Interconnection Framework for High–Throughput, Flexible LDPC Decoders

Federico Quaglio, Fabrizio Vacca, Cristiano Castellano, Alberto Tarable, Guido Masera
CERCOM – Dipartimento di Elettronica
Politecnico di Torino – Italy

Abstract

This paper presents a possible interconnection structure suitable for being used in a flexible LDPC decoder. The main feature of the proposed approach is the possibility of implementing parallel or semi–parallel decoders with a reduced communication complexity. To the best of our knowledge this is the first work detailing the implementation of a fully flexible LDPC decoder, able to support any type of code. To prove the effectiveness of this approach, a complete decoder has been implemented on a XC2V8000, achieving a decoding throughput of 529 Mbps on a (1920,640) code.

1 Introduction

Low Density Parity Check (LDPC) codes, firstly introduced for the first time by R. Gallager [1] in 1960, are a class of powerful, linear block codes able to perform very close to Shannon AWGN channel bound [2–4]. Beside their impressive error–correcting performances, LDPC codes show an intrinsic parallel structure suited to reach remarkable decoding throughputs [5, 6]. These two peculiarities have significantly contributed to make LDPC codes among one of the most promising candidates for next generation communication standards.

LDPC decoding process is usually represented through a bipartite graph, also called *Tanner Graph*, where two class of Processing Elements (PEs) are mapped onto two classes of graph's vertices, (Fig. 1). These processing elements are usually referred as *Variable Nodes* (VNs) and *Check Nodes* (CNs) and are strictly related to the $N \times M$ *Parity–Check Matrix* $\mathbf{H}$ associated with the code. VNs are related to the N columns of $\mathbf{H}$ matrix and represent the "code-block" of the specified LDPC code. The M CNs (which correspond with the rows of $\mathbf{H}$) represent the M parity-check equation of the code. Given two distinct processing elements VN_j and CN_i, they are interconnected through an edge on the Tanner Graph if and only if $\mathbf{H}(i, j) = 1$.

The decoding process is applied iteratively sending messages from VNs to CNs (and vice-versa) updating the received soft-information at each PE. To avoid probability multiplications, most decoders work in logarithmic domain employing "Log-Likelihood Ratio (LLR)" λ rather than the received bit, leading to the "Sum-Product" implementation of the more general "Belief Propagation" algorithm. According to the extrinsic information principle (i.e.a message is computed excluding any *a priori* information on the same message) [2, 6], VNs update the LLR values with the information received from the CNs. This process is expressed in equation (1) where k represents the current iteration, Q_{ji} is the message being sent from VN_j to CN_i, $R_{\alpha j}$ is an incoming message produced by CN_α and directed to VN_j (excluding the message R_{ij} received on edge ji) while $C[j]$ is the whole set of incoming messages for VN_j.

$$Q_{ji}[k] = \lambda_j + \sum_{\alpha \in C[j] \setminus \{i\}} R_{\alpha j}[k-1] \qquad (1)$$

The M Check Nodes evaluate the M parity-check equations working on the messages received from VNs ($\text{Q}_{\gamma i}$) excluding message Q_{ji} on the same edge ij. In equation (2), $R[i]$ is the set of all messages received by the CN_i from the VNs connected to it and δ_{ij} depends on the signs of incoming messages.

$$R_{ij}[k] = \psi^{-1}\left[\sum_{\gamma \in R[i] \setminus \{j\}} \psi(Q_{\gamma i}[k])\right] \cdot \delta_{ij} \qquad (2)$$

The ψ in (2) is an non–linear function which actually performs the CN processing according to the Belief Propagation algorithm. Its general expression is shown by equation (3).

$$\psi = -\ln\left(\tanh\left|\frac{x}{2}\right|\right) = \ln\frac{\left(1+e^{-|x|}\right)}{\left(1-e^{-|x|}\right)} \qquad (3)$$

Analyzing Fig. 1 a possible implementation of an LDPC decoder may be deduced in a straightforward manner. Depending on the number of processing elements actually in-

3-9810801-0-6/DATE06 © 2006 EDAA

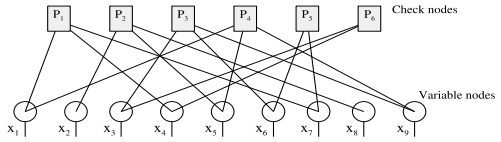

Figure 1. Tanner Graph for small LDPC code

stantiated in the hardware implementation, three decoder structures have been already proposed in the literature:

- Serial Decoders;
- Partially–Parallel (also called semi–parallel) Decoders;
- Fully–Parallel (also called Parallel) Decoders.

The Serial solution is made of a single VN/CN pair. The exchanged message are stored in a central memory. This architecture exhibits an high flexibility degree: if the implemented code has to be changed, it is sufficient to change the number of VN and CN operations provided that the memory is large enough to hold the messages. Unfortunately the throughput achievable with a Serial solution tends to be fairly poor. For this reason Serial solutions are practically not viable for actual decoder implementations.

At the farthest end there are Fully–Parallel solutions: in this case the decoder is made of N VNs and M CNs respectively. In other words, Fully–Parallel structures directly maps the Tanner Graph in hardware, trying to exploit as much as possible the algorithm parallelism. Even if this solution is able to reach extreme throughputs, the associated hardware costs tend to become prohibitive. In particular, one can try to evaluate the number of message exchanged between VNs and CNs in a simplistic case. If b_e is the average edge number entering in a generic VN, and letting c_e be the correspondent number for a generic CN, then the total number of messages can be roughly evaluated as:

$$MESS_{number} = N \times b_e = M \times c_e$$

This complexity issue becomes particularly relevant in case of LDPC codes with large block size. In such cases the relative area required by the interconnection fabric tends to be the limiting factor in delay and area of VLSI implementations. An example of this can be found in the work of Blanksby et al. [5] where a custom technique had to be developed in order to route the interconnections between VNs and CNs. Unfortunately, beside the complexity factor these structures suffer also from their total lack of flexibility. Being designed and tailored around a specific case makes parallel structures hardly adaptable to different codes, since the number of PEs and their interconnections remain fixed.

Partially–Parallel structures are a sort of compromise between the previous two classes. Generally they are composed by $N_{pp} \leq N$ VNs and by $M_{pp} \leq M$ CNs. This reduced number of PEs makes Partially–Parallel structures well suited for practical LDPC decoders implementations: high throughput can be achieved using a proper number of PEs, while the interconnection among them tends to be less critical than in the Fully–Parallel case. Unfortunately the main problem of semi-parallel implementations is the risk of collisions during the access to the memory which stores the decoding partial results. Considering N units working in parallel, N memory banks can be employed to store the information exchanged between them. The use of memory instead of dedicated interconnections enables the possibility to design more flexible structures. For instance all the edges pertaining to a same subset of VNs can be assigned to the same memory bank. In this case every VN processor can access without any particular problem to its memory bank, but the CN processors should retrieve their data from different memory banks in a scrambled order, according to the parity check matrix, thus leading to possible collisions.

2 Reconfigurable Interconnection Structure: A Space–Time Permutation Network

Although several semi-parallel implementations were recently proposed [6–8], they all rely on sub-classes of codes designed in order to simplify both encoder and decoder implementation. This means that even if the proposed approaches are able to achieve remarkable decoding performances, they cannot be applied to a generally designed code. To overcome these limitations, Tarable et al. [9] proposed a fairly different approach: rather than a joint code-decoder design, they introduce, in any previously defined code, a *Permutation Network* that properly scrambles the messages in order to avoid memory collisions. This is achieved without imposing any constraints in the designed code. To do this, the basic structure of a semi-parallel decoder has to be modified. In particular, before and after the memory banks, used to store the exchanged metrics, two *Crossbar Switches* (XBars) are inserted to scramble data (Fig. 2). This scrambling is obtained by means of *permutation laws* (denoted with β and β' in Fig. 2), which control the switches in the crossbars, to prevent two messages from being stored in the same memory bank in the same time slot. The combination of memories and crossbars performs both a spatial (crossbars) and temporal (memories) permutation of the exchanged messages.

It is supposed that, at both sides of the decoder (VNs and CNs), a node partitioning is given, each subset of nodes being assigned to a different processor. The partitioning should be made so that each processor at a given side outputs about the same number of messages. We suppose that

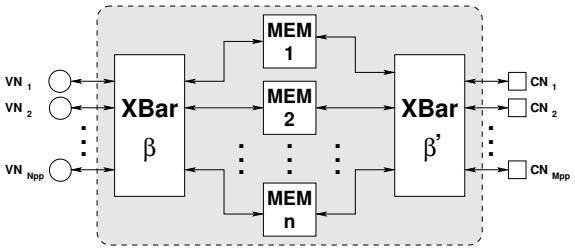

Figure 2. Implementation of the Permutation Network

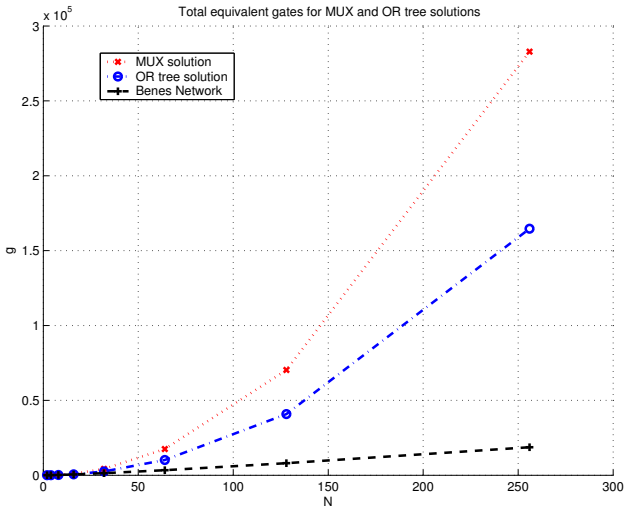

Figure 3. Benes Network vs. Crossbar Switch architectures

the scheduling of the processor outputs is also given.

In [9], an algorithm to compute the mapping function is described, working for any LDPC code. Let us introduce a *mapping function* $\mathcal{M}$, with the following meaning: given an ordering of messages, the i-th message is stored in the $\mathcal{M}(i)$-th memory bank. The permutation laws β and β' directly depend on $\mathcal{M}$. The algorithm, which accepts as its input the node partitioning and the scheduling of the processor outputs, can be divided into two successive steps, described in the following:

First step: Any step that produces a *preliminary* mapping function with the property that no memory collisions are originated. However, there are some messages for which the mapping function is not determined yet, called *blanks*.

Second step: This step accepts the preliminary mapping function output in the first step and fills all blanks. The iterative procedure of completing the mapping function is called *annealing*. The result is a valid collision-free mapping function for the given LDPC code.

Given the mapping function, the permutation laws β and β', as well as the timing of memory banks accesses, can easily be derived. Interested readers can find more details in [9].

3 Proposed Approach

As mentioned above, this mapping algorithm is able to tackle any LDPC code since it is able to find proper permutation laws. Moreover for any given code the permutation laws is always the same, so they can be computed offline each time a new code is addressed and then "downloaded" in the decoder. Thus it seems a very promising candidate for the implementation of flexible structure. On the other hand the proposed algorithm requires two Crossbar Switches to scramble the messages. This structure have to be fully connected, since each input could be connect to any output. This means that a straightforward implementation of these blocks requires an interconnection matrix of $N \times N = N^2$ multiplexers (muxes), where N is the number of input/output of the Crossbars. Moreover each multiplexer have to dispatch messages represented over a finite number of bits. Even if simpler switching elements, e.g. OR gates, are employed rather than muxes, the crossbar switches are already to far complex for actual implementation since complexity is still dominated by the fairly high amount of required switching elements. From all these premises it easy to infer that for actual implementation of decoders the straightforward realization of the proposed scheme (based on Crossbar Switches) is infeasible when the number of I/O assume values in the order of 128 or more ports. This means that actual implementations must rely on *switching fabrics* that can provide full connections from inputs to output, but with a complexity lower than $O(N^2)$.

The problem of low-complexity, fully-connected, non-blocking switching structures was already addressed in the Networking community some years ago. When, as in our case, auto-routing capabilities are not required (permutation laws are always the same for a specific code and are compute offline only when the code varies) effective solution are *Benes Networks* [10]. This structure provide full connectivity requiring only $2\log_2(N) - 1$ layers of $N/2$ 2×2-switching elements thus an overall complexity proportional to $O(N \log_2(N))$. In Figure 3 we report the great complexity improvements achievable, in terms of equivalent gates vs. required inputs N, when adopting Benes Network rather than Crossbar Switches. Even when OR gates rather than muxes are employed as switching elements the Benes Network overcomes the Crossbar Structures due to the reduced number of basic routing elements involved.

3.1 Low–Traffic Belief Propagation Algorithm

Node processing equations (1) and (2) can be rearranged as in eq. (4) and (5):

$$\begin{aligned} Q_{ji}[k] &= \lambda_j + \sum_{l \in C[j]} R_{lj}[k-1] - R_{ij}[k-1] \\ &= S_j - R_{ij}[k-1] \end{aligned} \quad (4)$$

$$\begin{aligned} R_{ij}[k] &= \psi^{-1}\left[\sum_{\gamma \in R[i]\setminus\{j\}} \psi(Q_{\gamma i}[k])\right] \cdot \delta_{ij} \\ &= \psi^{-1}\left[\sum_{l \in R[i]} \psi(Q_{li}[k]) - \psi(Q_{ji}[k])\right] \cdot \delta_{ij} \\ &= \psi^{-1}\left[PS_i - \psi(Q_{ji}[k])\right] \cdot \delta_{ij} \end{aligned} \quad (5)$$

In eq (4) S_j is the total sum of all messages entering VN_j, including the intrinsic λ, and similarly the symbol PS_i in (5) represents the sum of terms obtained by applying operator ψ to all messages incoming in CN_i. Both equations can be obtained by means of a tree of adders, which calculate S_j and PS_i amounts, followed by a final stage of subtracters to evaluate the output messages; in addition CN processing also requires that the ψ operator is applied to the subtracter output.

Starting from these rearranged processing equations, in [11], a sensible reduction in the amount of data to be exchanged among processing nodes is achieved by dispatching the total sums S_j and PS_i instead of Q_{ji} and R_{ij} messages; this simplification enables the implementation of flexible decoding architectures. These modification in the decoding algorithm lead to a different version of the traditional Belief Propagation; as proposed by the authors we will refer to this approach as *Low–Traffic Belief Propagation Algorithm* or LTBPA.

The adoption of the Benes Networks as switching structure can give significant improvements, in terms of required complexity; nevertheless a straightforward implementation of Tarable's scheme as in Fig. 2 is still too complex for a practical implementation since two Benes Networks are still necessary. On the other hand, the adoption of the low complexity interconnection framework [11], can be beneficial also when Tarable scheme is implemented. This is because the simplified formulation relying on the modified Variable and Check Nodes, as they derive from equations (4) and (5), is independent from the decoder implementation.

Since in this formulation only global messages S_i and PS_j are dispatched great simplifications are achievable since the outputs of the memories of Fig. 2 must not be scrambled again. This means that only one Benes Network is required in the actual decoder implementation. Moreover also the memory requirements are reduced since only global amounts are store rather than all the messages. Starting from these premises, a block scheme of the simplified flexible decoder is easily obtainable as in Fig. 4. Moreover, the adoption of LTBPA strategy in Tarable's scheme has another beneficial side effect. In the preliminary version of this algorithm, memory access should be performed accordingly to a well-defined scheduling; in the LTBPA version, on the other hand, the adoption of particular scheduling is not so central in terms of reduction of memory access, i.e. the latency of the interconnection network. Thus simple sequential accesses (for each memory bank) it is the most effective strategy to reduce "interconnection" latency. This is mainly related to the storage of global amount S_i and PS_j, rather than single messages, that is able to greatly simplify the overall interconnection structure with fairly little increase of complexity in the processing elements.

Table 1. Complexity comparison between BPA and LTBPA versions of an 8–input CN processor.

Architecture	Frequency [MHz]	Area $\mu\mathrm{m}^2$
BPA CN	307	51 109
LTBPA CN	265	59 208

4 Implementation results

As introduced in the previous section, LTBPA is able to reduce the interconnection requirements moving some complexity into the nodes. As a direct consequence, VNs and CNs internal architecture become pretty identical (eq. (4) and (5)), opening the possibility to reuse the same processor performing different tasks. An interesting point is to evaluate how these modifications impact on the CN/VN area. In this way it is possible to devise if and how the LTBPA represents a viable way to obtain a reduced-complexity, flexible

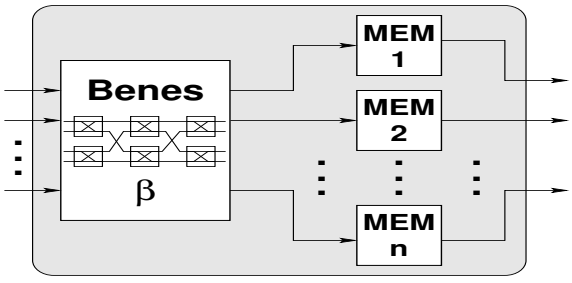

Figure 4. Tarable's Scheme with LTBPA strategy: simplified block scheme

Table 2. Complexity estimations for the LTBPA algorithm. The percentages are referred to a traditional BPA implementation.

PE Area	Increase	$+50\%$
PE Number	Decrease	-50%
Benes Network	Decrease	-50%
Number	Decrease	$1/Wc$
RAM bits		

LDPC decoder.

As a case study we then described in VHDL an 8-input, parallel CN employing the "original" structure as well as an LTBPA version. Each of these nodes has been completed characterized by performing a logical synthesis using an ST Microelectronics 0.13 μm standard-cell library. To better understand the hardware impact of this variation two scenarios have been investigated:

- Logical synthesis without timing constraints, useful to highlight how the increased complexity impacts on timing performance of the node;
- Logical synthesis with time constraints. This second experiment is useful to understand how many resources are needed to achieve the same timing performance of the simpler case.

In table 1 the results for the first scenario are shown. In this case we observe a reduction in the maximum clock frequency of nearly 15% with respect to the original BPA CN, associated with an increase in area occupation of the same percentage.

In the second investigation we have found that the LTBPA node is actually able to reach the same timing performance of the first one but at the expense on an increase in area of almost 50%. Even if the figure can be read as a significant complexity increase, it is important to remark how the total number of processors is decreased by a factor of two, due to the fact that a CN processor can now play the role of a VN as well. From all the considerations, it is possible to derive a sort of "rough" advantages estimation of the final decoder structure. The summary of this study is reported in table 2. As anticipated in 3.1, LTBPA allows to save one Benes Network since data can be read from memories in natural order. This figure represents a dramatic save in terms of area occupation, especially if one thinks that a 128×128 Benes network requires 388722 $\mu\mathrm{m}^2$ (ST Microelectronics 0.13 μm). On the last row of the table the number of required RAM bits is shown. Thanks to LTPBA reduced traffic, this number decreases of a factor that is at least equal to the minimum between the column weight W_c and the row weight W_r. Since generally for LPDC codes

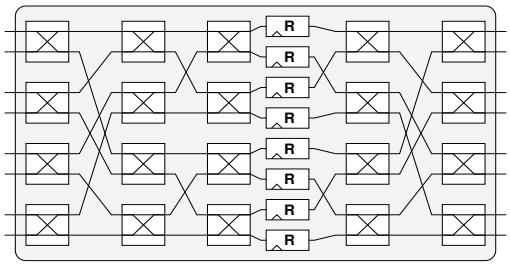

Figure 5. Pipelined Benes Network. Simple Example: 8×8 Network

$W_c < W_r$, we can assume that the gain here is proportionally to W_c.

Finally, to validate the proposed approach, a complete LTBPA–based LDPC decoder have been implemented using as a target an high–end FPGA device from XILINX. As a case study we choose to implement a partially parallel LTBPA decoder with 128 PE. From a first logical synthesis it turns out that the architecture critical path is located in the Benes Network itself. In particular using 8 bit messages the maximum operative frequency varies from 109 MHz (8 PEs case) down to 54 MHz when 128 PEs are used. For the aforementioned reasons we decided to insert a variable number of pipeline stages inside the Benes Network, trying to mitigate the severe critical path performance limit. In Figure 5 we report the structure of the pipelined solution. It is interesting to note how with the insertion of just one level of pipeline registers the network is able to operate again at 100 MHz. A deeper investigation shown that the insertion of additional levels of registers doesn't increase significantly the maximum frequency, since the critical path moves from the data path to the internal control signals. In the end we decided to use a single pipeline stage for 128 case. Moreover, we decide to pipeline also the control signals needed by the switching network. In this way it has been possible to obtain a completely pipelined design, able to accept a new set of 128 messages and the associated permutation every clock cycle. In table 3 we report the logical

Table 3. Logical synthesis results for a 128–PEs array on a X2V8000 FPGA from XILINX.

Block RAMs	48	28%
Register bits	3 323	3%
Total LUTs	18 723	20%
Max. Freq.	99 MHz	

synthesis results for the 128 PEs array. We use a Virtex-II FPGA from Xilinx reaching a maximum clock frequency of

99 MHz.

Then we decide to map on the array two different LDPC codes in order to obtain an estimate of the maximum achievable throughput. The former is a (504,252) LDPC code with rate $1/2$, while the latter is a (1920,640) LDPC with rate $1/3$. The most important feature of the proposed decoder architecture is the possibility of being adapted to different codes without having to change the hardware used. Basically all it's needed is to change the control bits and the address employed by the Benes Network. These changes can be pre–computed for each given code and stored in a proper memory: then these data can be downloaded on the on–chip memories which holds the routing informations for the array.

As far as the decoding performance are concerned, the proposed core was able to achieve a throughput of 657 Mbps and 529 Mbps per iteration for the two considered cases. This data represents a remarkable achievement since it has been reached on such a flexible platform, not tailored around any specific class of codes. A flexible solution has already been proposed in the literature by Kienle et al. [12]: they present an ASIC implementation able to achieve a throughput of 22 Mbps for a (10200, 5100) LDPC code. A fair comparison is hard to obtain since the target technology is completely different, the employed code is not comparable, etc .... Nonetheless the performance of the presented core enable us to be confident about this solution.

5 Conclusions and future works

In this work we present a flexible LDPC decoder able to being adapted to a wide variety of codes. We tackle the connectivity problem using a recently proposed method in conjunction with a low–complexity switching network. To improve the decoder efficiency we also resort to a promising modified version of the traditional BPA, called LTBPA. The resulting architecture exhibits remarkable performance.

As far as the future developments are concerned three possible directions have been identified. First we are interested in using the same structure also for last–generation international standard codes (such as DVB-S2 or IEEE 802.16e). Then a next step will be to move towards a complete ASIC implementation in order to leverage the performance of this core. Finally we will possibly look into the design and the implementation of an hardware accelerator to speed-up the simulation of large LDPC codes.

6 Acknowledgment

This work was partially supported by the Italian Department of Education, University and Research (MIUR) under Project FIRB PRIMO.

References

[1] R. G. Gallager, "Low Density Parity Check Codes," *IRE Trans. Information Theory*, vol. IT-8, no. 1, pp. 21–28, 1962.

[2] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 33, no. 6, pp. 457–458, 1997.

[3] D. J. C. MacKay, "Good Error-Correcting Codes Based on Very Sparse Matrices," *IEEE Trans. Inform. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

[4] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of Capacity-Approching Irregular Low-Density Parity-Check Codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[5] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, Rate 1/2 Low-Density Parity-Check Code Decoder," *IEEE J. Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, Mar. 2002.

[6] M. M. Mansour and N. R. Shanbhag, "High Throughput LDPC Decoders," *IEEE Trans. VLSI Syst.*, vol. 11, no. 6, pp. 976–996, Dec. 2003.

[7] T. Zhang and K. K. Parhi, "Joint Code and Decoder Design for Implementation-Oriented $(3, k)$-regular LDPC Codes," in *Proc. Asilomar Conference on Signals, Systems and Computers*, vol. 2, Nov. 2001, pp. 1232–1236.

[8] D. E. Hocevar, "LDPC code construction with flexible hardware implementation," in *Proc. IEEE Int. Conf. on Communications 2003, (ICC 2003)*, vol. 4, Anchoarage, USA, May 2003, pp. 2708–2712.

[9] A. Tarable, S. Benedetto, and G. Montorsi, "Mapping interleaver laws to parallel turbo and LDPC decoders architectures," *IEEE Trans. Inform. Theory*, vol. 50, no. 9, pp. 2002–2009, Sept. 2004.

[10] V. E. Benes, "Optimal Rearrangeable Multistage Connecting Networks," *Bell System Technical Journal*, vol. 43, pp. 1641–1656, 1964.

[11] F. Quaglio, F. Vacca, and G. Masera, "Low Complexity, Flexible LDPC Decoders," Proc. 14th IST Mobile Summit 2005, June 2005.

[12] F. Kienle, M. J. Thul, and N. Wehn, "Implemntation Issue of Scalable LDPC-Decoders," in *Proc. 3rd Int. Symp. on Turbo Codes & Related Topics*, Brest, France, Sept. 2003, pp. 291–294.