Block-Interlaced LDPC Decoders With Reduced Interconnect Complexity

Ahmad Darabiha, *Student Member, IEEE*, Anthony Chan Carusone, *Member, IEEE*, and Frank R. Kschischang, *Fellow, IEEE*

***Abstract*—Two design techniques are proposed for high-throughput low-density parity-check (LDPC) decoders. A *broadcasting* technique mitigates routing congestion by reducing the total global wirelength. An interlacing technique increases the decoder throughput by processing two consecutive frames simultaneously. The brief discusses how these techniques can be used for both fully parallel and partially parallel LDPC decoders. For fully parallel decoders with code lengths in the range of a few thousand bits, the half-broadcasting technique reduces the total global wirelength by about 26% without any hardware overhead. The block interlacing scheme is applied to the design of two fully parallel decoders, increasing the throughput by 60% and 71% at the cost of 5.5% and 9.5% gate count overhead, respectively.**

***Index Terms*—10-GB Ethernet, channel coding, decoder architectures, iterative message-passing, low-density parity-check (LDPC) codes, very-large-scale integration (VLSI).**

I. Introduction

THIS brief investigates VLSI architectures and design methodologies for multi-Gbit/s low-density parity-check (LDPC) decoders. LDPC codes were originally proposed by Gallager [1]. They have recently attracted much attention due to their high bit-error rate (BER) performance and unique potential for very high-throughput and low-latency decoding. As a result, they have been recently adopted for the 10GBase-T [2], the DVB-S2 [3], and the Mobile WiMAX [4] standards.

Despite all their desirable properties, LDPC codes' random parity-check matrices impose serious implementation challenges for both parallel and partially parallel LDPC decoders [5], [6]. In this brief, we describe two techniques to reduce the interconnect complexity and increase the throughput of LDPC decoders. These techniques are applicable to any LDPC code and impose no degradation on error-correction performance. The two techniques can be used individually or together. The remainder of this brief is organized as follows. Section II gives a brief introduction to LDPC codes and prior work on implementation of iterative LDPC decoders. Section III describes half- and full-broadcasting and compares them in terms of reducing the complexity of node-to-node communications and hardware overhead. Section IV then explains block interlacing, showing the results for two implemented block-interlaced LDPC decoders.

Manuscript received January 2, 2007; revised April 27, 2007. This paper was recommended by Associate Edtior G. M. Maggio.

The authors are with The Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto, Toronto M5S 3G4, Canada (e-mail: ahmadd@eecg.utoronto.ca; tcc@eecg.utoronto.ca; frank@comm.utoronto.ca).

Digital Object Identifier 10.1109/TCSII.2007.905328

II. Background

A binary LDPC code, C, can be described as the null space of a sparse $M \times N$ $\{0,1\}$–valued parity-check matrix, H. It can also be described by a bipartite graph, or *Tanner graph*, in which check nodes $\{c_1, c_2, \ldots, c_M\}$ represent the rows of H and variable nodes $\{v_1, v_2, \ldots, v_N\}$ represent the columns. An edge connects the check node c_m to the variable node v_n if and only if H_{mn} is nonzero.

Suppose that a binary codeword $W = (w_1, w_2, \ldots, w_N) \in C$ is transmitted over a communication channel and that $Y = (y_1, y_2, \ldots, y_N)$ is received. Let $z_{mn}^{(i)}$ and $q_{mn}^{(i)}$ represent the message sent from v_n to c_m and from c_m to v_n in the ith iteration, respectively. Let $N(m) = \{n : H_{mn} = 1\}$ and $M(n) = \{m : H_{mn} = 1\}$. Let I_M denote the maximum number of iterations. The sum–product decoding algorithm [7] consists of the following steps.

1) Initialize the iteration counter, i, to 1.

2) Initialize $z_{mn}^{(0)}$ to the *a posteriori* log–likelihood ratios (LLR), $\lambda_n = \log\big(P(v_n = 0|y_n)/P(v_n = 1|y_n)\big)$ for $1 \le n \le N$, $m \in M(n)$.

3) Update the check nodes, i.e., for $1 \le m \le M$, $n \in N(m)$, compute

$$q_{mn}^{(i)} = 2\tanh^{-1}\left(\prod_{n' \in N(m)\backslash n} \tanh\left(\frac{z_{mn'}^{(i-1)}}{2}\right)\right). \quad (1)$$

4) Update the variable nodes, i.e., for $1 \le n \le N$, $m \in M(n)$, compute

$$z_{mn}^{(i)} = \lambda_n + \sum_{m' \in M(n)\backslash m} q_{m'n}^{(i)}. \quad (2)$$

5) Make a hard decision, i.e., compute $\hat{W} = (\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_N)$, where element $\hat{w}_n$ is calculated as

$$\hat{w}_n = \begin{cases} 0, & \text{if } \lambda_n + \sum_{m' \in M(n)} q_{m'n}^{(i)} \ge 0 \\ 1, & \text{otherwise} \end{cases}$$

for $1 \le n \le N$. If $\hat{W}H^T = 0$ or $i \ge I_M$ output $\hat{W}$ as the decoder output and halt; otherwise set $i = i + 1$ and go to step 3.

In partially parallel LDPC decoders a limited number of check and variable processing units are used multiple times per iteration to compute extrinsic messages for a group of check

and variable nodes. A common challenge in partially parallel architectures is to manage the large number of memory accesses and prevent memory collisions since multiple messages must be accessed simultaneously by the check and variable nodes. Examples of partially parallel decoders include [8] and [6].

In fully parallel decoders, each node in the code's Tanner graph is assigned a dedicated hardware processing unit and messages are communicated between nodes by wires. The drawbacks of a fully parallel architecture are the large circuit area required to accommodate all the processing units, as well as a complex and congested global interconnect network. Routing congestion leads to longer interconnect delays and can degrade decoder timing performance and dynamic power dissipation. Examples of fully parallel decoders include [5], [9] .

III. Broadcasting

This section introduces a technique that reduces the node-to-node communication complexity in LDPC decoding. We will show how this technique can be applied to both fully parallel and partially parallel decoder architectures. We start by re-writing (1) and (2) as follows. Let

$$q_{mn}^{(i)} = 2\tanh^{-1}\left(\frac{P_m^{(i)}}{\tanh\left(\frac{z_{mn}^{(i-1)}}{2}\right)}\right) \tag{3}$$

where

$$P_m^{(i)} = \prod_{n' \in N(m)} \tanh\left(\frac{z_{mn'}^{(i-1)}}{2}\right). \tag{4}$$

Let

$$z_{mn}^{(i)} = S_n^{(i)} - q_{mn}^{(i)} \tag{5}$$

where

$$S_n^{(i)} = \lambda_n + \sum_{m' \in M(n)} q_{m'n}^{(i)}. \tag{6}$$

Fig. 1 shows the block diagram of a check and variable processing unit using (3)–(6). Symbols ⊟ and ⊞ denote the operations that are performed in (3) and (4), respectively. Similarly, symbols ⊘ and ⊙ denote operations performed in (5) and (6), respectively.

Half-broadcasting is a repartitioning of the computations in Fig. 1. In this new partitioning, shown in Fig. 2(a), the ⊟ functions are moved to the variable nodes without affecting the message-passing algorithm. This is because extrinsic messages, $q_{mn}^{(i)}$, are reconstructed in the variable nodes from the received $P_m^{(i)}$ and the $z_{mn'}^{(i-1)}$'s from iteration $i-1$. So, unlike the schemes in [5], [10], and [11] in which each degree-d_c check node generates d_c separate messages, one for each neighboring variable node, in this scheme each check node *broadcasts* a single message (i.e., $P_m^{(i)}$) to all of its neighbors. This approach reduces the amount of information that needs to be conveyed from check nodes to variable nodes. In a fully parallel decoder, this translates into a reduction in global interconnect. In a partially parallel decoder it translates into fewer memory accesses.

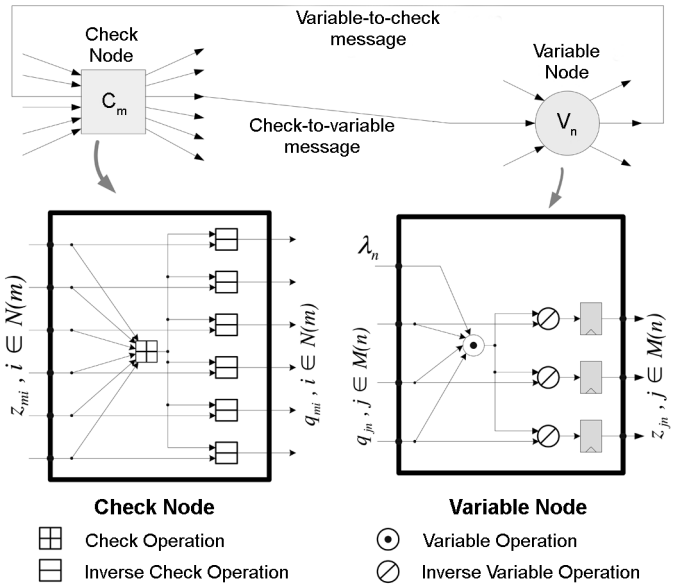


Fig. 1. Conventional fully parallel message-passing LDPC decoder with generic functions for check and variable nodes.

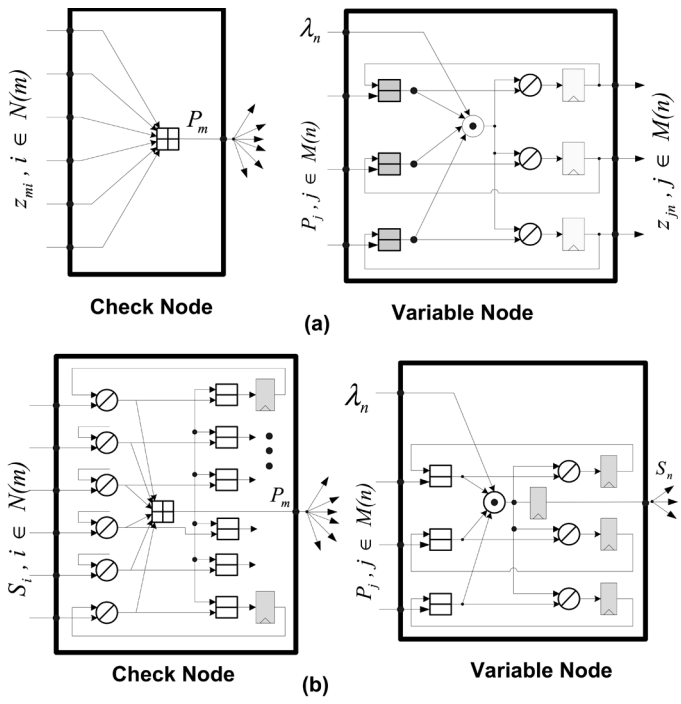
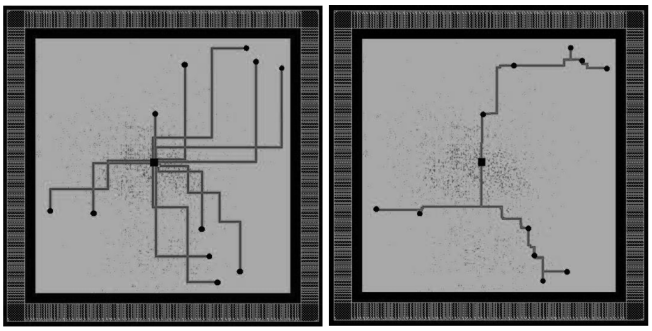


Fig. 2. (a) Half-broadcast architecture. The check node c_m broadcasts a single message, P_m, to all neighboring variable nodes. (b) Full-broadcast architecture. The check node c_m broadcasts P_m to the neighboring variable nodes. The variable node v_n broadcasts S_n.

Although the broadcasting technique above was described using sum–product algorithm, the same technique can be applied to other variations of message-passing decoding such as min-sum decoding and bit-flipping. For example, in the case of min-sum decoding the variable nodes would be designed to broadcast the total value, S_n, to their neighbors.



(a) (b)

Fig. 3. Routed nets for one check node output highlighted in a fully parallel LDPC decoder layout: (a) without broadcasting and (b) with broadcasting.

TABLE I
WIRELENGTH REDUCTION FOR GLOBAL NETS IN FULLY PARALLEL LDPC DECODERS

Code	P&R tool	Predicted
(992,829) RS-LDPC	-	27%
(2048,1723) RS-LDPC	26%	26%
(4096,3403) RS-LDPC	-	27%

Fig. 3 shows the effect of the broadcasting technique on a fully parallel decoder layout. These are real layouts obtained by automated P&R tools from Cadence using a floorplan similar to [5] where the check nodes are instantiated in the center and the variable nodes are instantiated on the sides of the decoder layout. One check node and its neighboring 11 variable nodes and the nets for conveying check-to-variable intrinsic messages between them are highlighted in the figure for clarity. Fig. 3(a) shows the case where no broadcasting is applied. Fig. 3(b) shows that by using the broadcasting scheme of Fig. 2(a) significant amount of interconnect wiring can be shared, hence mitigating the complexity of interconnections. As an example, in the 2048-bit decoder design presented in this brief the broadcasting scheme reduces the average node-to-node wirelength by 26% from 1.88 to 1.40 mm.

Table I lists the percentage wirelength reduction obtained from half-broadcasting compared with the conventional case where no broadcasting is applied. The technique is applied to LDPC codes with various lengths. The values in the first column are obtained from automated P&R tools. The values in the second column are obtained from a prediction algorithm which approximates a Steiner tree [12] solution to predict the wirelength savings by using the actual floorplan of the fully parallel decoder and the silicon area dedicated to each variable and check-update unit in the floorplan. Table I shows that for code lengths of few thousand bits half-broadcasting yields similar savings in global wirelength. For fully parallel decoders there is no hardware overhead associated with half-broadcasting; we are simply shifting some logic from one node to the other.

For partially parallel LDPC decoders, broadcasting reduces the number of shared memory write accesses. This is because in a conventional partially parallel decoder, each check-processing unit (CPU) reads the extrinsic messages, z_{ij}, generated in the previous iteration from the memory and writes the resulting q_{ij} messages to another shared memory to be read by variable-processing units (VPUs), and so on. Thus, the CPU for a check node with degree d_c, needs to perform d_c reads and d_c writes. By using a broadcast scheme for the check nodes, the CPU still needs to read d_c input values, but since the CPU generates only one output value, just one write operation is required. In total, the number of read/write memory accesses per CPU per iteration is reduced from $2d_c$ to d_c+1. Unlike fully parallel decoders, there is some hardware overhead for half-broadcasting in partially parallel decoders since the z_{mn}'s need to be also stored locally in the VPU for use in the next iteration. This additional local storage, however, does not add to the global node-to-node communication complexity.

We call the architecture of Fig. 2(a) *half-broadcasting* because we applied the broadcasting technique only to check-to-variable messages while the variable-to-check messages were kept unchanged. The same idea can be extended to a *full-broadcasting* scheme in which both check-to-variable and variable-to-check messages are broadcast. Fig. 2(b) shows the variable and check node architecture capable of full-broadcasting. In this figure, the inverse-check and inverse-variable operations $\boxminus$ and $\oslash$, are duplicated in order to be able to reconstruct the individual messages from the interim variable and check totals. Memory-based versions of full-broadcasting are proposed in [13] and [14]. As one can expect, full-broadcasting results in further simplification in interconnect complexity; however, this comes with a relatively large logic overhead. The exact amount of this overhead depends on the exact type of variable and check update functions, but since most of the calculations are duplicated the overhead can be as much as 2x [13].

Depending on the type of update functions, the designer may need to assign a larger word length for the broadcast values [e.g., $P_m^{(i)}$ in (4) and $S_n^{(i)}$ in (6) in the case of sum–product message-passing] compared with the word length needed for the actual extrinsic values [e.g., $q_{mn}^{(i)}$'s and $z_{mn}^{(i)}$'s in (3) and (5)]. In these cases, the effect of increased word length for broadcast messages must be taken into account. As an example, if λ_n and $q_{m'n}$'s in (6) are quantized with q bits, then $q+\lceil\log(d_v+1)\rceil$ bits would be required to represent S_n. However, since the word length of z_{mn}'s in (5) is generally limited to only q bits by clipping as in [15], S_n can be represented with only $q+1$ bits without loss of accuracy. So, the variable-to-check messages will have $0.5(1/q)\times 100\%$ additional wiring due to the increased word length in S_n.[1] For $q=6$, the overhead becomes 8%. A similar analysis can also be made for the check-to-variable broadcasting of Fig. 2(a) since the multiplications and divisions in (3) and (4) are usually transformed into summations and subtractions in the logarithm domain. One particular case is the hard-decision message passing decoding [16] in which the z_{mn}'s and q_{mn}'s in Fig. 1 are 1-bit messages, and $\boxplus$ and $\boxminus$ symbols both indicate exclusive-OR operations. As a result, the broadcast message, P_m, in Fig. 2 is also a 1-bit value, hence no word length increase is required.

To compare the effectiveness of different broadcasting schemes in a partially parallel LDPC decoder, we define the node-to-node communication complexity as the number of unique LLR messages being read/written from/to the shared memory per iteration. For an LDPC code with E edges in the graph, $2E$ global read operations are involved in each iteration: E reads in the check-update phase and E reads in

[1]The 0.5 coefficient is because half-broadcasting in this case only affects the variable-to-check messages and keeps check-to-variable messages unchanged.

the variable-update phase, independent of the type of broadcasting. The number of write operations, however, varies with the choice of broadcasting. In a conventional decoder, each variable node generates d_v unique messages, so $Nd_v = E$ write operations are needed in variable-update phase. Similarly, E write operations are needed for the check-update phase. In a half-broadcasting scheme in which the check nodes broadcast a single LLR message, the number of variable-update phase memory writes continues to be E, however, the number of check-update phase write operations is reduced to M. Finally, in a full-broadcast the number of required write operations for variable and check-update phase is reduced to N and M, respectively. To summarize, one decoding iteration in a no-broadcast scheme requires $2E + 2E = 4E$ read/write operations. With half-broadcasting this is reduced to $2E + E + M = (3 + 1/d_c)E$. With full-broadcasting this is further reduced to $2E + N + M = (2 + 1/d_v + 1/d_c)E$. For moderately large values of d_v and d_c, half-broadcasting and full-broadcasting result in close to 25% and 50% memory access reduction, respectively, compared to a no broadcasting scheme. As an example, for a (6–32)-regular (2048,1723) LDPC code in 10GBase-T standard ($E = 12288$) the number of global memory accesses per iteration is reduced by 24% and 45% using half-broadcasting and full-broadcasting, respectively.

The above comparisons suggest that in fully parallel decoders half-broadcasting provides a better trade-off between relaxing node-to-node communication complexity and logic overhead. Meanwhile, the full-broadcasting can be the preferred choice in low-parallelism decoders where logic area constitutes a small portion of the total decoder area and hence the logic overhead due to full-broadcasting can be tolerated.

IV. Block Interlacing

As explained in Section III, each iteration of message-passing decoding consists of updating check and variable node outputs based on (3) and (5). Due to the data dependency, computation of $z_{mn}^{(i)}$ in (5) cannot be started until all the $q_{m'n}^{(i)}, m' \in M(n)$ are calculated from (3). Similarly, the check-update phase cannot be started before completion of the variable update phase of the previous iteration. In [17] an overlapped message-passing scheduling is proposed for quasi-cyclic LDPC codes decoded in a memory-based architecture. The idea is to perform the check (variable) node update phase in an order such that the variable (check) node update phase can be started for some variable (check) nodes before all the check (variable) node updates are complete. A modified scheduling algorithm for overlapped message passing is proposed in [13] which can be applied to any LDPC code. Fig. 4(a) and (b) shows how overlapped message passing reduces the iteration delay compared with conventional scheduling with no overlapped message passing. The algorithm in [13] is based on permuting rows and columns of H so that the sub-matrices in the lower left and upper right corners of the H are all zeros. The problem with this algorithm is that as the number of parallel variable and check processing units increases, the potential increase in throughput is decreased. (In the case of a fully parallel architecture, the saving becomes zero).

The pipelined scheme in Fig. 4(c) doubles the decoder throughput compared with the throughput in Fig. 4(a). Instead of overlapping the variable-update phase and check-update phases of the same frame, we *interlace* the decoding of two

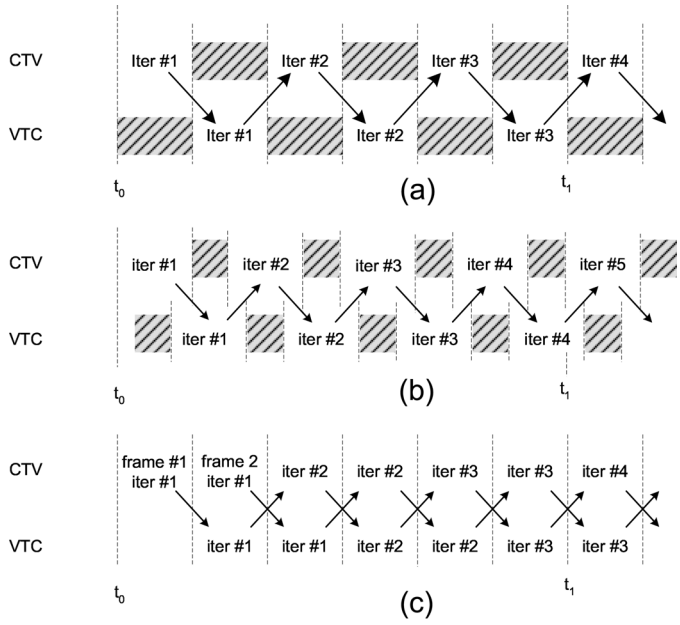


Fig. 4. Timing diagram for the message-passing algorithm. (a) Conventional. (b) Overlapped message passing [17]. (c) Block interlacing.

consecutive frames such that when variable (check) node update phase is being applied to one frame, another frame is in the check (variable) node update phase and vice versa.

From Fig. 4, it can be seen that from t_0 to t_1 the conventional scheme in Fig. 4(a) has finished three iterations and the overlapped message passing has completed 4 iterations (or it is ready to start fifth iteration) whereas in the same time period the block interlaced scheme in Fig. 4(c) has finished 3 iterations for frame 1 and 3 iterations for frame 2, i.e., 6 iterations in total. This improvement in decoder throughput results from eliminating the idle times in the timing diagram, or equivalently, increasing the hardware utilization of the decoder.

The block-interlacing is similar to the fully pipelined architecture in [10] where logic utilization is increased by having one set of VPUs and CPUs for each iteration. Here, a single set of VPUs and CPUs exchange information back and forth while performing the iterations. As a result the memory size does not increase with number of iterations. The block interlacing technique can be used both in fully parallel and partially parallel decoders and, unlike the overlapped message-passing technique of [13], the throughput improvement is independent of the H matrix. For fully parallel decoders, the only additional requirement are the pipeline registers at the node outputs. In these decoders, the gate-count overhead due to the pipeline registers is relatively small compared with the large logic gate count. As an example, for the two decoders presented below, the hardware overhead is less than 10%. This overhead is easily justified by the throughput improvement.

For partially parallel decoders, block interlacing keeps the CPU and VPU logic unchanged but requires doubling the size of the LLR memories so that the CPUs and VPUs can switch between two memory banks as they switch between the two different frames. In decoders with a small number of CPUs and VPUs (i.e., low parallelism) memory is the dominant portion of the decoder, so block interlacing will nearly double the power and area but, in high-parallelism decoders, the logic size becomes dominant and overhead is reduced.

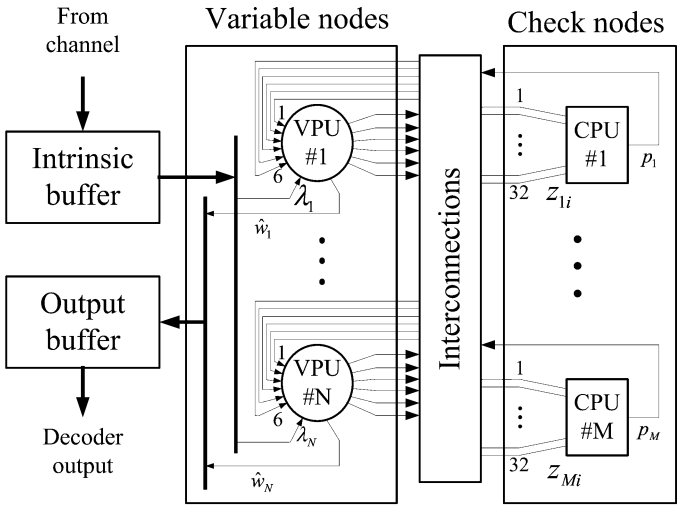


Fig. 5. Top level block diagram for the fully parallel (2048, 1723) LDPC decoder with half broadcasting. ($N = 2048$, $M = 384$).

TABLE II
SUMMARY OF LDPC DECODER CHARACTERISTICS

	design 1A	design 1B	design 2A	design 2B
CMOS process	0.18 μm	0.18 μm	90 nm	90 nm
Block interlacing	no	yes	no	yes
Code length (bits)	2,048	2,048	660	660
Iter. per frame	32	32	16	16
Cycles per iter.	1	2	1	2
quantization (bits)	1	1	4	4
Max freq. (MHz)	60	96	83	142
Core area (mm^2)	9.8	11.4	2.72	3.06
Gate count (k)	522	551	685	750
Throughput (Gbps)	3.84	6.14	3.42	5.86

To demonstrate block interlacing, we compare the characteristics of two implemented fully parallel LDPC decoders. Each of these two decoders was implemented twice: once without block interlacing and once with interlacing. The first decoder is a rate-0.84 (2048, 1723) RS-based (6,32)-regular LDPC decoder in a 0.18-μm CMOS-6M technology. The decoder performs 32 iterations of hard-decision message-passing decoding [16]. The top-level block diagram of this decoder is shown in Fig. 5. The second decoder uses a similar top-level diagram as in Fig. 5 but is based on a (660,480) (4,15)-regular LDPC code generated using the progressive edge growth algorithm [18] and performs 16 iterations of min-sum decoding per frame. Although the second code is shorter and has lower average node degree, the gate count in the second decoder is higher than the first decoder because of the need for computing and storing the 4-bit quantized LLR values. Table II summarizes post-layout simulation results for all decoders. For the 2048-bit decoder, block interlacing increases the total throughput by 60% at the cost of just 5.5% more gates. For the 660-bit decoder, throughput is increased by 71%, with a 9.5% gate count overhead. Notice that block interlacing has not exactly doubled the throughput as one would ideally expect from Fig. 4. The reason is that the pipeline registers do not exactly break the critical path into two identical paths. So, the maximum clock frequency is limited by the delay in the longer path.

V. CONCLUSION

We have presented two techniques for improving the throughput and hardware efficiency of LDPC decoders. We have described half- and full-broadcasting and shown that half-broadcasting provides a better trade-off between node-to-node communication complexity and logic overhead in fully parallel decoders. Half-broadcasting resulted in 26% global wirelength reduction in fully parallel decoders and 24% memory access reduction in partially parallel decoders. We have also described a block interlacing scheme that maximizes logic utilization and increases the decoder throughput compared with conventional schemes. This method requires no change in the structure of the code or the decoding algorithm. We have demonstrated this technique in two fully parallel LDPC decoder designs. Post-layout simulations show that the throughput was improved by 60% and 71% at the cost of only 5.5% and 9.5% more gates, respectively.

REFERENCES

[1] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT press, 1963.
[2] *World Wide Web*, IEEE 802.3an Task Force, Jun. 8, 2004 [Online]. Available: http://www.ieee802.org/3/an/index.html
[3] *Draft European Telecommunication Standards Institue*, EN 302 307 V1.1.1, Jun. 2004.
[4] *IEEE Standard for Local and Metropolitan Area Networks*, IEEE 802.16e Standard, Feb. 28, 2006 [Online]. Available: http://standards.ieee.org/getieee802/download/802.16e-2005.pdf
[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*, vol. 37, no. 3, pp. 404–412, Mar. 2002.
[6] M. M. Mansour and N. R. Shanbhag, "High-throughput LDPC decoders," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 6, pp. 976–996, Dec. 2003.
[7] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, "Factor graphs and the sum–product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.
[8] T. Zhang and K. K. Parhi, "Joint code and decoder design for implementation-oriented (3, k)-regular LDPC codes," in *Proc. IEEE Asilomar Conf.*, Nov. 2001, pp. 1232–1236.
[9] A. Darabiha, A. Chan Carusone, and F. R. Kschischang, "Multi-Gbit/sec low density parity check decoders with reduced interconnect complexity," in *Proc. ISCAS*, Kobe, Japan, May 2005, vol. 5, pp. 5194–5197.
[10] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, "High throughput low-density parity-check decoder architectures," in *Proc. IEEE GLOBECOM*, 2001, pp. 3019–3024.
[11] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, "Efficient implementation of the sum–product algorithm for decoding LDPC codes," in *Proc. IEEE GLOBECOM*, 2001.
[12] E. N. Gilbert and H. O. Pollak, "Steiner minimal trees," *SIAM J. Appl. Math.*, vol. 16, no. 1, pp. 1–29, Jan. 1968.
[13] S.-H. Kang and I.-C. Park, "Loosely coupled memory-based decoding architecture for low density parity check codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 5, pp. 1045–1056, May 2006.
[14] G. Masera, F. Quaglio, and F. Vacca, "Implementation of a flexible LDPC decoder," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 6, pp. 1057–7130, Jun. 2007.
[15] F. Zarkeshvari and A. Banihashemi, "On implementation of min-sum algorithm for decoding low-density parity-check (LDPC) codes," in *Proc. IEEE GLOBECOM*, 2002, pp. 1349–1353.
[16] T. R. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.
[17] Y. Chen and K. K. Parhi, "Overlapped message passing for quasi-cyclic low-density parity check codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 6, pp. 1106–1113, Jun. 2004.
[18] X.-Y. Hu, E. Eleftheriou, and D.-M. Arnold, "Progressive edge-growth tanner graphs," in *Proc. IEEE GLOBECOM*, 2001, vol. 2, pp. 995–1001.