Non-fractional parallelism in LDPC Decoder implementations

John Dielissen, Andries Hekstra

NXP Semiconductors, High Tech Campus 31, 5656 AE Eindhoven, The Netherlands
E-mail: john.dielissen@nxp.com

Abstract

Because of its excellent bit-error-rate performance, the Low-Density Parity-Check (LDPC) decoding algorithm is gaining increased attention in communication standards and literature. Also the new Chinese Digital Video Broadcast standard (CDVB-T) uses LDPC codes. This standard uses a large prime number as the parallelism factor, leading to high area cost. In this paper we present a new method to allow fractional dividers to be used. The method depends on the property that consecutive sub-circulants have one memory row in common. Several techniques are shown for assuring this property, or solving memory conflicts, making the method more generally applicable. In fact, the proposed technique is a first step towards a general purpose LDPC processor. For the CDVB-T decoder implementation the method leads to a factor 3 improvement in area.

1. Introduction

Since the rediscovery of Low-Density Parity-Check (LDPC) codes [8] in 1996, many publications of their implementations have appeared. LDPC codes are included in the DVB-S2 standard, which is a digital satellite video broadcast standard. For this standard, implementations have been suggested in [4, 5, 9]. Also for the Chinese Digital Video Broadcast standard (CDVB-T) LDPC codes are proposed. Because of the excellent bit-error-rate (BER) performance of the decoding algorithm, LDPC-codes are expected to be part of many future standards as well.

In the DVB-S2 standard, the interconnection between computation kernels is grouped in circulant blocks of 360 codewords. Fully exploiting this circulant parallelism results in an implementation with 360 computation kernels. Such an implementation can achieve a throughput rate of more than 800 Mbps while only 90 Mbps is required. In [4] an architecture is proposed in which the parallelism factor is reduced, resulting in an area reduction of more than a factor 2. However the choice of parallelism factors is limited to dividers of the circulant size. This limitation reduces the freedom of choosing a parallelism factor, leading to higher implementation cost. Moreover, the technique cannot be applied for standards in which the circulant size is chosen to be prime number. In this paper we propose a new method in which the parallelism factor does not have to be a divider of the circulant size. As an example we show the new method for the CDVB-T standard, in wich the circulant size is chosen to be a mersenne prime viz 127.

In this paper, we first discuss the state-of-the-art in LDPC decoding implementations. In Section 3, we briefly explain the LDPC algorithm. We explain the mixture of known technologies that are used in our new LDPC-decoder. The new LDPC architecture is based upon an state-of-the art architecture [4] which is explained briefly in Section 4. Although the technology explained in this paper can be applied to a wider range of applications, the results and the benchmarks are presented for the CDVB-T standard, which is explained in Section 5. Section 6 shows that the state-of-the art architecture leads to an area of 3 mm^2, and that this can be improved by a factor 3 if CDVB-T had dividers. All area numbers are based upon 90 nm technology and for logic only the synthesis results are reported. In section 7, the method to allow fractional dividers to be used is explained. In this section we also present the resulting improvements in area for the different achievable parallelism factors. We end this paper with conclusions and recommendations for future work.

2. State-of-the-art

One of the most common measures to compare LDPC decoders, is the level of parallelism. This ranges from fully parallel [1, 6], to (sub)circulant-level parallel [4, 5, 9], grouped sequentially [3], and fully sequential. In a fully parallel implementation, all symbol and check-node calculations (see Section 3), are directly realised in hardware. All

978-3-9810801-2-4/DATE07 © 2007 EDAA

units are interconnected via many wires, leading to congestion in the layout. A fully parallel implementation of the CDVB-T decoder is impractical. For CDVB-T, even at 10 MHz, this solution yields a throughput of 1Gbps, performing 10^{12} operations/second. In a group-level parallel implementation, the code needs the property that interconnect between the symbol- and check-node calculations is structured. Two published implementations [5, 9] of a LDPC-DVB-S2 decoder are examples of this group-level parallelism. A decoder using sub-group-level parallelism is descibed in [4]. This architecture divides the group of parity equations into smaller sub-groups, processed sequentially, allowing lower parallelism factors. The decoder architecture described in this paper falls into this category.

In a grouped sequential solution [3], the algorithm loops sequentially through all symbol-nodes, executing the connected check-node calculations simultaneously. The grouped sequential solution requires as many memory accesses as there are check-nodes connected to the symbol-node. Since in CDVB-T the number of connected check-nodes varies from 3 to 16, this is not an efficient solution. Moreover, it would lead to a throughput of 10 Mbps (at 300 MHz) which is too low for the CDVB-T application. Fully serial solutions, traversing symbol-nodes and check-nodes consecutively, would result in an even lower throughput, hence this solution is also out of scope for CDVB-T.

3. LDPC decoding algorithm

In this paper we use the *min-sum* LDPC decoding algorithm, which is described in [2, 7]. This algorithm achieves a performance level very close to, or sometimes even out performing that of belief propagation (BP) decoding, while offering significant hardware advantages. Note however that the architectural solution can also be applied with other decoding algorithms.

Let N be the codeword length and M be the number of parity check equations. The parity check matrix H consists of M rows and N columns with elements "0" or "1". The rows in the matrix are the parity check equations, and the set of elements which have a '1' in a row are the arguments of the equation. For a parity check equation with index m, $0 \leq m < M$, define the set $N(m)$ of codeword symbol positions that it checks,

$$N(m) = \{n | n = 0, 1, \ldots, N-1; H_{mn} \neq 0\}.$$

The number of elements in $N(m)$ is refered to as K_m. Similarly, for a codeword symbol position n, $0 \leq n < N$, define the set $M(n)$ of indices of parity check equations that check the symbol position n,

$$M(n) = \{m | m = 0, 1, \ldots, M-1; H_{mn} \neq 0\}.$$

The number of elements in $M(n)$ is refered to as J_n. The parity check matrix can be associated with a bipartite graph (V, E) called the Tanner graph, shown in Figure 1. The set of vertices (V) is the union of the set of N symbol-nodes and the set of M parity check-nodes. The set of edges (E) consisting of all edges (m, n) for which $H_{mn} = 1$. Classical iterations of the LDPC algorithm consist of information send from symbol-nodes (N) via the edges (E) to the check-nodes (M), and back.

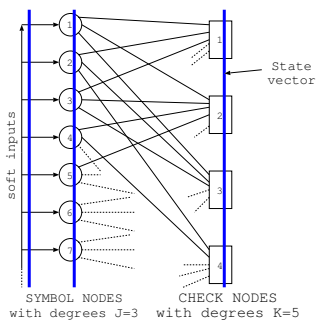

Figure 1. Tanner graph of LDPC code

For a given iteration of the *min-sum* algorithm, we define the following variables:

- L_n - The x bit, signed input message into symbol-node n.

$$L_n = \frac{2y_n}{\sigma^2} \tag{1}$$

  y_n being the received BPSK symbol value, and σ^2 being the noise variance.

- λ^i_{nm} - The message sent from symbol-node n to check-node m in the i^{th} iteration.

$$\lambda^i_{nm} = L_n + \sum_{m' \in M(n) \backslash m} \Lambda^{i-1}_{m'n} \tag{2}$$

- Λ^i_{mn} - The message sent from to check-node m to symbol-node n in the i^{th} iteration.

$$\begin{aligned} \Lambda^0_{mn} &= 0, \\ \Lambda^i_{mn} &= \underset{n' \in N(m) \backslash n}{XOR} \left\{ sign(\lambda^i_{n'm}) \right\} \\ &\quad * \underset{n' \in N(m) \backslash n}{MIN} \left\{ |\lambda^i_{n'm}| \right\} \end{aligned} \tag{3}$$

XOR is defined as sign equivalent of the boolean xor function, e.i. $XOR(-,-) = +$

- λ_n - The decoder output messages. Unlike the λ_{nm}'s, the decoder output message λ_n uses all information available in a symbol-node n, and is only necessary in the last iteration I.

$$\lambda_n = L_n + \sum_{m \in M(n)} \Lambda_{mn}^I \tag{4}$$

Additionally, we change this algorithm to include Gauss-Seidel iterations, a technique also known as "staggered decoding" [10], "turbo decoding LDPC", "shuffled decoding", and "layered decoding". For this we use the check-node centric processing of LDPC. A variable λ_n^i is used, which consists of the sum of L_n and the most up to date messages between check-nodes and symbol nodes:

$$\lambda_n^i(m) = L_n + \sum_{\substack{m' \in \\ U(n,m)}} \Lambda_{m'n}^i + \sum_{\substack{m' \in \\ R(n,m)}} \Lambda_{m'n}^{i-1} \tag{5}$$

In this equation, the set $U(n,m) \subset M(n)$, relates to the messages which have already been updated in the current iteration i before processing check-node m, and $R(n,m) = M(n) \backslash U(n,m)$ e.g. the remaining set. For Jacobi iteration $U(n,m) = \emptyset$, and for Gauss-Seidel $U(n,m)$ is defined as:

$$\emptyset \quad = \quad U(n,m_1) \subset U(n,m_2) \subset \ldots \subset U(n,m_{J_n}) \tag{6}$$

For calculating λ_{nm}^i we use $\lambda_{nm}^i = \lambda_n^i(m) - \Lambda_{mn}^{i-1}$, and after calculating Λ_{mn}^i in the current check node m, λ_n^i is updated as:

$$\lambda_n^i(m_{x+1}) = \lambda_{nm_x} + \Lambda_{m_x n}^i \tag{7}$$

4. State-of-the-art (sub)group LDPC decoding architecture

The kernel of the architecture in [4] is formed by the data path, shown in Figure 2. As explained in Section 3, the K_m λ_{nm}^{i-1}'s are formed by subtracting the Λ_{mn} from the sequentially arriving $\lambda_n^i(m)$'s. The λ_{nm}'s are used to calculate Equation (3) by means of running minimum, one-but minimum, index, and xor calculations. As a result all Λ_{mn}^i's are compressed into a vector Λ_m^i containing the mentioned elements. Simultaneously λ_{nm}'s are stored in a FIFO for later use. During the next K_m cycles this is repeated for the next parity check equation, while for the current parity check equation, the individual Λ_{mn}^i's are calculated by decompressing the vector Λ_m^i. These Λ_{mn}^i are added to the λ_{nm} from the FIFO, which results in the new $\lambda_n^i(m)$ (Eq. 7). The data path thus produces the K_m λ_n's of one parity equation, while receiving the K_m λ_n's for the next equation. This data-path can handle one connection between a check-node and a symbol-node per clock cycle. To avoid a read of λ_{n_1} before a write of λ_{n_2} for $n_1 = n_2$ the order of the parity equations must be statically scheduled. When not achieving this no-operations must be inserted, leading to lower throughputs.

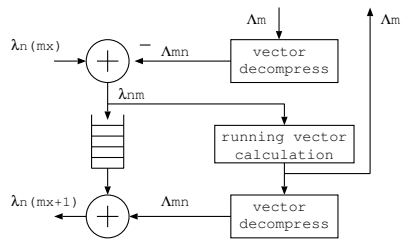

Figure 2. data path of architecture

Most LDPC codes are composed of circulant structures such that parity equations can be calculated simultaneously. When complying to the parallelism factor of the LDPC-code, the decoder can compute multiple parity check equations without memory conflicts[1]. The top-level architecture, which includes D data paths is shown in Figure 3.

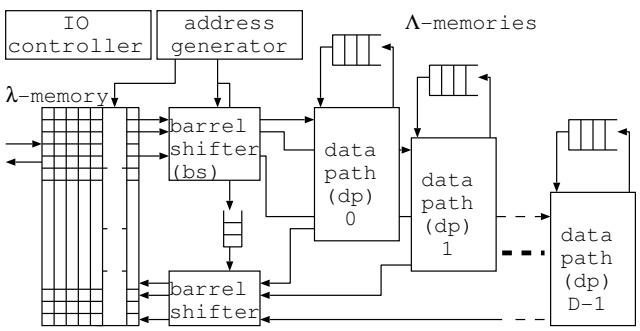

Figure 3. top-level architecture

The (memory) efficiency of the proposed architecture is achieved by using the property that the set of data to/from the data paths always exists in one word in the memory. When applying the technique described in [4], the circulants are split into multiple smaller sub-circulants, for which the diagonal property prevails.

5. LDPC code for CDVB-T

The specification of a LDPC code can in general be done by presenting its H-matrix. The rows in the matrix are the parity check equations, and the set of elements which have

1 multiple diagonals per circulant are excluded

a '1' in a row are the arguments in that equation. If e.g. there is a '1' in the second column of a row, and a '1' in the column 365, the symbols '2', and '365' participate in one equation. The CDVB-T code is specified for 3 different rates (0.4, 0.6 and 0.8) by means of 3 different matrices. The structure of the CDVB-T H-matrices, can be observed from Figure 4. Note that the code consists of circulants of size $z = 127$. The height of the H-matrix, and thus the number of parity equation groups equals 35, 23, and 11 respectively. The codeword length is in all 3 cases $59 * 127 = 7493$ bits. The diagonals should be seen as the line of '1's in the matrix, and contrary to DVB-S2, no multiple diagonals per circulant occur. The position where the diagonal starts is refered to as the angle of the circulant.

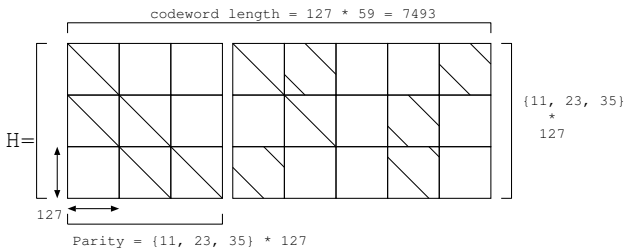

Figure 4. H-matrix of CDVB-T LDPC code

The distribution of the check-node degree differs for each rate, and for equations within a rate two values for the check-node degree are used. Note that check-node degree is also the number of circulants participating in a equation group. The distribution of the symbol-node degree differs for each rate, and for symbols within a rate values between 3 and 16 are used.

The architecture presented in this paper can handle all codes where the parity matrix has the single diagonal quasi cyclic structure shown in Figure 4.

6. Opportunities of fractional divider solution

The state of the art LDPC decoding architecture has been discussed in section 4 and in [4]. This architecture only allows for parallelism factors which are a divider of the circulant size z. For CDVB-T this leads to only one solution, namely with $D = z = 127$ data paths. The λ-memory height of this instantiation is only 59 words, which is low for efficient SRAM memory design. Such a memory will not exceed an efficiency rate of 23%, leading to 0.3 mm^2 in a 90 nm technology. The total area cost for $D = 127$ equals 3 mm^2 (including overhead).

The instantiated decoder with $D = 127$ data paths has a throughput which is too high. The ratio between the maximum throughput and the required throughput can be calculated by $\frac{F*C}{T*I*g*K_m}$. In this equation F is the clock frequency (300MHz), T is the throughput (45 Mbps), I is the number of iterations (25), C is the codeword length (7493 bits), and g is the number of equation groups. For CDVB-T the most critical rate is 0.6 with $g = 23$ groups and $K_m \leq 13$, resulting in overdimensioning ratio of 6.6. In order to reduce control overhead, equations are calculated group-wise. Figure 5 shows how this factor 6.6 relates to the calculation time per group. A decoder with $D = 127$ data paths could consume the data in K_m clock cycles, produce data in K_m clock cycles, and being idle for $\frac{5.6}{6.6}$ of its time.

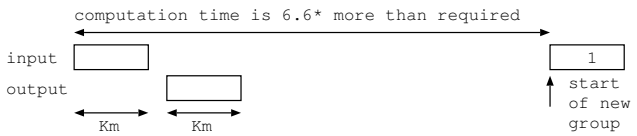

Figure 5. Timing diagram for processing one group of equations

In order to calculate the possible area gain, we assume no problems with the fractional divider, and choose to process a equation group in $r = 6$ sub-groups, resulting in $D = \lceil \frac{127}{6} \rceil = 22$ data paths. Note that the 127 symbol-nodes of a circulant are then divided over r memory rows, each containing D symbol-nodes. The area cost of such a decoder equals 0.9 mm^2 in 90 nm technology. This shows that potentially a factor 3 can be gained by using sub-group parallelism.

7. Method to allow fractional divider solution

When choosing D such that $D * r > z$, there are always rows containing less symbol-node variables. As a tutorial-example take a circulant size of $z = 7$, and a parallelism factor of $D = 3$. This leads to $r = 3$ sub-circulants. No matter how it is organized, it is not possible to find a memory mapping such that, independent of the 7 rotation angles, the required [3;2;2] variables are each in one row of the memory. At best, these exist in two rows. When using the given circulant and grouping the equations linearly e.i. first the (row)equations $(0 \ldots D-1)$, then $(D \ldots 2D-1)$ etc. a solution is found. For this solution, the memory is organised as rows containing D or $D-1$ symbols, organised linearly, as is shown in Figure 6. In fact this memory organisation occurs automatically after writing the results of an equation set. Note that the property only holds if each equation set contains either D or $D-1$ equations.

The method we propose here uses the property that consecutive (modulo $\lceil \frac{z}{D} \rceil$) rows are read during processing. More restrictive, we require that when one sub-circulant uses rows $x-1$ and x, the next sub-circulant uses data from rows x and $x+1$. The usage of row x in both sub-circulants leads to advantages in memory access. When ac-

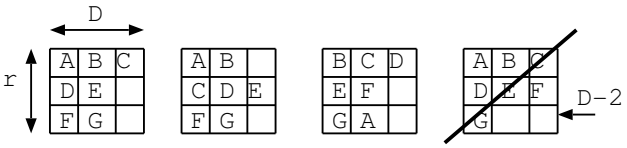

Figure 6. Possible and impossible memory allocations for $z = 7$, $D = 3$, and $r = 3$

cessing rows $x - 1$ and x for one sub-circulant, row x is placed into a FIFO. When processing the next sub-circulant after K_m cycles, row x is to be retrieved from the FIFO, and only row $x + 1$ must be retrieved from the λ memory. The double access during the first sub-circulant is solved by filling the FIFO during an initial phase. The proposed architecture, shown in Figure 7, has a throughput efficiency of $\frac{r}{r+1}$. In this architecture only data routing is different.

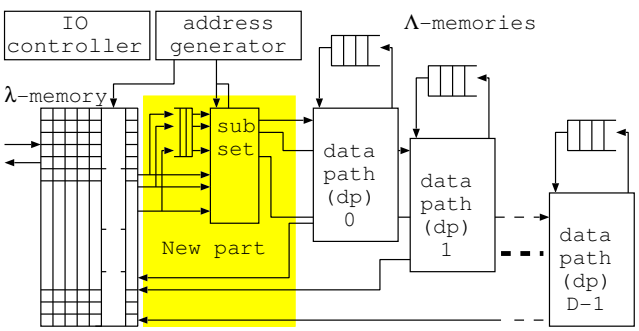

Figure 7. top-level architecture with buffering

Note that the word is written back in a different order compared to the read order. To illustrate this, take the left most memory organisation in Figure 6. When processing this circulant starting with symbols CDE, followed by FG and AB, the result after writting back is represented by the second memory organisation in Figure 6. When initiating the circulant memory allocation similarly as the last access in one iteration leaves it, the subset selections are the same for all iterations. The starting address might be different for iterations and needs to be stored. Note that the second "barrelshifter/sub-set selection" can be removed in those cases, where it gives an advantage in area.

As stated before we require that when one sub-circulant uses rows $x-1$ and x, the second sub-circulant uses rows x and $x+1$. The left part of Figure 8 shows a tutorial-example which results in a violation of the requirement. When using HIJK (from row 1 and 2) in one sub-circulant, and LMNA (from row 3 and 0) in the following, 4 instead of 3 rows are required.

Figure 8. Memory allocations leading to 4-row-conflicts

In order to find the number of 4-row-conflicts and the circulant rotation angle under which it occurs, all situations are simulated. Under the constraint that all equations using D columns are executed first, followed by the equations with $D-1$ columns the number of conflicting angles equals $(m-1)(r-m-1)$, where $m = z \bmod r$. This number is independent of how the memory is (linearly) ordered. Where the left example in Figure 8 gives a conflict for the angle at 8, organizing the memory as shown in the right example in the figure gives a conflict for angle at 9. If multiple conflicts occur, the number of conflicts and the relation between the conflicting angles remains the same. Simulations further showed that multiple conflicting angles per allocation can occur if z is very small.

When opting for no conflicts, e.g. by means of $m = r-1$ which is equal to $r * D = z + 1$ this limits the choice of D significantly. For the CDVB-T case, the options for D are 32 and 64. The area of the $D = 32$ solution equals 1.12 mm^2. The decomposition of the area is shown in Table 1 and includes an additional 0.05 mm^2 for the FIFO. Note that the choice of 32 data paths results in having $5 * K_m$ cycles per equation group, whereas we calculated a budget of $6.6 * K_m$ cycles per group before. This gives space to finish the processing of one group before continuing with the next, as is shown in Figure 9.

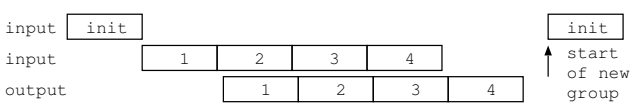

Figure 9. Timing diagram with initialisation for $D = 32$

The 4-row-conflict can be solved by slightly modifying the control of the architecture in the case of conflict: when retrieving row x from the memory, the next clockcycle row $x + 1$ is retrieved and this one is pushed into the FIFO instead of x. This solution leads to a no-operation in the data paths, leading to lower throughputs. The occurence of the 4-row-conflict depents on the memory organisation at the moment of reading and the rotation angle of the circulant. Both

can be influenced by rotating equations groups and scheduling the order of equations and equation groups.

Since the 4-row-conflict can be fully circumvented in CDVB-T by means of compile time scheduling, also the $D = 26$ case can be used. This choice leads to an area of 1.03 mm^2, including the 0.04 mm^2 for the FIFO. e.g. a gain of 0.1 mm^2 compared to $D = 32$, and 0.1 mm^2 away from the target presented earlier.

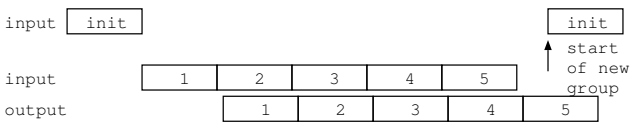

Figure 10. Timing diagram for $D = 26$

The timing of the $D = 26$ solution is shown in Figure 10. This figure also shows that the initialisation of the next group starts while the first group is still busy. When the two groups use the same variables, this leads to conflicts. Analysis of the CDVB-T case showed that the overlap between groups after scheduling for minimum overlap remains 15 of the 27 participants(for rate 0.8). Since each group is divided into 5 sub-groups, the probability that the initial phase reads the word that must still be written by the output of the last sub-group equals $\frac{1}{5}$. For the $D = 26$ case in Figure 10 this implies that the output part "5" and the second "init" part have -on average- 3 rows in common. Carefull scheduling and adjusting the group angles might reduce this. Scheduling the common rows early in the output part "5" and late in the "init" part, combined with the additional space of $0.6 * 27 = 16$ cycles[2] of empty space gives more than enough room for avoiding the conflicts at all.

D	λ mm^2	Λ mm^2	dp mm^2	bs/ subset mm^2	Total incl overhead mm^2
127	.30	.30	1.0	1.2	3.00
32	.36	.20	.26	.05	1.12
26	.34	.20	.21	.04	1.03
22	.30	.20	.17	.03	0.90

Table 1. Estimated area cost of LDPC decoder for various parallelism factors.

Table 1 summarizes the area cost in 90 nm technology of all options discussed in this paper. The separated elements can be found in the architectures in Figure 3 and 7. Note that the $D = 22$ is only a target area, not presenting a valid design point. The resulted areas show that proposed fractional divider solution leads (for CDVB-T) to an area improvement of a factor 3.

2 Note that 2 additional cycles are necessary due to pipelining, and additional cycles could be necessary for angle conflict solving

8. Conclusions

In this paper we showed a method to divide the big groups of equations of the CDVB-T standards into smaller sub-groups. The method depends on the relation that consecutive sub-circulants have one memory-row in common. By means of an initialisation phase the first row is retrieved resulting in a situation where sub-circulants only need to retrieve one additional row from the memory. Unfortunately there are violations of this property and we showed several ways to solve them, making the method more generally applicable.

This method improves the area by a factor 3 compared to a straighforward method. It is the aim of future research to incorporate this method in a multistandard LDPC decoder that will be able to process all standards. The proposed methode can be applied to find lower parallelism factors both for standards with prime numbered parallelism and for standards with non-prime parallelism factors.

References

[1] A.J. Blanksby and C.J. Howland. A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder. In *IEEE Journal of Solid-State Circuits*, volume 37, pages 404–412, 3 2002.

[2] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X Hu. Reduced-complexity decoding of LDPC codes. In *IEEE Transactions on Communications*, volume 53, pages 1288–1299, 2005.

[3] M. Cocco, J. Dielissen, M. Heijligers, A. Hekstra, and J. Huisken. A scalable architecture for LDPC decoding. In *IEEE Proceedings of DATE*, pages 88–95, 2004.

[4] J. Dielissen et.al. Low cost LDPC decoder for DVB-S2. In *IEEE Proceedings of DATE*, 2006.

[5] P. Urard et.al. A 135Mb/s DVB-S2 compliant codec based on 64800b LDPC and BCH codes. In *IEEE Proceedings of ISSCC*, 2005.

[6] L. Fanucci, C. Pasquale, and C. Giulio. VLSI design of a full-parallel high-throughput decoder for turbo gallager codes. In *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, volume E89, pages 1976–1986, 2006.

[7] M. Fossorier and Jinghu Chen. Near optimum universal belief propagation based decoding of low-density parity check codes. In *IEEE Transactions on Communications*, volume 50, pages 406–414, 2002.

[8] R.G. Gallager. Low density parity check codes. In *IRE Transations on Information Theory*, volume 8, pages 21–28, 1962.

[9] F. Kienle, T. Brack, and N. Wehn. A synthesizable IP core for DVB-S2 LDPC code decoding. In *IEEE Proceedings of DATE*, 2005.

[10] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam. High throughput low-density parity-check decoder architectures. In *IEEE proceedings of GLOBECOM*, volume 5, pages 3019–3024, 2001.