loops for adaptive processing and processing under poor jamming environments. Thus, the compact and fast encoder can also play an essential role in the receiver design. In [8], the proposed encoder coding rate reaches 125 Mbps given the maximum clock frequency of 125 MHz but still needs to achieve full DVB-S2 compliance as it supports only one code rate for the 16200 bits code length and 3/5 code rate. A pipelined approach is mentioned, but more detail is needed regarding its actual implementation. Similarly, the architecture in [9] supports only 13 (out of 21) code rates and has a high resource consumption per code rate. In [10], an encoder compliant with DVB-S2 and its extension DVB-S2X is proposed. The output is limited to one bit per cycle in the natural order. Due to a maximum clock frequency of 350 MHz, the throughput is restricted to a maximum of 350 Mbps. The encoder circuit proposed in [11] for DVB-S2 is fully compatible with all LDPC codes defined in the standard. It takes advantage of the features of the codes, presenting a fast-encoding algorithm based on parallelism of 360 bits, and addresses the critical issue of realising low complexity quasi-cyclic encoder using RAMs instead of shift registers. It works on a 50 MHz clock on an Altera Stratix EPI S80 with a maximum throughput of up to 400 Mbps. The I/O interfaces are also programmable but need to be more suitable for higher-order output parallelism; the system was tested only for continuous byte-wise input/output interfaces. The work [12] proposes an FPGA implementation of a FEC encoder core supporting all different DVB-S2 codes in LEO satellite-ground communications. The architecture utilises parallel computation but reaches only a maximum throughput of 1.19 Gbps. Furthermore, this architecture does offer the flexibility to reconfigure the I/O datapath, even on the fly. Still, once again, this is limited to the set 2, 3, 4, 5 of small input and output parallelism values, which have been selected to match the modulation efficiency. The highly influential paper in reference [13] also implements an LDPC Encoder Core based on a fixed 360 bits parallelism (as in [11], but with a different approach), showcasing a fast-encoding algorithm that takes advantage of the structure of the LDPC code and allowing them to reach a high throughput of 10 Gbps at 135 MHz clock frequency. Interestingly, this architecture does not save the entire input message; the input 360 bits participate immediately in the encoding process. This allows for better memory utilisation and lower latency. However, latency is not usually an issue in practical applications, as the ACM configuration varies very slowly (even under prohibitive weather phenomena, it may take a few seconds to change the ACM configuration). This comes at the cost of throughput: when the input circuitry has provided the last 360 bits of the input message, it must wait until the encoder has finished the entire encoding process to give the first 360 parity bits. Furthermore, the output 360 parity check bits are not in the natural and correct order, as defined in the DVB-S2 standard: the authors say that an interleaver can resolve this issue. Still, they do not describe its structure nor its impact in terms of resources, which is expected to be high if the fixed parallelism of 360 is to be maintained at the output of such an interleaver. Another design, [14], has a similar structure to [15], exhibiting a high data rate of 10 Gbps for the relatively low clock frequency of 100 MHz. However, more information about the number of resources used must be shown. Finally, researchers in [16] have developed a highly innovative encoding algorithm and architecture based on a Recursive Encoder Core tailored to one specific code rate and frame length combination. They have shown that up to an outstanding 48.5 Gbps throughput can be achieved with a 270 MHz clock. However, full compliance with DVB-S2 is only partially achievable. One Recursive Encoder Core is based on $(n_{ldpc} - k_{ldpc})$ registers, one register per parity check bit, and an optimised number of adders. Since 21 Recursive Encoder Cores should be utilised parallel to cover the entire DVB-S2 functionality, more than 300,000 FFs would be necessary. A summary of the main characteristics of the state-of-the-art DVB-S2 LDPC encoder is shown in Table 1.

TABLE 1. State of the art DVB-S2 LDPC encoders.

Ref.	*Full DVB-S2 Compl.*	*Hardware*	*Through. [Gbps]*	*Freq [MHz]*	*Config.*
[8]	NO	Altera Cyclone II	0.125	125	NO
[10]	YES	Xilinx Kintex Ultrascale	0.350	350	NO
[11]	YES	Altera Stratix EP1S80	0.400	50	YES (Only parallelism 8 is shown)
[12]	YES	Xilinx XC7K325t	1.19	389.5	YES (parallelisms 2,3,4,5)
[13]	NO	Xilinx XC2VP30	Xilinx	131.7	NO
[17]	YES	Xilinx LX155T	10	100	NO
[16]	NO	Xilinx Kintex 7	47.5	280	NO

D. OBJECTIVES

This work focuses on designing and implementing a DVB-S2 LDPC Encoder on a Xilinx Radiation Tolerant Kintex UltraScale XQRKU060 FPGA, a high-performance monolithic FPGA focusing on performance. The selected hardware provides high Digital Signal Processing (DSP) and block Random Access Memory (RAM)-to-logic ratios, next-generation transceivers, and space-grade packaging for vibration and handling requirements during launch and

operation. It targets applications like on-board processing, digital payloads, remote sensing, and many more. The innovation of the proposed LDPC encoder architecture is the support of the following features in a unique solution, providing an encoder with configurable parallelism and unmatched flexibility and performance:

- Full Compliance with DVB-S2 Standard: The encoder will adhere to the DVB-S2 Standard, ensuring compatibility across various coding rates and frame lengths. This will enable the utilization of ACM and VCM techniques, optimizing transmission efficiency by adapting to varying channel conditions.
- Highly Reconfigurable I/O Interfaces: The input and output interfaces of the LDPC Encoder will be designed with high reconfigurability, allowing easy adaptation to different operational requirements and seamless integration into diverse systems. A wide range of input and output parallelism options will offer flexibility and scalability.
- High Throughput: The design will prioritize high data processing speed to enable efficient payload transmission and reception while maintaining optimal resource utilization. The encoder will deliver parity check bits at up to approximately 20 Gbps, making it one of the fastest DVB-S2 LDPC Encoders available.

The rest of this work will explore these design challenges, structured in the following chapters. Chapter II analyses the specific LDPC code and the encoding algorithms utilized in the LDPC code specified by the DVB-S2 Standard, as they form the foundation for comprehending the subsequent Register Transfer Level (RTL) Design. Chapter III comprehensively describes the proposed architecture, highlighting the flexibility the reconfigurable I/O interfaces enable. Chapter IV will present the implementation results obtained, including assessing performance metrics such as clock frequency, throughput, and resource utilization. Finally, Chapter 5 will provide the concluding remarks and insights derived from the conducted work, summarizing the key strengths of the proposed design.

II. DVB-S2 ENCODING ALGORITHMS

A. DVB-S2 ENCODING ALGORIHMS

LDPC Codes [18] implemented by the DVB-S2 standard are linear systematic codes. Thus, we define a codeword c to have the systematic structure

$$c = [i_0, i_1, i_2 \cdots i_{k-1}, p_o, p_1 \cdots p_{n-k-1})] \tag{1}$$

where $[i_0, i_1, i_2 \cdots i_{k-1}]$ are the k input information bits and $[p_o, p_1 \cdots p_{n-k-1}]$ are the $n-k$ parity check bits that must be appended to the input information bits to construct the codeword. There are two choices for n and 11 for the code rate $r = \frac{k}{n}$. Thus, the standard [3] requires different (n, k) LDPC code parameters. A codeword is valid if and only if:

$$H_{(n-k)\times n} \cdot c = 0 \tag{2}$$

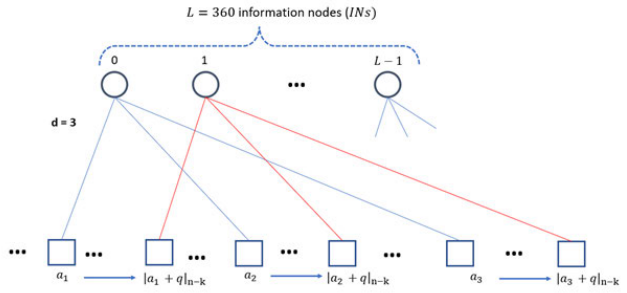

FIGURE 1. Graphical representation of Eq. 4 on a Tanner Graph.

where H is the Parity Check Matrix. Because of the large dimensions of H, its structure is better illustrated through an equivalent geometrical description through a Tanner Graph [19]. The Tanner graph comprises two sets of n - k check and n-bit nodes, corresponding to the rows and columns of the parity check matrix H, respectively. An edge is drawn between a check node and a variable node only if the corresponding element in the H matrix is '1'. The n bit nodes (VN) are divided into:

- n - k parity nodes, denoted PN
- k information nodes, denoted IN

The information nodes are divided into t groups: each contains $L = 360$ information nodes. Thus:

$$t = \frac{k}{L} \tag{3}$$

The parameter $L = 360$ is crucial for the Parity Check Matrix structure: it is constant for every Modulation and Coding (MODCOD), i.e., for any combination of code rates and codeword lengths. We can also see that the $L = 360$ information nodes inside the same group all have the same number of connections. In contrast, such a number may vary between groups. As long as the addresses $[a_0, a_1, \ldots a_{w-1}]$ of the w Check Nodes (CNs) connected to the first information node IN_0 in each group are determined, then the location indices of the w CNs connected to the i^{th} information node IN_i, where $i \in \{0, 1 \ldots L-1\}$, can be obtained by the following equation:

$$\begin{cases} |a_0 + i \cdot q|_{n-k} \\ |a_1 + i \cdot q|_{n-k} \\ \vdots \\ |a_{w-1} + i \cdot q|_{n-k} \end{cases} \tag{4}$$

where:

$$q = \frac{n-k}{L} \tag{5}$$

Eq. 4 can be represented graphically on the Tanner Graph [19] in Figure 1.

We shall now list the important parameters $t = \frac{k}{L} = \frac{r \cdot n}{L}$ and $q = \frac{n-k}{L} = \frac{n \cdot (1-r)}{L}$ with regards to all the available code rates r and codeword lengths n.

The five short frames marked by $*$ have been modified offline; their code rates are lower than the nominal ones. For

TABLE 2. Values of t and q.

Normal frames (N = 64 800)			*Short frames (N = 16 200)*		
Code Rate r	*t*	*q*	*Code Rate r*	*t*	*q*
1/4	45	135	1/4*	9	36
1/3	60	120	1/3	15	30
2/5	72	108	2/5	18	27
1/2	90	90	1/2*	20	25
3/5	108	72	3/5	27	18
2/3	120	60	2/3	30	15
3/4	135	45	3/4*	33	12
4/5	144	36	4/5*	35	10
5/6	150	30	5/6*	37	8
8/9	160	20	8/9	40	5
9/10	162	18	9/10		-

TABLE 3. Nominal and actual code rates for DVB-S2 short codes.

Nominal Code Rate	Actual Code Rate
1/4	1/5
1/2	4/9
3/4	11/15
4/5	7/9
5/6	37/45

the computation of the correct values of t and q, it is necessary to refer to the actual code rates instead of the nominal ones (see Table 3.3). This ensures that the fundamental parameter $L = 360$ remains constant for every MODCOD.

The $[a_0,\ a_1,\ \ldots a_{w-1}]$ addresses corresponding to each group are found in tables in the European Telecommunications Standards Institute (ETSI) DVB-S2 standard [5]. Each row of the table corresponds to one of the groups of INs, i.e., each table has t rows. As an example, for code rate $r = \frac{1}{4}$ and codeword length $n = 64\,800$, Table B.1 from the ETSI DVB-S2 standard [3] has $t = \frac{k}{L} = \frac{k}{360} = \frac{16200}{360} = 45$ rows. This table, shown in Figure 2, can be seen as the union of a $t_1 \times w_1$ table and a $t_2 \times w_2$ (e.g. a 15×12 table and a 30×3 table, respectively). Thus, in this case, there are only two values for the degree w_c of each group: 12 and 3. The other standard tables (with different code rates and frame lengths) also have a very limited set of column-weight values. If the values in these matrices are to be stored in a Read Only Memory (ROM), this could be useful in developing a simpler and faster encoder: the total number of elements in each standard table is denoted by W.

The parameter $(t_1,\ w_1)$, (t_2, w_2) and W of these standard tables are described in Table 4 for Normal frames and Short Frames.

Finally, we can deduce from the Tanner Graph that the parity check matrix's structure is shown in Figure 3.

The **H** matrix can be divided into two sub-matrices: a sparse matrix $A_{(n-k)\times k}$, corresponding to the permutation matrix in the Tanner Graph, and $B_{(n-k)\times(n-k)}$, a lower staircase matrix corresponding to the *zig-zag structure* of the Tanner Graph. The **A** matrix is divided into t groups of $L = 360$ columns. For each of these groups, Eq 4 can be translated as follows: the first column of each group has non-zero elements at indices $[a_0,\ a_1,\ \ldots, a_{w-1}]$. The i^{th} column inside each group can be obtained by a downward logical rotation of the first column by $i \times q$. The first t_1

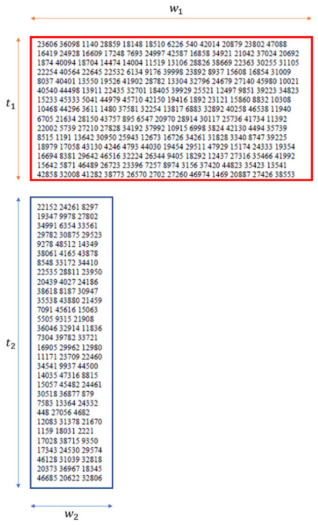

FIGURE 2. DVB-S2 Standard Table for n=64800 and code rate r=1/4.

TABLE 4. DVB-S2 standard Table parameters.

Normal frame (N = 64 800)						
Code Rate	*t*	t_1	w_1	t_2	w_2	*W*
1/4	45	15	12	30	3	270
1/3	60	20	12	40	3	360
2/5	72	24	12	48	3	432
1/2	90	36	8	54	3	450
3/5	108	36	12	72	3	648
2/3	120	12	13	108	3	480
3/4	135	15	12	120	3	540
4/5	144	18	11	126	3	576
5/6	150	15	13	135	3	600
8/9	160	20	4	140	3	500
9/10	162	18	4	144	3	504
Short frame (N = 16 200)						
Code Rate	*t*	t_1	w_1	t_2	w_2	*W*
1/4	9	4	12	5	3	63
1/3	15	5	12	10	3	90
2/5	18	6	12	12	3	108
1/2	20	5	8	15	3	85
3/5	27	9	12	18	3	162
2/3	30	3	13	27	3	120
3/4	33	1	12	32	3	108
4/5	35	0	0	35	3	105
5/6	37	1	13	36	3	121
8/9	40	5	4	35	3	125

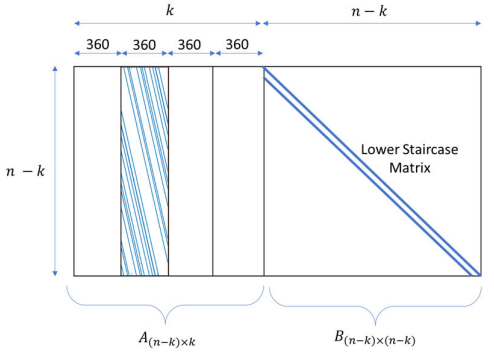

FIGURE 3. DVB-S2 parity check matrix (structure).

L-columns group all have the same column weight w_1; the latter t_2 L-columns group all have the same column weight $w_2 = 3$.

B. REPEAT AND ACCUMULATE ENCODING

As we have seen, the **H** matrix is defined in the standard as:

$$H_{(n-k)\times n} = [A_{(n-k)\times k} \;\; B_{(n-k)\times(n-k)}] \tag{6}$$

If we expand $A_{(n-k)\times k}$ *and* $B_{(n-k)\times(n-k)}$, we obtain the following identity:

$$H_{(n-k)\times n} = \begin{bmatrix} a_{0,0} & \cdots & a_{0,k-1} & 1 & 0 & 0 & \cdots & 0 & 0 \\ a_{1,0} & \cdots & a_{1,k-1} & 1 & 1 & 0 & \cdots & 0 & 0 \\ a_{2,0} & \cdots & a_{2,k-1} & 0 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-k-2,0} & \cdots & a_{n-k-2,\,k-1} & 0 & 0 & 0 & \cdots & 1 & 0 \\ a_{n-k-1,0} & \cdots & a_{n-k-1,\,k-1} & 0 & 0 & 0 & \cdots & 1 & 1 \end{bmatrix} \tag{7}$$

By applying the fundamental equation $H \cdot c^T = \mathbf{0}$ where $c = [i_0,\, i_1,\, \cdots\, i_{k-1},\, p_0,\, p_1,\, \cdots,\, p_{n-k-1}]$ is the codeword in systematic form, we obtain the following system of n-k parity check equations:

$$\begin{gathered} (a_{0,0}i_0 + a_{0,1}i_1 + \ldots + a_{0,k-1}i_{k-1}) + p_0 = 0 \\ (a_{1,0}i_0 + a_{1,1}i_1 + \ldots + a_{1,k-1}i_{k-1} + p_0) + p_1 = 0 \\ \vdots \\ (a_{n-k-1,0}i_0 + a_{n-k-1,1}i_1 + \ldots + a_{n-k-1,K-1}i_{k-1} + \\ + p_{n-k-2}) + p_{n-k-1} = 0 \end{gathered} \tag{8}$$

Since $x + x = 0$ in the GF(2) field, we can add the bracketed quantities to both sides of the equations and obtain the following:

$$\begin{gathered} p_0 = (a_{0,0}i_0 + a_{0,1}i_1 + \ldots + a_{0,k-1}i_{k-1}) \\ p_1 = (a_{1,0}i_0 + a_{1,1}i_1 + \ldots + a_{1,k-1}i_{k-1}) + p_0 \\ \vdots \\ p_{n-k-1} = (a_{n-k-1,0}i_0 + a_{n-k-1,1}i_1 + \ldots + \\ + a_{n-k-1,k-1}i_{k-1}) + p_{n-k-2} \end{gathered} \tag{9}$$

The input information sequence **i** mainly participates in the encoding process through the sub-matrix **A**, which can be expressed with the vector $S_{1\times(n-k)}$:

$$S_{1\times(n-k)} = i_k \cdot A^T_{k\times(n-k)} = [\, s_0,\; s_1 \ldots s_{n-k-1}] \tag{10}$$

For $r \in 0,\; 1 \ldots, n-k-1$:

$$s_r = a_{r,0}i_0 + a_{r,1}i_1 + \ldots + a_{r,k-1}i_{k-1} = \sum_{j=0}^{n-k-1} i_j \cdot a_{r,j} \tag{11}$$

By substituting back s_r into Eq. 3.7 we obtain the following equations.

$$\begin{gathered} p_0 = s_0 \\ p_1 = s_1 + p_0 = s_1 + s_0 \\ p_2 = s_2 + p_1 = s_2 + s_1 + s_0 \\ \vdots \\ p_{n-k-1} = s_{n-k-1} + p_{n-k-2} = s_{n-k-1} + s_{n-k-2} + \\ + \cdots + s_1 + s_0 \end{gathered} \tag{12}$$

The i^{th} parity check bit thus can be computed recursively as:

$$p0 = s0,\, pi = si + pi - 1 \tag{13}$$

Thus, we can obtain any parity check bit p_i through an accumulation of the s_r bits:

$$p_i = \sum_{r=0}^{i} s_r = s_i + s_{i-1} + \cdots + s_1 + s_0 \tag{14}$$

Thus, the LDPC encoding algorithm can be divided into two steps: computation of the $s_r^{prime}s$ (ii) accumulation of the $s_r^{prime}s$. For this reason, the DVB-S2 LDPC code is referred to as an Irregular Repeat Accumulate (IRA) code. However, it can be easily demonstrated that this simple encoding algorithm is not only inefficient in terms of speed, as it is only capable of outputting one parity bit at a time, but it is also utterly infeasible in terms of hardware complexity and resource utilization. To perform Eq. 11, we would need to store the entire $(n-k) \times kA$ matrix and perform a multiplication between input as the whole message (k bits) and one row of the **A** matrix (k bits). The main flaw of this algorithm is that it does not use either the sparseness of the **A** matrix or its structure. In the following sections, we shall present two more advanced algorithms that exploit both features.

C. VECTORISED IRA CODING

For $r \in 0,\; 1 \ldots N-K-1$:

$$s_r = \sum_{j=0}^{n-k-1} i_j \cdot a_{r,j} \tag{15}$$

The non-zero $a_{r,j}$ correspond to the j^{th} information node linked to check node r. We define IN(r) as the set of information nodes that are linked to check node r (r-th row of **A**). Thus, we can rewrite s_r in Tanner Graphs terminology as:

$$s_r = \sum_{z \in IN(r)} i_z \tag{16}$$

We can define the following algorithm:

Instead of processing the CNs individually, which requires a priori knowledge of all information bits, we can exchange the processing order from horizontal to vertical. We define CN(c) as the set of check nodes that are linked to the information node c (c^{th} column of **A**). So, for each new

```
Initialize s_r = 0
for r = 0 to (N − K − 1) do
  for each z ∈ IN(r) do
    s_r = s_r + i_z
  end for
end for
```

information bit i_c received by the encoder, the associated s_r values are updated according to:

```
Initialize s_r = 0
for c = 0 to (K − 1) do
  for each r ∈ CN(c) do
    s_r = s_r + i_c
  end for
end for
```

Now, considering the code periodicity, we shall show how each inner cycle of this algorithm can be performed simultaneously for $L = 360$ CNs, increasing the encoder throughput. We shall divide the input message $i = [i_0, i_1, \ldots i_m, \ldots i_{t-1}]$ into t groups comprised of consecutive 360 bits. Each group is defined as:

$$i_m = [i_{360m}, i_{360m+1}, i_{360m+2}, \ldots i_{360m+359}] \tag{17}$$

with $m \in 0, 1, \cdots, t-1$ The accumulation vector $S_{1\times(n-k)}$ in Eq. 10 shall be rearranged as $S\prime_{1\times(n-k)}$:

$$S\prime_{1\times(N-K)} = \left[S_0, S_1, \ldots, S_{q-1}\right] \tag{18}$$

where each S_j is a L=360 bits vector, defined as:

$$S_j = [s_j, s_{j+q}, s_{j+2q}, \ldots s_{j+359q}] \tag{19}$$

with $j \in 0, 1, 2, \ldots q-1$ Finally, we construct a $q \times L$ matrix, called S_M, with the S_j as its rows, as follows:

$$S_M = \begin{bmatrix} S_0 \\ S_1 \\ \vdots \\ S_{q-1} \end{bmatrix} = \begin{bmatrix} s_0 & s_q & \cdots & s_{359q} \\ s_1 & s_{1+q} & \cdots & s_{359q+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{q-1} & s_{2q-1} & \cdots & s_{360q-1} \end{bmatrix} \tag{20}$$

We only need to know CN(c) in order to find $CN(c + j)$, for $j = 1, 2, .., L - 1$. In fact, if CN_r is connected to IN_c, i.e. $r \in CN(c)$, then, $| r + j \cdot q|_{n-k} \in CN(c + j)$.

It is desirable for an implementation to store the 360 values of $[s_r, s_{|r+q|_{N-K}}, s_{|r+2q|_{N-K}} \cdots s_{|r+359q|_{N-K}}]$ values in the S_M matrix row-wise: this way we shall be able to store 360 bits word in a $q \times 360$ RAM, represented by S_M. However, $r \in \{0, 1, 2, \ldots N - K - 1\}$, so the S_j do not automatically line up with the q rows of S_M. Two ways to solve this issue are logical rotation of S_j or logical rotation of the input i_m. The second approach is preferable since we can easily vectorize the operation on the S_M matrix. We denote the left logical rotation of a vector by the bracketed apex $x^{(\ )}$ and modify the algorithm this way:

```
Initialize s_r = 0
for m = 0 to (t − 1) do
  for each r ∈ CN(c = 360m) do
    s_r = s_r + i_{360m}
    s_{|r+q|_{N−K}} = s_{|r+q|_{n−k}} + i_{360m+1}
    s_{|r+2q|_{N−K}} = s_{|r+2q|_{n−k}} + i_{360m+2}
    s_{|r+3q|_{N−K}} = s_{|r+3q|_{n−k}} + i_{360m+3}
    ⋮
    s_{|r+359q|_{N−K}} = s_{|r+359q|_{n−k}} + i_{360m+359}
  end for
end for
```

```
Initialize S_M = 0
for m = 0 to (t − 1) do
  for each r ∈ CN(c = 360m) do
    j = |r|_q (address)
    x = ⌊r/q⌋ (shift amount)
    Read old S_j value from S_M
    S_j = S_j+
    textbfi_m^(x)
    Write new S_j value to S_M
  end for
end for
```

Assuming that the S_M matrix is stored in a Single Port RAM, the read and write operations in the inner loop take two clock cycles (1 clock cycle each). For a given frame length and code rate, "each $r \in CN(c = 360m)$" means that for the m^{th} group of 360 input bits, at each iteration of the inner loop r assumes all the values stored in the corresponding m^{th} line in the Standard Table identified by the frame length and code rate. Thus, we can estimate the clock cycles and resource utilisation for implementing this algorithm on FPGA. We need $2 \times W$ clock cycles, a $q \times 360$ Single Port RAM and 21 Standard Tables, each with W rows and 19 bits words (8 bits for address j and 9 bits for shift amount x). One further fundamental observation can be made. The implementation of this algorithm relies on a complex addressing control module that uses a $q \times 360$ Single Port RAM inside a computational loop: in the context of this work, however, we strive to achieve a unidirectional data flow as much as possible as one of the requirements of the design is to employ an AXI Stream pipelined approach. We shall address these issues by exploring a different approach to the encoding problem. Having shown how to compute the S_j, we shall now define a vectorized algorithm to accumulate the S_j into the parity bits. Computing the checksum of all the columns of S_M, we obtain the following checksum vector:

$$s_c = \left[\sum_{i=0}^{q-1} s_i \ \sum_{i=q}^{2q-1} s_i \ \sum_{i=2q}^{3q-1} s_i \cdots \sum_{i=359q}^{360q-1} s_i\right] \tag{21}$$

It is important to notice that the checksum vector s_c can be computed in parallel to the computation of the S_j:

Initialize $S_c = \mathbf{0}$
for $m = 0$ to $(t-1)$ **do**
  for each $r \in CN(c = 360m)$ **do**
    $x = \left\lfloor \frac{r}{q} \right\rfloor$
    Read old s_c value from s_c register
    $s_c = s_c + i_m^{(x)}$
    Write new s_c value to s_c register
  end for
end for

Moving on, let $L_{L\times L}$ be a lower triangular matrix of ones:

$$L = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 1 & 1 & 0 & \dots & 0 & 0 \\ 1 & 1 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & 1 & \dots & 1 & 0 \\ 1 & 1 & 1 & \dots & 1 & 1 \end{bmatrix}_{L\times L} \tag{22}$$

By applying the linear transformation specified by $\mathbf{L}$ to the checksum vector, we obtain the following result:

$$\begin{aligned} &L \cdot s_c^T \\ &= \left[\textstyle\sum_{i=0}^{q-1} s_i \ \sum_{i=0}^{2q-1} s_i \ \sum_{i=0}^{3q-1} s_i \ \cdots \ \sum_{i=0}^{360q-1} s_i\right]^T \end{aligned} \tag{23}$$

Next, the vector s_c is logically shifted left of one bit to obtain the following vector, referred to as the parity initialization vector:

$$\begin{aligned} p_{init} &= \left[0 \ \textstyle\sum_{i=0}^{q-1} s_i \ \sum_{i=0}^{2q-1} s_i \ \cdots \ \sum_{i=0}^{359q-1} s_i\right] \\ &= \left[0 \ p_{q-1} \ p_{2q-1} \ \cdots \ p_{359q-1}\right] \end{aligned} \tag{24}$$

In this identity, we have applied Eq. 14. Furthermore, we notice that p_{init} can readily be obtained from the checksum vector s_c using a simple combinatorial lesson. Finally, we can now calculate L parity bits at a time by using the following procedure:

$$\begin{cases} [p_0 p_q \cdots p_{359q}] = \\ = [0 p_{(q-1)} \cdots p_{(359q-1)}] + [s_0 s_q \cdots s_{359q}] \\ [p_1 p_{(q+1)} \cdots p_{(359q+1)}] = \\ = [p_0 p_q \cdots p_{359q}] + [s_1 s_{(1+q)} \cdots s_{(1+359q)}] \\ [p_2 p_{(q+2)} \cdots p_{(359q+2)}] = \\ = [p_1 p_{(q+1)} \cdots p_{(359q+1)}] + [s_2 s_{(2+q)} \cdots s_{(2+359q)}] \\ \vdots \\ [p_{(q-1)} p_{(2q-1)} \cdots p_{(n-k-1)}] = \\ = [p_{(q-2)} p_{(2q-2)} \cdots p_{(n-k-2)}] + [s_{(q-1)} s_{(2q-1)} \cdots s_{(n-k-1)}] \end{cases} \tag{25}$$

Similarly to S_j, we define P_j as:

$$P_j = \left[p_j \ p_{j+q} \ p_{j+2q} \ \cdots \ p_{j+359q}\right] \tag{26}$$

Which can be rewritten recursively as:

$$\begin{cases} P_0 = p_{init} = [0 p_{q-1} p_{2q-1} \cdots p_{359q-1}] \\ P_j = S_j + P_{j-1} \end{cases} \tag{27}$$

Similarly to S_M, we shall define a $q \times L$ matrix containing the parity bits as follows:

$$\begin{aligned} &P_M = \\ &\begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ \vdots \\ P_{q-1} \end{bmatrix} = \begin{bmatrix} p_0 & p_q & p_{2q} & \cdots & p_{359q} \\ p_1 & p_{q+1} & p_{2q+1} & \cdots & p_{359q+1} \\ p_2 & p_{q+2} & p_{2q+2} & \cdots & p_{359q+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{q-1} & p_{2q-1} & p_{3q-1} & \cdots & p_{N-K-1} \end{bmatrix}_{q\times M} \end{aligned} \tag{28}$$

Finally, we can define the computation algorithm of the P_j.

Initialize $P_j = p_{initi}$
for $j = 0$ to $(q-1)$ **do**
  Read S_j from memory at address j
  $P_j = S_j + P_{j-1}$
  Write P_j to memory at address j
end for

Two crucial observations must be made. The first one is that p_{init}, being obtained from the checksum vector s_c, can be computed only after all S_j have been calculated. Secondly, this algorithm computes L=360 parity check bits at a time, which is desirable for high throughput architectures; however, these bits are not output in the natural order. Therefore, a reordering process is necessary. Although this is an important topic, much of the relevant literature underrates this problem: we shall propose a solution to this issue further on. Finally, other $2\times$ q cycles are needed for accumulation, for $2 \times (W + q)$ under the assumption that Simple One Port RAM is utilized.

D. VECTORIZED QUASI-CYCLIC ENCODING

We shall now present an alternative approach to the encoding algorithm based on performing a row permutation on $\mathbf{A}$. Extract a $q \times k$ matrix, denoted as A'_r, from $\mathbf{A}$, with $r \in 0, 1, \dots q-1$

$$A'_r = \begin{bmatrix} a_{r,0} & a_{r,1} & \cdots & a_{r,k-1} \\ a_{r+q,0} & a_{r+q,1} & \cdots & a_{r+q,k-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r+359q,0} & a_{r+359q,1} & \cdots & a_{r+359q,k-1} \end{bmatrix} \tag{29}$$

Reorganize the A'_r submatrices into $\mathbf{C}$. $\mathbf{C}$ is a row-wise permutation of $\mathbf{A}$.

$$C = \begin{bmatrix} A_{0'} \\ A'_1 \\ \vdots \\ A'_{q-1} \end{bmatrix} \tag{30}$$

The reshaped matrix $\mathbf{C}$ is composed of $q \times t$ cyclic matrices:

$$C = \begin{bmatrix} C_{0,0} & C_{0,1} & \cdots & C_{0,t-1} \\ C_{1,0} & C_{1,1} & \cdots & C_{1,t-1} \\ \vdots & \vdots & \ddots & \vdots \\ C_{q-1,0} & C_{q-1,1} & \cdots & C_{q-1,t-1} \end{bmatrix} \tag{31}$$

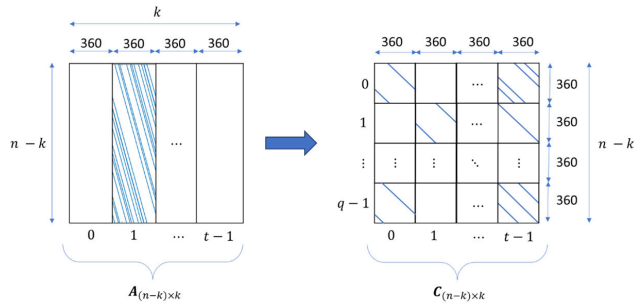

FIGURE 4. Reshaping of A matrix.

Each submatrix $C_{j,m}$ is a square $L \times L$, i.e. 360×360, cyclic matrix (with $j \in \{0,\ 1,\ \ldots q-1\}$, $m \in \{0,\ 1,\ \ldots t-1\}$), that is each row (or column) is the right logic rotation of the previous row (or column). The reshaping operation of the **A** matrix into a quasi-cyclic matrix **C** is represented in Figure 4:

The accumulation vector $S_{1\times(n-k)}$ in Eq. 10 is consequently rearranged as $S\prime_{1\times(n-k)}$:

$$S\prime_{1\times(N-K)} = \left[S_0,\ S_1,\ \ldots,\ S_{q-1}\right] \tag{32}$$

where each S_j is a L=360 bits vector, defined as:

$$\begin{aligned} S_j &= [s_j, s_{j+q}, s_{j+2q}, \ldots s_{j+359q}] \\ &with j \in 0,\ 1,\ 2,\ \ldots q-1 \end{aligned} \tag{33}$$

We shall also divide the input message $i = [i_0,\ i_1,\ \ldots\ i_m,\ \ldots i_{t-1}]$ into t groups comprised of consecutive 360 input bits, denoted by i_m. Eq. 10 can be rewritten as:

$$\begin{aligned} S'_{1\times(N-K)} &= i \cdot C^T \\ &= [i_0,\ i_1,\ \ldots, i_{t-1}] \\ &\cdot \begin{bmatrix} C^T_{0,0} & C^T_{1,0} & \cdots & C^T_{q-1,0} \\ C^T_{0,1} & C^T_{1,1} & \cdots & C^T_{q-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ C^T_{0,t-1} & C^T_{1,t-1} & \cdots & C^T_{q-1,t-1} \end{bmatrix} \end{aligned} \tag{34}$$

Thus, we deduce that each S_j can be computed as:

$$\begin{aligned} S_j &= \left[s_j, s_{j+q}, s_{j+2q}, \ldots s_{j+359q}\right] \\ &= [i_0,\ i_1,\ \ldots, i_{t-1}] \cdot \begin{bmatrix} C^T_{j,0} \\ C^T_{j,1} \\ \vdots \\ C^T_{j,t-1} \end{bmatrix} \end{aligned} \tag{35}$$

If the entire input message $i = [i_0,\ i_1,\ \ldots i_m,\ \ldots i_{t-1}]$ is stored in a memory, then it is possible to compute 360 bits in parallel and store them in a $q \times 360$ RAM as follows:

$$S_M = \begin{bmatrix} S_0 \\ S_1 \\ \vdots \\ S_{q-1} \end{bmatrix} = \begin{bmatrix} s_0 & s_q & \cdots & s_{359q} \\ s_1 & s_{1+q} & \cdots & s_{359q+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{q-1} & s_{2q-1} & \cdots & s_{360q-1} \end{bmatrix} \tag{36}$$

Furthermore, it can be shown that the computation of the S_j a simple task because of the sparseness of the matrix **C**, inherited by the parity check matrix, and because of the quasi-cyclic structure of the square sub-matrices $C_{i,j}$. By expanding Eq. 35 we get:

$$S_j = i_0 \cdot C^T_{j,0} + i_1 \cdot C^T_{j,1} + \cdots + i_{t-1} \cdot C^T_{j,t-1} = \sum_{m=0}^{t-1} i_m \cdot C^T_{j,m} \tag{37}$$

The first observation that can be made is that not all t transformations of the i_m vectors need to be executed, as the vast majority of the $C_{j,m}$ submatrices have all zero elements. Secondly, it is trivial to compute the product $i_m \cdot C_{j,m}$. Let us consider a quasi-cyclic $L \times L$ submatrix **D** and let $\alpha \in 0,\ 1, \cdots, L-1$ be the index of the only non-zero element in the first row of **D**. For example, if $\alpha = 0$, then **D** is simply an $L \times L$ identity matrix **I**. The rows of **D** are L-vectors over GF(2) denoted by $[d_0, d_1, d_2, \cdots, d_{L-1}]$. Let **u** and **v** be two L-vectors over GF(2) and consider the following identity:

$$D \cdot u^T = v^T \tag{38}$$

This equation can be expressed equivalently by the system of equations:

$$\begin{cases} d_0 \cdot u^T = v_0 \\ d_1 \cdot u^T = v_1 \\ \vdots \\ d_{L-1} \cdot u^T = v_{L-1} \end{cases} \tag{39}$$

Since the rows of **D** are rotated replicas of the first row, we can indicate a logical left rotation by the bracketed apex $x^{(\ )}$ and rewrite the equation above as follows:

$$\begin{cases} d_0 \cdot u^T = v_0 \\ d_0^{(1)} \cdot u^T = v_1 \\ \vdots \\ d_0^{(L-1)} \cdot u^T = v_{L-1} \end{cases} \tag{40}$$

Since d_0 only has one non-zero element in position α, then $d_0 \cdot u^T = v_0 = u_\alpha$. Consequently, it is clear that $d_0^{(1)} \cdot u^T = v_1 = u_{|\alpha+1|_L}$ where the modulo-L is a mathematical description of the fact that the circular non-zero element in **D** rotates over to the 0^{th} position after it has reached the $(L-1)^{th}$ position. By repeating the same process for all the other terms, we obtain the following result:

$$v^T = D \cdot u^T = \begin{bmatrix} u_\alpha \\ u_{|\alpha+1|_L} \\ u_{|\alpha+2|_L} \\ \vdots \\ u_{|\alpha+359|_L} \end{bmatrix} \Rightarrow v = u^{(-\alpha)} \tag{41}$$

In simpler terms, Eq. 41 means that the result of the multiplication between an L-vector **u** and a quasi-cyclic matrix **D** is an L-vector **v** that is the logical proper rotation of **u** by α, i.e. the position of the only non-zero bit in the first row of **D**. Returning to the discussion of Eq. 3.34, we note

TABLE 5. Index-Alfa Table (frame length = 16200, code rate =1/2).

m	α	m	α	m	α	m	α	m	α
5	0	9	26	19	222	-	-	-	-
1	125	2	132	2	323	6	0	-	-
3	217	3	248	4	112	7	0	14	45
1	107	4	280	8	0	17	239	-	-
0	106	9	0	-	-	-	-	-	-
6	246	10	0	13	237	-	-	-	-
11	0	13	176	-	-	-	-	-	-
2	220	12	0	18	318	-	-	-	-
0	154	8	314	13	0	14	175	-	-
5	83	14	0	15	205	-	-	-	-
4	313	15	0	16	3	-	-	-	-
0	198	0	265	16	0	19	64	-	-
0	318	0	332	7	352	17	0	-	-
2	263	4	310	18	0	18	121	-	-
1	237	8	223	17	330	19	0	-	-
2	233	4	155	10	349	-	-	-	-
3	317	6	358	-	-	-	-	-	-
3	174	4	171	11	302	12	271	-	-
1	259	2	213	15	86	-	-	-	-
2	350	7	93	-	-	-	-	-	-
0	0	0	159	3	180	12	48	-	-
1	168	2	0	4	101	9	184	-	-
1	131	1	267	3	0	-	-	-	-
3	148	3	183	4	0	10	124	11	199

that some of the non-zero $C_{j,m}$ are, in fact, quasi-cyclic $L \times L$ submatrix with only one circulating '1', whereas others may have more than one circulating '1'. In the first case, all we need to know to compute the product $i_m \cdot C_{j,m}$ is m, i.e. the index identifying what group of 360 input bits is currently selected, and the α value relative to $C_{j,m}$. Thus, we have:

$$i_m \cdot C_{j,m} = i_m^{(-\alpha)} \tag{42}$$

In the latter case, we can keep the index value m constant and express $C_{j,m}$ as a sum of multiple quasi-cyclic $L \times L$ submatrix with only one circulating '1', each characterized by a distinct α value. For example, if there are three circulating '1's, we shall define three αvalues, i.e. α_1, α_2, α_3, and compute $i_m \cdot C_{j,m}$ as follows:

$$i_m \cdot C_{j,m} = i_m^{(-\alpha_1)} + i_m^{(-\alpha_2)} + i_m^{(-\alpha_3)} \tag{43}$$

Each S_j can be computed by an accumulation of 360-bit entries selected from the input message, i.e. i_m, rotated by an amount α. Therefore, we must extrapolate all the valid (m, α) couplets for any q rows of $L \times L$ quasi-cyclic matrices. These couplets are a complete and equivalent description of the **C** matrix. One example of a Look-Up Table (LUT) for $framelength = 16200$ and $nominalcoderate = 1/2 (actualcoderate = 4/9)$ containing the (m, α) couplets are given in Table 5: it is composed of q=25 rows and a varying number of columns, which is equal to the row-weight w_r of each of the selected row of $L \times L$ quasi-cyclic matrices: in this case, w_r assumes every value between 2 and 5.

The total number of elements (with two entries) in the Index-Alfa Tables could be computed by summing all the row-weights w_r of **C**. However, there is a more straightforward and more meaningful method. Let us compare the

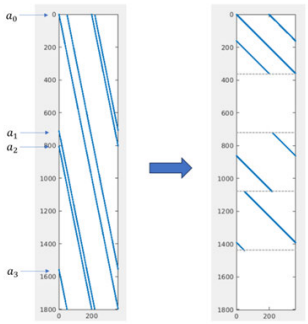

FIGURE 5. Comparing the first 360 columns of A (left) and C (right).

first 360 columns of **A** with the first 360 columns of **C**, as exemplified in Figure 5 for $framelength = 16200$ and $nominalcoderate = 8/9 (q = 5, \ n - k = 1800)$.

We can quickly identify four $L \times L$ quasi-cyclic matrices with only one rotating '1' and their (m, α) couplets. They correspond to the original four "rotating diagonals" specified by the four elements $[a_0, a_1, a_2, a_3]$ from the first row of the Standard Table. Therefore, by repeating the same observation for all the 360-column groups, we may deduce that there are as many (m, α) couplets in the Index-Alfa Table as there are elements in the Standard Tables, i.e. the Index-Alfa table contains $W(m, \alpha)$ couplets (for the W values, see Table 4). Finally, we may define the following algorithm.

Initialize $P_j = p_{initi}$
for $j = 0$ to $(q - 1)$ **do**
  Initialize $S_j = 0$
  for each $(m, \alpha) \in j - throwoftheIndex - AlfaTable$ **do**
    $S_j = S_j + i_m^{(-\alpha)}$ (now we can store S_j in j^{th} row of S_M)
  end for
end for

Finally, the second step of the algorithm, computing the parity check bits, is identical to Vectorized IRA Encoding. From a timing standpoint, this algorithm requires only W+q cycles, which is better than the previous one. We also showed that the Standard Table and the Index-Alfa tables are equivalent. This approach is more suitable for unidirectional data flow as it does not rely on a complex loop involving RAM as a computational building block. These benefits come at the cost of an increased memory footprint as the entire input message needs to be stored. As already discussed, latency also increases, but it is not an issue in practical applications.

III. REGISTER TRANSFER LEVEL DESIGN

The proposed encoder architecture introduces the following key innovations. **I) Full Compliance with DVB-S2 Standard**. The encoder will be designed to comply with the DVB-S2 Standard for all MODCODs. This will enable the utilization of ACM and VCM techniques, optimizing transmission efficiency by adapting to varying channel conditions. **II) Highly Reconfigurable I/O Interfaces**. The input and

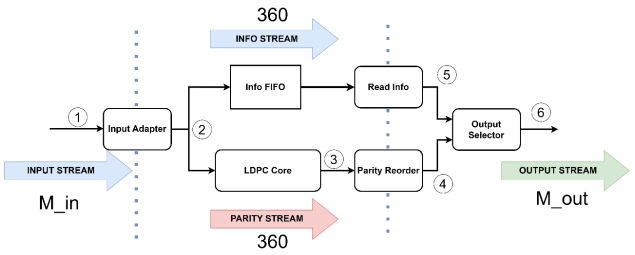

FIGURE 6. Block Level Architecture.

output interfaces of the LDPC Encoder will be designed for high reconfigurability, allowing easy adaptation to different operational requirements and facilitating seamless integration into diverse systems. A set of 12 values for the input parallelism and another set for the output parallelism, also of 12 values, parallelism, is available. The values of M_{in} and M_{out} are the following:

- $M_{in} = 2,\ 3,\ 4,\ 6,\ 8,\ 12,\ 16,\ 24,\ 32,\ 48,\ 64,\ 96$
- $M_{out} = 2,\ 3,\ 4,\ 6,\ 8,\ 12,\ 18,\ 24,\ 36,\ 60,\ 72,\ 120$

We shall refer to the input parallelism as M_{in} and the output parallelism as M_{out}. M_{in} and M_{out} are chosen at compile time independently from each other (for a total of 144 possible configurations). However, it is preferable to choose between two reasonably close values. No comparable architecture in the literature shows such a range of reconfigurability while maintaining high throughput for the higher parallelism configurations. **III) High Throughput**: The design will prioritize achieving high throughput to achieve the throughput requirements while maintaining acceptable resource utilization. To achieve this goal, the architecture shall be based on a non-reconfigurable LDPC Encoder Core utilizing a 360-bit stream to implement the high-speed Vectorized Quasi-Cyclic Encoding algorithm described in Chapter 3.4. However, this choice challenges adapting the reconfigurable I/O interfaces to work with the Core for all M_{in} and M_{out} values.

A. BLOCK LEVEL ARCHITECTURE

As depicted in Figure 6, the proposed architecture is based on four streams: Input Stream, Info Stream, Parity Stream, and Output Stream

The Input Stream is characterized by a M_{in} bits data payload. Since our circuit is meant to be interfaced with the BCH, Encoder proposed in [20], which only adds as much as 192 bits of redundancy, M_{in} was conveniently chosen as a divisor of 192. This parameter can be adjusted to different values to support other BCH stages. Ideally, the Input Stream frame should be comprised of $N_{bch} = K_{ldpc}$ bits; however, depending on the MODCOD, N_{bch} is not always divisible by M_{in}. Therefore, the output of the BCH Encoder operates zero padding in such a way that the actual frame length, denoted by $N'_{bch} = K'_{ldpc}$, is divisible by M_{in}. The Input Adapter will eliminate the zero padding and adapt from M_{in} to L=360 to fulfil the requirements of the LDPC Core component. The output of the Input Adapter is a $t \times 360$ frame. Because the

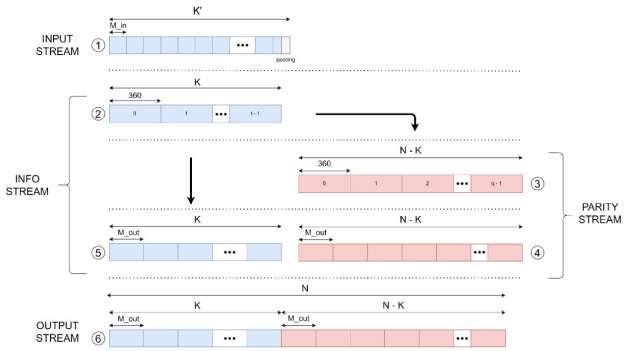

FIGURE 7. DVB-S2 LDPC Encoder Framing Structure.

DVB-S2 LDPC Code is systematic, the Input Stream can be divided into two streams. The first one, denoted as Parity Stream, undergoes the high-speed Quasi-Cyclic Encoding algorithm performed by the LDPC Core, which is divided into two stages: the first stage computes the S_j vectors; the second stage accumulates the S_j vectors into the P_j vectors. Since the P_j vectors contain the parity bits out of order, a Parity Reorder block is introduced to provide the output parity bits in natural order. This block also adapts the $L = 360$ bits Parity Stream to the output parallelism M_{out}, whose values are conveniently chosen to be all divisors of 360. On the other hand, the Info Stream contains the same information as the Input Stream but with an L=360 bits data payload instead, facilitating output adaptation. No operation needs to be carried out on the Info Stream; thus, it is only buffered on a FIFO, denoted as Info FIFO. The Read Info module is then used to read 360-bit words from the Info FIFO, which are, in turn, adapted to the output parallelism. Finally, an Output Selector switches over the Info Stream and the Parity Stream to provide the Output Stream. The framing structures of the proposed encoder architecture are shown in Figure 7: for the numbered references, see Figure 4.5.

B. INPUT ADAPTER

The Input Adapter takes M_{in} bits as inputs and outputs L=360 bits. As explained earlier, the Input Adapter shall execute two tasks: I) eliminate the zero-padding added by the output of the BCH Encoder; II) adapt from M_{in} to the core parallelism of L=360. The Input Adapter may be divided into two stages to execute these tasks. The first stage performs adaption from M_{in} to the first divisor of L=360 that is greater than M_{in} (i.e. 32 to 36), which we shall denote as $M'_{in} \geq M_{in}$. This first stage is omitted if $M_{in} = M'_{in}$ is already a divisor of 360. The second stage accumulates M'_{in} bits from the first stage into L=360 bits.

C. PARITY STREAM

The sub-modules of the LDPC Core are shown in Figure 8. The Sum Core stage reads 360-bit words from the Input Message RAM based on the addresses from the Index-Alfa Tables and computes q vectors, the S_j. Since the S_j are computed from 0 to q-1 and must be read with the same

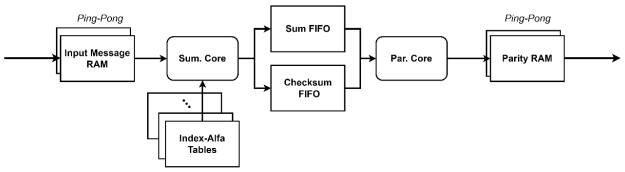

FIGURE 8. LDPC Core Architecture.

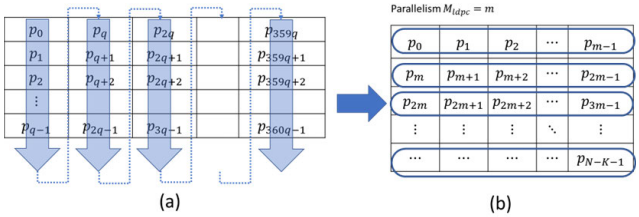

FIGURE 9. Parity Memory Reorder Issue.

order, a FIFO, denoted as Sum FIFO, can be used as a buffer. The Sum Core stage also computes the checksum vector s_c in parallel and writes it into the small Checksum FIFO, which has only two 360-bit words. Since the checksum vector s_c only becomes valid in correspondence with the last S_j, i.e. S_{q-1}, the second stage of the Core, denoted as Parity Core, must wait until it is possible to read the checksum vector s_c from the Checksum FIFO. Having read s_c, the initialization vector p_{init} may be computed. Then, the Parity Core shall pop the S_j from the Sum FIFO and accumulate them into the P_j, written into the Parity RAM.

1) PARITY REORDER

The proposed quasi-cyclic encoder generates the parity bits in order of the rows of $\mathbf{C}$ in the form of Eq. 28 ($P_M matrix$), which is a permutation of the original parity check matrix $\mathbf{H}$. On the other hand, the output requires its input bits in the order of the original, i.e. the natural order. This difference in the row order raises the following design challenge. Parity bits of the quasi-cyclic code is in the permuted matrix's order and stored in the single port RAM, as indicated in Figure 9 (a). However, the output requires parity bits to be read from the RAM in the order indicated in Figure 9 (b). Since multiple words cannot be accessed simultaneously at a RAM, we need more ideas for this ordering.

We solved this problem by partitioning the Parity Memory RAM into $360/M_{out}$ sections along the word direction (i.e. this is the hardware equivalent of dividing the Parity Matrix P_M in groups of M_{out} columns): using a counter (ranging from 0 to $360/M_{out} - 1$) and a $360/M_{out}$: 1 MUX with $M_{out} - bits$ lines, we shall read M_{out} bits at a time. After q cycles, the entire partition is read: we shall store it in a buffer memory using Flip-FLops (FF) registers. If we transpose the $q \times M_{out}$ partition into an $M_{out} \times q$ matrix, we have observed that not only the q bits from one column are reordered, but the entire partition may also be reordered. Then, we may read the transposed partition M_{out} bits at a time: this also takes q cycles. Using a Ping-Pong strategy, we may use q cycles to read and transpose the matrix and one cycle to copy the

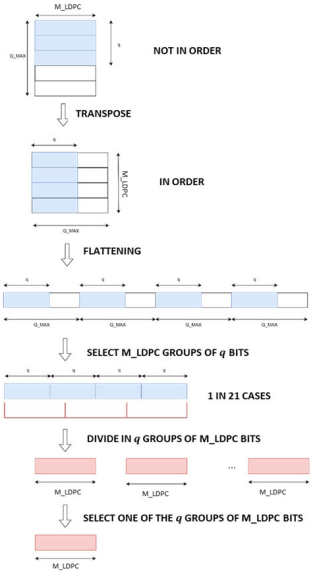

FIGURE 10. Simple Process of Transpose & Selection: Routing issue.

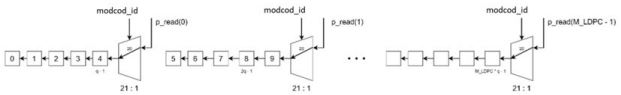

FIGURE 11. Reorder with shift register for q = 5.

transpose matrix in a buffer. Now, we may read the buffer in q cycles and the subsequent partition of the Parity Memory. Implementing the matrix transpose operation in hardware is not a trivial task due to the dependence of q on the MODCOD: 21 different transpose operations must be implemented and configurable on the fly. The easiest way to implement the transpose operation and the output bit selection is shown in Figure 10: after the transpose, a flattening operation (which is merely a rearrangement of bits) produces a long flattened vector with $q_{MAX} \times M_{out}$ bits. A selection of these bits should then be performed according to the specific q value; however, this is where problems arise, as we would need to use at least $\sum_{i=0}^{21} (q_i \times M_{out})$ wires to create 21 configurations of $q_i \times M_{out}$ bits in order. In the worst case, for $M_{out} = 120$, at least 110400 connections should be routed in the FPGA.

We must find a way to perform the flattening operation. The idea is to implement a reconfigurable shift register, which we call a reorder vector, with $q_{MAX} \times M_{out}$ bits in which the input M_{out} bits are shifted for q cycles. At the end of this shifting operation, we shall have $q \times M_{out}$ valid bits in the lower part of the register. For example, for q=5 (Short Frame, Code Rate 8/9), we may have the architecture represented in Figure 11, where *p_read(i)* are the bits read from the partition of the Parity Memory.

Now, we shall show how to make the structure in Figure 11 reconfigurable on the fly for all MODCODs. First, we have computed all the multiples of q from $q \times 1$ to $q \times M_{out}$ for all possible q values. Many values are repeated, as shown in Figure 12 for a subset of the likely multiples. It is then possible to extract a set of non-repeating multiples of q.

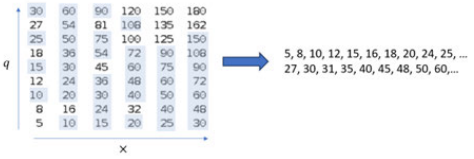

FIGURE 12. Multiples of q.

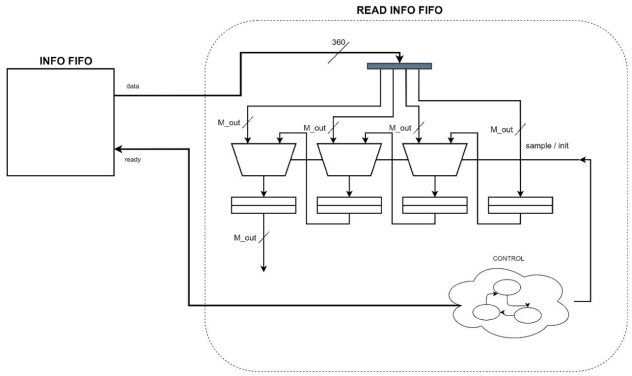

FIGURE 13. Read Info Architecture.

Two fields are associated for each non-repeating multiple of q. The first field is given by all the possible Physical Layer (Signaling) (PL(S))-Codes related to the q multiple: since there are only 21 valid combinations, a number ranging from 0 to 20, denoted as $modcod_id$, may be obtained from the PL(S)-Code. We also need to know what bit shall be selected from the input M_{out} bits read from the Parity Memory: this field is denoted as an index and ranges from 0 to $M_{out} - 1$. Then, we can define the following LUTs for every output parallelism M_{out} and optimize the MUXs involved in the reordering process.

It is important to note that this table is not stored in the FPGA. Instead, it is read at compile time to define the following structure of MUXs that feed data to the reorder vector.

D. INFO STREAM

The Read Info FIFO module shall be designed with the following architecture. The controller shall issue a pop of one 360-bit word from the FIFO. The word shall be sampled in $360/M_{out}$ shift registers with M_{out} bits. After the sampling phase, the controller issues a shift operation by toggling the input selection of the MUXes in Figure 4.24: the output is then read out from the last shift register M_{out} bits at a time.

Furthermore, the Info FIFO must be dimensioned with care. Suppose we assume that the row-column interleaver, placed after the LPDC Encoder, always provides backpressure to the encoder (worst-case scenario). In that case, it is still desirable that the data proceed forward through the pipeline stages in the Parity Stream and the Info Stream as much as possible. As we have seen, the Parity Stream may contain as high as:

- 2 frames in the Ping-Pong Input Message RAM

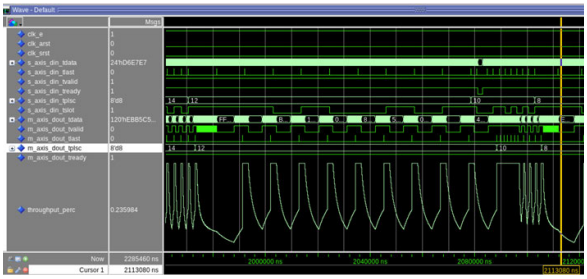

FIGURE 14. Throughput Simulation for $M_{in} = 24$ and $M_{out} = 120$.

- 2 frames in the Sum FIFO (it is, in fact, the parallel Checksum FIFO that limits the Sum FIFO to accommodate only two frames)
- 2 frames in the Ping-Pong Parity Memory

Therefore, we must design the Info FIFO to accommodate six information frames, i.e. six frames of $t \times 360$ bits. In the worst case, for $t = t_{MAX} = 162$, we need a 972×360 RAM.

IV. RESULTS

A. THROUGHPUT SIMULATION

In this section, we present a way to estimate the output throughput of our design since, even in full throughput mode (downstream ready always asserted), the proposed encoder may not always be able to provide valid data at every cycle. Thus, we cannot simply compute the output throughput according to the following equation:

$$T_{out} = M_{out} \times f_{ck} \tag{44}$$

Considering Figure 14, it is evident that the output valid waveform is not human-readable. Additionally, due to the requirement of evaluating throughput for all 144 combinations of input and output parallelism, it becomes apparent that conducting a simulation is essential. To do so, we introduced in our verification flow the capability to generate multiple frames (10) with the same PL(S)-Code. Then, we configured the VHDL testbench to achieve full throughput. We exploited the simulation to measure the percentage of throughout effectively used, which we call throughput_percent. We can then modify Eq. 44 by defining an equivalent output parallelism $\bar{M}_{out} = M_{out} \times throughput\_percent$ and compute the output throughput as:

$$T_{out} = \bar{M}_{out} \times f_{ck} = M_{out} \times throughput_percent \times f_{ck} \tag{45}$$

To guarantee that the encoder has achieved a consistent operating condition before saving the throughput measurements, the throughput percentage values are stored in a text file only when the last frame of every block of five frames raises the previous flag. By simulating every input and output parallelism, we can generate look-up tables that map equivalent output parallelism to every valid code. $M_{out} = 36$ is the lowest available output parallelism greater or equal to $M_{in} = 32$. In this case, we can see that the output valid is almost always asserted: using the simulation method just described, we can prove that the equivalent output parallelism

TABLE 6. Equivalent output parallelism for $M_{in} = 24$ and $M_{out} = 120$.

M_{in}	M_{out}	Nominal Code Rate	Actual Code Rate	Frame Length	Through. Percent	Equiv. Output Paral.
24	120	1/4	1/4	Normal	0.80	96.0
24	120	1/3	1/3	Normal	0.60	72.0
24	120	2/5	2/5	Normal	0.50	60.0
24	120	1/2	1/2	Normal	0.40	48.0
24	120	3/5	3/5	Normal	0.33	40.0
24	120	2/3	2/3	Normal	0.30	36.0
24	120	3/4	3/4	Normal	0.27	32.0
24	120	4/5	4/5	Normal	0.25	30.0
24	120	5/6	5/6	Normal	0.24	28.8
24	120	8/9	8/9	Normal	0.23	27.0
24	120	9/10	9/10	Normal	0.22	26.7
24	120	1/4	1/5	Short	1.00	120.0
24	120	1/3	1/3	Short	0.60	72.0
24	120	2/5	2/5	Short	0.50	60.0
24	120	1/2	4/9	Short	0.45	54.0
24	120	3/5	3/5	Short	0.33	40.0
24	120	2/3	2/3	Short	0.30	36.0
24	120	3/4	11/15	Short	0.27	32.7
24	120	4/5	7/9	Short	0.26	30.9
24	120	5/6	37/45	Short	0.24	29.18
24	120	8/9	8/9	Short	0.23	27.0
1	168	2	0	4	101	9
1	131	1	267	3	0	-
3	148	3	183	4	0	10

is always higher or equal to $M_{in} = 32$. We have verified that, for any given M_{in} if we select M_{out} to be the lowest available output parallelism greater or equal to M_{in}, then the equivalent output parallelism $\bar{M}_{out}$ is always greater or equal to M_{in}. This means that if the previous block in the transmitter chain, i.e. the BCH encoder, is capable of guaranteeing an input throughput of $M_{in} \times f_{ck}$, then our proposed encoder is capable of maintaining at least this throughput value. This important observation shall be the basis for the choice of the input and output parallelism in implementation. What happens if we select a higher output parallelism? We shall explain it through an example throughput simulation, as shown in Figure 14 for $M_{in} = 24$ and $M_{out} = 120$ where we have demonstrated throughput_percent using the analogue format option. The throughput per cent waveform is only low. There is no underlying issue in our design. The reason for this waveform shape is that any systematic block encoder with an input throughput T_{in} can only reach an output throughput as high as $T_{out} = \frac{T_{in}}{r_c}$. Therefore the equivalent output parallelism shall be given by $\bar{M}_{out} = \frac{M_{in}}{r_c}$. This simulation's Equivalent Output Parallelism Table for $M_{in} = 24$ and $M_{out} = 120$ is given below.

Analysing the equivalent output parallelism values in Table 6, we have verified that the equation $\frac{\bar{M}_{out}}{M_{in}=\frac{1}{r_c}}$ is verified. The proposed encoder may be able to raise its output equivalent parallelism and its output throughput even higher than this boundary, but only for a limited amount of time: an example of this behaviour is illustrated in Figure 14 for PL(S)-Code = 10, as this short frame (with code rate 1/3) follows a normal frame (with code rate 2/5).

B. IMPLEMENTATION

Design synthesis and implementation were conducted in Vivado on the target FPGA Xilinx Kintex Ultrascale XQRKU-060. Since the proposed encoder is an internal transmitter module, mapping the encoder I/O interfaces to the FPGA is not of concern. We have set the synthesis and implementation in out-of-context mode to account for this. Moreover, since in the course of the encoder design, we have placed much emphasis on throughput optimization, we have elected to instruct the synthesizer to use more hardware resources (within reason) to minimize the critical register-logic-register path, thus reducing the Worst Negative Slack (WNS) and boosting the maximum clock frequency achievable by the encoder. As indicated in the previous section, selecting M_{out} as the lowest output parallelism so that $M_{out} \geq M_{in}$ is sufficient to avoid a bottleneck scenario. Considering that the core is monolithic and the impact of input adaption on resources is relatively minor compared to the significant effect of the complex reordering module, opting for this choice is suggested, as it represents the optimal trade-off between throughput and resource utilization. Therefore, we implemented the encoder in twelve configurations, characterized by the following (M_{in}, M_{out}) couplets: $(2,\ 2)$, $(3,\ 3)$, $(4,\ 4)$, $(6,\ 6)$, $(8,\ 8)$, $(12, 18)$, $(16, 18)$, $(24, 24)$, $(32, 36)$, $(48, 60)$, $(64, 72)$, $(96, 120)$. We can visualize these couplets as points in an x-y plane (Figure 15).

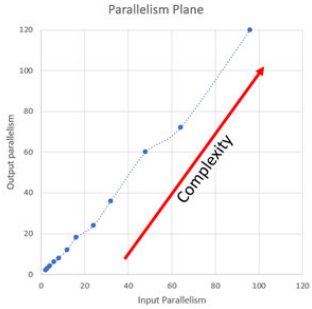

FIGURE 15. Parallelism Plane (Implementation).

C. MAXIMUM CLOCK FREQUENCY AND THROUGHPUT

Different clock frequency targets were selected based on the parallelism couplet to define the design constraints, ranging from 180 MHz to 280 MHz. We observe an initial plateau for low input and output parallelism: in this region, the limiting factor is the monolithic core. However, as the output parallelism increases, the Reorder Parity module becomes increasingly complex and resource-consuming, becoming the limiting factor in achieving higher clock frequencies. Clock Frequency and Estimated Max Clock Frequency are shown in Figure 16.

As discussed earlier, if the output parallelism is identical to the input parallelism, then we can compute the output throughput simply as $T_{out} = M_{out} \times f_{ck}$. However, if the output parallelism is strictly more significant than the input parallelism, then the output throughput becomes code rate dependent; however, this dependency is limited to a small set of codes for which we can compute the equivalent output

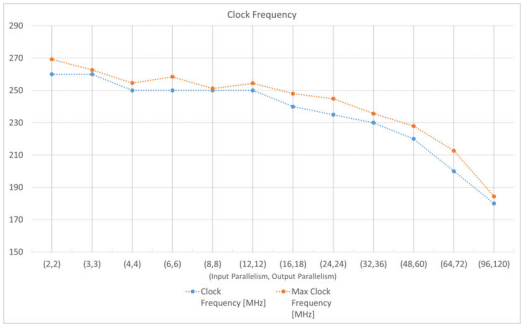

FIGURE 16. Clock Frequency over Parallelism Couplet.

TABLE 7. Equivalent output parallelism (Implementation).

M_{in}	M_{out}	Equivalent Mout	Frame Length	Code Rate
32	36	35.6	Normal	9/10
48	60	57.6	Normal	5/6
48	60	54.0	Normal	8/9
48	60	53.3	Normal	9/10
48	60	54.0	Short	8/9
64	72	71.1	Normal	9/10
96	120	115.2	Normal	5/6
96	120	108.0	Normal	8/9
96	120	106.7	Normal	9/10
96	120	108.0	Short	8/9

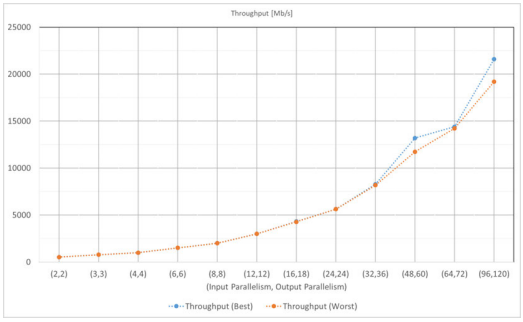

FIGURE 17. Throughput over Parallelism Couplet.

TABLE 8. Clock frequency and throughput over parallelism couplet.

M_in	M_out	Clock Frequency [MHz]	Throughput [Mb/s]	
			(Best)	(Worst)
2	2	260	520	520
3	3	260	780	780
4	4	250	1000	1000
6	6	250	1500	1500
8	8	250	2000	2000
12	12	250	3000	3000
16	18	240	4320	4266.7
24	24	235	5640	5640
32	***36***	230	8280	8177.8
48	60	220	13200	11733.3
64	72	200	14400	14222.2
96	120	180	21600	19200

parallelism, as reported in Table 7. The throughput curves over the parallelism couplet are shown in Figure 6.4, where the worst throughput is referred to normal frames with a code rate of 9/10.

Finally, we obtained the results reported in Table 8.

TABLE 9. Block RAM utilization.

Memory	Utilization [Tiles]	Utilization [%]	Utilization [Footprint]
Input Message BRAM	10	0.92	360 Kb
Info FIFO	10.5	0.97	378 Kb
Index-Alfa Tables	10.5	0.97	378 Kb
Sum FIFO	5.5	0.51	198 kB
Parity Memory	10	0.92	360 Kb
Read Reorder Vector	0.5	0.046	18 Kb
Tot.	**47**	**4.35**	**1 692 Kb**

TABLE 10. Resource utilization (LUT, FF).

Input parallelism	Output parallelism	LUTs		FFs	
		%	Tot.	%	Tot.
2	2	2.18	7229	1.22	8063
3	3	2.22	7369	1.26	8328
4	4	2.27	7527	1.30	8601
6	6	2.36	7841	1.38	9136
8	8	2.45	8127	1.46	9684
12	12	2.63	8716	1.62	10768
16	18	2.91	9643	1.88	12456
24	24	3.17	10500	2.12	14039
32	*36*	5.14	17054	2.62	17392
48	60	7.16	23756	3.61	23966
64	72	8.19	27153	4.11	27283
96	120	12.17	40354	6.11	40526

TABLE 11. Vivado power report.

In. paral.	Out. paral.	Clock Freq. [MHz]	Total [W]	Static [W]	Dynamic [W]
2	2	260	1.4	0.633	0.767
3	3	260	1.42	0.633	0.787
4	4	250	1.379	0.633	0.746
6	6	250	1.384	0.633	0.751
8	8	250	1.413	0.633	0.78
12	12	250	1.385	0.633	0.752
16	18	240	1.385	0.633	0.752
24	24	235	1.361	0.633	0.728
32	36	230	1.446	0.633	0.813
48	60	220	1.467	0.633	0.834
64	72	200	1.411	0.633	0.778
96	120	180	1.403	0.633	0.77

1) RESOURCE UTILIZATION

Because the core is monolithic, the Block RAM utilization is constant for all configurations: 47 Block RAM tiles are utilized according to Table 9. One tile is a 36 Kb RAM or two fully independent 18 Kb RAM (half tile mode). There are 1080 tiles for a total of 38.880 Mb.

In contrast, FF Utilization and LUT Utilization, reported in Table 10, demonstrate an approximately exponential increase as the system's complexity rises (see Figure 18 and Figure 19). In the Kintex UltraScale XQRKU060 FPGA, the LUTs can be configured as either one 6-input LUT with one output or two 5-input LUTs with separate outputs but common inputs. This is useful to ensure a fair and unbiased evaluation between different technologies and manufacturers.

2) POWER REPORT

The Power Report was obtained from Vivado for all configurations. The Static Power only depends on the selected FPGA

TABLE 12. Comparison between State-of-the-art DVBS2 Encoder and the proposed encoder (Resource Utilization).

Ref.	Hardware	Clock Frequency [MHz]	Throughput [Mb/s]	Resources			Comments
				LUT	FF	RAM [Kb]	
[8]	EP2C35F484C8 (Altera Cyclone II)	125	125	75	958	100	
[10]	Xilinx Kintex Ultrascale	350	350	4796	2988	216	Supports all 55 DVB-S2X codes
[11]	Altera Stratix EP1S80	50	400	11840	2000*	44	*2000 FFs is a measure of complexity.
[12]	Xilinx XC7K325t	390	1190	6618	4568	1224	
[13]	XC2VP30 Virtex2P Xilinx	132	10780	N/A	821	489.6	Monolithic core only.
[17]	Xilinx Virtex5-LX155T	100	10000	N/A	36801	N/A	
[16]	Xilinx Kintex-7 FPGA	280	47500	37542	51139	36	Worst case for one DVB-S2X code rate and frame length
Proposed Enc.	FPGA Xilinx Kintex Ultrascale XQRKU-060	260 to 180	520 to 21600	7229 to 40354	8063 to 40526	1692	Ranges depend on parallelisms

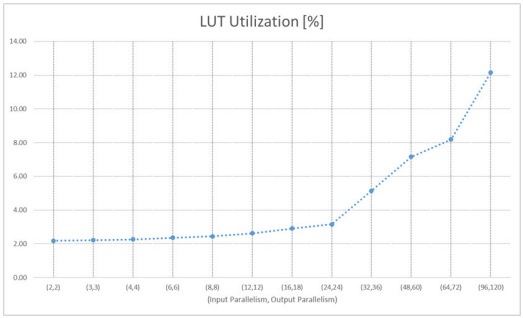

FIGURE 18. LUT utilization over parallelism couplet.

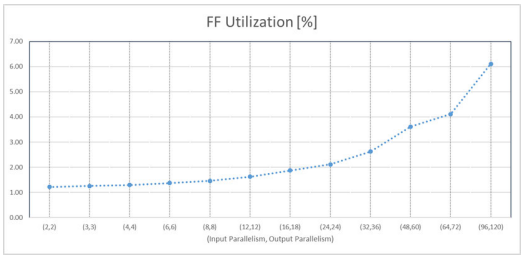

FIGURE 19. Flip-Flop utilization over parallelism couplet.

and equals 0.633 W. Conversely, the Dynamic Power depends on the clock frequency (ideally linearly) and resource utilization. The clock frequency constraint is steadily diminished after the configuration with $(M_{in}, M_{out}) = (12,\ 12)$. The BRAM utilization does not scale up well with higher input and output parallelisms, and it has the most significant impact on Dynamic Power, as it ranges from 57% for $(M_{in}, M_{out}) = (2,\ 2)$ at $f_{ck} = 260\ MHz$ to 40% for $(M_{in}, M_{out}) = (96,\ 120)$. Therefore, even though the resource utilization rises in a quasi-exponential way for LUTs and FFs, the combination of the lowering of the frequency and the impact of non-scalable BRAM resources on the Dynamic Power is responsible for the fact that the Dynamic Power also does not scale up well with higher configurations of (M_{in}, M_{out}). The Dynamic Power ranges from 0.751 W to 0.834 W.

V. CONCLUSION

The main features of the proposed encoder are Full DVB-S2 compliance, complete characterisation on state-of-the-art flight hardware (XQRKU-060), throughput configurable from 0.52 to 21 Gbps largely reconfigurable (up to 144 degrees of reconfigurability) By comparing these results with state-of-the-art LDPC Encoders, we can see that the proposed encoder presents unprecedented levels of reconfigurability while achieving very high throughput (although the technological advancements in hardware complicate a fair comparison between the proposed encoder and the high throughput encoders in [13] and in [15]). The proposed architecture offers the end-user significant flexibility, allowing them to choose the input and output parallelism independently. As a result, our encoder provides a high degree of reusability, enabling it to adapt to different requirements and scenarios and making it an ideal choice for developing a reconfigurable IP. To achieve excellent results in throughput, the high-speed encoding algorithm was implemented in hardware using an AXI-Stream Pipelined approach. Table 12 compares the resource utilization of the state-of-the-art DVBS2 Encoder and the proposed encoder about resource utilization. The proposed encoder does not stack up well with encoders that are resource-optimized for relatively low throughput applications, such as the one presented in [10], which in turn offers no reconfigurability and indeed very low throughput even with a high 350 MHz clock. However, the proposed encoder is, in fact, comparable to [12] when configured with $(M_{in}, M_{out}) = (4,\ 4)$, offering 1000 Mbps of throughput. On the other hand, for high throughput scenarios, the proposed encoder is far better than [16] when we consider that [16] is not fully compliant. The values shown in the table refer to only one frame length and code rate combination. It is also better than [17] for the only metric that is provided (FF utilization) when setting $(M_{in}, M_{out}) = (48,\ 60)$, although it probably has a worse memory footprint. Comparison with [13], which implements a vectorized IRA encoding algorithm, is difficult, as Gomes et. Al. said it does not provide a method for reordering the output parity nor the resources necessary for this operation. Furthermore, the high throughput of 10 Gbps is computed, assuming that the input bits may be provided with parallelism equal to 360. Since the encoder is operated at $f_{ck} = 131\ MHz$, the input throughput must be 47 Gbps, higher than the 10 Gbps at the output.

Finally, the encoder design is also scalable to the DVB-S2X extension. The same encoding algorithm also applies to DVB-S2X frames. To support all 55 DVB-S2X code rate

combinations and frame lengths, more Index-Alfa Tables can be added inside the Core, and more MUXs shall be employed in the reordering module.

REFERENCES

[1] M. Jefferies, K. Maynard, P. Garner, J. Mayock, and P. Deshpande, "26-GHz data downlink and RF beacon for LEO in orbit demonstrator satellite," in *Proc. Int. Workshop Tracking, Telemetry Command Syst. Space Appl. (TTC)*, Sep. 2016, pp. 1–5.
[2] *Current and Future Eumetsat Meteorological Satellite Networks 'Second ITU/WMO Seminar Use of Radio Spectrum for Meteorology: Weather Water and Climate Monitoring and Prediction, Geneva*, World Meteorological Org., Geneva, Switzerland, 2017.
[3] *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and Other Broadband Satellite Applications; Part 2: DVB-S2 Extensions (DVB-S2X) Part 2: DVB*, Standard (ETSI) EN, 302(307), V1, E, 2005, p. 7.
[4] *Specific Attenuation Model for Rain for Use in Prediction Methods*, document Recommendation ITU-R P.838-3, 1992.
[5] *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and Other Broadband Satellite Applications—Part*, Standard (ETSI) EN, 302(307), 2014, pp. 1–307.
[6] *CCSDS Space Link Protocols Over ETSI DVB-S2 Standard, Blue Book*, document C. 131.3-B-2, 2013.
[7] J.-C. Sibel, M. Crussiere, and J.-F. Helard, "Analysis of decoding failures of DVB-S2 LDPC codes," in *Proc. IEEE 80th Veh. Technol. Conf. (VTC-Fall)*, Sep. 2014, pp. 1–6.
[8] J. W. Sun and S. S. Yin, "Design of DVB-S2 LDPC coder and decoder implemented in FPGA," *Appl. Mech. Mater.*, vols. 380–384, pp. 3093–3097, Aug. 2013.
[9] Y. Yuhuang, Z. Wen, and Z. Minmin, *Design and Implementation of DVBS2*. San Jose, CA, USA: LDPC Encoder, 2012.
[10] A. V. Lazarenko, "FPGA design and implementation of DVB-S2/S2X LDPC encoder," in *Proc. IEEE Int. Conf. Electr. Eng. Photon. (EExPolytech)*, Petersburg, Russia, Oct. 2019, pp. 98–102.
[11] T. Yokokawa, M. Nakane, and M. Kan, "A low complexity and programmable encoder architecture of the LDPC codes for DVB-S2," in *Proc. 4th Int. Symp. Turbo Codes Rel. Topics; 6th Int. ITG-Conf. Source Channel Coding*, Apr. 2006, pp. 1–6.
[12] J. Kang, J. An, and B. Wang, "An efficient FEC encoder core for VCM leo," *IEEE Access*, vol. 8, pp. 125692–125701, 2020,
[13] M. Gomes, G. Falcao, A. Sengo, V. Ferreira, V. Silva, and M. Falcao, "High throughput encoder architecture for DVB-S2 LDPC-IRA codes," in *Proc. Internatonal Conf. Microelectron.*, Dec. 2007, pp. 271–274.
[14] *AOS Space Data Link Protocol*, document CCSDS 732.0-b-4, 4, Recommendation for Space Data Syst. Blue Book, 2006.
[15] *TM Space Data Link Protocol. Recommendation for Space Data System Standards, Blue Book*, CCSDS. 132.0-B-3, 2005.
[16] D. Liu, Y. Luo, Y. Li, Z. Wang, Z. Li, Q. Zhang, J. Zhang, and Y. Li, "An LDPC encoder architecture with up to 47.5 gbps throughput for DVB-S2/S2X standards," *IEEE Access*, vol. 10, pp. 19022–19032, 2022.
[17] J. W. Jung and G. Y. Park, "High speed LDPC encoder architecture for digital video broadcasting systems," in *Computer Applications for Database, Education, and Ubiquitous Computing*, T.-H. Kim, J. Ma, W.-C. Fang, Y. Zhang, and A. Cuzzocrea, Eds. Berlin, Germany: Springer, 2012, pp. 233–238.
[18] A. Shokrollahi, "LDPC codes: An introduction," in *Coding, Cryptography and Combinatorics*. Cham, Switzerland: Springer, 2004, pp. 85–110.
[19] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. IT-27, no. 5, pp. 533–547, Sep. 1981.
[20] G. Quintarelli, M. Bertolucci, and P. Nannipieri, "Design and implementation of a DVB-S2 reconfigurable datapath BCH encoder for high data-rate payload data telemetry," *IEEE Access*, vol. 11, pp. 120281–120291, 2023.

PIETRO NANNIPIERI (Member, IEEE) received the Ph.D. degree (cum laude) in information engineering from the University of Pisa, in 2020. He is currently an Assistant Professor with the University of Pisa. In 2019, he was a Visiting Researcher with the TEC-EDP Section in ESTEC (ESA), where he carried out different qualification tests on the SpaceFibre technology. His research interests include digital and VLSI design, electronics for space applications, cryptography, hardware IPs for satellite onboard data handling, signal processing, and hardware cryptography. Indeed, he is also working on the European Processor Initiative (EPI) Project.

GIACOMO BARTOLACCI received the master's degree (cum laude) in electronics engineering from the University of Pisa, in 2023. During his academic career, he pursued the embedded design and mechatronics curriculum. He conducted his master's thesis research in collaboration with IngeniArs.srl and the Department of Information Engineering (DII) in Pisa, focusing on digital signal processing (DSP) and space communications. His research interests include the fields of digital and VLSI design and integration.

MATTEO BERTOLUCCI received the Ph.D. degree in information engineering from the University of Pisa, in 2022. His study focused on implementing digital designs for high-speed satellite communications, such as CCSDS 131.2-B transmitters and receivers on FPGA platforms. He is currently a part of IngeniArs, where he works as a DSP and a Hardware Engineer for space communications. His latest developments focused on the implementation of a giga-symbol-per-second transmitter for CCSDS 131.2-B, which is capable of achieving over 6 Gbit/s of actual telemetry downlink, thus enabling the full and spectrally efficient use of Ka-band.

LUCA FANUCCI (Fellow, IEEE) received the Ph.D. degree in electronic engineering from the University of Pisa, in 1996. From 1992 to 1996, he was a Research Fellow with European Space Agency. From 1996 to 2004, he was a Senior Researcher with the Italian National Research Council, Pisa. He is currently a Full Professor in microelectronics with the University of Pisa. His research interests include design technologies for integrated circuits and electronic systems. He is the coauthor of more than 400 journals and conference papers and he is a co-inventor of more than 40 patents. He served on several technical programme committees for international conferences.

Open Access funding provided by 'Università di Pisa' within the CRUI CARE Agreement