High-Throughput VLSI Implementations of Iterative Decoders and Related Code Construction Problems *

Vijay Nagarajan†, Stefan Laendner†, Nikhil Jayakumar‡, Olgica Milenkovic†, and Sunil P. Khatri‡
†University of Colorado, Boulder
‡Texas A&M University, College Station

December 18, 2006

Abstract

We describe an efficient, fully-parallel Network of Programmable Logic Array (NPLA)-based realization of iterative decoders for structured LDPC codes. The LDPC codes are developed in tandem with the underlying VLSI implementation technique, without compromising chip design constraints. Two classes of codes are considered: one, based on combinatorial objects derived from difference sets and generalizations of non-averaging sequences, and another, based on progressive edge-growth techniques. The proposed implementation reduces routing congestion, a major issue not addressed in prior work. The operating power, delay and chip-size of the circuits are estimated, indicating that the proposed method significantly outperforms presently used standard-cell based architectures. The described LDPC designs can be modified to accommodate widely different requirements, such as those arising in recording systems, as well as wireless and optical data transmission devices.

Index Terms: Code Construction, Fully-Parallel VLSI implementation, Iterative Decoding, Low-Density Parity-Check Codes, Network of PLAs

1 Introduction

One of the most prominent capacity-approaching error-control techniques in communication theory is coding with low-density parity-check (LDPC) matrices, coupled with decoding of the form of belief propagation on a graphical representation of the code. Currently, long random-like LDPC codes offer the best quality error-control performance for a wide range of standard channels [5, 6], channels with memory [10, 15], and channels with inter-symbol interference (ISI) [19]. In addition to their excellent performance, LDPC codes have decoders of complexity linear in their code length and of an inherently parallel nature. This makes them amenable for implementation using parallel VLSI architectures. The primary performance-limiting factor of most known parallel implementations is the complexity of the graph connectivity associated with random-like LDPC codes. Additional problems arise from the fact that LDPC codes of random structure also require large block sizes for good error correction performance, leading to prohibitively large chip sizes. Despite these bottlenecks, there were several attempts to come up with high throughput implementations [3] and implementation-oriented code constructions [51, 52]. The drawbacks of most of these proposed techniques are that the code-design and VLSI implementation

*Part of this work was presented at Globecom 2004, Dallas, Texas. This work is supported in part by a fellowship from the Institute for Information Transmission, University of Erlangen-Nuremberg, Germany, awarded to Stefan Laendner.

issues are considered in a somewhat decoupled manner, resulting in increased chip dimension and reduced data throughput. As an example, the standard-cell based approach adopted in [3] has a die area of 7.5 mm x 7 mm for a rate 1/2 code; the design strategy followed in that and other reports is based on choosing some known random or structured coding scheme, and developing a good parallel, serial, or partly-parallel implementation for it [3, 26, 51, 52]. Some of these strategies rely on utilizing complicated optimization techniques that fail to be efficient for code lengths beyond several thousands. In addition, they do not address the need of high throughput, low-to-moderate redundancy codecs used in recording and optical communication systems and some wireless architectures. For the applications mentioned above, the decoder is usually only one part of a significantly larger system including other components such as channel detectors/estimators, timing recovery circuits etc. Hence, it is very important to develop low hardware complexity coders/decoders that operate as efficient as possible. Despite all the above described issues, no systematic investigation of different VLSI implementation problems arising in the context of LDPC decoder and encoder design has been performed so far.

We address the problem of LDPC code construction, analysis, and VLSI implementation from a different and significantly broader perspective. The crux of the proposed approach is that VLSI implementation-aware code design can lead to an exceptional increase in data throughput and overall code performance by means of careful choices of VLSI implementation and circuit design techniques. In this context, a joint optimization of code-related and hardware-imposed code constraints is performed. The first set of constraints includes characteristics such as large girth and minimum distance of the codes; the second set of constraints is related to VLSI issues such as routing congestion, cross-talk minimization, uniform processing delay in one iteration, power conservation, and chip size reduction. For the purpose of fast prototyping, FPGA implementations of the proposed coding scheme can be devised, relying only on the *structure of the code graphs* and not on the actual VLSI layout.

The proposed work is aimed at devising a fully parallel implementation based on NPLAs. Implementing a circuit using a medium sized network of PLAs was shown to result in fast and area-efficient designs [20, 21]. As will be seen, the check and variable nodes in an LDPC decoder can be decomposed into such a network configuration, resulting in a fully parallel LDPC decoder architecture. This fully-parallel implementation also eliminates the need for *storing* the code description - the code structure is implicit in the wiring of the chip itself. The obtained implementation results indicate that PLA-based designs have a very small chip size and low power consumption even for codes of long length and that they offer a high level of operational flexibility. The system throughput is only limited by the rate at which the integrated circuit (IC) is able to read in serial data, which is approximately 10Gbps in modern CMOS technology, but it could support order of magnitude increased serial decoding rates as well. If however, the input data for the decoder is transferred to the data in parallel, then our approach can deliver decoding rates of several hundreds of Gbps.

The rest of the paper is organized as follows. Section 2 discusses problems related to the design of structured LDPC

decoder integrated circuits (ICs). Section 3 presents an overview of one possible implementation approach. Section 4 introduces the technical details needed for describing the proposed VLSI architecture. Section 5 contains an overview of the proposed layout while section 6 explains the structure of the LDPC codes supporting the proposed layout. The chip power, area, and throughput estimates are presented in section 7. Section 8 introduces generalized LDPC (GLDPC) codes and related VLSI design issues, while section 9 describes some reconfigurability problems. Section 10 discusses possible applications of the designed codecs while the concluding remarks are given in section 11.

2 LDPC Codes: Implementation bottlenecks

In 1963, Gallager [14] introduced a class of linear block codes known as low-density parity-check codes, endowed with a very simple, yet efficient, decoding procedure[1]. These codes, popularly referred to as LDPC codes, are described in terms of bipartite graphs. In the bipartite graph of a designed-rate $1-m/n$ code, the m rows of the parity-check matrix H represent check nodes ("right nodes"), while its n columns represent variable nodes ("left nodes"). The edges of the graph are placed according to the non-zero entries in the parity-check matrix. If all variable nodes have the same degree, the code is called *left-regular*. Similarly, if all check-nodes have the same degree, the code is termed *right-regular*. The decoding complexity is directly proportional to the number of edges and hence to the number of ones in the parity-check matrix, justifying the use of sparse matrices.

A consequence of the graphical representation of LDPC codes is that these codes can be efficiently decoded in an iterative manner. More specifically, decoding is performed in terms of belief propagation (BP) [22, 37], with log-likelihood ratios of bits and checks iteratively passed between the two classes of nodes until either all parity-check equations are satisfied or a maximum number of iterations is reached. The iterations are initiated at the variable nodes, which usually receive soft input information from the channel. At the end of message passing decoding, the bits are estimated based on the final reliability information of the variable nodes. We mostly focus our attention on the sum-product version of the belief propagation (BP) algorithm. The same type of design philosophy can be used for other classes of iterative algorithms, such as min-sum decoding. Furthermore, the design methods proposed in this work can be applied to both regular and irregular codes.

The operations performed at each variable and check node can be summarized as follows:

Variable nodes (VN):

Denote[2] the set of all neighboring check nodes incident to variable node v as C_v, the set of all variable nodes connected to check node c as V_c, a message on an edge going from variable node v to check node c in the l^{th} iteration as $m_{vc}^{(l)}$, and a message on the edge going from check node c to variable node v in the l^{th} iteration as $m_{cv}^{(l)}$. In this case, at each iteration of

[1] We assume that the reader is familiar with basic notions from coding theory. All definitions relevant for this work can be found in [25].
[2] In this section, we follow the notation in [37], p. 626.

the sum-product algorithm, $m_{vc}^{(l)}$ is computed as the sum of the channel information at variable node v, m_0, and the incoming messages $m_{cv}^{(l)}$ on the edges coming from all other check nodes $c' \in C_v \backslash \{c\}$ incident to v. Since there are no prior messages from the check nodes at the zeroth iteration, the algorithm is initialized to $m_{vc}^{(0)} = m_0$. Formally,

$$m_{vc}^{(l)} = \begin{cases} m_0, & \text{if} \quad l = 0 \\ m_0 + \sum\limits_{c' \in C_v \backslash \{c\}} m_{c'v}^{(l)}, & \text{if} \quad l \geq 1 \end{cases}, \tag{1}$$

where y denotes the channel output and $p(y|x=i), i=0,1$ represents the channel transition statistics, while $m_0 = \log \frac{p(y|x=1)}{p(y|x=0)}$ denotes the channel output log-likelihood ratio of the variable v.

Check nodes (CN):

From the duality principle [13] it follows that the message $m_{cv}^{(l)}$ is computed based on the messages from all other incoming edges at the previous iteration, $m_{v'c}^{(l-1)}$, according to

$$\tanh(m_{cv}^{(l)}/2) = \prod_{v' \in V_c \backslash \{v\}} \tanh(m_{v'c}^{(l-1)}/2). \tag{2}$$

The computations in Equation (2) will be referred to as the *log/tanh* operations.

The implementation bottlenecks of the decoding process can be easily identified from the previous discussion, as summarized below.

- *Large wiring overhead and routing congestion of the code graph implementation.* These problems become particularly apparent for low-rate, long and random-like codes.

- *Approximate computations performed at check nodes, involving tanh and arctanh functions.* These approximations have to be implemented for every incoming edge of a check node and they have a two-fold effect: first, they may compromise the decoder performance, and second, they can lead to a large increase in the chip size.

- *Finite precision arithmetic and finite computational time imposed on the hardware implementation.* For many codes these constraints have a significant impact on the error-correcting performance. Capacity-approaching random-like, irregular codes [38] are usually very long and take a large number of iterations (typically around 1000) ([37],p. 624) to converge to a stable solution. This has a significant bearing on the throughput of the implementation. On the other hand, restricting the maximum number of iterations performed can in certain cases lead to significant degradations of the error performance.

Current implementations fail to provide solutions to one or more of these problems. Ideally, one would like to use codes with near-capacity performance that also bound the worst-case (longest) wire length desired, and that have chip-area and chip-delay characteristics as good as possible. Most known approaches for handling these obstacles deal with code design

and implementation problems as separate issues thereby leading to non-optimal solutions [3] [3]. Also, most known implementation schemes use *standard-cell circuitry*. It was shown in [20, 21] that an implementation of a circuit using a network of medium-sized PLAs has better area and delay characteristics compared to a standard cell design. Hence, we propose to investigate PLA-based decoders and compare their performance with those of known standard-cell implementations.

3 The Proposed Approach: Structure and Full Parallelism

Our proposed implementation of a *fully-parallel* LDPC decoding system utilizes extremely fast and area-efficient NPLAs [20, 21]. The major features of the proposed system are :

- Full parallelism with the code structure "embedded" in the wiring;
- Area and delay efficient implementation with PLAs;
- A unified approach of tackling the LDPC code design and VLSI implementation problem.

This approach can yield a throughput of the order of several hundred Gbps. As a consequence, it can be used in most modern recording and wireless systems. Given the placement and routing constraints arising out of the NPLA architecture, LDPC codes are tailor-made to meet these and performance-related constraints. Such an approach yields an overall solution of the problem that demonstrates a significant improvement over prior attempts to implement LDPC codecs in VLSI.

4 LDPC Codec Architecture

4.1 Encoder Implementation

The central problem of the paper – a fully parallel decoder design – has to be viewed in the context of a scheme that deals jointly with the encoding and decoding process. LDPC *encoding* can be realized in terms of operations involving matrix multiplications that can be implemented in terms of tree-based XOR operations in hardware. This ensures that encoding delays for the codes investigated are logarithmic in the code length. Additionally, for certain LDPC codes of the form presented in the forthcoming sections, encoders based on shift registers and addition units can be used as well. In this setting, the parity check matrix itself is used for the encoding process. This significantly simplifies the overall implementation of the codec, and as a consequence, the LDPC encoding process is not expected to present a stumbling block of the architecture.

4.2 Decoder Implementation

In the proposed approach, the parallel nature of the iterative decoding process is directly exploited in the hardware implementation. Since each of the variable and check nodes makes use of information available from their counterparts only

[3] It is widely believed that the proprietary chip by Flarion Technologies [12](now Qualcomm) is a notable exception.

from the previous cycle, it is possible to let these units operate in parallel and complete their operations in one clock cycle. The main challenge in this implementation is to reduce the complexity of the inter-connects. This problem is solved at the code design level itself. The LDPC codes are hardwired into the chip and have a structure that results in small wiring overhead. The fully parallel design helps avoid storing the code parity-check matrix in a look-up table or some other way. The hardware architectures used for the variable and check nodes of the decoder are described next.

4.2.1 Variable Node Architecture

The variable node operations are specified by Equation (1). The outgoing information through any edge is the sum of the log-likelihood values of the channel information and the information coming into the variable node from all other edges. Hence, at a variable node a series of additions of log-likelihood values is performed. The channel information and check messages are quantized to values that can be represented by 5 bits. Extensive computer simulations show that 5-bit quantization results in very small degradation of the decoder performance in the waterfall region [5, 31], for most types of sufficiently long LDPC codes. Nevertheless, quantization can have a significant impact on the codes' performance in the error-floor region – see for example [33, 35, 46], but this issue will not be dealt with in this paper. Assuming 5-bit quantized messages both from the channel and the checks, a total of $\lceil \log(d_v + 1) \rceil + 1$ stages (levels) of two-input adders is needed to perform the variable operations. For this purpose, Manchester adders described in [33] are used. At the beginning of the evaluate period of a clock cycle, the messages from the previous iterations are used to perform a series of additions. The results of these additions are latched and sent as inputs to the check nodes during the next clock cycle. The sign of the sum represents the current estimate of the decoded bit. Figure 1 illustrates the described variable node architecture. Though it is possible to increase the throughput by stopping the iterative process for a given block by checking for its parity, the proposed architecture does not incorporate this feature. This feature is dictated by the constant throughput requirement imposed by

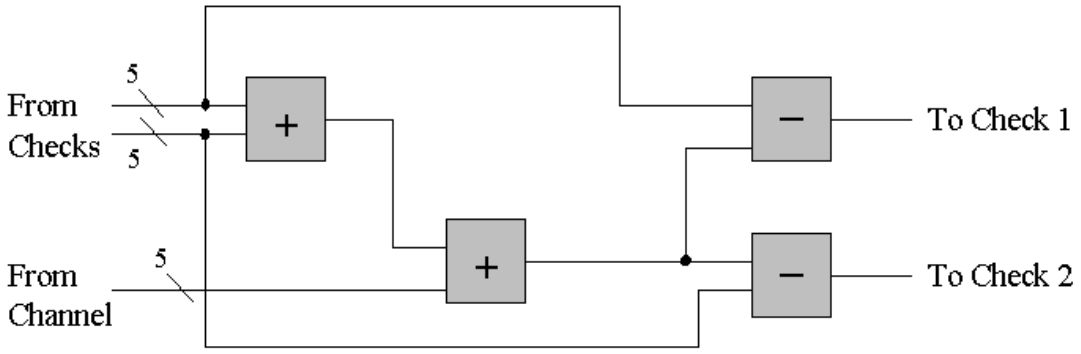

Figure 1: Variable node architecture (d_v =2)

most applications. Hence, the number of iterations performed is fixed, and chosen depending on the convergence speed

of the decoding process. To increase the throughput, this number is typically set to 16; in general, a number of 16 or 32 iterations was found to be most appropriate for the proposed code structures. For codes with a very small gap to capacity, the number of iterations would have to be significantly larger, of the order of several thousands. This follows based on the fundamental trade-off between complexity and performance of error-control codes [27]. Due to these facts, such codes are not suitable for practical implementation. A gap to capacity of approximately 1dB is usually considered a good choice regarding the trade-off between performance and complexity and the stability of operation of the decoder [36].

4.2.2 Check Node Architecture

At the check nodes, two types of operations are performed: parity updates and reliability updates. Since the parity update operation implementation has been dealt with in [3], and since it has a very small influence on the chip area and power overhead, it will not be discussed in this paper.

The reliability operations described in Equation (2) are – as are the variable node operations – performed in the log-likelihood domain in order to avoid multiplication and division operations. The system blocks are required to:

- Perform *log/tanh* operation on each incoming edge;
- Add all values obtained from these operations on a check node;
- Subtract the incoming value on each edge from the result obtained in the previous step;
- Perform an inverse *log/tanh* operation on the messages on each of the edges, in order to obtain the "outgoing" information from the variable nodes at the end of an iteration.

Figure 2 shows the reliability update architecture of a check node for the case d_c=3. Finite precision arithmetic is used to develop a PLA-based look-up for the *log/tanh* and *log/arctanh* operations, as described below.

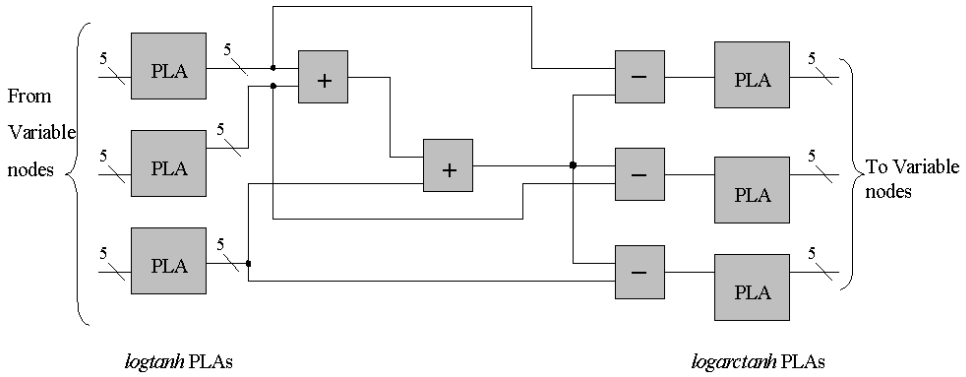

Figure 2: Architecture for reliability update in check node

4.2.3 PLA Design

The design of a good PLA layout[4] plays a crucial role in efficiently implementing the check-node circuitry. The problem of designing good PLA layouts was addressed by one of the authors in [21]. For the sake of completeness, the most important features of the PLAs are described in this section.

A PLA can be considered as a means to directly implement a conjunctive (product of sum) or disjunctive (sum of product) expression of a set of switching functions. A PLA has an "AND" plane followed by an "OR" plane. In practice, either NAND or NOR arrays are used, with the resulting PLA said to be a NAND/NAND or a NOR/NOR device.

Let us describe the functionality of a PLA of the NOR-NOR form with w rows, n input variables $x_i, i \in \{1,2,\ldots,n\}$, and m output variables $y_j, j \in \{1,2,\ldots,m\}$. Define a *literal* L_i as an input variable or its complement. A function g is described by a sum of cubes $g = \sum_{i=1}^{w} C_i$, where each cube is the product of literals $C_i = L_i^1 \cdot L_i^2 \cdots L_i^{t_i}$, according to:

$$\overline{g} = \overline{\sum_{i=1}^{w}(C_i)} = \overline{\sum_{i=1}^{w}(\overline{\overline{C_i}})} = \overline{\sum_{i=1}^{w}(\overline{\overline{L_i^1 \cdot L_i^2 \cdots L_i^{t_i}}})} = \overline{\sum_{i=1}^{w}(\overline{\overline{L_i^1} + \overline{L_i^2} + \cdots + \overline{L_i^{t_i}}})} \tag{3}$$

In words, the PLA output $\overline{g}$ is obtained as the logical NOR of a series of expressions, each corresponding to the NOR of the complement of the literals present in the cubes of g. As can be seen from the schematic view of the PLA core in Figure 3, the outputs of the PLA are implemented by vertically running *output lines* (f and g in Figure 3), which are connected to the horizontal *word lines* implementing the cubes of g. Each cube combines the vertically-running *bit-lines* (a, $\overline{a}$, b, $\overline{b}$, c and $\overline{c}$ in Figure 3) implementing the two literals for each input variable, the variable itself and its complement.

Note that in general, a PLA can implement more than one output using the same circuit structure. As an example, the PLA in Figure 3 implements 2 outputs f and g. Also, a NOR-NOR PLA yields an extremely high-speed realization of the underlying logic function, which is the reason we choose it for this work.

For the message passing algorithm, literals represent the 5-bit quantized message input log-likelihoods, so a NOR-NOR layout of the function g involving $2^5 = 32$ terms is designed accordingly. For the check node PLAs, a logic function consisting of at most 32 terms is used to implement the log-tanh operations. Based on the underlying logic sharing operations, this number can be modified. The corresponding outputs are retrieved from the output plane through their designated output drivers.

For our proposed decoder design, pre-charged NOR-NOR PLAs [20, 21] are used. This is motivated by the fact that NOR-NOR PLAs are extremely fast compared to traditional design approaches.

When a word line of a PLA switches to "high", it may happen that some neighboring lines switch to low. The worst case switching delay occurs when all neighboring lines of one line, set to "high", are in a "low" state. For a pre-charged NOR-NOR PLA, and for every word-line, its neighbors are restricted to either switch with it or remain static. This re-

[4]The design of a PLA layout in the remainder of this section follows closely the discussion in [21].

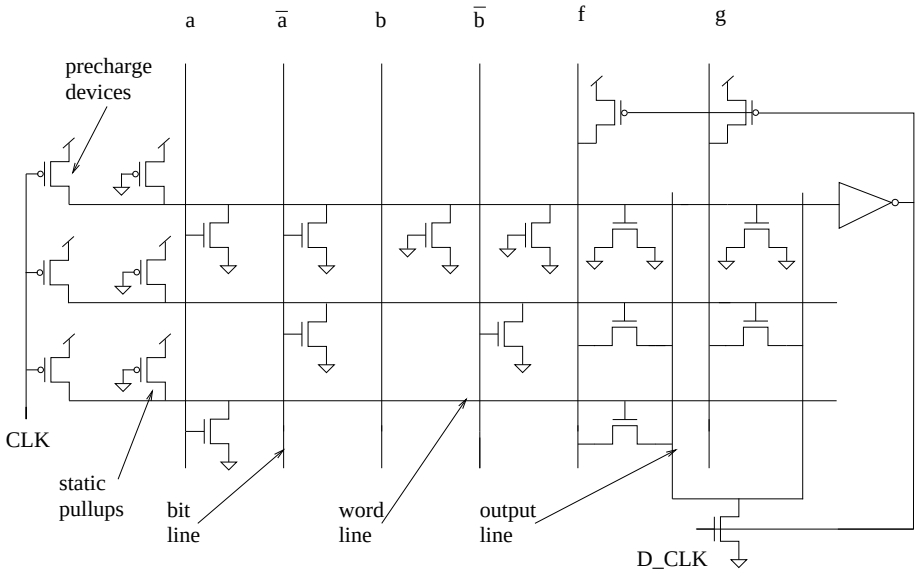

Figure 3: Schematic view of the PLA core

sults in reduced delay deterioration due to cross-talk, since adjacent word-lines never switch in opposite directions. As a consequence, in a pre-charged NOR-NOR PLA, a word-line of the PLA must switch from "high" to "low" at the end of any computation, or remain pre-charged. In order to ensure that the output of the PLA is sampled only after the slowest word-line has switched, one maximally loaded[5] word-line is designed to switch "low" in the evaluate phase of every clock. It effectively generates a delayed clock, D_CLK, which delays the evaluation until the other word-lines have reached their final values. The described PLA core was implemented using two metal layers, where the horizontal word lines were implemented in metal layer METAL2 [18] (see Figure 4).

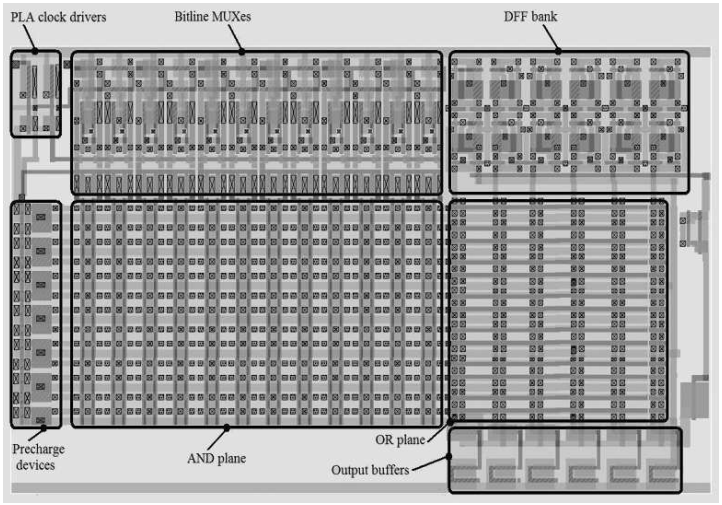

Figure 4: Structure of the PLA (layout) used in the check nodes

[5]The maximally loaded word-line has the maximum number of diffusion and gate loads possible in the PLA (see topmost word line of Figure 3)

In order to perform a valid comparison between a *single* PLA implemented in our layout style and the standard-cell layout style, we implemented both styles for four examples. The delay results were obtained utilizing SPICE [32], while the area comparison was obtained from actual layouts of both styles using two routing layers. The standard-cell style layout was done by technology-independent optimizations in SIS [44], afterwards mapping the circuit using a library of 11 standard-cells, which were optimized for low power consumption. Placement and routing was done using the *wolfe* tool within *OCT* [4], which in turn calls *TimberWolfSC-4.2* [43] for placement and global routing, and YACR [34] for completion of the detailed routing.

The examples for the PLA layout style were flattened, then the *magic* [16] layout for the resulting PLA was generated using a *perl* script. In order to perform the delay computation, a maximally loaded output line pulled down by a single output pull-down device was simulated.

				PLA implementation		Standard-cell		Ratios	
Example	*n*	*m*	*w*	*D*	*A*	*D*	*A*	*D*	*A*
cmb	16	4	15	160.3	53.3k	300	159.8k	0.534	0.334
cu	14	11	19	189.1	69.5k	420	186.5k	0.450	0.373
x2	10	7	17	164.8	45.3k	290	136.8k	0.568	0.331
z4ml	7	4	59	200.5	95.2k	575	118.3k	0.349	0.805

Table 1: Comparison of Standard-cell and PLA implementation styles

The comparison of the two layout styles is summarized in Table 1. We compare four test examples, cmb, cu, x2, and z4ml, taken from the MCNC91 benchmark suite. The parameters in the columns are:

- *n* denotes the number of input lines or variables;
- *m* denotes the number of output lines or variables;
- *w* denotes the number of rows in the PLA;
- *D* denotes delay in picoseconds;
- *A* denotes the layout area of the resulting implementation in square grids.

The values of *D* for the standard cell layout style were obtained as the maximum values after simulating about 20 input test vectors. It has to be taken into consideration that wire resistances and capacitances, which would increase the delay in the standard-cell implementation, were not accounted for. The delay numbers and area sizes for the PLA layout style are taken as worst-case values (after accounting for wire resistances and capacitances). Although this leads to a bias in comparison (in favor of the standard-cell approach), impressive improvements of the PLA layout style over the standard-cell layout style can still be observed. The PLA layout requires only an area between 33 and 81 per cent of the the standard-cell layout

area, while the average area requirement of the PLAs is 46 per cent and the average delay is 48 per cent of the standard-cell layout style. This favorable area and delay characteristics of the PLA is due to the following reasons:

- In the standard-cell implementation, traversing different levels (i.e. gates) of the design leads to considerable delays, while the PLA logic functions have a compact 2-level form with superior delay characteristics, as long as w is bounded.
- Local wiring delays and wire delay variations due to crosstalk are reduced in the PLA, since it is collapsed into a compact 2-level core.
- Extremely compact layout is achieved in the PLA by using minimum-sized devices.
- In a standard-cell layout, both PMOS and NMOS devices are used in each cell, leading to a loss of layout density due to the PMOS-to-NMOS diffusion spacing requirements. In contrast, NMOS devices are used exclusively in the PLA core, avoiding area overheads due to P-diffusion to N-diffusion spacing rules
- Finally, PLAs are dynamic, and hence faster than static standard-cell implementations.

In summary, the advantages of the proposed realization are favorable delay and area characteristics, as well as improved cross-talk immunity, compared to traditional standard-cell based ASICs. By utilizing these novel PLAs, interconnected in the manner of [21], all these characteristics can be exploited to implement fast, fully parallel LDPC codecs. For each check node, $2d_c$ PLAs and $(\lceil \log(d_c) \rceil + 1)$ 2-input adders have to be used to perform its underlying operations. The checks and the variables are hard-wired with separate wiring in either direction. As already pointed out, uniform 5-bit quantization is performed on the messages, although it is also possible to implement non-uniform quantization schemes suited to the particular channel noise density function. Accuracy of operation can be improved by using non-uniform quantization that can be adaptively changed based on the evolution of the check and variable message densities. The PLA design needs minimal modification to allow for such flexibility.

If one is willing to somewhat compromise the decoding performance of a code, an alternative belief propagation algorithm can be implemented: the sum-product algorithm can be approximated by the min-sum algorithm, for which the outgoing check-node messages are computed as

$$u_i = \left(\prod_{\substack{j=1 \\ j \neq i}}^{d_c} \operatorname{sign}(v_j) \right) \min_{\substack{j \in \{1,\ldots,d_c\} \\ j \neq i}} |v_j|. \tag{4}$$

This min-sum approximation leads to an underestimate of the true message values [50], but the simpler implementation of the *min* and *sign* functions largely reduces the check node complexity requiring less complicated circuitry and chip area of the PLAs.

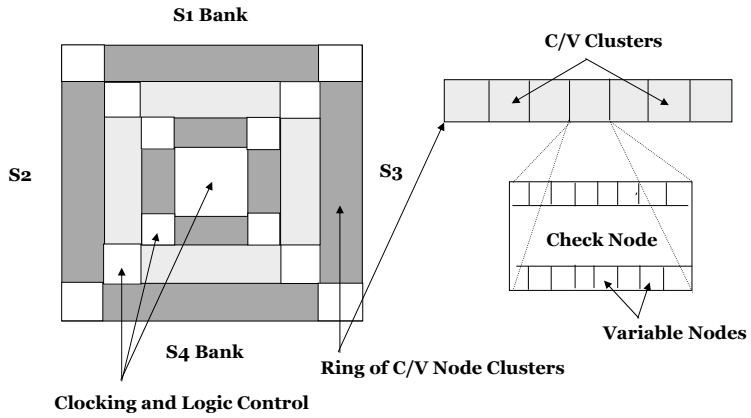

Figure 5: Concentric implementation of LDPC codes

5 VLSI Implementation of LDPC CODECs

In order to utilize the IC area most efficiently, a decoder implementation with a square aspect ratio is sought. The proposed die floor plan is shown in Figure 5. The implementation consists of banks of check and variable (C/V) node clusters, arranged in a concentric configuration. White spaces in Figure 5 are reserved for clock drivers and control logic. There are four sets of banks shown in the figure, denoted by S_1, S_2, S_3 and S_4, respectively. Each bank of C/V nodes consists of several C/V node clusters, shown in the right side of Figure 5. A cluster consists of a single check node, and several variable nodes. A typical high-rate code has a large number of variable nodes for each check node. For example, a rate 0.9 code has 10 variable nodes for each check node. Check node computations are assumed to be more complex, as indicated by the larger area devoted to these nodes' logic in the figure.

A set of clusters arranged along the sides of a square will be called a *ring*. The size of the ring is the number of banks of clusters on one side of the square. Denoting the size of a bank of C/V node clusters in ring i by $a+2i$, and the total number of check nodes by m, one obtains the following formula for the number of rings r in the above described *concentric construction*:

$$r = \left\lceil \frac{\sqrt{a^2-2a+1+m}\ +1-a}{2} \right\rceil . \tag{5}$$

Alternative C/V cluster packing with different variable to check node ratios can be used for the min-sum version of the iterative decoding algorithm, making the number of packed blocks dependent on the decoding algorithm; it also makes the C/V cluster structure more amenable for lower-rate codes. Furthermore, different variable to check-node packing ratios can be used for generalized LDPC codes, described in more detail in section 8.

As described before, the PLAs for the reliability operations of check nodes require a large chip area, which allows arrangements of C/V node clusters with a large number of variable nodes neighboring a check node as shown in Figure 5.

The regularity inherent in the IC architecture of Figure 5 represents an input constraint for the code construction prob-

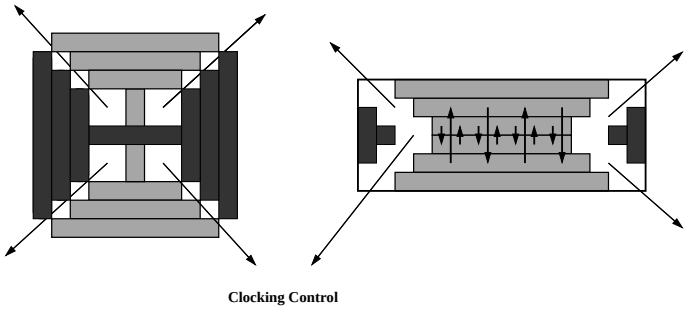

Figure 6: Alternative implementations of LDPC codes

lem. In particular, the locality of a check node and several variable nodes in a cluster is exploited during the code construction process. In order to minimize the length of long wires between check and variable nodes, the codes are additionally constrained in such a way that nodes in the S_1 bank do not communicate with nodes in the S_4 bank, and likewise, and that the nodes in S_2 do not communicate with nodes in the bank S_3. Prototype codes of this kind have been constructed, and custom IC implementations of these codes have been developed with very good results presented in section 7. The resulting design has the property that wiring is sparse and that long wire lengths are minimized due to the fact that the codes are constructed so as to exploit the regularity of the above architecture. At the same time, code performance does not have to be significantly compromised by introducing this constraint, as will be seen in the subsequent sections.

For the purpose of achieving more flexibility in the code design process, and hence in the achievable error-correcting performance, alternative layouts can be considered as well. The layouts introduce some losses in desirable VLSI implementation characteristics, which are to be compensated by the improvements in code performance. First, the node "communication constraint" can be relaxed insofar that a small number of blocks within opposite banks of the concentric construction are allowed to interact with each other. The number of units communicating across the central region of the chip will depend on the number of units per side on the innermost ring of the architecture. For example, if this number is set to 10 and only the 3 innermost rings were allowed to communicate, 36 clusters per side would be allowed to communicate with each other across the chip. This number is very small compared to the total number of clusters and cannot cause a major change in code performance. On the other hand, if the innermost ring were to contain a much higher number of blocks, the number of layers would be small resulting in a large central clocking area. This implies that a large portion of the chip is inefficiently utilized. Furthermore, it would no longer help to have the inner rings communicate across the chip, as it would imply potentially significantly longer wire lengths, resulting in routing and delay issues. This motivates the design of two possible alternative layout schemes depicted in Figure 6.

The idea is to introduce a bridge connecting the basic units across the clocking control region in the center of the chip. This can increase the percentage of variable nodes communicating across the central region of the chip and lead to improved code performance. Another approach is to make use of a chip with a $2:1$ aspect ratio, rather than a square aspect

ratio, and to additionally eliminate the central clocking control unit. The proposed architecture is shown in Figure 6. This architecture also allows for larger flexibility in the code design process by ensuring the communication of a larger fraction of units across the chip without the constraints imposed by routing and delay issues.

6 LDPC Codes for the Concentric Construction

6.1 Constraints on LDPC Codes from VLSI Implementation Structure

For the concentric VLSI implementation described in the previous section, an LDPC code can be constructed based on the following set of constraints:

- Variable and check nodes on opposite sides of the chip should not be mutually connected, or less restrictively, very few connections should exist between them; this ensures that no wires cross the central region of the block or very few do so.

- Only nodes on the border of two neighboring sides of the chip are allowed to exchange messages during the decoding process; this ensures highly localized wiring.

Posed as constraints on the code design process, these requirements take the following form. Assume that U denotes the set of variable nodes of the code, and that W denotes the set of parity-check nodes. We seek a code with good error-correcting characteristics that allows for a partition of the set U into four subsets U_1, U_2, U_3, U_4, approximately of the same size. If S_i denotes the subset of parity-check nodes in W that are adjacent to the variable nodes in U_i, i = 1,2,3,4, then one should limit the intersection between those subsets to:

$$|S_1 \cap S_2| \leq s,\ |S_3 \cap S_4| \leq s, |S_1 \cap S_3| \leq s,\ |S_2 \cap S_4| \leq s,\ |S_1 \cap S_4| \leq c,\ |S_2 \cap S_3| \leq c, \tag{6}$$

for some integers s and c such that $c \ll s$, and c sufficiently small. In this setting, the check nodes in S_1, S_2, S_3, and S_4 will be assigned to the four different sides of the chip, and there will be very limited or absolutely no interaction between these sides. Furthermore, the variables in the intersection of sets S_1 and S_2, say, will be placed on the edge between the two corresponding sides. For a code of interest, a structure satisfying these constraints can be obtained by selectively deleting some non-zero entries in the parity-check matrix. This has to be done in such a way as neither to make the code graph disconnected nor to have a large number of variables of degree less than or equal to two. Furthermore, one can devise a code construction methods that would directly address the constraints posed in Equation (6).

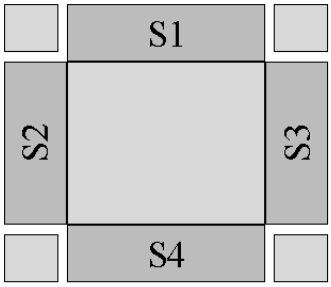

Figure 7: Layout from a coding perspective

$$H = \begin{array}{c} \\ \mathbf{S_1} \\ \mathbf{S_4} \\ \mathbf{S_3} \\ \mathbf{S_2} \\ \mathbf{S_2} \\ \mathbf{S_1} \\ \mathbf{S_4} \\ \mathbf{S_3} \\ \mathbf{S_3} \\ \mathbf{S_2} \\ \mathbf{S_1} \\ \mathbf{S_4} \end{array} \begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} & \mathbf{9} & \mathbf{10} & \mathbf{11} & \mathbf{12} & \mathbf{13} & \mathbf{14} & \mathbf{15} & \mathbf{16} & \mathbf{17} & \mathbf{18} & \mathbf{19} & \mathbf{20} & \mathbf{21} & \mathbf{22} & \mathbf{23} & \mathbf{24} \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix},$$

$$\text{i.e., } H = \begin{bmatrix} I & P & P^2 & P^3 & I & P \\ P^3 & I & P & P^2 & P & P^2 \\ P^2 & P^3 & I & P & P^2 & P^3 \end{bmatrix}, P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{7}$$

To clarify the code-design ideas, we consider a "toy-example" of a rate 1/2 code with parity-check matrix given in Equation (7). In this example, P is used to denote a circulant permutation matrix of dimension p (in the given example, $p = 4$). It is to be observed that the code described by H is of no practical use, since it is of length 24 only and its graphical representation contains a very large number of four-cycles. It can also be seen that the matrix in Equation (7) contains linearly dependent and repeated rows. Nevertheless, it is straightforward to explain all the underlying constraints and design issues on such a simple structure.

The vertical labels in the matrix of Equation (7) represent the banks of the chip-layout and the horizontal labels represent the variable nodes. All check-nodes with the same label are in the same bank of the layout. Thus, for this case one has:

$$\begin{gathered} S_1 = \{1,6,11,16,17,19,22,24\}, S_4 = \{2,7,12,13,18,20,21,23\}, \\ S_3 = \{3,8,9,14,17,19,22,24\}, S_2 = \{4,5,10,15,18,20,21,23\}, \\ |S_1| = |S_2| = |S_3| = |S_4| = 8, \\ S_1 \cap S_4 = \emptyset, S_1 \cap S_3 = \{17,19,22,24\}, S_1 \cap S_2 = \emptyset, \\ S_3 \cap S_4 = \emptyset, S_2 \cap S_4 = \{18,20,21,23\}, S_2 \cap S_3 = \emptyset. \end{gathered} \tag{8}$$

Based on Equation (8), one can see that the code matrix in Equation (7) can be used without any modifications for the proposed design approach. As a result, no wires will be crossing the central region of the chip. Furthermore, although this scenario is not directly applicable in this case, one can make the desired codes parity-check matrix slightly irregular, by deleting certain ones in H, in order to meet the implementation constraints of Equation (6). This process is to be performed in such a in such a way as to eliminate edges that result in wirings between opposite banks. In addition, such "sparsifying" could also be performed to reduce, rather than completely eliminate, the number of wires crossing the central section of the chip. Consequently, only few entries in the parity-check matrix would be modified, ensuring that with overwhelming probability the overall code characteristics and parameters are not compromised.

The variables in the intersections of adjacent banks can be placed at the "diagonals" of the concentric chip. Placement within the S_i, $i = 1,..,4$, banks themselves can be governed by known *proximity-preserving* space-filling curves, such as the Hilbert-Peano (HP) or Moor's version of the HP curve (HP-M) [42]. The square-traversing structure for these two curves (dimension four) are depicted below.

$$HP: \begin{array}{cccc} 6 & 7 & 10 & 11 \\ 5 & 8 & 9 & 12 \\ 4 & 3 & 14 & 13 \\ 1 & 2 & 15 & 16 \end{array} \qquad HP-M: \begin{array}{cccc} 7 & 8 & 9 & 10 \\ 6 & 5 & 12 & 11 \\ 3 & 4 & 13 & 14 \\ 2 & 1 & 16 & 15 \end{array} \tag{9}$$

For example, for the H matrix in Equation (7) one can take eight variables and three checks per node bank. If two variable nodes from a given bank are glued to one check, then one obtains three blocks, and two variable blocks can be grouped independently. Denote these blocks by $C_1(S_i), C_2(S_i), C_3(S_i), C_4(S_i)$, respectively, and the corresponding variable nodes by $B_{1,i}, B_{2,i}, B_{3,i}, B_{4,i}$. Then, for example, one can choose $B_{1,1} = \{1,6\}$, $B_{1,2} = \{16,19\}$, $B_{1,3} = \{17,22\}$ and $C_4(S_1) = \{11,24\}$.

An example of a practically important code parity-check matrix, with the partition property described in Equation (6) and with $c = 0$ is shown below,

$$H_S = \begin{bmatrix} H_{1,1} & H_{1,2} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & H_{2,1} & H_{2,2} \\ H_{3,1} & \mathbf{0} & \mathbf{0} & H_{3,2} \\ \mathbf{0} & H_{4,3} & H_{4,2} & \mathbf{0} \end{bmatrix}. \tag{10}$$

The question of interest is how to *choose the blocks* $H_{1,1},\ldots,H_{4,2}$ so that the resulting code has good performance under iterative message passing, and at the same time has a simple structure amenable for practical implementation also allowing for easy encoding. This problem is addressed in detail in the next section.

6.2 Code Construction Approach Based on Difference Sets

Several design strategies for H_S are described below. The sub-matrices $H_{i,j}$, $i = 1,\ldots,4$; $j = 1,2$ are chosen to be row/column subsets of "basic" parity-check matrices H based on permutation blocks, as described in more detail by one of the authors

in [48]. For the first technique the "basic" parity-check matrix H is of the form

$$H = \begin{bmatrix} P^{i_{1,1}} & P^{i_{1,2}} & \dots & P^{i_{1,s-1}} & P^{i_{1,s}} \\ P^{i_{2,1}} & P^{i_{2,2}} & \dots & P^{i_{2,s-1}} & P^{i_{2,s}} \\ \dots & \dots & \dots & \dots & \dots \\ P^{i_{m,1}} & P^{i_{m,2}} & \dots & P^{i_{m,s-1}} & P^{i_{m,s}} \end{bmatrix}, \tag{11}$$

where P is of dimension N, $i_{k,l} \in \mathcal{N} \cup \{-\infty\}$ and $P^{-\infty}$ stands for the zero matrix of dimension N. The integers $i_{k,l}$ form a so-called *Cycle-Invariant Difference Set* (CIDS) of order h, or cyclic shifts thereof [30]. CIDSs are a subclass of Sidon sets [30] which can be easily constructed according to the formula

$$\Theta = \{0 \le a \le q^h - 1 : \ \omega^a + \omega \in GF(q)\}, \tag{12}$$

where $GF(q)$ denotes a finite field with a prime number of elements q. For ($N = 5, h = 2$) and ($N = 7, h = 4$) two such sets are $\{i_1, i_2, i_3, i_4, i_5\} = \{23, 72, 244, 313, 565\}$ (mod 624) and $\{i_1, i_2, i_3, i_4, i_5, i_6, i_7\} = \{431, 561, 1201, 1312, 1406, 1579, 1883\}$ (mod 2400). The resulting codes have girth six. The last claim is a consequence of the result proved by one of the authors in [11].

Next, we choose the first two block-rows of the CIDS-based LDPC codes to represent $H_{1,1}$, and then form the other sub-blocks of H from block-rows and block-column subsets of the parity-check matrices of these CIDS codes. Two examples for CIDS-based parity-check matrices are shown below. The first corresponds to a rate $R = 1/3$ code with d_v=4, d_c=6, while the second corresponds to a rate $R = 1/2$ code with d_v=3, d_c=6. In both cases, the dimension of P, the basic circulant permutation matrix, is $7^4 - 1 = 2400$.

$$H_1 = \begin{bmatrix} P^{i_1} & P^{i_2} & P^{i_3} & P^{i_4} & P^{i_5} & P^{i_6} & 0 & 0 & 0 & 0 & 0 & 0 \\ P^{i_6} & P^{i_1} & P^{i_2} & P^{i_3} & P^{i_4} & P^{i_5} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & P^{i_1} & P^{i_2} & P^{i_3} & P^{i_4} & P^{i_5} & P^{i_6} \\ 0 & 0 & 0 & 0 & 0 & 0 & P^{i_6} & P^{i_1} & P^{i_2} & P^{i_3} & P^{i_4} & P^{i_5} \\ 0 & 0 & 0 & P^{i_1} & P^{i_2} & P^{i_3} & P^{i_4} & P^{i_5} & P^{i_6} & 0 & 0 & 0 \\ 0 & 0 & 0 & P^{i_6} & P^{i_1} & P^{i_2} & P^{i_3} & P^{i_4} & P^{i_5} & 0 & 0 & 0 \\ P^{i_4} & P^{i_5} & P^{i_6} & 0 & 0 & 0 & 0 & 0 & 0 & P^{i_1} & P^{i_2} & P^{i_3} \\ P^{i_3} & P^{i_4} & P^{i_5} & 0 & 0 & 0 & 0 & 0 & 0 & P^{i_6} & P^{i_1} & P^{i_2} \end{bmatrix} \tag{13}$$

$$H = \begin{bmatrix} P^{i_1} & P^{i_2} & P^{i_3} & P^{i_4} & P^{i_5} & P^{i_6} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & P^{i_1} & P^{i_2} & P^{i_3} & P^{i_4} & P^{i_5} & P^{i_6} \\ 0 & 0 & 0 & P^{i_1} & P^{i_2} & P^{i_3} & P^{i_4} & P^{i_5} & P^{i_6} & 0 & 0 & 0 \\ 0 & 0 & 0 & P^{i_6} & P^{i_1} & P^{i_2} & P^{i_3} & P^{i_4} & P^{i_5} & 0 & 0 & 0 \\ P^{i_4} & P^{i_5} & P^{i_6} & 0 & 0 & 0 & 0 & 0 & 0 & P^{i_1} & P^{i_2} & P^{i_3} \\ P^{i_3} & P^{i_4} & P^{i_5} & 0 & 0 & 0 & 0 & 0 & 0 & P^{i_6} & P^{i_1} & P^{i_2} \end{bmatrix} \tag{14}$$

Both codes have length $2 \times 6 \times (7^4 - 1) = 28800$, and are free of cycles of length four and six (i.e. the girth of the codes g is at least eight). Lower bounds on the minimum distances d of the codes of rate 1/2 and 1/3 can be obtained from the well-known formula due to Tanner [45],

$$d \ge 2\frac{(d_v - 1)^{g/4} - 1}{d_v - 2}, \tag{15}$$

and are equal to eight and six, respectively. Figure 8 shows the BER curves for these codes for different number of decoding iterations. For the simulations, 5-bit quantized messages were used. Observe that the LDPC code of rate $1/2$ with VLSI-implementation imposed constraints exhibits an error-floor type behavior at very high BERs - i.e. at BERs of the order of 10^{-5}. The rate $1/3$ code represents an interesting example of a rare code which exhibits multiple error floors in its performance curve. One possible combinatorial explanation for this phenomena is the decrease in the *diameter* of the code graphs represented by matrices in (13) and (14), as compared to the original code graph. The diameter of the graph is the maximum of the lengths of the shortest distance between any pair of variable nodes, and it measures the quality of "information mixing" in the code graph. The error floors might also be due to the emergence of different small trapping sets in the code. Despite their good code parameter properties (such as fairly large girth), these codes show a surprisingly weak performance and are not considered for implementation purposes.

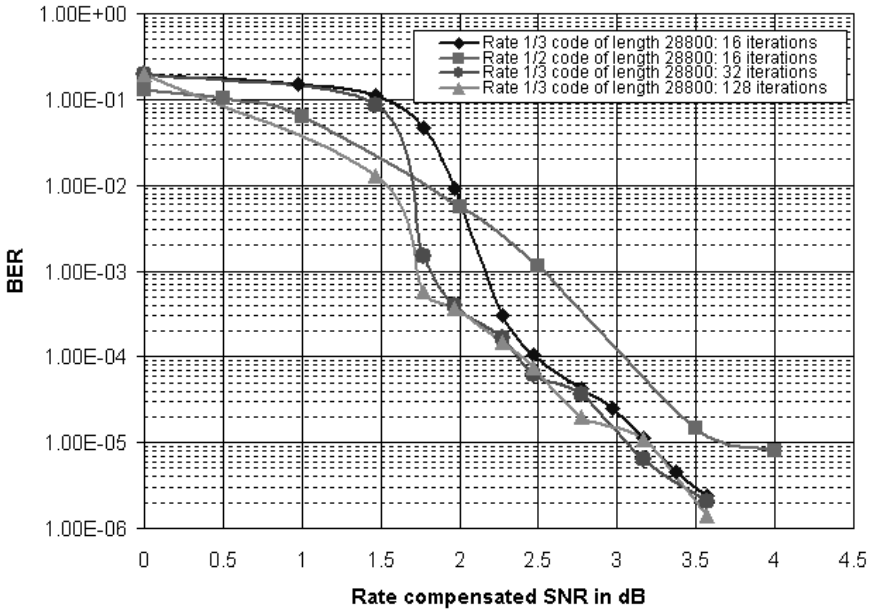

Figure 8: Error performance of regular rate-1/3 and rate-1/2 concentric codes

For the alternative constructions described in section 5, one can use codes with parity-check matrices of the form shown below.

$$H_{alt} = \begin{bmatrix} P^{i_1} & P^{i_2} & 0 & P^{i_4} & P^{i_5} & P^{i_6} & 0 & 0 & P^{i_3} & 0 & 0 & 0 \\ P^{i_6} & P^{i_1} & P^{i_2} & 0 & P^{i_4} & P^{i_5} & 0 & 0 & 0 & P^{i_3} & 0 & 0 \\ 0 & 0 & P^{i_3} & 0 & 0 & 0 & P^{i_1} & P^{i_2} & 0 & P^{i_4} & P^{i_5} & P^{i_6} \\ 0 & 0 & 0 & P^{i_3} & 0 & 0 & P^{i_6} & P^{i_1} & P^{i_2} & 0 & P^{i_4} & P^{i_5} \\ 0 & 0 & 0 & P^{i_1} & P^{i_2} & 0 & P^{i_4} & P^{i_5} & P^{i_6} & 0 & 0 & P^{i_3} \\ P^{i_3} & 0 & 0 & P^{i_6} & P^{i_1} & P^{i_2} & 0 & P^{i_4} & P^{i_5} & 0 & 0 & 0 \\ P^{i_4} & P^{i_5} & P^{i_6} & 0 & 0 & P^{i_3} & 0 & 0 & 0 & P^{i_1} & P^{i_2} & 0 \\ 0 & P^{i_4} & P^{i_5} & 0 & 0 & 0 & P^{i_3} & 0 & 0 & P^{i_6} & P^{i_1} & P^{i_2} \end{bmatrix} \tag{16}$$

The small improvement in the error-correcting ability of the resulting code in this case is not large enough to justify the

introduction of longer length wires, as was observed during extensive simulations.

If one is willing to compromise the throughput in order to achieve better quality of error-protection, the number of iterations can be increased to several hundreds. For the example of the rate 1/3 codes shown in Figure 8, Table 2 shows the trade-off between code performance, number of decoding iterations and the resulting throughput for one representative noise level corresponding to an SNR value of 2.27dB (here, SNR is defined as $10\log(E_b/N_0)$).

Number of iterations	BER	Throughput (Gbps)
16	3.00×10^{-4}	958.9158
32	1.65×10^{-4}	479.4579
128	1.49×10^{-4}	119.8645

Table 2: BER and throughput for 2.27 dB as a function of the number of iterations for the rate-1/3 code (50% duty cycle)

6.3 Construction Approach Based on Array Codes

A different technique for designing H_S of the form shown in (10) is based on array codes [48], described in terms of a parity-check matrix of the form:

$$H_A = \begin{bmatrix} P^{0\cdot0} & P^{0\cdot1} & \ldots & P^{0\cdot(q-1)} \\ P^{1\cdot0} & P^{1\cdot1} & \ldots & P^{1\cdot(q-1)} \\ P^{2\cdot0} & P^{2\cdot1} & \ldots & P^{2\cdot(q-1)} \\ \ldots & \ldots & \ldots & \ldots \\ P^{i\cdot0} & P^{i\cdot1} & \ldots & P^{i\cdot(q-1)} \end{bmatrix}, \tag{17}$$

where q is some odd prime, and P has dimension q. To construct a code with non-interacting banks, all that is needed is to retain an appropriate set of block-row labels $A = \{a_0, a_1, \ldots\} \in \{0,1,\ldots,i\}$ and block-column labels $B = \{b_0, b_1, \ldots\} \in \{0,1,\ldots,(q-1)\}$ and to delete all other permutation matrices from the matrix. To ensure good code performance, we suggest the use of improper array codes (IAC), a type of shortened array codes described by one of the authors in [29]. IACs of column weight four ($d_v = 4$) can be constructed so as to have girth at least ten, provided that the chosen sets of exponents of P avoid solutions to *cycle-governing equations* [29]. The parity-check matrices of codes of girth ten are obtained by selecting a set of block-rows from H_A and by deleting block-columns from this selection (i.e. shortening the code) in a structured manner: only those block-rows a_i and block-columns b_j are retained that are indexed by numbers from the sequences in [29], Table 5, starting as $A = \{0,1,3,7\}$ and $B = \{0,1,9,20,46,51,280,\ldots\}$ for q=911. Codes obtained from this construction have girth equal to ten.

The parity-check matrix for array-based codes of rate 1/3, of the special structure given by Equation (10), is specified

in terms of exponents of P which are products of the form $a_i \cdot b_j,\ i = 0,1,2,3,\ j = 0,1,2,3,4,5$:

$$H = \begin{bmatrix}
P^{a_0 \cdot b_0} & P^{a_0 \cdot b_1} & P^{a_0 \cdot b_2} & P^{a_0 \cdot b_3} & P^{a_0 \cdot b_4} & P^{a_0 \cdot b_5} & 0 & 0 & 0 & 0 & 0 & 0 \\
P^{a_0 \cdot b_0} & P^{a_1 \cdot b_1} & P^{a_1 \cdot b_2} & P^{a_1 \cdot b_3} & P^{a_1 \cdot b_4} & P^{a_1 \cdot b_5} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & P^{a_0 \cdot b_0} & P^{a_0 \cdot b_1} & P^{a_0 \cdot b_2} & P^{a_0 \cdot b_3} & P^{a_0 \cdot b_4} & P^{a_0 \cdot b_5} \\
0 & 0 & 0 & 0 & 0 & 0 & P^{a_1 \cdot b_0} & P^{a_1 \cdot b_1} & P^{a_1 \cdot b_2} & P^{a_1 \cdot b_3} & P^{a_1 \cdot b_4} & P^{a_1 \cdot b_5} \\
0 & 0 & 0 & P^{a_2 \cdot b_0} & P^{a_2 \cdot b_1} & P^{a_2 \cdot b_2} & P^{a_2 \cdot b_3} & P^{a_2 \cdot b_4} & P^{a_2 \cdot b_5} & 0 & 0 & 0 \\
0 & 0 & 0 & P^{a_3 \cdot b_0} & P^{a_3 \cdot b_1} & P^{a_3 \cdot b_2} & P^{a_3 \cdot b_3} & P^{a_3 \cdot b_4} & P^{a_3 \cdot b_5} & 0 & 0 & 0 \\
P^{a_2 \cdot b_0} & P^{a_2 \cdot b_1} & P^{a_2 \cdot b_2} & 0 & 0 & 0 & 0 & 0 & 0 & P^{a_2 \cdot b_3} & P^{a_2 \cdot b_4} & P^{a_2 \cdot b_5} \\
P^{a_3 \cdot b_0} & P^{a_3 \cdot b_1} & P^{a_3 \cdot b_2} & 0 & 0 & 0 & 0 & 0 & 0 & P^{a_3 \cdot b_3} & P^{a_3 \cdot b_4} & P^{a_3 \cdot b_5}
\end{bmatrix}. \quad (18)$$

Codes of different rate (e.g. 1/2) can be obtained by deleting block-columns, as described in [29].

The performance of shortened (IAC) array codes of rate $1/3$ defined by Equation (18) is shown in Figure 9. Since $q = 911$, the resulting length of the code is $12 \times 911 = 10932$. Simulations showed no error floor up to a BER of 10^{-7}. For performance comparison, we used a random-like (irregular) code of length 10800 constructed in terms of the progressive edge-growth (PEG) algorithm [17], and for an optimized degree distributions obtained from [47]. Denoting the fraction of variable nodes of degree $d_v = i$ by λ_i, the chosen variable degree distribution is $\{\lambda_2, \lambda_3, \lambda_5, \lambda_7, \lambda_{15}\} = \{0.5509, 0.2386, 0.1320, 0.000052, 0.0784\}$. As can be seen, at a bit error rate close to 10^{-5}, the IAC code with the special VLSI structure has a performance gap of approximately 1dB compared to random-like codes. This, of course, is compensated by the array codes' simplicity of implementation.

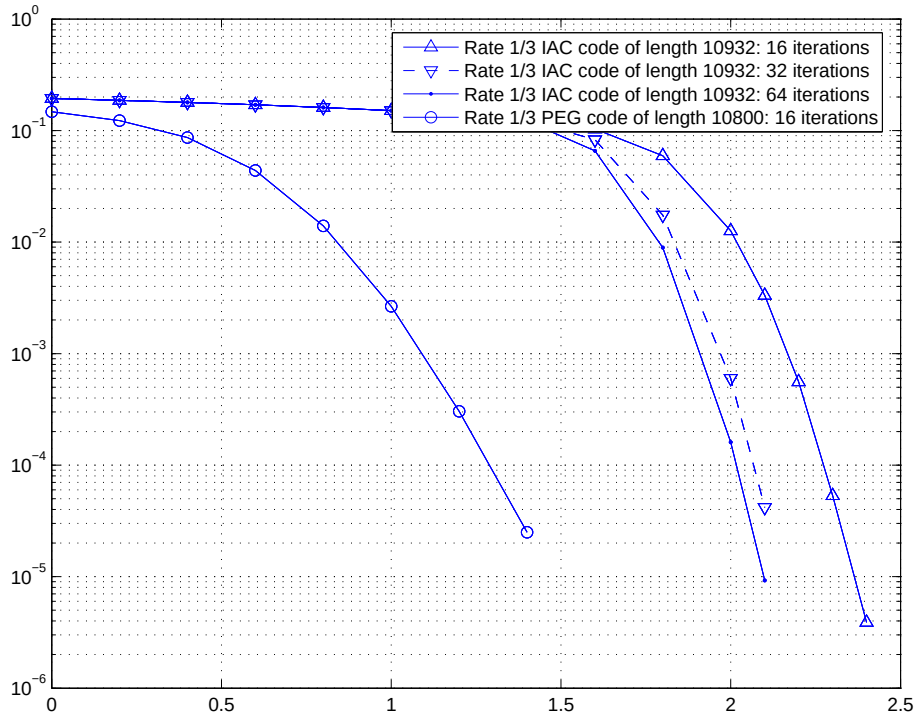

Figure 9: Error performance of rate-1/3 concentric codes from shortened array codes in comparison to random-like codes

6.4 Construction Approach based on PEG Codes

Since VLSI-implementation oriented codes based on cycle-invariant difference sets exhibit high error floors, we propose to relax some of the design constraints in order to improve the system performance. The relaxations pertain to the regularity of the code, the wiring structure within the banks S_1, S_2, S_3 and S_4, and the ease of encoding. The resulting design has a somewhat more complex, but still highly localized wiring structure, and a slightly increased chip area size.

Besides using permutation matrices to construct H_S, one can also develop VLSI-implementation oriented codecs based on random-like, irregular LDPC codes constructed by progressive edge-growth (PEG) techniques [17]. The PEG code construction algorithm can produce random-like, irregular codes with optimized degree distributions that have excellent BER characteristics [47].

A *welding PEG code* of length n is constructed in two steps. In the first step, the parity-check matrix of a PEG optimized code of length $\lfloor n/4 \rfloor$ is placed on the diagonal of an all-zero matrix, as shown in Figure 10a). In the second step, half of the non-zero entries in each row is cyclically shifted $\lfloor n/4 \rfloor$ positions to the right. The entries to be shifted are selected randomly. Performing the same set of shifts for each of the four block-rows does not change the row-weights of the matrix, nor the optimal column degree distribution. The resulting parity-check matrix structure in Figure 10b can be easily seen to fit the structure governed by Equation (10). The welding algorithm was described in a different setting in [9], where it was shown that welded codes can outperform PEG codes.

Figure 11 compares the performance of VLSI-implementation oriented codes of different lengths and rates to standard PEG codes of the same length and with the same degree distribution. As can be seen, for rate $1/4$ codes of length $n = 48000$ (standardly used for mobile communications [8]) as well as for length $n = 10800$ codes of rates $1/3$ and $3/4$, there exists only a small performance degradation for welded codes compared to PEG codes. PEG codes have error-floors that cannot be detected by means of standard Monte-Carlo simulation techniques. Furthermore, the lengths of the codes shown in Figure 11 are such that no known methods for estimating the height of the error-floor are applicable. Nevertheless, extensive computer simulations show that welded PEG codes of length several thousand should not have error-floors for BERs above 10^{-9}. These findings suggest that welded PEG codes represent excellent candidates for use in the decoder architectures proposed in this paper.

7 Estimation results

We applied the proposed method of decoder implementation using a 0.1μ process [1]. The delay and size estimates of the PLA were based on [20, 21], while the size estimate of adders were taken from [33]. An accurate delay/power evaluation of both these hardware units based on SPICE simulations was performed. It should be noted that in computing the size/delay/power estimates of adders and PLAs, wiring overhead, routing delays and the parity update operations at the

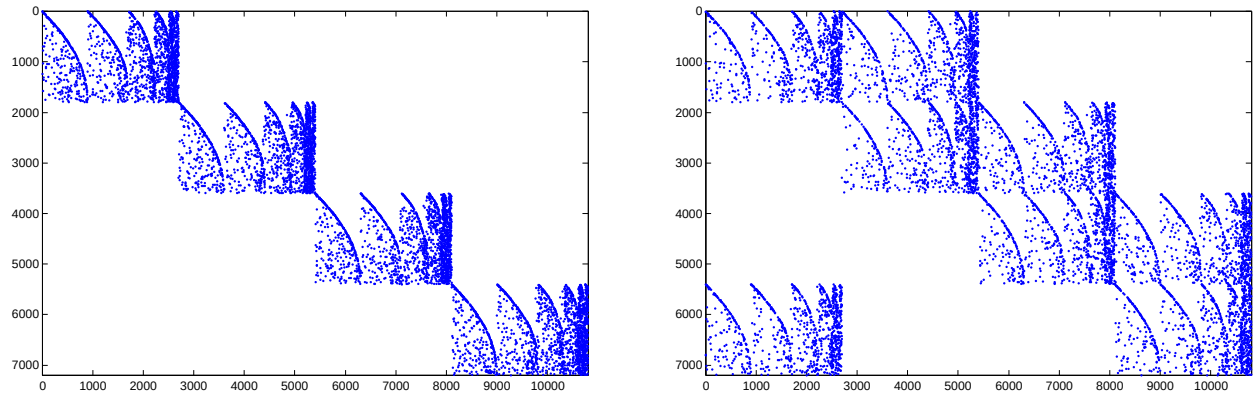

Figure 10: Construction of VLSI-implementation oriented LDPC codes by welding of PEG parity-check matrices: (a) before and (b) after welding.

checks were not accounted for. A minimal overhead is incurred upon incorporating these schemes.

	Throughput (Gbps)	Side of chip (mm)	Power (W)
Flat-out (max. duty cycle of 77.14%)	1479.4	11.0923	104.5185
50% Duty cycle	958.9158	11.0923	83.6372
Lower clock for practical applications	2	11.0923	13.3214

Table 3: Estimates for n=28800, Rate 1/2

	Throughput (Gbps)	Side of chip (mm)	Power (W)
Flat-out (max. duty cycle of 77.14%)	369.8537	5.5461	30.4711
50% Duty cycle	239.7289	5.5461	20.9093
Lower clock for practical applications	2	5.5461	3.4406

Table 4: Estimates for n=7200, Rate 1/3

As an example, rate $1/3$, $1/2$ and $3/4$ codes, suited for a variety of applications, are considered. In the first case, the column weight d_v was set to four, while the number of decoding iterations was set to 16. Tables 3, 4, and 5 show throughput, chip size, and power estimates for these given rates and lengths 28800, 7200, and 8992, respectively.

The tables show that the maximum achievable throughput is between *one and two orders of magnitude* higher than that demanded by most applications. By lowering the clock speed, the power consumption can be brought down as shown in Tables 3, 4, and 5. Consequently, power dissipation does not represent a bottleneck for practical communication system applications. The power can be reduced even further if the number of iterations were to be decreased. For example, for 32 iterations, the power consumption is estimated to be 1.8697 Watts. Alternative techniques for reducing the power

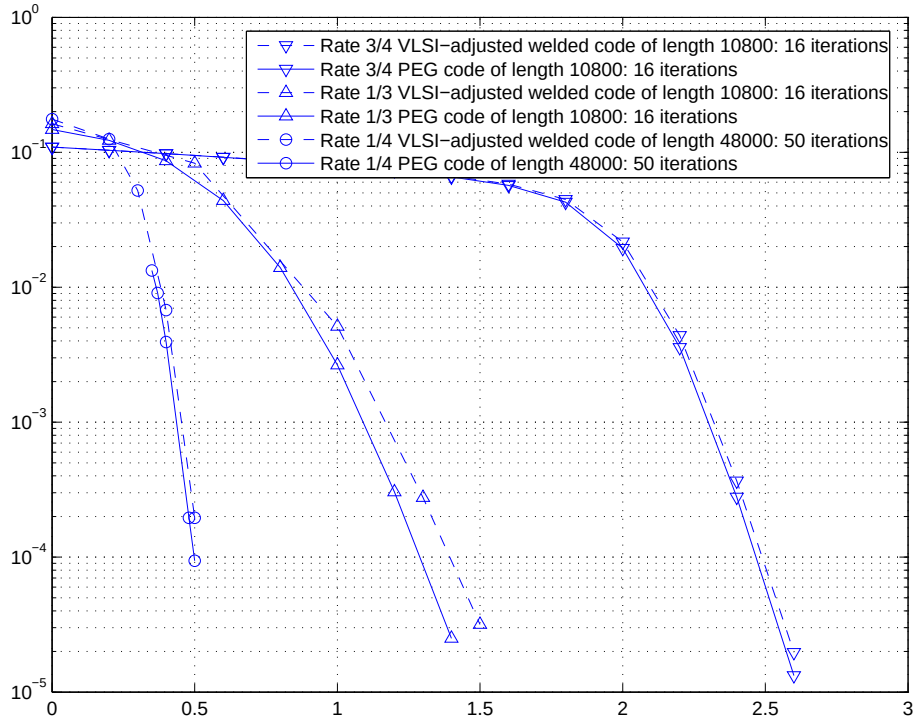

Figure 11: Error performance of rate-1/3 concentric codes constructed from welded PEG codes and random-like PEG codes

	Throughput (Gbps)	Side of chip (mm)	Power (W)
Flat-out (max. duty cycle of 77.14%)	357.4476	6.198	30.3792
50% Duty cycle	231.6876	6.198	21.1382
Lower clock for practical applications	2	6.198	4.2603

Table 5: Estimates for n=8992, Rate 3/4

consumption even further are currently under investigation.

The length $n = 28800$ code is probably not acceptable for most practical applications, with typical requirement of 1.5 Gbps throughput and power consumption within 4 Watts. A more appropriate code length is approximately 8000, for which the corresponding estimates are presented in Tables 4 and 5.

In order to compare the proposed approach with the standard-cell based implementation in [3], the estimates for a regular rate 1/2 code on 0.16μ technology are provided as well. The parameters of the design are $n = 1024$, $d_v = 3$, 64 iterations of BP decoding, and a power supply voltage of 1.5 V. For a throughput of 1 Gbps, the side of the square chip based on the proposed implementation is 2.956 mm with a power dissipation of 0.723 Watts. This is a tremendous improvement on the area figures provided in [3], where a similar code dissipated 0.690 Watts with a chip size of 7.5mm x 7mm. The reason for the drastically reduced size of our implementation is two-fold: first, we utilize extremely dense and compact implementation approaches (PLAs), and second, we perform the code construction and VLSI implementation tasks in

tandem, resulting in significantly reduced circuit areas. It should again be pointed out that the size estimates in Tables 3, 4 and 5 are for values of n that are an order of magnitude larger than the ones for the codes reported in [3]. Reducing the value of n reduces the chip-size and power consumption, at the cost of minor error performance loss.

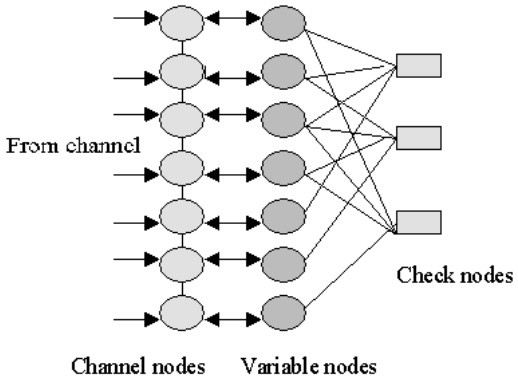

Figure 12: Modified decoder graph with channel detector

As a concluding remark, we would like to observe that in the proposed implementation, the delay introduced by the variable nodes is almost three times smaller than that of a check node. It is therefore possible to further reduce the size of the chip by using multiplexers that allow a single variable node unit to perform calculations for two variable nodes in a single clock cycle. This strategy would involve using additional multiplexers, de-multiplexers, and latches, but lead to a reduction of the number of variable node units to one half of its current value.

In most modern applications, it might also be necessary to incorporate the channel detection block into the bi-partite graph structure as shown in Figure 12. In such a case, the channel nodes perform the same set of operations as the variable node and present a minor overhead in terms of area and power dissipation. As an example, we considered a length $n = 7200$ code with a channel detection scheme added to the decoder, and a total number of 32 iterations. Such a code would have a chip size of 6.1806 mm and power dissipation of 3.6386 Watts. An inclusion of all overheads arising from timing recovery circuits, serial-parallel, and parallel-serial conversion blocks is not expected to increase the side of the chip beyond 15% of its current value, based on a very conservative estimate.

8 Generalized LDPC Codes

The implementations proposed in the previous sections can be easily adapted to accommodate generalized LDPC (GLDPC) codes [24]. GLDPC codes show excellent performance under a combination of iterative message passing and belief propagation algorithms, and for a wide variety of channels [7, 28]. There are two variants of GLDPC codes that one can consider. The first is the case when *each check* in the global parity check matrix is a short LDPC code itself (alternatively, each one in a row is replaced by a different column of a smaller length LDPC code). In the first setting, a natural generalization of

the proposed architectures is a *fractal concentric architecture*. In this realization, each "local" code is implemented as a concentric sub-unit. These units can now be looked at as the basic building blocks of the "global" code. It is to be noted that the "check" blocks in this case each have a bigger area compared to the blocks of a standards LDPC code implementation. In addition, GLDPC codes usually have a much larger overall parity check matrix. These characteristics impose a constraint on the smallest achievable size of a fractal-like chip. Consequently, a *partly parallel implementation* seems to be a more attractive solution for this problem. For example, by considering a GLDPC with 80000 variable nodes, it is possible to apply the concept of *semi-parallelism*. It would be reasonable to scale down the level of parallelism by a factor of 16, to have only 5000 variable node units and a corresponding decreased number of check units as well. Of course, in this case the throughput will decrease by the same factor, but would be still be comparable to the same value of its LDPC counterparts of the same rate. Hence, with this approach, it is possible to improve the error performance for the same throughput and almost the same chip-size and power consumption. Another variant of GLDPC codes has the property that each check node represents a short algebraic code, for which an appropriate MAP decoder is used during global iterative decoding [24]. In this case, at each check node the standard *tanh* and *arctanh* operations are replaced by MAP decoding circuits (this also justifies using PLA circuits, rather than standard-cell ones, since MAP decoding operations tend to be complex). Thus, the area of each check logic will increase based on the size of the MAP-decoder unit. For example, if a simple $[7, 4, 3]$ Hamming code is used as a local code, a 128×7 table look-up may be required. Similarly as in the previous scenario, a partly parallel implementation would provide for a solution with practical chip sizes, while allowing for good code performance.

9 Reconfigurability

In the context of LDPC decoding, circuit reconfigurability can be achieved by implementing the codes using reconfigurable wiring, and multiplexed *tanh* and *arctanh* nodes. Given a fixed number and arrangement of check and variable nodes, one can develop several codes that differ in their connectivity of check and variable nodes, but have a "nested" structure. The latter allows for the wiring differences between the codes to be minimized, resulting in a maximally area-efficient design. Using these ideas, the predictions are that such an architecture can operate with a throughput of 25Gbps and a power consumption of about 0.7W, for code lengths approximately 20,000 and rates 1/6, 1/3, 1/2, 2/3 and 5/6. The overall chip size is estimated to be 14mm on one side.

10 Applications of the proposed LDPC Code Implementations

The extremely powerful and yet fairly simple error control coding schemes of the form of codes on graphs are currently considered for applications in storage systems, optical communications, as well as wireless systems. We will briefly discuss

some potential applications of the practical design scheme proposed in this work.

Since the emergence of magnetic, optical and solid-state recording technologies, the main force supporting their progress was the improvement of areal storage density. The most promising storage systems that have emerged in the recent past are multi-layer and multi-level recorders and nanoscale-probe storage techniques [49]. Especially the class of systems based on atomic force microscopy (AFM), e.g. the "Millipede", a thermo-mechanical data-storage system based on AFM and micro electro-mechanical systems (MEMS), having data in the system recorded in blocks of 1024×1024 arrays, require powerful error control techniques. First results for utilizing codes on graphs for modern storage systems, namely LDPC codes with iterative decoding for both transversal and perpendicular magnetic recording have been presented in [39, 40, 41], while joint message-passing decoding of LDPC codes over partial response channels was addressed in [23]. The results of these investigations suggest that very large performance gains can be achieved from utilizing such coding schemes instead of Reed-Solomon (RS) codes, the well-known and by now standard coding schemes in tape and disk systems.

A debate is still going on as of how to conduct a fair comparison of complexity and performance for soft-decision LDPC, which have inherently more complex decoders, and hard-decision RS codecs, whose circuitry is complex due to their operation over *finite fields of large order*. Since quantized soft information can be used for iterative decoding (3-5 bits suffice for this purpose), the fact that all operations are performed over a binary field makes codes on graphs an attractive scheme compared to RS codes.

For the proposed code design technique, the decoder chip size can be made very small and power-efficient, and the decoder can also be easily incorporated into a larger system involving channel state estimating/equalization and timing recovery, as described in [2]. Code constraints imposed in storage systems, such as high code rate (usually exceeding 0.8), lead to an even smaller implementation complexity, due to the fact that such codes have a small number of check nodes. For possible applications in nano-storage systems, fractal-like generalized LDPC codes developed by one of the authors [7] can be used instead of LDPC codes, since they represent extensions of product codes well suited for two-dimensional recording systems.

For wireless communication systems, there already exists a prototype vector-LDPC architecture developed by Flarion Technologies [12]. The central block of the architecture is a programmable parallel processor that reads a *description of the particular LDPC code* from memory. Several codes can reside in the device at once, and switching between them incurs no overhead. The Flarion LDPC technology was integrated into a mobile wireless communications system for end-to-end Internet Protocol (IP)-based mobile broadband networking. The modulation schemes supported by Flash-OFDM include QPSK and 16QAM. The coding rates currently used are 1/6, 1/3, 1/2, 2/3, and 5/6, and the system uses adaptive modulation to rapidly switch between codes. The current maximum data throughput in the Flash-OFDM system is 3 Mbits/sec, but the

decoder actually supports speeds of up to 45 Mbits/sec. Several technical aspects of their design, such as code construction, power consumption information and chip-size are not disclosed. Also the FPGA and ASIC based implementations of the Flarion solution suggest that the throughput of their design is substantially lower compared to a custom IC implementation such as the one described in this work.

The ideas described in this paper propose an LDPC coding scheme construction with a significantly broader perspective. The described architecture can be extended or modified in order to cover a very wide range of other system architectures, for example, in concatenation with Multiple-Input Multiple-Output (MIMO) wireless systems. As opposed to the Flarion technique, the idea in this paper is based on a fully parallel implementation and the use of PLAs with a low wiring overhead. Also, in contrast to the Flarion implementation, the custom IC based solution proposed here can have the property of on-the-fly reconfigurability between codes, with significantly improved throughput, as described in the previous sections. Some additional initial experimental results show a decoding throughput of 25 Gbps and a power consumption of about 0.7W, for a code of length 20000 and rates $1/6, 1/3, 1/2, 2/3$ and $5/6$ with a die size 14mm on a side. Nevertheless, one has to point out that the Flarion implementation includes other functionalities, such as channel estimation and automatic repeat request (ARQ) controls, which can account for their observed performance.

LDPC codes are also becoming increasingly important in modern high-speed long-haul wavelength-division multiplexing (WDM) systems; there, they can be used to provide a necessary system performance margin or they can effectively increase the amplifier spacing, transmission distance and system capacity. Optical networking interface device employing a rate $1/2$ block length $n = 1024$ low-density parity-check (LDPC) code were recently developed by Agere Systems. As for the case of storage systems, high code rates and relatively short code lengths are important design parameters for these applications, which can be easily accomplished by the code architecture proposed in this paper. Full details regarding code implementations for these applications will be described elsewhere.

11 Conclusions and Future Research

A general high throughput VLSI architecture was proposed that can be used to design LDPC decoder chips for specific applications as wireless communications, magnetic recording, or optical communications. By using an efficient code design criterion and a regular chip floor plan, which is exploited during code construction, a high speed, low area design was developed. Furthermore, based on some preliminary estimates, it was concluded that practical size and power constraints can be met based on the proposed setting. The current problem of interest is to develop techniques for reducing the power consumption of the chip even further.

Acknowledgements

The authors wish to thank Thorsten Hehn for his help with the PEG algorithm and PEG code simulations and for helpful discussions.

References

[1] BSIM3 Homepage. `http://www-device.eecs.berkeley.edu/~bsim3/intro.html`.

[2] J. Barry, A. Kavcic, S. McLaughlin, A. Nayak, and W. Zeng. Iterative timing recovery. *IEEE Signal Processing Magazine*, 21:89–102, 2004.

[3] A. J. Blanksby and C. J. Howland. A 690-mw 1024-b, rate 1/2 low-density parity-check code decoder. *IEEE Journal of Solid-State Circuits*, 37(3):404–412, March 2002.

[4] A. Casotto, editor. *Octtools-5.1 Manuals*, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, Sept. 1991.

[5] S.-Y. Chung, G. D. Forney, Jr, T. J. Richardson, and R. Urbanke. On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit. *IEEE Communication Theory Letters*, 5:58–60, February 2001.

[6] S.-Y. Chung, T. Richardson, and R. Urbanke. Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation. *IEEE Transactions on Information Theory*, 47(2):657–670, February 2001.

[7] I. Djordjevic, O. Milenkovic, and B. Vasic. Generalized LDPC codes for long-haul optical communication systems. *IEEE Journal of Lightwave Technology*, 23(5):1939–1946, May 2005.

[8] A. Doenmez, T. Hehn, S. Laendner, and J. Huber. Comparison of high-performance codes on AWGN channel with erasures. In *Proceedings of 4th International Symposium on Turbo Codes in connection with the 6th International ITG-Conference on Source and Channel Coding*, Munich, Germany, April 2006.

[9] A. Doenmez, T. Hehn, S. Laendner, and J. B. Huber. Improved optimum-degree randomized LDPC codes of moderate length by Welding. In *Proceedings of the 44th Allerton Conference on Communications, Control, and Computing*, Allerton House, Monticello, Illinois, USA, September 2006.

[10] A. W. Eckford, F. R. Kschischang, and S. Pasupathy. Analysis of low-density parity-check codes for the Gilbert-Elliott channel. *IEEE Transactions on Information Theory*, 51(11):3872–3889, November 2005.

[11] J. Fan. Array codes as low-density parity-check codes. In *Proceedings of the 2nd International Symposium on Turbo Codes and Related Topics*, pages 543–546, Brest, France, September 2000.

[12] Flarion Technologies. http://www.flarion.com.

[13] J. Forney, G.D. Codes on graphs: normal realizations. *IEEE Transactions on Information Theory*, 47(2):520–548, February 2001.

[14] R. Gallager. *Low-Density Parity-Check Codes*. MIT Press, 1963.

[15] J. Garcia-Frias. Decoding of low-density parity-check codes over finite-state binary Markov channels. *IEEE Transactions on Communications*, 52(11):1840–1843, November 2004.

[16] G. T. Hamachi, R. N. Mayo, and J. K. Ousterhout. Magic: A VLSI Layout system. In *21st Design Automation Conference Proceedings*, 1984.

[17] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold. Regular and irregular progressive edge-growth Tanner graphs. *IEEE Transactions on Information Theory*, 51:386–398, January 2005.

[18] N. Jayakumar and S. Khatri. A METAL and VIA maskset programmable VLSI design methodology using PLAs. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, November 2004.

[19] A. Kavcic, X. Ma, and M. Mitzenmacher. Binary intersymbol interference channels: Gallager codes, density evolution and code performance bounds. *IEEE Transactions on Information Theory*, 49(7):1636–1652, July 2003.

[20] A. Khatri, R. Brayton, and A. Sangiovanni-Vincentelli. *Cross-talk noise immune VLSI design using regular layout fabrics*. Kluwer Academic Publishers, 2000. Research Monograph, ISBN #0-7923-7407-X.

[21] S. Khatri, R. Brayton, and A. Sangiovanni-Vincentelli. Cross-talk immune VLSI design using a network of PLAs embedded in a regular layout fabric. In *IEEE/ACM Internactional Conference on Computer-Aided Design*, pages 412–418, November 2000.

[22] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor graphs and sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, February 2001.

[23] B. Kurkoski, P. Siegel, and J. Wolf. Joint message passing of LDPC codes and partial response channels. *IEEE Transactions on Information Theory*, 48(6):1410–1423, June 2002.

[24] M. Lentmaier and K. Zigangirov. On generalized low-density parity-check codes based on hamming component codes. *IEEE Communication Letters*, 3(8):248–259, August 1999.

[25] F. MacWilliams and N. Sloane. *The Theory of Error-Correcting Coding*. North-Holland, 1977.

[26] M. Manour and N. Shanbhag. Memory-efficient turbo decoder architectures for LDPC codes. In *IEEE Workshop on Signal Processing Systems (SIPS '02)*, pages 159–164, November 2002.

[27] R. McEliece. Turbo-like Codes for Nonstandard Channels. *ISIT Plenary Talk, Washington*, 2001.

[28] O. Milenkovic, I. Djordjevic, and B. Vasic. Block-circulant low-density parity-check codes for optical communication systems. *IEEE Journal of Selected Topics in Quantum Electronics*, 10(2):294–299, April 2004.

[29] O. Milenkovic, D. Leyba, and N. Kahyap. Shortened array codes of large girth. *IEEE Trans. on Inform. Theory*, 5(8):3707–3722, August 2006.

[30] O. Milenkovic, K. Prakash, and B. Vasic. Regular and irregular low density parity check codes for iterative decoding based on cycle-invariant difference sets. In *Proceedings of the 43rd Annual Conference on Communications, Computing and Control, Allerton, IL*, October 2003.

[31] G. Murphy, E. Popovici, R. Bresnan, and P. Fitzpatrick. Design and implementation of a parameterizable LDPC decoder IP core. In *Proceedings of the 24th International Conference on Microelectronics*, volume 2, pages 747–750, 2004.

[32] L. Nagel. SPICE: A computer program to simulate computer circuits. In *University of California, Berkeley UCB/ERL Memo M520*, May 1995.

[33] J. Rabaey. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall Electronics and VLSI Series. Prentice Hall, 1996.

[34] J. Reed, M. Santomauro, and A. Sangiovanni-Vincentelli. A new gridless channel router: Yet Another Channel Router the second (YACR-II). In *Digest of Technical Papers International Conference on Computer-Aided Design*, 1984.

[35] T. Richardson. Error floors of LDPC codes. In *Proceedings of the 41st Allerton Conference on Communications, Control, and Computing*, Allerton House, Monticello, IL, USA, October 1-3 2003.

[36] T. Richardson. Workshop on applications of statistical physics to coding theory, Discussion. Santa Fe, New Mexico, January 2005.

[37] T. Richardson, M. Shokrollahi, and R. L. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Transactions on Information Theory*, 47(2):619–637, February 2001.

[38] T. Richardson and R. L. Urbanke. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Transactions on Information Theory*, 47(2):599–618, February 2001.

[39] W. Ryan. Performance of high rate Turbo codes on a pr4-equalized magnetic recording channel. In *Proceedings of the IEEE International Conference on Communications (ICC), Atlanta, GA*, pages 947–951, June 1998.

[40] W. E. Ryan, S. W. McLaughlin, K. Anim-Appiah, and M. Yang. Turbo, LDPC, and RLL codes in magnetic recording. In *Proceedings of the 2nd International Symposium on Turbo Codes and Related Topics*, Brest, France, September 2000.

[41] W. E. Ryan, L. L. McPheters, and S. W. McLaughlin. Combined turbo coding and turbo equalization for pr4-equalized lorentzian channels. In *Proceedings of the Conference on Information Sciences and Systems*, March 1998.

[42] H. Sagan. *Space-Filling Curves*. Springer Verlag, 1991.

[43] C. Sechen and A. Sangiovanni-Vincentelli. The TimberWolf Placement and Routing Package. *IEEE Journal of Solid-State Circuits*, X-20(2), April 1985.

[44] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical Report UCB/ERL M92/41, Electronics Research Laboratory, Univ. of California, Berkeley, CA 94720, May 1992.

[45] R. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, IT-27(9):533–547, April 1981.

[46] J. Thorpe. Low-complexity approximations to belief propagation for LDPC codes. 2003. Available: http://www.ee.caltech.edu/∼jeremy/research/papers/research.html.

[47] R. Urbanke. LdpcOpt - a fast and accurate degree distribution optimizer for LPDC ensembles. http://lthcwww.epfl.ch/ research/ldpcopt/index.php.

[48] B. Vasic and O. Milenkovic. Combinatorial constructions of LDPC codes. *IEEE Transactions on Information Theory*, 50(6):1156–1176, June 2004.

[49] P. Vettiger and G. Binnig. The Nanodrive Project. *Scientific American*, pages 46–54, January 2003.

[50] M. R. Yazdani, S. Hemati, and A. H. Banihashemi. Improving belief propagation on graphs with cycles. *IEEE Communications Letters*, 8(1):57–59, January 2004.

[51] T. Zhang and K. Parhi. Joint (3,k)-regular LDPC code and decoder/encoder design. *IEEE Transactions on Signal Processing*, 52(4):1065–1079, April 2004.

[52] H. Zhong and T. Zhang. Design of VLSI implementation-oriented LDPC codes. In *Proceedings of the IEEE 58th Vehicular Technology Conference*, volume 1, pages 670–673, October 2003.