# An Efficient NB-LDPC Decoder Architecture for Space Telecommand Links

Ángel Álvarez, Víctor Fernández and Balázs Matuz, *Member, IEEE*

***Abstract*—In the framework of error correction in space telecommand (TC) links, the Consultative Committee for Space Data Systems (CCSDS) currently recommends short block-length BCH and binary low-density parity-check (LDPC) codes. Other alternatives have been discarded due to their high decoding complexity, such as non-binary LDPC (NB-LDPC) codes. NB-LDPC codes perform better than their binary counterparts over AWGN and jamming channels, being great candidates for space communications. We show the feasibility of NB-LDPC coding for space TC applications by proposing a highly efficient decoding architecture. The proposed decoder is implemented for a (128,64) NB-LDPC code over GF(16) and the design is particularized for a space-certified Virtex-5QV FPGA. The results prove that NB-LDPC coding is an alternative that outperforms the standardized binary LDPC, with a coding gain of 0.7 dB at a reasonable implementation cost. Given that the maximum rate for TC recommended by the CCSDS is 2 Mbps, the proposed architecture achieves a throughput of 2.03 Mbps using only 9615 LUTs and 5637 FFs (no dedicated memories are used). In addition, this architecture is suitable for any regular (2,4) NB-LDPC (128,64) code over GF(16) independently of the H matrix, allowing flexibility in the choice of the code. This work places NB-LDPC codes as the excellent candidates for future versions of the telecommand uplink standard.**

***Index Terms*—VLSI design, FPGA, non-binary LDPC codes, decoder architecture, space communications.**

## I. Introduction

SHORT block-length error correcting codes have been largely discussed and developed during the last years regarding the transmission of telecommand (TC) messages in the uplink channel in space communications. The Consultative Committee for Space Data Systems (CCSDS) recommendation for TC synchronization and channel coding [1] currently proposes an outdated Bose-Chaudhuri-Hocquenghem (BCH) (63,56) code and binary low-density parity-check (LDPC) codes with block lengths (128,64) and (512,256). Short block codes show a larger gap to the respective theoretic limits than their long counterparts. Therefore, more sophisticated decoders/code constructions are usually considered to mitigate this gap. For example, LDPC under most reliable basis (MRB) decoding [2] and non-binary LDPC (NB-LDPC) codes were investigated by CCSDS. However, these were discarded due to hardware complexity limitations, which put a significant constraint on the choice of the channel code.

Manuscript received Month XX, 2020; revised Month XX, 2020. This work has been supported by Project TEC2017-86722-C4-3-R, funded by Spanish MICINN/AEI.

A. Alvarez and V. Fernandez are with the Microelectronics Engineering Group, University of Cantabria, 39005 Santander, Spain (e-mail: {alvarez,victor}@teisa.unican.es).

B. Matuz is with the Institute of Communications and Navigation of the German Aerospace Center (DLR), Münchner Strasse 20,82234 Weßling, Germany (e-mail: balazs.matuz@dlr.de).

NB-LDPC codes achieve higher coding gain, better burst error correction capabilities and lower error floors than their binary counterparts. The above makes them extraordinary candidates for space communications. NB-LDPC codes with excellent performance have been recently proposed by DLR [3] and NASA [4]. However, these coding schemes are not typically used in practice owing to the high cost in terms of hardware resources of the state-of-the-art implementations.

This paper shows the feasibility of NB-LDPC coding for space TC links as an alternative to the standardized binary LDPC codes. We propose an optimized hardware implementation for a (128,64) (parameters in bits) NB-LDPC decoder over GF(16). The designed architecture is suitable for any regular (2,4) parity-check matrix of a GF(16) NB-LPDC code with block length (128,64), giving flexibility to code designers. The design is particularized for a space-grade Virtex-5QV FPGA device. The results show that the implementation achieves a 0.7 dB coding gain over the binary LDPC code in the standard with affordable implementation complexity. The selected algorithm will be detailed in the next sections. Although MRB-decoded binary LDPC schemes achieve even higher coding gain, the implementation cost turns out to be prohibitive for the required throughput and target device, as detailed in the results section. To the best of the authors' knowledge, there are no comparable implementations targeting short block-length NB-LDPC codes for space applications at a reasonable cost.

The rest of this brief is organized as follows. In Section II, the chosen NB-LDPC decoding method is introduced. Section III describes the design procedure and proposed architecture for the necessary modules and the overall NB-LDPC decoder. The implementation results are given in Section IV. Finally, the conclusions are drawn in Section V.

## II. NB-LDPC Decoding Algorithm

NB-LDPC iterative decoding algorithms are based on message passing between variable and check nodes along the edges of a Tanner graph. Simplifications of the original belief propagation (BP), also known as q-ary sum-product algorithm (QSPA) [5], are needed in order to reduce the high complexity of the check node operations. The decoding method of choice for implementation is the Min-Max algorithm [6], which operates in the logarithmic domain. In comparison with other alternatives, such as BP with Fast Fourier Transform (BP-FFT) [7] or extended min-sum (EMS) [8], it has the least decoding complexity with very small loss of performance.

Trellis versions of the algorithms [9]–[11] allow a higher degree of parallelism, targeting throughputs far above the ones used in telecommand links. Space telecommand rates vary from hundreds of kbps to 2 Mbps, which is the upper limit recommended by the CCSDS [12].

Next, the Min-Max algorithm is reviewed. Let $\mathbf{H}$ be the $M \times N$ parity-check matrix defining the code, with elements $h_{m,n}$ from GF($q$). Assume $a$ is a symbol of GF($q$). A q-ary codeword transmitted over a binary-input AWGN channel is denoted by $c = (c_1, c_2, \ldots, c_N)$. $\mathcal{M}(n)$ is the set of check nodes connected to the variable node $n$ and $\mathcal{N}(m)$ is the set of variable nodes connected to the check node $m$. $L_n(a)$ and $L_{post,n}(a)$ denote the a priori and a posteriori information of the variable node $n$ concerning the symbol $a$. The configurations set $\mathscr{L}(m \mid a_n = a)$ is defined as the set of sequences of GF($q$) symbols verifying the equation of the check node $m$: $h_{m,n}a + \sum_{n' \in \mathcal{N}(m)\setminus\{n\}} h_{m,n'}a_{n'} = 0$. Messages exchanged within Min-Max decoding represent log-likelihood ratios (LLRs). The messages from the check node *m* to the variable node *n*, $\mathbf{R}_{m,n}$, and the messages from the variable node *n* to the check node *m*, $\mathbf{Q}_{m,n}$, concerning every symbol of GF($q$) are computed and exchanged in each iteration until the algorithm converges to a codeword or a maximum number of iterations is reached.

Min-Max NB-LDPC Decoding Algorithm:
**Initialization**
*A priori information (initial LLRs):*

$$L_n(a) = \ln\left(P(c_n = s_n \mid \text{channel}) / P(c_n = a \mid \text{channel})\right)$$

where $s_n$ is the most likely symbol for $c_n$.
**Iterations**
*Check node processing:*

$$R_{m,n}(a) = \min_{a_{n'} \in \mathscr{L}(m|a_n=a)} \left\{ \max_{n' \in \mathcal{N}(m)\setminus\{n\}} Q_{m,n'}(a_{n'}) \right\}$$

*Variable node processing:*

$$Q'_{m,n}(a) = L_n(a) + \sum_{m' \in \mathcal{M}(n)\setminus\{m\}} R_{m',n}(a)$$

$$Q_{m,n}(a) = Q'_{m,n}(a) - \min_{a \in \text{GF}(q)} Q'_{m,n}(a)$$

*A posteriori information and hard decision:*

$$L_{post,n}(a) = L_n(a) + \sum_{m \in \mathcal{M}(n)} R_{m,n}(a)$$

The decision on $c_n$ is $\tilde{c}_n = \arg\max_a L_{post,n}(a)$

Since building and storing the set of configurations is complicated in practice, the check node processing will be implemented with a forward-backward (FB) method, which recursively operates as shown in the algorithm below (the i-th connected variable node is denoted by $n_i$ and the check node degree by $d_c$). Although the FB algorithm is a serial process due to data dependency, we will later show that the required throughput for the target application can be reached and exceeded. The complexity of the Min-Max algorithm is dominated by $\mathcal{O}(q^2)$ comparison operations for each check node processing, $q$ being the Galois field cardinality. The architecture in the following section is designed for NB-LDPC codes over GF(16) since they offer a good trade-off between correcting performance and complexity, when compared with codes over other fields [13]. The degrees of the variable and check nodes are $(d_v, d_c) = (2, 4)$, a common choice for short length NB-LDPC codes.

```
Forward-Backward Algorithm:
Initialization:
$F_1(a) = Q_{m,n_1}(h^{-1}_{m,n_1}a)$; $B_{d_c}(a) = Q_{m,n_{d_c}}(h^{-1}_{m,n_{d_c}}a)$;
Forward Process:
for $i = 2$ to $d_c - 1$ do
    $F_i(a) = \min_{a=a_1+h_{m,n_i}a_2} \max\{F_{i-1}(a_1), Q_{m,n_i}(a_2)\}$
end for
Backward Process:
for $i = d_c - 1$ to $2$ do
    $B_i(a) = \min_{a=a_1+h_{m,n_i}a_2} \max\{B_{i+1}(a_1), Q_{m,n_i}(a_2)\}$
end for
Merging Process:
$R_{m,n_1}(a) = B_2(a)$; $R_{m,n_{d_c}}(a) = F_{d_c-1}(a)$
for $i = 2$ to $d_c - 1$ do
    $R_{m,n_i}(a) = \min_{h_{m,n_i}a=a_1+a_2} \max\{F_{i-1}(a_1), B_{i+1}(a_2)\}$
end for
```

## III. Proposed Decoder Architecture

In the following subsections, the design of the most important modules and the top-level architecture is explained. Based on [14], messages are quantized with 5 bits in the iterative algorithm with negligible degradation of performance with respect to a floating point representation.

### A. Check Node Unit (CNU)

The biggest bottleneck of NB-LDPC decoding is the check node processing. In the FB algorithm, the processing is made up of elementary steps (1) which take two input vectors $L_1$ and $L_2$ and compute an output vector $L_o$.

$$L_o(a) = \min_{a=a_1+a_2} \max\{L_1(a_1), L_2(a_2)\} \tag{1}$$

In order to get an efficient architecture for computing the elementary min-max step, the LLRs in message vectors are stored according to their corresponding GF symbol in power representation [15]. By storing the LLRs of $L_1$, $L_2$ and $L_o$ into shift registers inside the min-max module and taking advantage of the power representation, it is possible to implement a fixed interconnection network between the 'min' and 'max' comparators (while an expensive switching network would be needed with LLRs ordered in polynomial representation). Moreover, the multiplications and divisions by the non-zero elements of $H$ can be integrated into the check node processing with no overhead other than a shift of the LLRs in one of the input vectors in the first elementary step of the forward and backward stages. If this technique is not applied, finite field multipliers or barrel shifters are required.

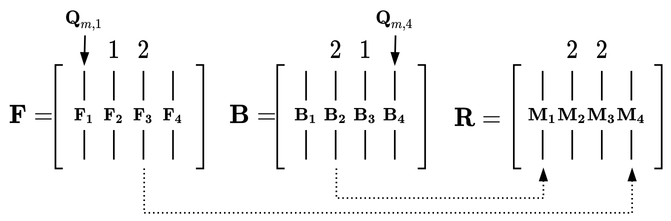
$\mathbf{Q}_{m,1}$
1 2
$\mathbf{F} = [\mathbf{F}_1\ \mathbf{F}_2\ \mathbf{F}_3\ \mathbf{F}_4]$
$\mathbf{Q}_{m,4}$
2 1
$\mathbf{B} = [\mathbf{B}_1\ \mathbf{B}_2\ \mathbf{B}_3\ \mathbf{B}_4]$
2 2
$\mathbf{R} = [\mathbf{M}_1\mathbf{M}_2\mathbf{M}_3\mathbf{M}_4]$

Fig. 1. Column vectors to compute in the forward-backward algorithm for the code of interest ($d_c = 4$) and data dependency between them.

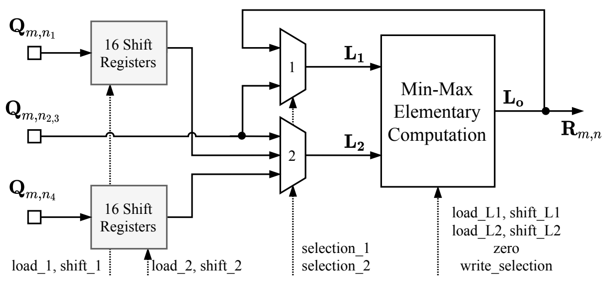

Fig. 2. Proposed low-area architecture for a Check Node Unit (CNU). Signals represent $q$=16 element vectors.

For a regular GF(16) NB-LDPC code with $d_c = 4$, the FB algorithm for the processing of one check node must compute the elementary step six times. Data dependency is shown in Fig. 1. Numbers above the columns denote dependency – column vectors marked with '1' have to be computed before those marked with '2'. Vectors $F_3$ and $B_2$ are directly the first and last output messages, respectively. Every message vector contains $q = 16$ elements (5 bits each).

In order to widen the design space exploration at the top-level decoder architecture, high-throughput and low-area versions of the CNU and VNU are designed. A maximum-throughput check node architecture is implemented using 4 min-max elementary units, exploiting the maximum degree of parallelism available. On the other hand, a low-area architecture is designed with only one min-max elementary unit (Fig. 2). We focus on the low area version, since we will later decide to use it in the proposed decoder as a result of a latency-area analysis. In the FB serial processing, the forward and backward stages are typically computed first and the results are finally combined in the merging process. For this particular $d_c = 4$ case, we propose a different ordering which allows to avoid the storage of the intermediate vectors produced in the first forward and backward steps ($F_2$ and $B_3$). The new scheduling is described in Fig. 3, showing the input ($L_1, L_2$) and output ($L_o$) vectors of the min-max unit in every step. In some steps, the input vectors to $L_1$ and $L_2$ have been interchanged so that, in case one vector is used in consecutive steps, it is needed in the same register array in which it is already stored inside the min-max unit. Then, reading the messages requires less memory accesses. Also, the size of the multiplexors is reduced since a lower number of different message vectors needs to be connected to a single input of the min-max unit. Notice that every signal in the architecture of Fig. 2 represents a $q = 16$ element vector.

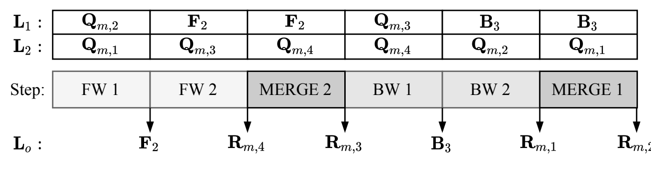
$\mathbf{L}_1$ : $\mathbf{Q}_{m,2}$ $\mathbf{F}_2$ $\mathbf{F}_2$ $\mathbf{Q}_{m,3}$ $\mathbf{B}_3$ $\mathbf{B}_3$
$\mathbf{L}_2$ : $\mathbf{Q}_{m,1}$ $\mathbf{Q}_{m,3}$ $\mathbf{Q}_{m,4}$ $\mathbf{Q}_{m,4}$ $\mathbf{Q}_{m,2}$ $\mathbf{Q}_{m,1}$
Step: FW 1 FW 2 MERGE 2 BW 1 BW 2 MERGE 1
$\mathbf{L}_o$ : $\mathbf{F}_2$ $\mathbf{R}_{m,4}$ $\mathbf{R}_{m,3}$ $\mathbf{B}_3$ $\mathbf{R}_{m,1}$ $\mathbf{R}_{m,2}$

Fig. 3. Proposed serial scheduling for the check node architecture with one min-max elementary unit (low-area architecture).

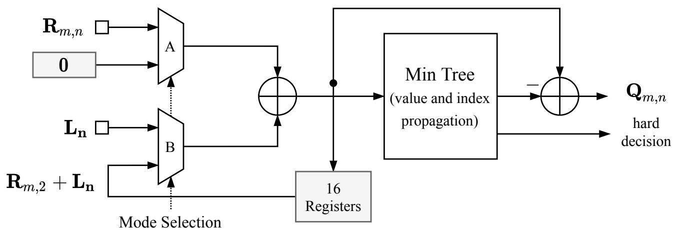

Fig. 4. Proposed low-area architecture for a Variable Node Unit (VNU). Signals represent $q$=16 element vectors.

## B. Variable Node Unit (VNU)

Variable nodes of degree $d_v = 2$ are connected to two check nodes. The message, $\mathbf{Q}$, to be sent to a check node is computed as addition of the message received from the other check node and the a priori information from the channel (element-wise addition of two 16-element vectors). Then, the message vector is normalized so that all the values represent log-likelihood ratios with respect to the most likely symbol. As the lowest LLR value is associated with the highest probability, the minimum value in the vector must be found and subtracted to all elements. Only the a priori information vectors $L_n$ are sent when decoding starts. The architecture proposed in Fig. 4 is capable of computing the above operations in one clock cycle per message. To reuse both the logic (adders and comparators) and computed data, this architecture is also used to get the hard decision symbol associated to the variable node with just an additional clock cycle. By storing the last sum vector in a register array, just one more addition is needed to get the a posteriori information and the most likely symbol (hard decision) is found by reusing the comparators tree. A minimum latency VNU (all operations in a cycle) can be built if the hardware chain is replicated.

## C. Parity Check Module

Within the decoding process, it is checked whether the hard decision verifies all parity-check equations imposed by the check nodes. As $d_c = 4$, there are 4 non-zero elements in a row of $\mathbf{H}$. Then, evaluating the m-th parity-check equation $\sum_{n=1}^{32} h_{m,n}c_n = 0$ reduces to 4 products and 3 additions over GF(16). When a hard decision symbol belonging to the tentative codeword is generated in the VNU, it is stored (e.g. in a shift register) until the 32 symbols are available. All non-zero elements of $\mathbf{H}$ and their position inside the matrix are also stored in a ROM. Multiplications are carried out with look-up tables, with 4 asynchronous ROM implemented as logic in the FPGA LUTs. Additions are implemented in polynomial representation with bitwise XOR operations.

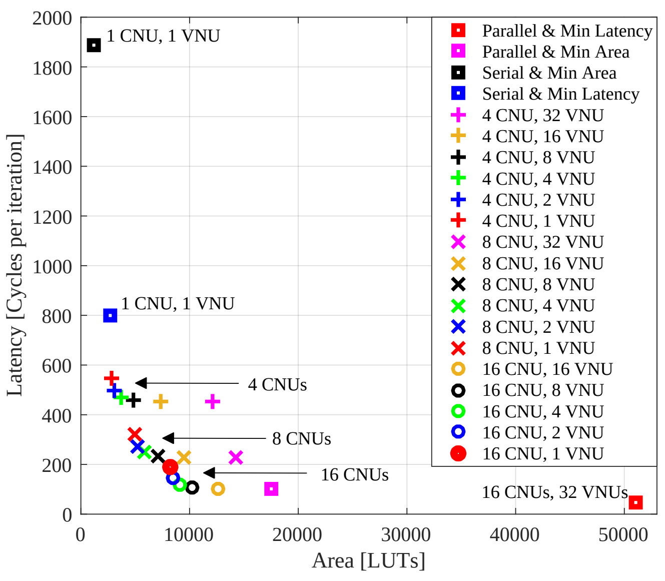

Fig. 5. Area-latency comparison of the candidate architectures for implementation of the (128,64) NB-LDPC decoder over GF(16).

## D. Overall Architecture of the NB-LDPC Decoder

Next, the top-level decoder architecture is discussed and options with different latency-area ratio are compared. Minimum area and minimum latency versions of the CNU and VNU modules have been designed. Also, different degrees of parallelism can be employed. In a (128,64) NB-LDPC code over GF(16), the $\mathbf{H}$ matrix has dimension $16 \times 32$. Fully parallel (16 CNUs and 32 VNUs) and fully serial (1 CNU and 1 VNU) architectures can be built using both minimum area or latency components. They are represented in the latency-area graph in Fig. 5. These extreme options will be discarded for practical applications, except for meeting special requirements. In between, several partial-parallel architectures with a better trade-off arise. The architectures with 8 and 16 CNUs and 1-to-8 VNUs are particularly efficient, as they are closer to the origin in the graph. Partial-parallel architectures represented in Fig. 5 are built with the low area modules since they meet typical telecommand throughputs while using the least hardware resources. Area overhead from memories, interconnections and control is not included in the graph. The needed additional resources have low impact and they affect all options in a similar way.

The architecture with 16 CNUs and 1 VNU, denoted with a red solid circle in Fig. 5, is chosen for implementation. It is the lowest-area choice meeting the 2 Mbps constraint. To calculate the throughput, a maximum of 18 decoding iterations has been selected, with performance shown in Section IV. Architectures with fewer CNUs do not comply with the throughput, and using less decoding iterations quickly leads to a performance degradation. The chosen point has some additional advantages when compared to other architectures. First, architectures with 16 CNUs don't need the large shift registers at the input of the CNUs, because the shift can be implemented hard wired due to having a dedicated CNU for every row of $\mathbf{H}$. Secondly, having just one dedicated VNU means less interconnection overhead between processors and message memories and some costly multiplexors can be avoided. One RAM is employed to store the a priori information and 16 dual port RAMs are used to store the algorithm messages, so that every CNU can read and write data in parallel. Moreover, an advantage of the (16 CNUs,1 VNU) architecture is the fact that it is suitable for any (128,64) NB-LDPC code with $(d_v, d_c) = (2, 4)$ over GF(16) with no memory conflicts and changes in the architecture. As a consequence, performance-oriented modifications or a novel code can be proposed by code designers while using the same architecture. Depending on the position of the non-zero elements in $\mathbf{H}$, other partial-parallel architectures may present issues such as various VNUs having to access the same message memory (when columns being processed in parallel contain non-zero elements in the same row of $\mathbf{H}$). This may result in extra hardware or latency and redesign of the control logic. The schematic of the proposed top-level architecture is shown in Fig. 6.

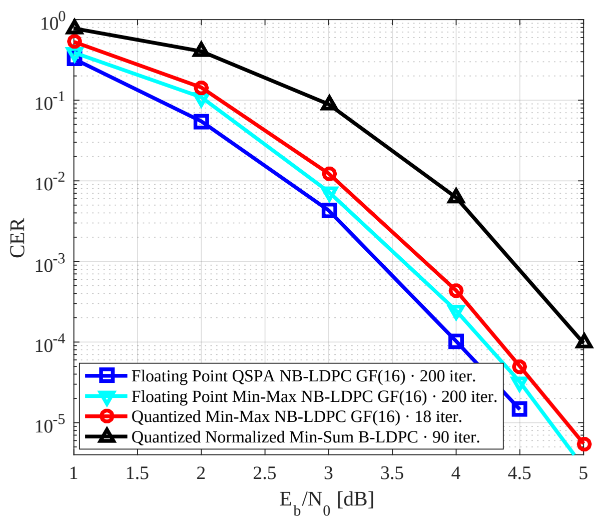

Fig. 7. Performance of proposed Min-Max implementation for the (128,64) NB-LDPC code over GF(16), on AWGN channel and BPSK modulation.

## IV. Implementation Results

The codeword error rate (CER) performance of a NB-LDPC code over GF(16) with parameters (128,64) (in bits) on a binary input AWGN channel is presented in Fig. 7. The NB-LDPC code's parity-check matrix was obtained by a circulant version of the progressive edge growth algorithm. Results in Fig. 7 include: (i) the QSPA and Min-Max floating-point models, (ii) the proposed quantized Min-Max implementation and (iii) the (128,64) CCSDS binary LDPC code under quantized min-sum decoding – with a maximum of 200, 18 and 90 decoding iterations, respectively. The iterations limit in quantized versions is chosen to meet the required throughput for the maximum (not average) number of iterations, with a small performance loss. Remarkably, the non-binary code shows a coding gain of 0.7 dB with respect to its binary competitor from the CCSDS standard.

In Table I, implementations of different decoding alternatives for the CCSDS telecommand recommendation are compared. In the first column, the proposed NB-LDPC decoder is

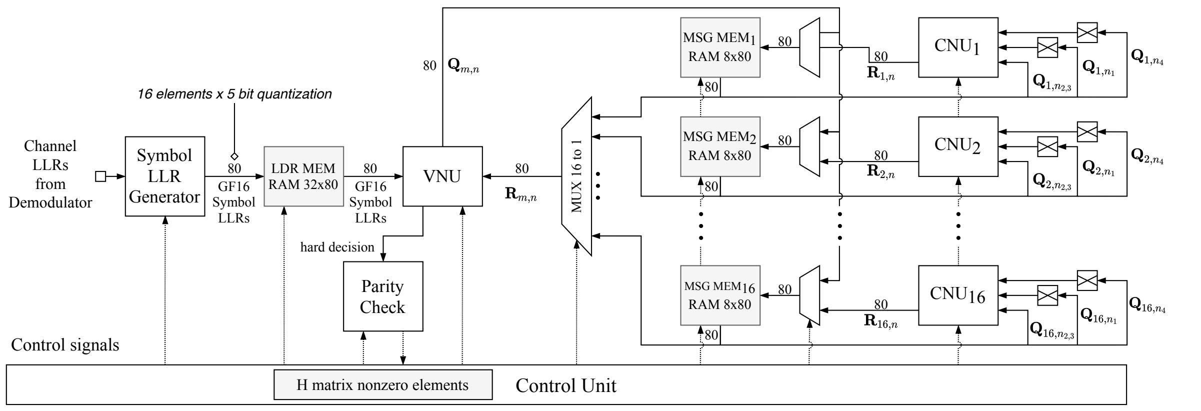

Fig. 6. Top-level proposed NB-LDPC decoder architecture. Signals represent $q$=16 element vectors.

presented. Our implementation uses 9615 Slice LUTs (11.7% out of the total) and 5637 FFs (6.9%) on a space-grade Virtex-5QV FPGA. The achieved throughput is 2.03 Mbps, exceeding the limit of 2 Mbps currently recommended for CCSDS agencies in high rate TC transmissions [12]. Using dedicated BRAMs would require data redundancy (e.g. triple vote) as a protection mechanism in space. Instead, memories have been implemented as distributed memory in the FPGA logic, which is protected against radiation. Therefore, memories are included in the above LUT utilization. In the second column, the implementation of the binary LDPC code which is currently in the standard is summarized. The design has been area optimized under normalized min-sum decoding and the same constraint of 2 Mbps. In the third column we consider binary LDPC with MRB decoding, motivated by its great error correcting capabilities. Assuming a throughput constraint of 1 Mbps and a 100 MHz clock, the authors in [2] indicate the number of processing units to be used. The amount of required LUTs and FFs is far bigger than the number available in the target FPGA, proving the MRB proposal to be only feasible for much slower applications (e.g. deep space communications).

## V. Conclusion

This work presents an efficient architecture which proves the feasibility of NB-LDPC coding for error correction in space telecommand communications. This scheme outperforms the standardized LDPC code over AWGN and jamming channels. Despite NB-LDPC codes not being included in the CCSDS recommendation for TC due to their high decoding complexity, the proposed decoder exceeds the maximum throughput of 2 Mbps recommended by the CCSDS for TC links while using only 9615 LUTs and 5637 FFs (including all memories) on a Virtex-5QV FPGA. To the best of our knowledge, this is the first implementation for short block-length NB-LDPC codes targeting space telecommand links at a reasonable cost. The results in this work show that NB-LDPC codes are a suitable alternative for the upcoming version of the CCSDS standard.

TABLE I
Comparison Between Different Decoding Implementations

| Class | **NB-LDPC** | Binary LDPC | Binary LDPC |
|---|---|---|---|
| Code | (128,64) GF(16) | (128,64) (CCSDS) | (128,64) (CCSDS) |
| Algorithm | Min-Max | Normalized Min-Sum | MRB4 |
| Device | Virtex-5QV | Virtex-5QV | Virtex-5QV |
| Slice LUTs | 9615 | 2446 | >available |
| Slice Registers | 5637 | 560 | >available |
| BRAM | 0 | 0 | 0 |
| Max. Iterations | 18 | 90 | NA |
| Max. $f_{\text{CLK}}$ | 60.9 MHz | 51.0 MHz | 100 MHz |
| Throughput | 2.03 Mbps | 2.01 Mbps | 1 Mbps |

## References

[1] CCSDS 231.0-B-3. "TC Synchronization and Channel Coding". Blue Book, Sep. 2017.
[2] M. Baldi et al., "On the use of ordered statistics decoders for low-density parity-check codes in space telecommand links", *Journal Wireless Communications and Networking 2016, Art. 272*, Nov. 2016.
[3] G. Liva et al., "Codes on high-order fields for the CCSDS next generation uplink", *Advanced Sat. Mult. Systems Conf. (ASMS) and Signal Proc. Space Comm. Workshop (SPSC)*, Baiona, 2012, pp. 44-48.
[4] B. Chang, D. Divsalar and L. Dolecek, "Non-binary protograph-based LDPC codes for short block-lengths", *2012 IEEE Information Theory Workshop*, Lausanne, Switzerland, 2012, pp. 282-286.
[5] M. C. Davey and D. MacKay, "Low-density parity check codes over GF(q)", *IEEE Commun. Letters*, vol. 2, no. 6, pp. 165-167, Jun. 1998.
[6] V. Savin, "Min-Max decoding for non binary LDPC codes", *Proc. IEEE Intl. Symp. on Info. Theory*, Toronto, Canada, Jul. 2008.
[7] L. Barnault and D. Declercq, "Fast decoding algorithm for LDPC over GF($2^q$)", *Proc. IEEE Inf. Theory Workshop*, Paris, 2003, pp. 70-73.
[8] D. Declercq and M. Fossorier, "Decoding Algorithms for Nonbinary LDPC Codes Over GF($q$)", *IEEE Transactions on Communications*, vol. 55, no. 4, pp. 633-643, Apr. 2007.
[9] E. Li, K. Gunnam and D. Declercq, "Trellis based Extended Min-Sum for decoding nonbinary LDPC codes", *ISWCS*, Aachen, 2011, pp. 46-50.
[10] Y. Ueng et al., "A High-Throughput Trellis-Based Layered Decoding Architecture for Non-Binary LDPC Codes Using Max-Log-QSPA" *IEEE Transactions on Signal Processing*, vol. 61, no. 11, pp. 2940-2951, 2013.
[11] M. Li et al., "An Efficient High-Rate Non-Binary LDPC Decoder Architecture With Early Termination", *IEEE Access*, vol. 7, 2019.
[12] CCSDS 401.0-B-28. "Radio Frequency and Modulation Systems – Part 1". Blue Book, Feb. 2018.
[13] D. Divsalar and L. Dolecek, "Non-binary Protograph LDPC Codes for Short Blocklengths", *CCSDS Fall Meeting*, Oct. 2012.
[14] X. Zhang and F. Cai, "Efficient Partial-Parallel Decoder Architecture for Quasi-Cyclic Nonbinary LDPC Codes", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 2, pp. 402-414, Feb. 2011.
[15] F. Cai and X. Zhang, "Efficient Check Node Processing Architectures for Non-binary LDPC Decoding Using Power Representation", *2012 IEEE Workshop on Signal Processing Systems*, pp. 137-142, Oct. 2012.