# *Interconnect-Efficient LDPC Code Design*

*Aiman El-Maleh, Basil Arkasosy, M. Adnan Al-Andalusi*

King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia
aimane@ccse.kfupm.edu.sa, basil_arkasosy@hotmail.com, andalusi@kfupm.edu.sa

***Abstract*—In this paper, we present a new, hardware-oriented technique for designing Low Density Parity Check (LDPC) codes. The technique targets to achieve an interconnect-efficient architecture that reduces the area and delay of the decoder implementation while maintaining good error correction performance. With a fully parallel implementation of the LDPC decoder, the proposed design assumes a constraint on the interconnect wire length which has a direct impact on the maximum signal delay and power dissipation. Furthermore, this design approach is shown to lower interconnect routing congestion, and hence reduce the chip area and maximize chip utilization.**

## I. INTRODUCTION

Forward Error Correcting (FEC) codes are an essential component of modern state-of-the-art digital communication and storage systems. In many of the recently developed standards, FEC codes play a crucial role for improving the error performance capability of digital transmission over noisy interference-impaired communication channels.

The two leading families of FEC codes are widely considered to be Turbo Codes [1] and Low Density Parity Check codes (LDPCs) [2]. Both families demonstrate performance very close to the information-theoretic bounds predicted by Shannon theory, while at the same time having the distinct advantage of efficient, near-optimal iterative decoding.

Recently, the trend in error correcting code implementation is focused on using custom chips for codecs. This is motivated by the need to maximize performance and increase codec throughput. This is particularly true for coding schemes which can use the parallelism offered by hardware: by using more devices, frequency can be reduced, and voltage scaled, thereby achieving the same throughput for less power [3]. In this regard, the sum-product iterative algorithm used in LDPC decoding is found to be amenable to high parallelization, thus offering a clear advantage over turbo decoders that use more sequential processing [4], and hence, don't fully benefit from hardware implementations. Recently, practical implementations of LDPC codecs with high throughput levels were reported [4, 5].

Most of LDPC code designs are based on random code generation which may result in long interconnect wires. Long interconnect wires increase the load capacitance resulting in slower designs and higher power dissipation. In addition, random code design results in interconnect routing congestion which reduces the chip area utilization. In [5], based on a parallel implementation of LDPC codes, it is reported that routing congestion has resulted in 50% chip area utilization and that most of the power dissipation is due to the switching activity of the long interconnect wires. In [6], an algorithm is proposed to generate LDPC codes with a bound on the 'height' of the VLSI layout of the decoder. It is shown that this approach limits the routing congestion with little impact on error correcting capability of the code.

In this paper, we present a new technique to design LDPC codes that are interconnect-efficient, which translates into substantial reduction in chip area and decoding latency. These advantages are also achieved while maintaining good error correction performance by constraining the girth (minimum loop size) of the codes' bipartite graph, as will be discussed in the subsequent sections.

## II. LDPC CODES

LDPC codes have been recently re-discovered and are undergoing very active research after their original introduction by Gallager in 1962 [3]. A tutorial review of the work of Gallager that highlights the powerful, capacity-achieving performance of LDPC codes can be found in [7]. LDPC codes are a type of linear block codes, and can be decoded by an efficient, linear-complexity decoding algorithm known as the belief propagation (or sum-product) algorithm, and shown to achieve near optimal performance approaching that of maximum likelihood decoding (when the code is cycle-free, as will be described next) [7].

An LDPC parity check matrix $\boldsymbol{H}$ represents the parity equations in a linear form, where any given codeword $\underline{\boldsymbol{c}}$ satisfies the set of parity equations $\boldsymbol{H}\,\underline{\boldsymbol{c}} = 0$. Each column in the matrix represents a codeword bit while each row represents a parity check equation. The 1's in this matrix indicate a relationship between the column and the row that contains this 1. For example, if a 1 exists in the ith row and the jth column, then the jth codeword bit is contained in the

ith parity check equation. LDPC codes can be classified as *regular* or *irregular*. The associated parity check matrix is said to be ($W_c$,$W_r$)-*regular Parity Check* when each parity check code contains the same number of codeword bits, $W_r$, and each codeword bit is contained in the same number of parity check equations, $W_c$.

LDPC codes can also be graphically represented by a special type of graphs called *Tanner Graphs.* A Tanner Graph is a bipartite graph that contains two types of vertices (or nodes): *Bit Vertices* and *Check Vertices,* with edges connecting them. Rows in the LDPC parity check matrix are represented by Check Vertices and columns are represented by Bit Vertices.

In the parity check matrix, the presence of 1's indicates the existence of an edge (or connection) between a given Bit Vertex and Check Vertex. Another parameter of interest is the *code rate* defined as the ratio of the number of information bits to the total number of bits in the codeword. This is also given by the ratio of the number of check nodes to the number of bit nodes in the code's Tanner graph.

## III. PROPOSED INTERCONNECT-EFFICIENT LDPC CODES DESIGN

It is known that randomly designed LDPC codes achieve good error correction performance. In this paper, different LDPC code constructions are investigated where some are purely random, while others are designed with a specified degree of randomness that is confined to a limited area of parity matrix (or code graph). The different scenarios that have been used to design these codes are described in the following sections.

### *A. Cycles-Free LDPC Codes*

A cycle or a loop of size *K* in a given graph is a closed path that is composed of *K* edges that visit a given vertex more than once, while visiting each edge in this path only once. Since we are dealing with a *Bipartite Graph* with bit nodes connecting only to check nodes, the possible loops are of even sizes, starting by four.

Previous studies have demonstrated that the existence of loops in LDPC codes decreases their efficiency [7]. Thus, we have considered the design of LDPC codes that are free from loops (or of reduced loops count) of different sizes and tested the performance of these codes.

Our objective in the first set of experiments was to design (3,6)-regular LDPC codes that are four-loops free. The rate of the graph was 1/2 of 1024 bit nodes. Connections between bit nodes and check nodes are made randomly as long as no 4-loops are created. The algorithm used to detect if a four loop will result after making a connection is given in Fig. 1. Similarly, in the second set of experiments we considered the design of (3,6)-regular LDPC codes that are six-loops free. The algorithm used to detect if a six loop will result after making a connection is given in Fig. 2. The third set of experiments was to design (3,6)-regular LDPC codes that are eight-loops free. We found that it is not possible to generate a (3,6)-regular eight-loops free LDPC codes with 1024 variable nodes, and a rate of 1/2. This is consistent with the results reported in [8]. Thus, we considered the design of (3,6)-regular LDPC codes that are four- and six-loops free and have minimal eight loops. The algorithm used for detecting an eight loop is similar to the six-loop detection algorithm with two more steps.

Checking if an edge between the **i**th bit node and the **j**th check node creates a **four loop**:

1. Find the set of all bit nodes **K** to which the **j**th check node is connected.
2. For *each* bit node k in **K (**$k \neq$ **i)**, find all the check nodes **L** that are connected to *that* node.
3. For *each* check node l in **L (**$l \neq$ **j)**, find all the bit nodes **M** that are connected to *that* node.
4. If node **i** is in **M**, then a four loop is detected and the edge should be removed.

Fig. 1 Four-loop detection algorithm.

Checking if an edge between the **i**th bit node and the **j**th check node creates a **six loop**:

1. Find the set of all bit nodes **K** to which the **j**th check node is connected.
2. For *each* bit node k in **K (**$k \neq$ **i)**, find all the check nodes **L** that are connected to that node.
3. For *each* check node l in **L (**$l \neq$ **j)**, find all the bit nodes **M** that are connected to that node.
4. For *each* bit node m in **M (**$m \notin \{$**i** , k$\}$), find all the check nodes **N** that are connected to that node.
5. For *each* check node n in **N (**$n \notin \{$**j** , l$\}$), find all the bit nodes **O** that are connected to that node.
6. If node **i** is in **O**, then a six loop is detected and the edge should be removed.

Fig. 2 Six-loop detection algorithm.

### *B. Area-Constrained LDPC Codes*

The design of LDPC codes that are free from loops of different sizes showed good error correcting performance results. However, due to the random connections between nodes, this might lead to an inefficient design, in terms of the area used and the delay.

As stated earlier, our objective in this work is to design LDPC codes that result in interconnect efficient decoder implementations while maintaining a good performance. This is achieved by the design of LDPC codes with a constraint on the interconnect wire length. The approach that we used in this work is to assume that bit nodes and check nodes are laid out in a two-dimensional structure as shown in Fig. 3. Then, with a given requirement on the degree of bit nodes, connections between bit nodes and check nodes will be constrained within a given window of a specified number of rows and columns. This guarantees that the wire length interconnecting a bit node with check nodes is bounded by a

maximum length. For example, as shown in the previous figure for a ½ rate 32-bit code, bit nodes 18 and 19 with the given constraint window of two rows and three columns can only connect to check nodes 4, 5, 6, 8, 9, and 10 bounded by the specified window. In addition, the code will be designed taking into account the girth of the graph i.e., it will be ensured that the designed code has no cycles of a predetermined length e.g. 4. Connections between bit nodes and check nodes will be done randomly as long as they do not violate the specified constraints. We discuss next how to determine the check nodes bounded by a window constraint for each bit node.

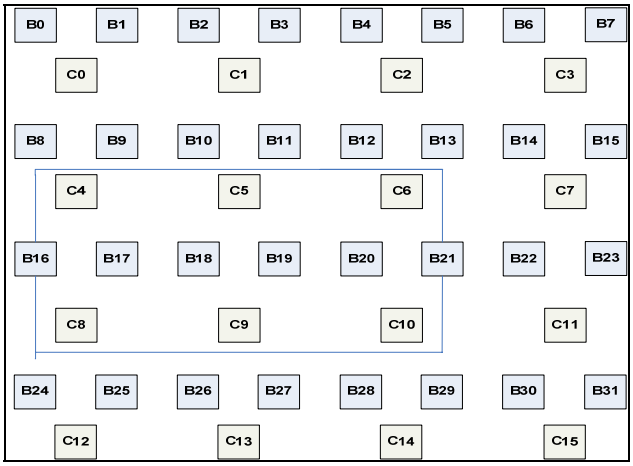


Fig. 3 Layout representation of a Tanner graph.

Considering the assumed two-dimensional layout of bit nodes and check nodes, each check node has two logical bit nodes associated with it. In other words, we can consider that for each bit node, there is an assigned check node, and every two bit nodes have a common assigned check node.

Given a layout of N x M bit nodes, N x M/2 check nodes and a window constraint (R, C), we need to determine the set of possible check nodes that the **i**th bit node can connect to. It should be observed that R is the number of rows of the given constraint window and should be an even number including the row of the assigned check node. Furthermore, C is the number of columns of the given constraint window and should be an odd number, including the column of the assigned check node.

The set of check nodes to which the **i**th bit node can connect to is determined by the algorithm given in Fig. 4. A similar algorithm can be used to determine the set of bit nodes to which the **j**th check node can connect to. In this work, we consider the design of LDPC codes with 32x32 layout of bit nodes and 32x16 layout of check nodes. A constraint window (R, C) =(16, 15) is assumed..

Limiting the area of connections to these dimensions will guarantee that the wire length does not exceed the length of eight rows and eight columns of check nodes.

Row#= $\left\lfloor \frac{i}{M} \right\rfloor$; Column #= **i** *mod* M; Vertical Domain= $\frac{R}{2}$

Assigned check node = $\left\lfloor \frac{Column\# + Row\#* M}{2} \right\rfloor$; **t** = 1;

for each **t** $\leq$ Vertical Domain
{
if (assigned check node + (**t-1**) * (M/2)) < number of check nodes Then
assigned check node = assigned check node + ((**t** -1)* (M/2))
Add_Horizontal(assigned check node , Row# + (**t-1**) )
if (assigned check node – **t** * (M/2)) $\geq$ 0 Then
assigned check node = assigned check node – (**t** * (M/2))
Add_Horizontal(assigned check node , Row# - **t** )
**t = t + 1**;
}

Add_Horizontal (assigned check node , Row#)
{
Horizontal Domain = $\frac{C-1}{2}$; add the assigned check node; **k** = 1;
for each **k** $\leq$ Horizontal Domain {
if (assigned check node + **k**) < ((M/2) * (Row# +1)) Then
add the check node whose number is "assigned check node + **k**"
if (assigned check node - **k**) $\geq$ ((M/2) * Row#) Then
add the check node whose number is "assigned check node - **k**"
**k = k + 1**;
}
}

Fig. 4 Set of check nodes for a given bit node**.**

## IV. EXPERIMETAL RESULTS

We have generated randomly a set of five LDPC codes for each criteria considered i.e. 4-loop free (4L), 6-loop free (6L), minimized 8-loop (M8L), and window constrained minimized 8-loop (WM8L). Then, we have selected the best performing code in each category for comparison purposes. Our comparison is based on the frame error rate (FER), the loop count, the area, delay and routing congestion for the implemented LDPC decoders. FER performance was obtained using matlab simulations at different SNR points with stopping criteria of 200 frame errors at SNR≤2 and 200,000 code words for SNR>2. Iterative decoding was performed for 64 iterations.

Hardware comparisons of the different LDPC codes were made based on a generic VHDL model that models a parallel implementation of the LDPC decoder for any given H matrix. The functions of the check nodes and variables nodes were assumed to be just a dummy single gate to simplify the design as the emphasis here is on interconnect complexity. Synthesis is performed using Xilinx synthesis tools mapping the design on Xilinx Spartan3 XC3S5000-fg900 FPGA optimized for area. Fig. 5 shows the FER of the four compared LDPC codes and Table 1 summarizes the loop analysis and synthesis results. As can be seen, the FER performance of the four LDPC codes is very close with a

noticeable difference at SNR=2.75. Apparently, the M8L LDPC code is achieving the best performance and the WM8L has a lower performance than M8L by around 0.1 dB. Thus, the performance impact of the window constrained LDPC code, WM8L, is considered small. As shown in Table 1, the WM8L LDPC code achieves the smallest delay and requires less FPGA slices, demonstrating the impact of constraining the interconnect wire length in the LDPC decoder. To further illustrate the impact of the window constrained LDPC code design, we show in Fig. 6 & 7 snapshots of the post place and route synthesis of the LDPC decoders for M8L and WM8L, respectively. It is noticeable from the figures that the routing congestion for WM8L LDPC decoder is less than that for M8L LDPC decoder.

TABLE 1 Loop count and synthesis results.

| LDPC Code Structure | No. of loops | | | Synthesis | |
|---|---|---|---|---|---|
| | **4** | **6** | **8** | Slices used out of 33,280 | Delay **ns** |
| 4L | 0 | 172 | 1273 | 2,562 | 14.191 |
| 6L | 0 | 0 | 1300 | 2,562 | 14.336 |
| M8L | 0 | 0 | 226 | 2,562 | 13.096 |
| WM8L | 0 | 0 | 870 | 2,527 | 11.879 |

## V. CONCLUSIONS

In this work, we have investigated the design of interconnect-efficient LDPC codes that reduce the area and delay of the decoder implementation while maintaining good error correction performance. We have demonstrated that it is possible to deign LDPC codes that are interconnect efficient with small performance impact compared to the randomly unconstrained generated LDPC codes.

## ACKNOWLEDGMENT

The authors gratefully acknowledge the support of King Fahd University of Petroleum & Minerals in this work.

## REFERENCES

[1] M. Bossert. *Channel Coding for Telecommunications.* John Wiley and Sons, Aug. 1999.
[2] R. G. Gallager, *Low Density Parity Check Codes*, Cambridge, MA: MIT Press, 1963
[3] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-Power CMOS Digital Design," *IEEE J. Solid-State Circuits*, pp. 473-484, Apr. 1992.
[4] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, "VLSI Architectures for Iterative Decoders in Magnetic Recording Channels," *IEEE Trans. Magnetics*, pp. 748-755, Jan. 2001.
[5] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, Rate-1/2 Low-Density Parity-Check Decoder," *IEEE J. Solid-State Circuits*, pp. 404-412, Mar. 2002.
[6] M. Mohiyuddin, A. Prakash, A. Aziz and W. Wolf, "Synthesizing Interconnect Efficient Low Density Parity Check Codes," Design Automation Conf., pp. 488-491, 2004.
[7] D.J.C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp.399-431, March 1999.
[8] J. A. McGowan and R. C. Williamson, "Loop removal from LDPC codes", Proc. IEEE Information Theory Workshop, pp. 230-233, 2003.

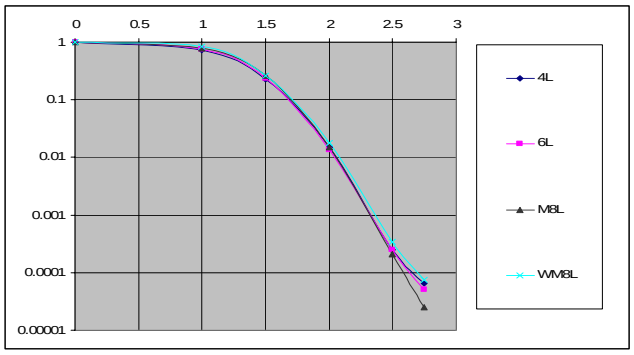


Fig. 5 FER Comparison of LDPC codes.

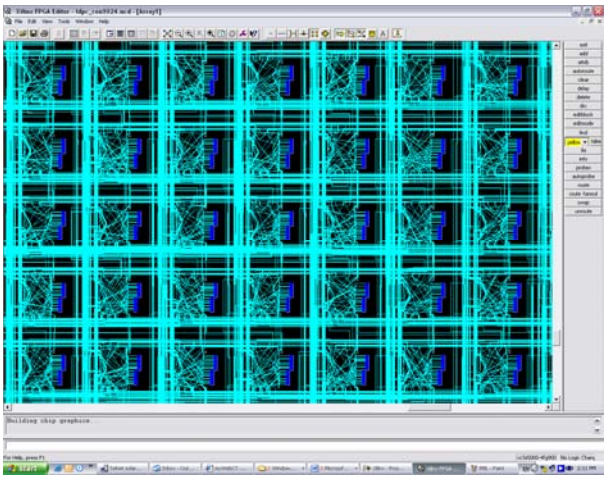

Fig. 6 Post place and route snapshot of synthesized M8L LDPC code.

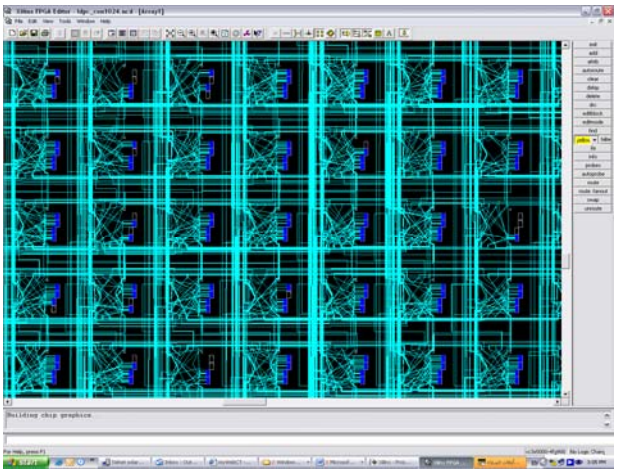

Fig. 7 Post place and route snapshot of synthesized WM8L LDPC code.