Int. J. Electron. Commun. (AEÜ) 67 (2013) 588–594

Contents lists available at SciVerse ScienceDirect

International Journal of Electronics and Communications (AEÜ)

journal homepage: www.elsevier.com/locate/aeue

Window-constrained interconnect-efficient progressive edge growth LDPC codes

Aiman H. El-Maleh[a], Mohamed Adnan Landolsi[b,*], Esa A. AlGhoneim[c]

[a] COE Department, KFUPM, Dhahran, Saudi Arabia
[b] EE Department, KFUPM, Dhahran, Saudi Arabia
[c] COE Department, KSU, Riyadh, Saudi Arabia

ARTICLE INFO

Article history:
Received 17 May 2012
Accepted 30 December 2012

Keywords:
LDPC codes
Progressive edge growth
Interconnect complexity
Routing congestion

ABSTRACT

One of the attractive features of low-density parity-check (LDPC) codes is the parallel iterative nature of their iterative belief propagation decoding, making them amenable to efficient hardware implementation. However, for an arbitrary code construction, the random-like connections between the code's Tanner graph variable and check nodes makes fully-parallel implementation a difficult task as this leads to complex interconnect wiring and routing congestion. In this paper, we present a novel LDPC code design approach, based on the progressive edge growth (PEG) Tanner graph construction, to solve the problem of dense connections between processing nodes. The approach is based on controlling the maximum connection length between processing nodes in order to make fully parallel implementation feasible. The proposed algorithm offers a good compromise between error correction performance and decoder complexity. Simulation results and FPGA-based implementation comparisons are presented to demonstrate the advantages of the proposed LDPC code constructions, and it is shown that, with proper window-constrained node placement design, an improvement of up to 40% in interconnect efficiency is achievable without any significant degradation in error correction capability.

1. Introduction

Forward error correcting (FEC) codes are an essential component of modern state-of-the-art digital communication and storage systems. low density parity check (LDPC) codes, originally introduced by Gallager [1], have recently found strong renewed interest, and are widely considered to be the leading family of FEC codes. LDPC codes demonstrate performance very close to the information-theoretic bounds [2,3], while at the same time having the distinct advantage of low-complexity and high throughput iterative decoding [4,5]. One of the main features of LDPC codes over other types, such as turbo codes, is the inherent parallelism involved in LDPC decoding, which facilitates efficient hardware implementation and enables decoder throughputs much higher than other serial decoders [7–9]. Because of these advantages, LDPC codes have recently been considered in many communication standards, including 10 Gigabit Ethernet (10GBASE-T), digital video broadcasting (DVB-S2), WiMAX (802.16e), Wi-Fi (802.11n), and 60 GHz WPAN (802.15.3c) [6].

LDPC codes are linear block codes that use a sparse parity-check matrix H [1,2]. The codes can be represented by bipartite graphs (Tanner graphs) having two types of nodes: *variable bit nodes* and *check nodes*, interconnected by edges. An edge indicates that a given information bit appears in the parity check equation of the corresponding check bit, as shown in Fig. 1.

LPDC code design is typically based on building a suitable H matrix to achieve desired error correction capability. There are two main types for LDPC code parity matrices: random-like and structured [23]. Random H matrix constructions do not impose many constraints, and can fit quite well to the parameters of the desired class such as code size and rate. In general, long random LDPC codes perform better than structured LDPC codes of comparable parameters in the waterfall region, but they do require intensive interconnections between processing elements. On the other hand, structured designs are more amenable to efficient implementation.

For hardware-oriented designs, there are several critical aspects that need to be taken into account, including reduced interconnect complexity, smaller die area, lower power dissipation, and design reconfigurability to support flexible code lengths and rates (see Refs. [10,11] and the references therein for a survey and comparison of the recent trends and trade-offs in hardware-oriented LDPC design). In particular, the problem of variable–check node interconnect complexity is of paramount importance, and this is the focus of the present work. Reduced interconnect complexity is crucial in achieving small-area, low-latency and high throughput LDPC decoder implementation, because belief propagation iterative decoding uses intensive message passing between the code variable and check nodes, and long interconnect wires will be a

* Corresponding author at: EE Department, Box 1413, King Fahd University, Dhahran, Saudi Arabia. Tel.: +966 3860 1498; fax: +966 3860 3535.
E-mail addresses: aimane@kfupm.edu.sa (A.H. El-Maleh), andalusi@kfupm.edu.sa (M.A. Landolsi), esa.alg@gmail.com (E.A. AlGhoneim).

1434-8411/$ – see front matter © 2013 Elsevier GmbH. All rights reserved.
http://dx.doi.org/10.1016/j.aeue.2012.12.006

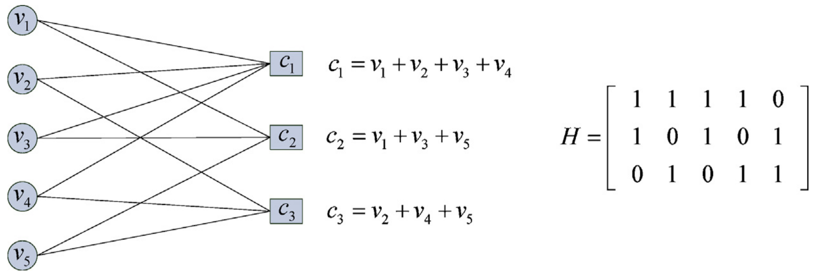
v_1
v_2
v_3
v_4
v_5
c_1
c_2
c_3
$c_1 = v_1 + v_2 + v_3 + v_4$
$c_2 = v_1 + v_3 + v_5$
$c_3 = v_2 + v_4 + v_5$
$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}$

Fig. 1. Representation of LDPC codes: Tanner graph form and matrix form.

major limiting factor in terms of power dissipation, interconnect routing congestion, and increased delay.

In recent works, some attempts have been made to tackle the interconnect problem in fully-parallel LDPC decoder implementations. Approaches based on low-complexity message-passing for reduced routing congestion in LDPC decoders and synthesizing interconnect efficient schemes are discussed in Refs. [15,16]. A window-constrained node placement approach was first introduced in Ref. [17]. A different approach based on bit-serial LDPC decoding is proposed in Ref. [18] to reduce the interconnect complexity. Block-interlaced LDPC decoders with reduced interconnect complexity are presented in Ref. [19]. Quasi-cycle LDPC (QC-LDPC) codes form another class of regular structured LDPC codes built from circulant permutation matrices [24] which offer the advantage of low encoding complexity in addition to partially parallel decoder implementation. Efficient implementations for high-rate QC-LDPC codes and area-efficient architectures are described in Refs. [20,21]. However, for fully parallel implementations, QC-LDPC codes do not retain the same low decoding complexity advantages.

In this paper, we present a different design approach suitable for fully parallel decoding and based on "window-constrained" node placement using the progressive edge growth (PEG) code construction technique. The PEG LDPC Tanner graph construction, first introduced in Ref. [12], has many attractive features as it produces practical moderate-length codes with flexible parameters and a minimized number of short cycles (which is critical for the error correction capability of LDPC codes). The general methodology in Ref. [12] does not effectively address hardware implementation issues and more recent works in Refs. [13,14] present some modified constructions to optimize the decoder hardware implementation. Improvements of the PEG approach were introduced in Refs. [25,26], and designs targeting block fading channel models were presented in Ref. [27]. An extension of QC-LDPC codes based on the PEG algorithm is also discussed in Ref. [28].

In our context, we specifically focus on hardware interconnect congestion aspects as discussed previously, and present novel PEG-LDPC constructions based on area-constrained, windowed node placement. In the proposed algorithm, variable and check nodes are laid out in a two dimensional structure. The layout imposes a maximum-size window constraint on the connections between variable and check nodes. This window constraint has a direct impact on the maximum signal delay and power dissipation aspects. Performance analysis and experimental results based on FPGA implementation show that the proposed schemes can offer a noticeable reduction in interconnect and routing congestion, while maintaining good error performance with a negligible SNR penalty (due to the restriction of "randomness" of node interconnections). The constructions also allow for design trade-offs between interconnect complexity and error correction capability, as will be demonstrated in the subsequent numerical results.

The rest of the paper is organized as follows. In Section 2, the PEG construction is introduced. The proposed window-constrained interconnect-efficient LDPC code design algorithm is then presented in Section 3. Experimental results are discussed in Section 4 to demonstrate the effectiveness of the proposed scheme, and final conclusions are given in Section 5.

2. Progressive edge growth (PEG) LDPC codes

For LDPC iterative decoding based on the belief propagation (BP) algorithm, cycles in the underlying code Tanner graph have a major impact on the decoder error performance [2]. A cycle is a path that starts and returns to the same node with edges in the path used only once. For example, in Fig. 1, the path ($v_2 \rightarrow C_1 \rightarrow v_4 \rightarrow C_3$) forms a cycle of length 4.

The BP algorithm is optimum for LDPC codes constructed with cycle-free Tanner graphs. If cycles exist, neighbors of a node are not independent; therefore, the BP algorithm is no longer optimum [2,4]. It is also found that the negative impact of cycles in an LDPC code Tanner graph increases as its girth (i.e. its minimum length cycle) decreases.

The progressive edge growth (PEG) algorithm, originally proposed in Ref. [12], is an LDPC code design approach based on maximizing the girth of the code. The PEG method is found to work well for moderate and short length codes, and provides flexibility in designing the code's parameters, which is desirable for practical applications. The PEG algorithm works by progressively establishing edges or connections between variable and check nodes in an edge-by-edge manner, while ensuring that the graph girth at each step is as large as possible.

Formally, a Tanner graph representing an H-matrix of size $M \times N$ is denoted as (V,E) where V is the set of graph vertices and E is the set of graph edges. The set V is partitioned into two sets, $V = V_v \cup V_c$, where $V_v = \{v_0, v_1, \ldots, v_{N-1}\}$ is the set of variable nodes and $V_c = \{c_0, c_1, \ldots, c_{M-1}\}$ is the set of check nodes. An edge (c_i, v_j) is a subset of E if and only if $h_{i,j} \neq 0$, $h_{i,j} \in H$, $0 \leq i \leq N-1$, $0 \leq j \leq M-1$. The set of edges E is partitioned as $E = E_{v_0} \cup E_{v_1} \cup \ldots \cup E_{v_{n-1}}$, with E_{v_j} containing all edges incident on variable node V_{v_j}. The kth edge incident on v_j is denoted by $E^k_{v_j}$, $0 \leq k \leq d_{v_j}$.

The degree of a variable node or check node is the number of edges incident on the node. The variable degree sequence is defined by $D_v = \{d_{v_0}, d_{v_1}, \ldots, d_{v_{N-1}}\}$ in which d_{v_j} is the degree of variable node v_j in increasing order, i.e. $d_{v_0} \leq d_{v_1} \leq \ldots \leq d_{v_{n-1}}$. Similarly for the check degree sequence $D_c = \{d_{c_0}, d_{c_1}, \ldots, d_{c_{M-1}}\}$ where d_{c_j} is the degree of check node c_j, and $d_{c_0} \leq d_{c_1} \leq \ldots \leq d_{c_{M-1}}$. Fig. 2 shows a Tanner graph example in which $D_v = \{2, 2, 2, 2, 3, 3, 3, 3\}$ and the check degree sequence is uniform with degree 5, i.e. $D_c = \{5, 5, 5, 5\}$.

For a given variable node v_j, its *neighbor set up to depth l*, $N^l_{v_j}$, consists of all check nodes reached by a tree spanning from variable node v_j up to depth l. Its complementary set $\bar{N}^l_{v_j}$ is defined as $V_c \backslash N^l_{v_j}$, or equivalently $N^l_{v_j} \cup \bar{N}^l_{v_j} = V_c$. To efficiently compute $N^l_{v_j}$ and $\bar{N}^l_{v_j}$, a binary flag is set for each check node. Initially, all flags are set

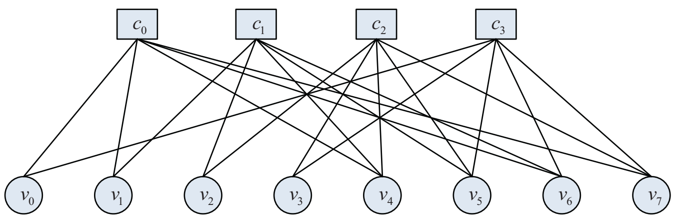

Fig. 2. An example of variable nodes degree sequence $D_v = \{2, 2, 2, 2, 3, 3, 3, 3\}$.

to 0. As the tree originated from node V_v proceeds to depth l, the indicators of check nodes included in the spanning tree are set to 1, indicating that these nodes belong to $N^l_{v_j}$. Likewise, all check nodes with flags of value 0 belong to $\bar{N}^l_{v_j}$. The PEG algorithm selects a check node among the nodes in the set $\bar{N}^l_{v_j}$ with the maximum possible depth l. The selected check node is the one having the smallest degree under the current partial graph. This ensures that the PEG Tanner graph has check node degrees as uniform as possible. However, other choices may still exist because multiple check nodes in $\bar{N}^l_{v_j}$ might have the same lowest degree, particularly at the initial construction step. In this case, a check node is randomly chosen among the lowest degree check nodes in the set $\bar{N}^l_{v_j}$. Based on the PEG algorithm, the shortest cycle passing through a new edge $E^k_{v_j}$ is $2 \times (l + 2)$, since the check node c_i is picked from level $l+1$, i.e. the set $\bar{N}^l_{v_j}$.

3. Proposed interconnect-efficient PEG LDPC algorithm

Although the PEG construction yields good girth-conditioned LDPC codes [9], it is difficult to implement these codes using fully-parallel architectures due to their inherent feature of maximizing variable and check nodes separation for the purpose of maximizing the graph girth. Based on the PEG algorithm, when a new connection is to be added to a variable node v_j, the selected check node for connection is the one in the farthest level of the tree originated from the variable node v_j. This results in dense and lengthy node interconnections (possibly exceeding those of random code construction methods). In our proposed approach, the standard PEG algorithm is modified in a way that minimizes interconnection lengths and routing congestion with a negligible impact on the error correction performance, as will be explained next.

3.1. Assumptions and definitions

The variable and check nodes are grouped in cells based on the code rate. For an LDPC code of rate $1 - p/q$, cells are constructed with each cell having p check nodes and q variable nodes. The cells are laid out on an X–Y grid. The 2D grid is selected for the layout to allow for a direct VLSI placement of the cells. Fig. 3 shows a layout example of a simple (32, 16) LDPC code. This code has a rate of 1/2; therefore, each cell contains one check node and two variable nodes, and the cells can be laid on a rectangular X–Y grid of size 4×4. The *width* of the grid in which the cells are laid out is an input parameter to the proposed algorithm. The layout size in this example being 4×4 implies that the coordinates of a cell can take values from the set {0,1,2,3}. Based on the layout width, each (variable or check) node is assigned (*x*,*y*) coordinates. The assignment of (*x*,*y*) coordinates to check and variable nodes for a rate $1 - p/q$ LDPC code is done as follows. For a check node c_j, the (*x*,*y*) coordinates are given by:

$$c^x_j = \left\lfloor \frac{j}{p} \right\rfloor \bmod \text{width} \tag{1}$$

$$c^y_j = \left\lfloor \frac{j}{p \times \text{width}} \right\rfloor \tag{2}$$

Similarly, the (*x*,*y*) coordinates of a variable node v_i are given by:

$$v^x_i = \left\lfloor \frac{i}{q} \right\rfloor \bmod \text{width} \tag{3}$$

$$v^y_i = \left\lfloor \frac{i}{q \times \text{width}} \right\rfloor \tag{4}$$

For the layout example in Fig. 3, the (*x*,*y*) coordinates of a check node c_j and a variable node v_i are: $c^x_j = j \bmod 4$, $c^y_j = \left\lfloor \frac{j}{4} \right\rfloor$, $v^x_i = \left\lfloor \frac{i}{2} \right\rfloor \bmod 4$ and $v^y_i = \left\lfloor \frac{i}{8} \right\rfloor$.

The main objective of the proposed algorithm is to design an interconnect-efficient LDPC code structure. To meet this, a constraint is set on the maximum connection length between any two variable and check nodes. The connection length between a variable node and a check node is computed based on the *Manhattan distance* between the two cells in which the nodes exist. The Manhattan distance (α) between a variable node v_i and a check node c_j is given by:

$$\alpha(v_i, c_j) = \left| v^x_i - c^x_j \right| + \left| v^y_i - c^y_j \right| \tag{5}$$

For example, in Fig. 3, the Manhattan distance between v_{10} and c_{11} is 3. The two nodes v_1 and c_0 have the same coordinates; therefore, the Manhattan distance between them is zero, and both v_1 and c_0 reside in the same cell. Finally, for each variable node v_i, S_i is defined as the pool of check nodes that can be connected to v_i without violating a maximum Manhattan distance ($\alpha_{\max}$) constraint, i.e. $S_i = \{c_j$: given that $\alpha(v_i, c_j) \le \alpha_{\max}\}$. For example, with $\alpha_{\max} = 2$ for the layout in Fig. 3, pools for v_0 and v_{10} are: $S_0 = \{c_0, c_1, c_2, c_4, c_5, c_8\}$ and $S_{10} = \{c_0, c_1, c_2, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{13}\}$, respectively.

3.2. Window-constrained interconnect-efficient PEG algorithm

The proposed interconnect-efficient PEG LDPC construction algorithm is given in Table 1, and its steps are described as follows. Initially, each check and variable node are assigned a placement attribute based on Eqs. (1)–(4), steps 1–5. In step 6, variable nodes are sorted according to their check nodes pool size. The sorted list determines the order in which variable nodes are processed. Variable nodes that have less possible connections options (usually placed near the layout grid boundaries) are given priority. For example, in the layout of Fig. 3, $|s_0| = 6$ while $|s_{10}| = 11$. This means that the possible check nodes that could be connected to v_0 are less than those that could be connected to v_{10}. The remaining steps are similar to the standard PEG approach, with the addition of maximum Manhattan distance constraint ($\alpha_{\max}$).

The purpose of step 15 is to avoid the condition when check nodes in $\bar{N}^l_{v_i}$ with smallest degree are all outside the pool of v_i, i.e. $B = \varnothing$. In this case, we keep incrementing the degree of the candidate check nodes and construct a new set (B) until $B \neq \varnothing$. Increasing the degree of candidate check nodes has a negative impact on the degree balance of check nodes in the resulting LDPC code. Another solution for avoiding an empty set of candidate check nodes (B) is to decrement the level in which check nodes are picked, i.e. instead of picking check nodes from $\bar{N}^l_{v_i}$, check nodes are picked from $\bar{N}^{l-1}_{v_i}$. However, this option has a negative impact on the error performance since it decreases the graph girth and average cycle lengths.

The proposed code constructions can be further improved by imposing initial connections between variable and check nodes that are close to each other. The initial connections can decrease the LDPC code interconnection complexity with very minor impact on its error correction performance. Three different variations of initial connections are studied:

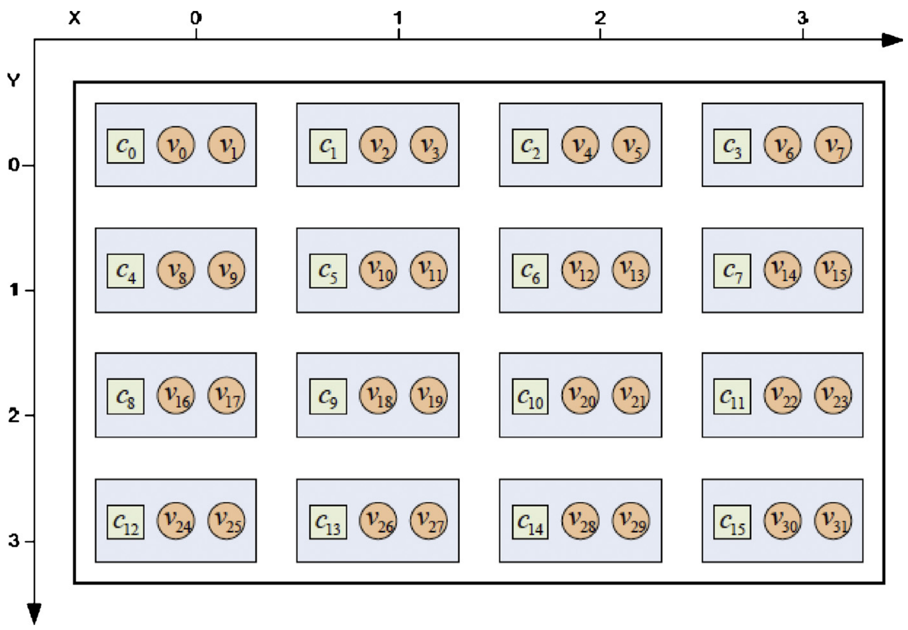

Fig. 3. Example of a (32, 16) LDPC code layout.

Table 1
Interconnect-efficient PEG LDPC code construction algorithm.

```
Inputs   $N$: number of variable nodes; $M$: number of check nodes
         $d_v$: degrees of variable nodes
         width: the width of the grid on which cells are placed
         $\alpha_{\max}$: the maximum Manhattan distance constraint.
Output   H-matrix
1.       $\frac{p}{q} = \frac{N-M}{N}$ // code rate
2.       for $j = 0$ to $M - 1$ do // perform check nodes placement
3.         $c_j^x = \left\lfloor \frac{j}{p} \right\rfloor$ mod width; $c_j^y = \left\lfloor \frac{j}{p \times \text{width}} \right\rfloor$
4.       for $i = 0$ to $N - 1$ do // perform variable nodes placement
5.         $v_i^x = \left\lfloor \frac{i}{q} \right\rfloor$ mod width; $v_i^y = \left\lfloor \frac{i}{q \times \text{width}} \right\rfloor$
6.       Sort variable nodes based on increasing order of their pool size, i.e.
         $|S_i|$
7.       For each variable node $v_i$ do
8.         for $k = 0$ to $d_v - 1$ do
9.           if $k = 0$ then // special case, first connection for $v_i$
10.            $A = \{c_w$: where $c_w$ are check nodes having the lowest degree
         in $S_i\}$
11.            $E_{v_i}^0 \leftarrow$ edge $(c_j, v_i)$, where $E_{v_i}^0$ is the first edge incident to $v_i$,
         and $c_j$ is chosen randomly from $A$.
           else
12.          Expand a tree originating from $v_i$ up to level $l$ under the
         current partial graph. Stop expansion until one of the two
         conditions is satisfied:
               (a) $\bar{N}_{v_i}^l \neq \varnothing$ but $\bar{N}_{v_i}^{l+1} = \varnothing$, or
               (b) The cardinality of $N_{v_i}^l$ stops increasing.
13.          $d$ = lowest check node degree in $\bar{N}_{v_i}^l$
14.          $B = \{c_w: c_w \subseteq \bar{N}_{v_i}^l$, degree of $c_w$ is $d$, and $c_w \subseteq S_i\}$
15.          if $B = \varnothing$ then increment $d$ and goto step 14
16.          $E_{v_i}^k \leftarrow$ edge $(c_j, v_i)$, where $E_{v_i}^k$ is the $k$th edge incident to $v_i$ and
         $c_j$ is chosen randomly from $B$.
```

Initial connections-I: for each cell, connect *one* variable node to a check node in the same cell. Fig. 4a shows an example for this connection.

Initial connections-II: for each cell, connect *two* variable nodes to a check node in the same cell. Fig. 4b shows an example for this connection.

Initial connections-III: connect two variable nodes to a check node in the same cell and one variable node to a check node in a neighbor cell, as shown in Fig. 4c.

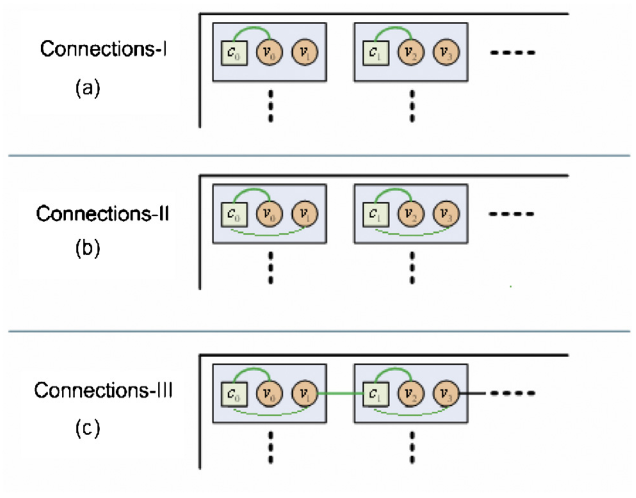

Fig. 4. Examples of different initial connections.

The proposed PEG algorithm is subsequently run to construct the remaining connections.

4. Experimental results

In order to demonstrate the effectiveness of the proposed algorithm, experiments are performed involving the construction of LDPC codes with different maximum Manhattan distance constraints ($\alpha_{\max}$). Each constructed code is evaluated using two metrics: error correction performance and efficiency of hardware implementation. The frame error rate (FER) is selected as an error performance metric. The hardware efficiency of the resulting LDPC code is evaluated using two methods: by computing the maximum and average interconnection length, and by modeling the LDPC decoder in VHDL and implementing it on an FPGA.

The LDPC codes used in the subsequent experimental results are rate-1/2 (1024,512) codes. The following constructions are considered:

Table 2
Hardware interconnection and speed efficiency for different LDPC codes.

Code	Maximum Manhattan connection length	Average Manhattan connection length	Max delay, ns (FPGA)
PEG	43	15.9	8.866
PEG-11	11	6.2	6.717
PEG-9	9	5.2	5.936
PEG-7	7	4.2	5.808
WM8L	27	11.4	7.064

PEG: LDPC code generated using the standard PEG algorithm of Ref. [9].
PEG-11: LDPC code generated using the proposed algorithm with a maximum Manhattan distance constraint ($\alpha_{\max}$) equal to 11.
PEG-9: LDPC code generated using the proposed algorithm with a maximum Manhattan distance constraint ($\alpha_{\max}$) equal to 9.
PEG-7: LDPC code generated using the proposed algorithm with a maximum Manhattan distance constraint ($\alpha_{\max}$) equal to 7.
WM8L: LDPC code based on non-PEG interconnect-efficient window constraint design in Ref. [4].

Table 2 shows hardware efficiency results for the different codes. The table shows three implementation efficiency measures: maximum Manhattan connection length, average Manhattan connection length and maximum interconnection delay (in nanoseconds) after FPGA place and route implementation. The maximum Manhattan distance constraint ($\alpha_{\max}$) has a direct impact on the propagation delay and thus speed performance of the code implementation.

The average connection length measures the interconnection complexity and reflects routing congestion. It is obvious that the standard PEG code generated without any hardware constraint is the worst in terms of maximum and average Manhattan connection length and FPGA implementation speed performance. It is also evident that the codes generated by the proposed algorithm (PEG-7, PEG-9 and PEG-11) have hardware efficiency better than the WM8L code.

Fig. 5 shows the error performance for the different codes using standard BP decoding. It is evident that the FER increases as the maximum Manhattan distance constraint ($\alpha_{\max}$) decreases. The PEG-11 code is about 0.1 dB from the standard PEG at FER = 10^{-6}.

Fig. 6 shows the error correction performance for the different codes under the modified BP decoding algorithm proposed in Ref. [22] (which implements “learning-based” tuned belief propagation decoding). The modified BP decoding algorithm further eliminated the slight performance gap between the PEG-11 and conventional PEG codes as shown in Fig. 8. This demonstrates that, by using tuned decoding algorithms, the proposed interconnect-efficient PEG constructions do not suffer any error performance penalty, while retaining their efficiency and hardware implementation advantages.

Fig. 7 shows the error correction performance of the proposed algorithm for three different initial connection assumptions and a maximum Manhattan distance constraint ($\alpha_{\max}$) equal to 10. The PEG-10 initial-I has similar performance to PEG-10 without initial connections. As the number of initial connections increases, a slight SNR degradation (by no more than 0.1 dB) of the resultant LDPC code is noticed compared to the PEG-10 code without initial connections. As the number of initial connections increases, the average connection length decreases significantly. Table 3 shows the percentage decrease in average Manhattan connection length of PEG-10 LDPC codes produced with different initial connections compared to the reference PEG-10 LDPC code generated without initial connections, and it is seen that up to 43% interconnect

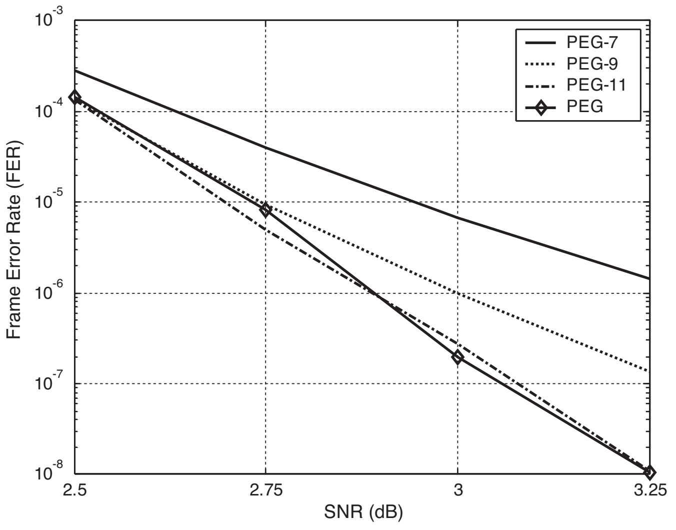

Fig. 6. Error correction performance using the learning-based BP algorithm of Ref. [22].

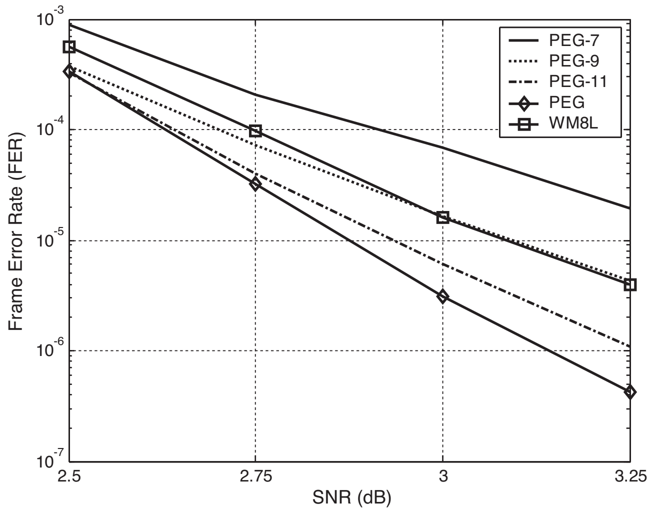

Fig. 5. Error correction performance using standard BP algorithm.

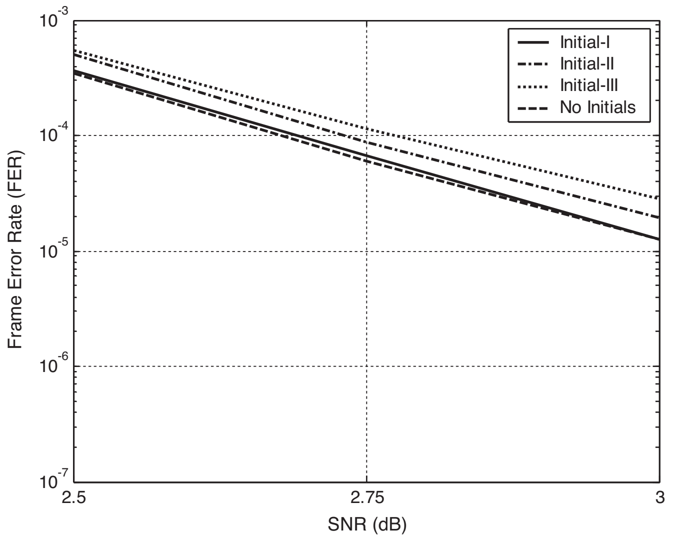

Fig. 7. Performance of PEG-10 with different initial connections using standard BP decoding.

Table 3
Interconnect efficiency of the proposed PEG LDPC codes with different initial connections.

Code	Maximum Manhattan connection length	Average Manhattan connection length	Improvement (%)
PEG-10	10	6.8	–
PEG-10 with initial-I	10	5.7	16
PEG-10 with initial-II	10	4.6	32
PEG-10 with initial-III	10	3.9	43

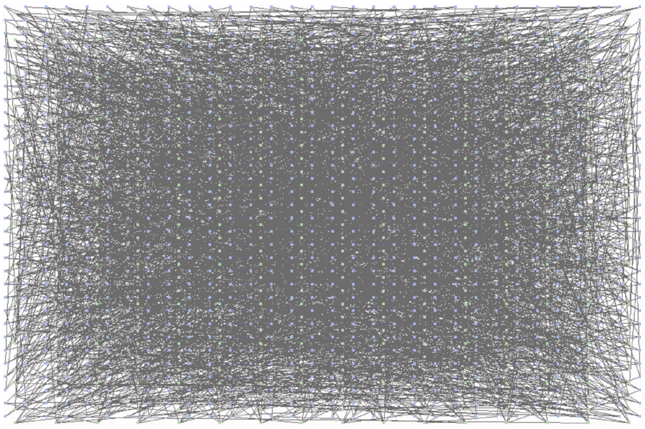
Fig. 8. Interconnection complexity for the conventional PEG code construction.

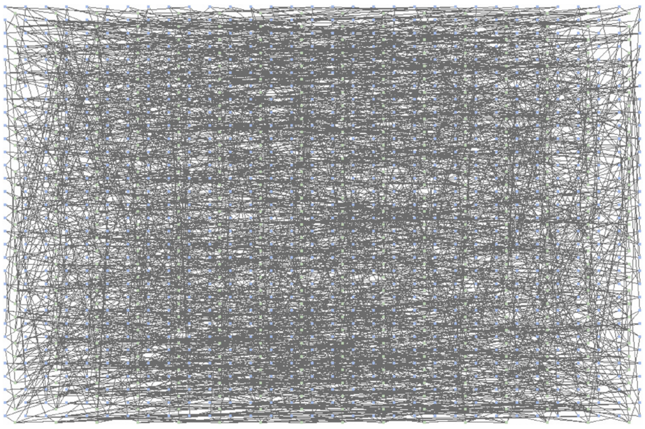
Fig. 9. Interconnection complexity for the proposed window-constrained PEG-9 code.

reduction is achieved by imposing more initial constraints. Finally, as an illustration of the interconnect complexity, Figs. 8 and 9 show a comparison between the interconnection density for the unconstrained standard PEG code and the proposed PEG-9 code. These two graphs show the nodes' interconnections after placement (before routing), and it is quite evident that the proposed PEG-9 construction has less interconnections density than the standard PEG code.

5. Conclusion

The paper presented a new technique to address the problem of interconnection complexity reduction in fully-parallel LDPC decoder implementation. Using the progressive edge growth LDPC construction, the code variable and check nodes are initially grouped into cells that are placed in a two-dimensional grid, where connections between nodes are confined to a given window, thereby imposing a constraint on the maximum inter-node span. The average connection length for the proposed designs was shown to be reduced down to 25% of that of standard PEG codes. Error performance simulation results of the novel window-constrained LDPC codes showed that the error correction capability decreases as the minimum Manhattan inter-node distance decreases. The proposed algorithm was further improved in terms of interconnection efficiency by imposing additional short-length initial connections. The error correction performance and hardware efficiency were investigated with one, two and three initial connections, and it has been found that interconnect complexity decreases remarkably by imposing simple initial connection constraints, with a negligible penalty in error correction performance. It is therefore concluded that, in addition to the maximum allowed connection length, the number of initial connections can be used as a design parameter to trade off efficiency of hardware implementation and error correction performance.

Acknowledgment

The authors would like to thank King Fahd University of Petroleum & Minerals for supporting this work under Project DSR-IN070376.

References

[1] Gallager RG. Low density parity-check codes. Cambridge, MA: MIT Press; 1963.
[2] MacKay DJ. Good error-correcting codes based on very sparse matrices. IEEE Trans Inf Theory 1999;45:399–431.
[3] Richardson T, Shokrollahi A, Urbanke R. Design of capacity approaching irregular low-density parity-check codes. IEEE Trans Inf Theory 2001;47: 619–37.
[4] Kschischang F, Frey B, Loeliger H. Factor graphs and the sum-product algorithm. IEEE Trans Inf Theory 2001;47:498–519.
[5] Chen J, Dholakia A, Eleftheriou E, Fossorier M, Hu X-Y. Reduced complexity decoding of LDPC codes. IEEE Trans Comm 2005;53:1288–99.
[6] Ohtsuki T. LDPC codes in communication and broadcasting. IEICE Trans Comm 2007;E90-B(3):440–53.
[7] Yeo E, Nikolic B, Anantharam V. Architectures and implementations of low-density parity check decoding algorithms. In: Proc. 45th Midwest Symposium on Circuits and Systems, vol. 3. 2002. p. 437–40.
[8] Howland CJ, Blanksby AJ. Parallel decoding architectures for low density parity check codes. In: Proc. IEEE Intl. Symp. Circuits and Systems. 2001. p. 742–5.
[9] Mansour MM, Shanbhag NR. High-throughput LDPC decoders. IEEE Trans VLSI Syst 2003;11:976–96.
[10] Mohsenin T, Baas BM. Trends and challenges in LDPC hardware decoders. In: Proc. 43rd Asilomar Conf. Signals, Systems and Computers. 2009. p. 1273–7.
[11] Roth C, Cevrero A, Studer C, Leblebici Y, Burg A. Area, throughput, and energy-efficiency trade-offs in the VLSI implementation of LDPC decoders. In: Proc. 2011 IEEE Int. Symp. Circuits and Systems (ISCAS). 2011. p. 1772–5.
[12] Hu X-Y, Eleftheriou E, Arnold DM. Progressive edge-growth Tanner graphs. In: Proc. IEEE GLOBECOM. 2001. p. 995–1001.
[13] Chen J, Zhao Z, Bao J. A modified construction method of low-density parity-check codes based on PEGPC. In: Proc. Intl. Conf. Comp. Inf. Sc. (ICCIS). 2011. p. 793–6.
[14] Healy CT, de Lamare RC. Decoder-optimised progressive edge growth algorithms for the design of LDPC codes with low error floors. IEEE Comm Lett 2012;16(6):889–92.
[15] Mohsenin T, Truong DN, Baas BM. A low-complexity message-passing algorithm for reduced routing congestion in LDPC decoders. IEEE Trans Circuits Syst 2010;57:1048–61.
[16] Mohiyuddin M, Prakash A, Aziz A, Wolf W. Synthesizing interconnect efficient low density parity check codes. In: Proc. Design Automation Conf. 2004. p. 488–91.
[17] El-Maleh A, Arkasosy B, Landolsi MA. Interconnect-efficient LDPC code design. In: Proc. 18th Int. Conf. Microlelectronics. 2006. p. 127–30.
[18] Darabiha A, Carusone A, Kschischang FR. A bit-serial approximate minsum LDPC decoder and FPGA implementation. In: Proc. IEEE Int. Symp. Circuits and Systems. 2006.
[19] Darabiha A, Carusone A, Kschischang F. Block-interlaced LDPC decoders with reduced interconnect complexity. IEEE Trans Circuits Syst 2008;51: 74–8.
[20] Yong N, Zhenyu X, Depeng J, Li S, Lieguang Z. Reduced-routing complexity decoder for high-rate QC-LDPC codes. In: Proc. Intl. Conf. Comp. Problem-Solving. 2011. p. 703–7.
[21] Kai Z, Xinming H, Zhongfeng W. An area-efficient LDPC decoder architecture and implementation for CMMB systems. In: Proc. 20th IEEE Intl. Conf. on Application-Specific Syst., Arch. & Proc. (ASAP). 2009. p. 235–8.

[22] AlGhoneim E, ElMaleh A, Landolsi MA. New techniques for improving the performance of LDPC codes in the presence of trapping sets. EURASIP J Wireless Comm Netw 2008 (Article ID 362897). doi:10.1155/2008/362897.
[23] Lin S, Costello Jr D. Error control coding. 2nd ed. NJ: Pearson Prentice Hall; 2004.
[24] Fossorier MPC. Quasi-cyclic low-density parity-check codes from circulant permutation matrices. IEEE Trans Inf Theory 2004;50(8):1788–93.
[25] Xiao H, Banihashemi AH. Improved progressive-edge-growth (PEG) construction of irregular LDPC codes. IEEE Comm Lett 2004;8(12):715–7.
[26] Lin Y-K, Chen C-L, Liao Y-C, Chang H-C. Structured LDPC codes with low error floor based on PEG Tanner graphs. In: Proc. IEEE ISCAS'08. 2008. p. 1846–9.
[27] Uchoa A, Healy C, de Lamare R, Souza R. Design of LDPC codes based on progressive edge growth techniques for block fading channels. IEEE Comm Lett 2011;15(11):1221–3.
[28] Fan Z, Zhang W, Liu X. An improved construction method of QC-LDPC codes based on the PEG algorithm. In: Proc. 3rd Pacific Asia Conf. on Comm. Circ. & Syst. (PACCS). 2011. p. 1–4, doi:10.1109/PACCS. 2011.599028.