A 690-mW 1-Gb/s 1024-b, Rate-1/2 Low-Density Parity-Check Code Decoder

Andrew J. Blanksby and Chris J. Howland

***Abstract*—A 1024-b, rate-1/2, soft decision low-density parity-check (LDPC) code decoder has been implemented that matches the coding gain of equivalent turbo codes. The decoder features a parallel architecture that supports a maximum throughput of 1 Gb/s while performing 64 decoder iterations. The parallel architecture enables rapid convergence in the decoding algorithm to be translated into low decoder switching activity resulting in a power dissipation of only 690 mW from a 1.5-V supply.**

***Index Terms*—CMOS digital integrated circuits, decoding, error correction coding, parallel architectures.**

I. Introduction

ERROR correcting codes are used to increase the bandwidth and power efficiency of communications systems [1]. The invention of turbo codes and turbo product codes has moved the limit of achievable coding gain much closer to the Shannon limit for low and high code rates,[1] respectively [2], [3]. However, the iterative nature of the decoding algorithms for turbo codes and turbo product codes present significant implementation challenges. Each pass through the data block to perform a decoder iteration requires the fetching, computation, and storage of large amounts of state information. Performing multiple iterations to achieve high coding gain reduces throughput and increases power dissipation [4].

The recent interest in iterative decoding algorithms has led to the rediscovery of low-density parity-check (LDPC) codes. LDPC codes were invented by Gallager in 1962 but were not pursued due to implementation complexity [5], [6]. It has been shown that LDPC codes are asymptotically superior to turbo codes with respect to coding gain [7]. The most powerful codes currently known are 1 million bit and 10 million bit, rate-1/2, LDPC codes, achieving a capacity which is only 0.13 dB and 0.04 dB, respectively, from the Shannon limit for a bit error rate (BER) probability of 10^{-6} [7], [8]. However, unlike turbo and turbo product codes, it is possible to decode LDPC codes using a block-parallel algorithm rather than a block-serial algorithm.

In this paper, a parallel architecture for decoding LDPC codes is presented that achieves excellent coding gain while delivering both extremely high throughput and very low power dissipation. It is the first published implementation of an LDPC code decoder. The parallel architecture is demonstrated through the implementation of a 1024-b, rate-1/2, soft decision LDPC decoder that achieves a coded throughput of 1 Gb/s. In Section II, LDPC codes and the message passing decoding algorithm are reviewed. In Section III, the design and performance of a 1024-b, rate-1/2 LDPC code are described. Section IV discusses two different families of LDPC decoder architecture and in Section V the detailed architecture of a parallel 1024-b, rate-1/2, soft decision LDPC decoder is presented. In Sections VI and VII, the implementation and measured performance of this decoder are discussed, respectively.

Manuscript received July 20, 2001; revised October 18, 2001.

The authors are with the High Speed Communications VLSI Systems Research Department, Agere Systems, Holmdel, NJ 07733 USA (e-mail: blanksby@agere.com).

Publisher Item Identifier S 0018-9200(02)01696-7.

[1]The rate of a code is defined as the ratio of the information bits to the sum of the information and parity bits. A low-rate code has a large redundancy overhead while a high-rate code has a small overhead.

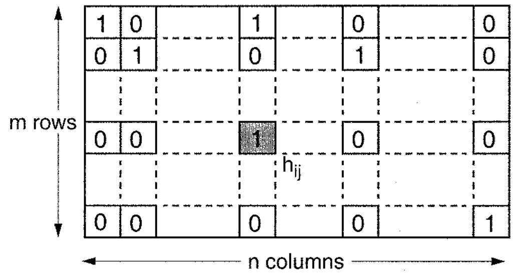

Fig. 1. General structure of an LDPC matrix $\boldsymbol{H}$.

II. Low-Density Parity Check Codes

A. Matrix Representation of LDPC Codes

LDPC codes are linear block codes and thus the set of all codewords, $x \in \boldsymbol{C}$, spans the null space of a parity check matrix $\boldsymbol{H}$

$$\boldsymbol{H}x^T = 0, \qquad \forall x \in \boldsymbol{C}. \tag{1}$$

The parity check matrix $\boldsymbol{H}$ for LDPC codes is a sparse binary matrix. The general structure of $\boldsymbol{H}$ is shown in Fig. 1. Each row of $\boldsymbol{H}$ corresponds to a parity check and a set element h_{ij} indicates that data symbol j participates in parity check i. In a block of n bits or symbols, there are m redundant parity symbols and the code rate r is given by

$$r = (n - m)/n. \tag{2}$$

The set row and column elements of $\boldsymbol{H}$ are chosen to satisfy a desired row and column weight profile, where the row and column weights are defined as the number of set elements in a given row and column, respectively [7]. In a *regular* LDPC code, all rows are of uniform weight, as are all columns. If the row and columns are not of uniform weight the LDPC code is said to be *irregular*.

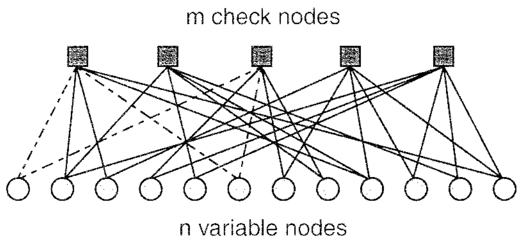

Fig. 2. Bipartite graph representation of an LDPC code. A short cycle of length 4 has been highlighted.

B. Graph Representation of LDPC Codes

LDPC codes can also be represented using a bipartite graph [9], where one set of nodes represents the parity check constraints and the other set represents the data symbols or variables as illustrated in Fig. 2. The parity check matrix is the incidence matrix of the graph where a variable node v_j, corresponding to column j in $\boldsymbol{H}$, is connected to check node c_i, corresponding to row i in $\boldsymbol{H}$, if the entry h_{ij} in $\boldsymbol{H}$ is set, i.e., non zero.

The algorithm used for decoding LDPC codes is known as the message passing algorithm. For good decoding performance with this algorithm, it is important that the length of *cycles* in the graph representation of the LDPC code are as long as possible [7], [9]–[11]. Short cycles, such as the length-4 cycle illustrated in Fig. 2, degrade the performance of the message passing algorithm.

C. The Message Passing Algorithm

The message passing algorithm is an iterative algorithm for decoding LDPC codes [5]. Both hard decision and soft decision forms of the algorithm have been developed. In hard decision decoding, each received symbol is thresholded to yield a single received bit as input to the decoding algorithm, and the messages passed between the variable and check nodes each consist of a single bit only. In soft decision decoding, multiple bits are used to represent each received symbol and the messages passed between the variable and check nodes. Soft decision decoding achieves substantially better coding performance because the confidence with which each decoder decision is made is forwarded to subsequent decoder iterations. The soft decision form of the message passing algorithm will now be described.

Soft Decision Decoding Algorithm (or Belief Propagation Algorithm) [5]–[7]:

1) Initialize all variable nodes and their outgoing variable messages to the value of the corresponding received bit represented as a log-likelihood ratio of the received symbol y defined as

$$\lambda = \ln\left[\frac{P(x=0|y)}{P(x=1|y)}\right]. \tag{3}$$

2) Propagate the variable messages from the variable nodes to the check nodes along the edges of the graph.
3) Perform a parity check (XOR) on the sign bits of the incoming variable messages at the check nodes to form the parity check result for the row. Form the sign of each outgoing check message for each edge of the graph as the XOR of the sign of the incoming variable message corresponding to each edge and the row parity check result.

In addition, compute an intermediate row parity reliability function defined as

$$\lambda_i = \prod_{j,\, h_{ij}=1} \tanh\left(\frac{\lambda_{ij}}{2}\right). \tag{4}$$

However, this computation and the subsequent calculation of the outgoing check message reliabilities are simplified by operating in the logarithmic domain where multiplications become additions and divisions become subtractions. Hence, (4) yields

$$\ln(\lambda_i) = \sum_{j,\, h_{ij}=1} \ln\left[\tanh\left(\frac{\lambda_{ij}}{2}\right)\right] \tag{5}$$

where $\ln(\lambda_i)$ is the log of the parity reliability for row i and λ_{ij} is the reliability of the message sent to check node i from variable node j.

The log of the intermediate row parity reliability function $\ln(\lambda_i)$ is then used to find all of the outgoing check message reliabilities according to

$$\begin{aligned}\lambda_{ik}^* &= 2\,\text{atanh}\left(\prod_{j,\, h_{ij}=1,\, j\neq k} \tanh\left(\frac{\lambda_{ij}}{2}\right)\right)\\ &= 2\,\text{atanh}\left\{\exp\left(\ln(\lambda_i) - \ln\left[\tanh\left(\frac{\lambda_{ik}}{2}\right)\right]\right)\right\}\end{aligned} \tag{6}$$

where λ_{ik}^* is the reliability of the check message from check node i to variable node k.

4) Pass the check messages from the check nodes back to the variable nodes along the edges of the graph.
5) At the variable nodes, update estimates of the decoded bit using a summation of the log-likelihood of the received bit and all of the check message log-likelihoods. The update can be considered as the check messages voting for the decoded bits value where the votes are weighted by their associated reliability. The decoded bit is taken to be the sign of the summation. Outgoing variable messages for the next decoding iteration are then formed using the same summation. However, each edge is updated without using the incoming message on the corresponding graph edge. This is easily implemented by subtracting the contribution of each edge from the group sum.
6) Repeat steps 2)–5) until a termination condition is met. Possible iteration termination conditions include the following.
    - The estimated decoded block $\hat{x}$ satisfies (1).
    - The current messages passed to the parity check nodes satisfy all of the parity checks. This does not guarantee that (1) is satisfied but is almost sufficient and is simple to test.
    - Stop decoding after a fixed number of iterations.

The message passing algorithm is optimal as long as the algorithm is propagating decisions from uncorrelated sources [5], [9]. This condition is true while the number of decoder iterations is less than half of the minimum cycle length of the graph representation of the LDPC code.

III. LDPC Code Design and Performance

A. LDPC Code Design

To explore the coding performance and implementation issues of LDPC codes, an LDPC code with a block size of 1024 b and a code rate of 1/2 was designed. This block size and code rate correspond to one of the Turbo codes proposed for third-generation (3G) wireless systems [12]. The 1024-b, rate-1/2, soft decision LDPC code was designed based on an irregular graph with an average column weight of 3.25 and an average row weight of 6.5. This weight profile was found through extensive simulation and yielded 3328 graph edges or set elements in $\boldsymbol{H}$. The column weights used were 3, 6, 7, and 8, and the row weights used were 256 weight 6 and 256 weight 7 rows. Based on the weight profile, a systematic code graph was constructed using an algorithm developed to maximize cycle lengths subject to the constraints of the block size and the column and row weight profile chosen. The effect of short cycles was further reduced by ensuring that they occur in distinct groups of received bits. This approach was used in preference to random permutation code construction techniques because the block size of 1024 b is relatively small [5], [7], [10].

Through software simulation, it was found that 64 decoder iterations were sufficient for good coding performance. It was also found that the decoder messages could be represented using only 4 b, one sign bit to represent the parity and three magnitude bits to represent the reliability, without significant coding performance penalty. The loss of coding gain with the fixed-point decoder performing 64 iterations is only 0.2 dB when compared to the decoder implemented using floating-point arithmetic and performing 1000 iterations.

B. LDPC Coding Performance Comparison

Although the coding gain of a rate-1/2 LDPC code with a block size of 1024 b is theoretically inferior to that of comparable turbo codes [7], implementation restrictions on the number of turbo code decoder iterations may not enable the realization of this difference. Most implementations of turbo codes to date have been of the lower complexity soft-output Viterbi algorithm (SOVA) performing only a small number of iterations [4]. In Fig. 3 the coding gain of the LPDC decoder exchanging 4-b messages is compared to the 1024-b, rate-1/2 turbo code in the 3G proposal decoding using the SOVA and maximum *a posteriori* (MAP) decoding algorithms implemented with full floating point precision [13], [14]. The performance of the LDPC code in Fig. 3 was obtained using a bit accurate software model of the parallel decoder implementation described in Section V. The coding performance of the LDPC code is comparable to that of the turbo code when decoded with six iterations of a MAP decoder for packet error rates (PERs) less than 1%.

IV. Architectures for LDPC Code Decoders

The main challenge when implementing the message passing algorithm for decoding LDPC codes is managing the passing of the messages. As the functionality of both the check and variable nodes is very simple, their respective realizations are straightforward and involve only a small number of gates. The issue requiring most consideration is the implementation of the bandwidth required for passing messages between the functional nodes. The message bandwidth $M_{\text{bandwidth}}$ measured in b/s of an LDPC code with average column weight Λ_{ave} can be computed according to

$$M_{\text{bandwidth}} = 2\Lambda_{\text{ave}} \cdot W \cdot N_{\text{iter}} \cdot T \tag{7}$$

where W is the number of bits used to represent each message, N_{iter} is the number of decoder iterations, T is the target coded throughput in b/s, and the factor of 2 is to count both variable and check messages. The realization of the message passing bandwidth results in very different and difficult challenges depending on whether a hardware sharing or parallel decoder architecture is pursued.

A. Hardware-Sharing Decoder Architecture

A hardware-sharing LDPC decoder architecture consists of a small number of units implementing either the check or variable node functionality and a memory fabric to store the messages and realize the graph connectivity. This approach has been proposed in [15] and a generalized form of the hardware-sharing architecture is illustrated in Fig. 4. The advantages of the hardware-sharing architecture are that it minimizes the area of the decoder and can be readily configured to support multiple block sizes and code rates. However, the throughput of the hardware-sharing architecture is limited by the need for the functional units to be reused and the memory fabric accessed multiple times to perform each decoder iteration. For example, using (7) it is found that to decode the 1024-b, rate-1/2 LDPC code introduced in Section III with 64 iterations using 4-b messages and achieve a coded throughput of 1 Mb/s requires a memory fabric with a bandwidth of 1.7 Gb/s. While it is possible to achieve this bandwidth with a single memory, higher throughputs are more problematic. For example, to achieve a coded throughput of 1 Gb/s requires 1.7 Tb/s of memory bandwidth which cannot be realized using a single memory. Using multiple memories to achieve the required memory bandwidth is difficult because the

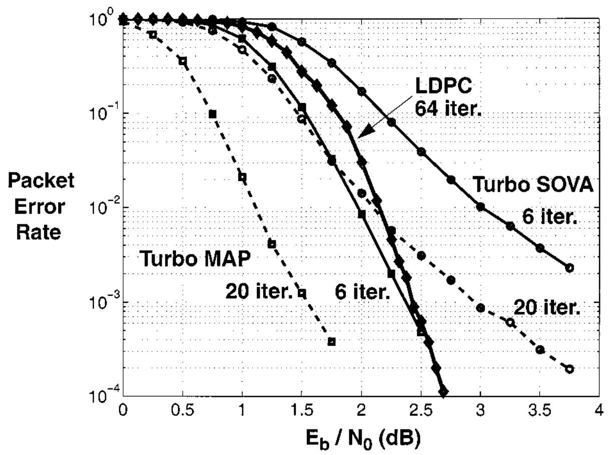

Fig. 3. Bit accurate simulation of the coding performance of the 1024-b, rate-1/2 LDPC code with 4-b messages and 64 decoder iterations. Also shown is a 3G wireless turbo code simulated using the MAP and SOVA decoding algorithms with full floating-point precision.

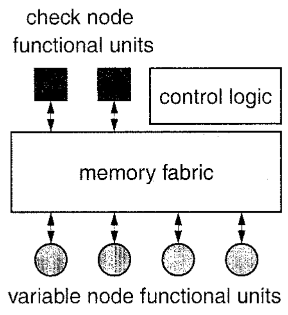

Fig. 4. Hardware-sharing LDPC decoder architecture.

essentially random or unstructured nature of the LPDC graph resists a memory architecture that allows both the variable node and check node messages to be addressed efficiently. Enforcing structure in the code graph to simplify the memory architecture typically introduces short cycles in the graph and reduces coding gain. High memory bandwidth requirements are also likely to translate into significant power dissipation. Another major issue with the hardware-sharing decoder architecture is the complexity of the control logic required for the representation of the graph connectivity and the corresponding address generation needed for fetching and storing the messages. The hardware sharing decoder architecture seems most suited for low-throughput applications where area is the major concern.

B. Parallel Decoder Architecture

The message passing algorithm maps extremely well to a parallel decoder architecture in which the graph (see Fig. 2) is directly instantiated in hardware. By this we mean that each of the variable and check nodes are realized once in hardware and routed together as defined by the LDPC code graph. To perform each decoder iteration, the variable and check node functional units are used once and messages are exchanged between them along the routed message wires. The graph representation of an LDPC code shows that the computational dependencies for any node depend only on nodes of the opposing type. This allows all variable nodes or all check nodes to be updated in a block-parallel manner enabling a large number of decoder iterations to be performed during a block period as well as allowing very high throughput. This is in stark contrast to the block-serial trellis dependencies inherent in turbo decoding. Furthermore, it can be shown that as the message passing algorithm iterates the percentage of messages changing rapidly converges to a very small value that is determined by the input SNR (E_b/N_0). This is illustrated in Fig. 5 for the 1024-b, rate-1/2 LDPC code designed in Section III.

While the parallel decoder architecture is necessarily larger than that of the hardware sharing architecture, the activity factor for the parallel architecture is very small, resulting in extremely low power dissipation. Very little control logic is needed for the parallel architecture when compared to the hardware-sharing architecture because the LDPC code graph is directly instantiated by the interconnection of the functional units. Higher throughput with a parallel decoder can be achieved by implementing a code with a larger block size and maintaining the same clock frequency. To increase the throughput in a hardware-sharing decoder, it is necessary to run the decoder at

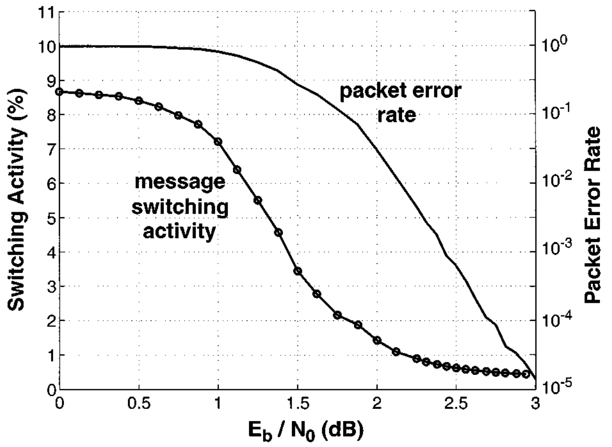

Fig. 5. Switching activity of message bits in a 1024-b, rate-1/2, soft decision LDPC decoder with 4-b messages.

a higher clock frequency, or in the case of turbo decoding to develop sliding window techniques to address the block-serial dependencies [16].

The main challenge when implementing a parallel decoder architecture for LDPC codes is the interconnection of the functional units at the top level. For an LDPC code to provide strong coding performance, the check nodes must necessarily connect to variable nodes distributed across a large fraction of the data block length. This results in a large number of long routes at the top level. However, by careful management of the physical design process, it is possible to solve routing congestion and timing closure problems.

The major drawbacks with the parallel decoder architecture are the relatively large area and the inability to support multiple block sizes and code rates on the same core. However, for applications that require high throughput and low power dissipation and can tolerate a fixed code format and large area, the parallel architecture is very suitable.

V. Architecture of a Parallel LDPC Decoder

To explore the performance and implementation issues of the parallel decoder architecture introduced in Section IV, a proof-of-concept device was designed and fabricated based on the 1024-b, rate-1 /2 LDPC code described in Section III. The number of variable nodes is set by the block size, in this case 1024, while the number of check nodes is determined by the code rate and the block size according to (2), in this case 512 for a rate-1/2 code. The interconnection of the variable and check nodes is determined by the LDPC code itself where for each edge in the code graph physical nets must be instantiated to carry messages between the variable and check nodes. The decoder architecture also requires a method to load new data packets into the decoder and write out packets once they have been decoded. The datapath, check node, variable node, and data input/output architecture of the parallel LDPC decoder are described in the following sections.

A. Decoder Datapath

The datapath of the parallel decoder is illustrated in Fig. 6 where for the purpose of clarity only one variable node and one

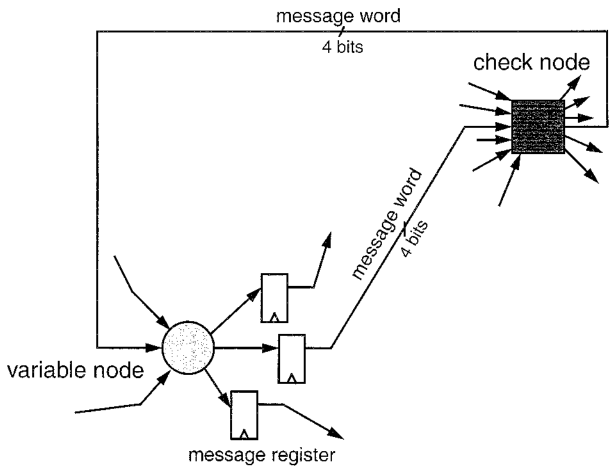

Fig. 6. Datapath architecture for the parallel decoder.

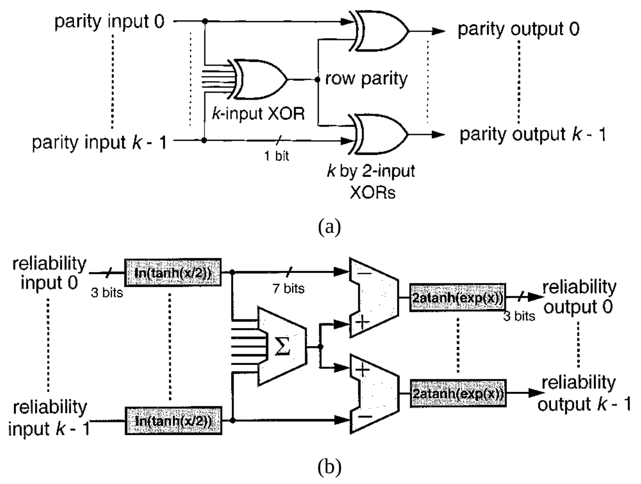

Fig. 7. Architecture for check node with k-inputs. (a) Parity update. (b) Reliability update.

check node have been shown. The messages from variable nodes to check nodes and the messages returning from the check nodes to the variable nodes are carried on distinct sets of wires. On first inspection, this doubles the number of message wires that need to be routed when compared to using a single set of wires in a time division multiplexing fashion. However, the control signal distribution, bidirectional buffers, and logic overhead associated with reusing a single set of wires negate the potential routing advantage of using a single set of wires. The total number of message wires required to connect between the variable and check nodes is calculated as

$$3328 \text{ graph edges} \times 4 \text{ b/message} \times 2 \text{ paths} = 26\,624 \text{ message wires.} \quad (8)$$

To ensure correct synchronous execution of the message passing algorithm, it is necessary to insert registers into the datapath to align the messages corresponding to each decoder iteration. By associating the registers with the variable nodes as shown in Fig. 6, the check nodes become purely combinatorial logic blocks which simplifies the floor planning of the overall decoder.

B. Check Node Architecture

Each check node performs a parity check across all variables in a row of $\boldsymbol{H}$. As shown in Fig. 7(a), the row parity is XOR-ed with each check node input to calculate the value that all other variables in the group imply each individual variable node should take. Along with the parity determination, an implied reliability of the parity in the log-likelihood domain is computed. The reliability update is performed in the log domain according to (5) and (6) so that it is multiplication- and division-free. At the output of the check node, each result is converted back into the log-likelihood domain. This arithmetic conversion is similar to the log-MAP implementation of MAP decoders [14]. The architecture of the reliability update is shown in Fig. 7(b) where the hyperbolic trigonometric functions required by (5) and (6) have been merged with the logarithmic and exponentiation functions. As the reliability messages have only a 3-b word size, the implementation of these mathematical functions is straightforward. The approximate logarithm at the inputs of the check node is realized using 10 combinatorial logic gates. The exponentiation at each of the outputs is approximated as a leading zeros count requiring 11 logic gates. A full adder compressor tree and ripple-carry adder were used to form the row reliability sum. All of the differences were computed using ripple-carry adders.

C. Variable Node Architecture

The architecture of the variable nodes is shown in Fig. 8. The variable nodes contain all of the registers in the decoder, including the decoder message registers and the shift registers that will be discussed in more detail in Section V-D. At the packet start signal, the decoding of a new packet is commenced and the previous packets results are loaded into the output shift registers. For the first decoding iteration, the messages sent to the check nodes are the sign-magnitude representations of the log-likelihood of the received value, since for a Gaussian channel the received values are the log-likelihoods up to a scaling factor. All messages passed between the variable and check nodes are represented as a sign bit and three magnitude bits. For subsequent iterations, each message entering the variable node together with the received value are converted to 2's complement and summed. The sign of the sum represents the current estimate of the decoded bit at each variable node. Outgoing messages are then formed as the group sum minus the input message of each individual edge. This is the value all other connected checks and the received value imply that each check should use for the next parity update. All values are converted back to a sign-magnitude representation and registered, to be used by the check nodes connected to the variable node in the next decoder iteration. In the case that the group sum is zero or the outgoing messages sum is zero, the sign bit used is that of the received bit, as this is the most probable value for the decoded bit. Note that for clarity the logic that performs the zero testing is not shown in Fig. 8.

D. Data Input/Output

The parallel decoder can be considered as a three-block pipeline. While one block is being iteratively decoded, the next

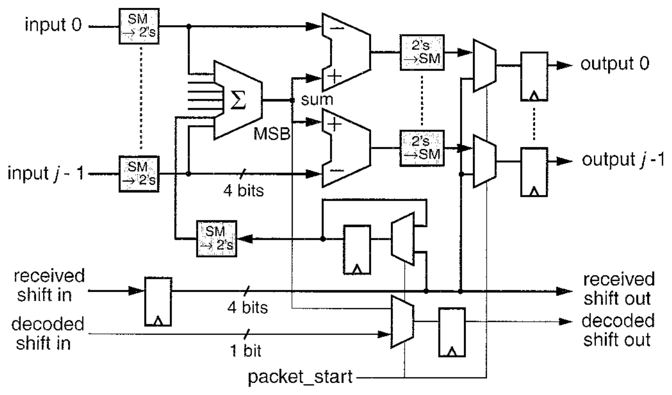

Fig. 8. Architecture for variable node with j-inputs.

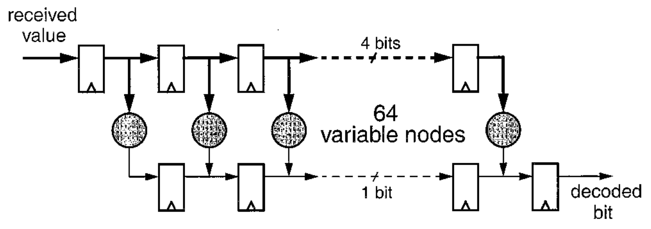

Fig. 9. Architecture of a variable node shift register group.

block is loaded into the decoder, and the previous block is read-out from the decoder. Data is loaded and unloaded from the decoder using w parallel shift registers, with a length of d samples, such that

$$w \cdot d = n. \tag{9}$$

The number of decoder iterations performed on all blocks is set to d. The columns of $\boldsymbol{H}$, and hence the variable nodes, are divided into w groups, each separated by w received symbols as shown in Fig. 9. The decoder was implemented with $w = 16$ and $d = 64$, thereby performing 64 decoder iterations for every packet. Using these parameters, a clock frequency of 64 MHz is required to achieve a coded throughput of 1 Gb/s.

E. Packet Error Detection

A packet error signal is derived by performing an OR operation of all row parities from the last decoder iteration, approximating a test of (1). The satisfaction of all parity checks in the final iteration is not equivalent to testing the parity using the final decoded bits but the difference is not significant.

VI. Implementation of a Parallel LDPC Decoder

The implementation of the decoder presented many unique challenges because the architectural characteristics of the decoder are so dissimilar to those of a typical ASIC. This required the development of special software tools to augment industry standard synthesis and place-and-route CAD tools. The floor planning and routing of the decoder are discussed in the following sections.

A. Floor Planning

The decoder was designed in a 0.16-μm 1.5-V CMOS process with five levels of metal. The variable and check node netlists

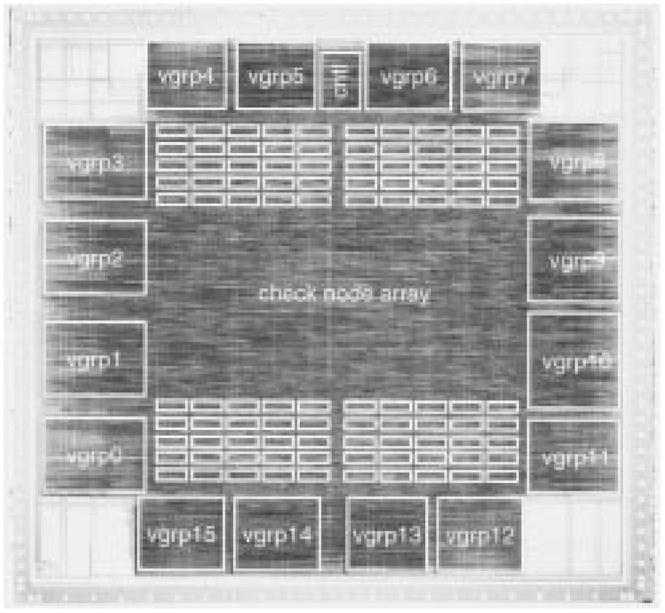

Fig. 10. Device microphotograph and floorplan.

were synthesized from a VHDL description. To simplify the physical design of the decoder, careful use of hierarchy was employed. As discussed in Section V, the 1024 variable nodes were grouped into 16 variable node shift register groups, denoted *vgrp0* to *vgrp15*, each containing 64 variable nodes. Each of the shift register groups was individually placed and routed to create a macro. Macros were also created for the weight 6 and weight 7 check nodes. To aid the routing of the top-level message nets, the variable node shift register group and check node macros were routed using only metals 1, 2, and 3, leaving metals 4 and 5 free for top-level message nets, clock, and power and ground distribution. The total gate count of the decoder was 1750K gates with the variable node and shift register logic contributing 986K gates and the check node logic requiring 686K gates.

As the check nodes do not contain any clocked elements, they were placed in the center of the chip and a clock ring containing clock buffers was placed around the periphery of the chip to distribute the clock and control signals to the variable node shift register groups. The floor plan of the decoder is shown superimposed on the device microphotograph in Fig. 10. The device dimensions are 7.5 mm × 7.0 mm giving a total area of 52.5 mm^2.

The floor planning and placement of the check nodes was critical to reducing the level of routing congestion. Each of the check nodes is connected to either six or seven variable nodes located in any of the 16 variable node shift register groups *vgrp0* to *vgrp15*. To minimize the length of the 26 624 message nets at the top level, the placement of each of the 512 check nodes was optimized using a software tool developed for that purpose. Even after optimized placement, the average length of these top level nets is 3 mm as shown in Fig. 11. As there are a large number of very long message nets, the power dissipation of the decoder is largely determined by the switching activity of these wires.

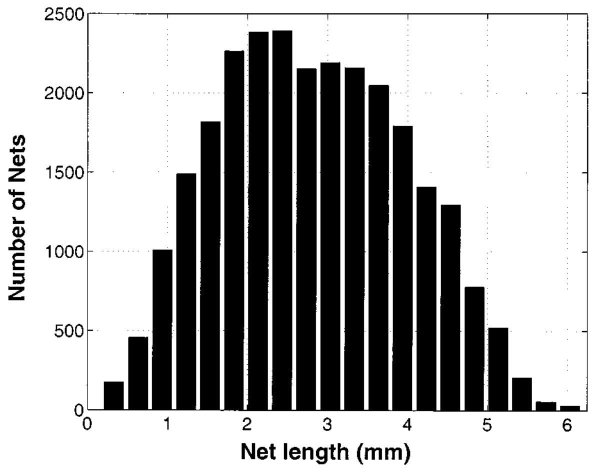

Fig. 11. Histograph of top level net lengths.

B. Routing

The size of the decoder was determined by routing congestion and not by the gate count. The key area of routing congestion was across the check node array in the center of the device. Given that the average message net length was 3 mm, buffer insertion was needed on almost every net to satisfy rise and fall time constraints. A software tool was developed to insert and place buffers for each net in such a way as to guide the router around areas of known routing congestion and enforce Manhattan geometry. As well as reducing routing congestion, the promotion of a Manhattan routing geometry had the added advantage of minimizing the number of vias on each net, resulting in substantial reduction in net resistance and hence net delay. The styles of buffer placement that were deemed "bad" and "good" for this approach are illustrated in Fig. 12(a) and (b), respectively.

With the combination of an optimized floor plan and the buffer placement technique to reduce routing congestion, timing closure for the decoder at the target clock frequency of 64 MHz was achieved under worst-case slow operating conditions. The utilization of the decoder was 50%. Significantly higher utilization could have been achieved with the availability of more levels of metal for signal routing.

VII. Measurement Results

The fabricated decoder was tested and was found to be fully functional at 64 MHz corresponding to a throughput of 1 Gb/s. The device characteristics are summarized in Table I.

A. Power Dissipation

The power dissipation of the decoder as a function of clock frequency and throughput was measured and is shown in Fig. 13 for input SNR values of 0 and 3 dB. For low SNRs, such as 0 dB, the decoder is not able to correct any packets and hence the switching activity of the message nets is relatively high (see Fig. 5). For higher values of SNR, such as 3 dB, when the decoder is functioning correctly, the switching activity on the message nets is substantially lower as the decoder is able to quickly correct packets and the message nets stop changing

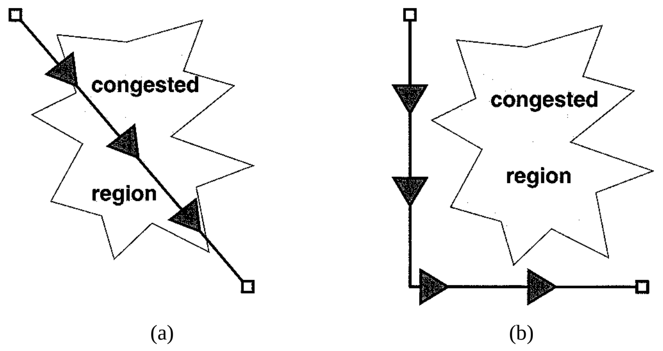

Fig. 12. Buffer placement strategy to reduce routing congestion. (a) "Bad" buffer placement. (b) "Good" buffer placement.

TABLE I
SUMMARY OF CHIP CHARACTERISTICS

Technology	0.16-μm CMOS 5-LM	
Maximum Clock Frequency	64 MHz	
Maximum Throughput	1 Gbit/s	
Total Power Dissipation	690mW (@1.5V)	
Digital Core	630mW	(91%)
Pad Frame	60mW	(9%)
Number of Gates	1750 K	
Variable Node Logic	986K	(56%)
Check Node Logic	686K	(39%)
Buffers	75K	(4.7%)
Control	5K	(0.3%)
Die Size	7.5mm × 7.0 mm	
Utilization	50%	

state after a small number of decoder iterations. At high SNRs and at low clock frequencies, it can be seen from Fig. 13 that the minimum power dissipation of the decoder is set by device leakage at around 4.5 mW. As the device contains around 7 million transistors, this corresponds to about 0.4 nA/device of leakage current.

If normal operation is considered at an SNR of 3 dB, the typical power dissipation of the decoder is 690 mW for a 1-Gb/s coded throughput. Lower power dissipation or higher throughput for the decoder can be achieved by voltage scaling. For operation below 1 Gb/s throughput, the supply voltage can be reduced below 1.5 V to as low as 0.75 V for throughputs of 10 Mb/s or less, with a corresponding saving in power dissipation. By increasing the supply voltage to 2.25 V, it was possible to run the decoder at 100 MHz to achieve a throughput of 1.6 Gb/s.

Also shown in Fig. 13 is the power dissipation estimated from the parasitics extracted from the decoder layout prior to fabrication. The switching activity factor of the message nets for this calculation was taken from Fig. 5 for an SNR value of 3 dB. It can be seen that the estimated power dissipation underestimates the measured power dissipation by a factor of 3. The reason for this discrepancy is that the message switching activity simulation used to produce Fig. 5 did not consider the substantial skew

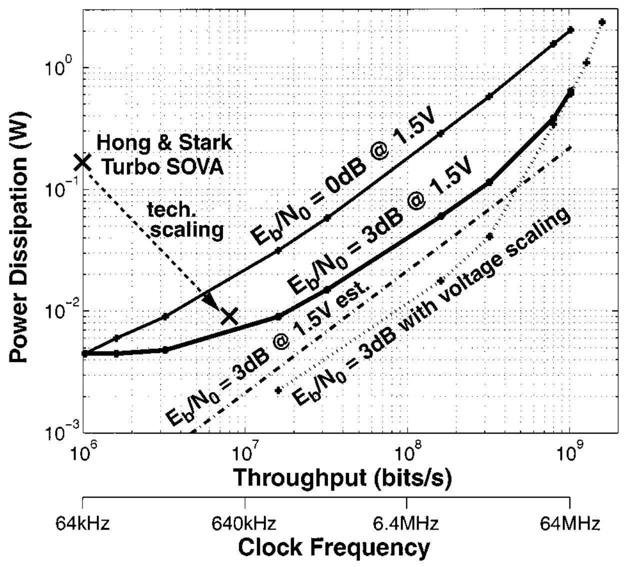

Fig. 13. Measured power dissipation as a function of throughput.

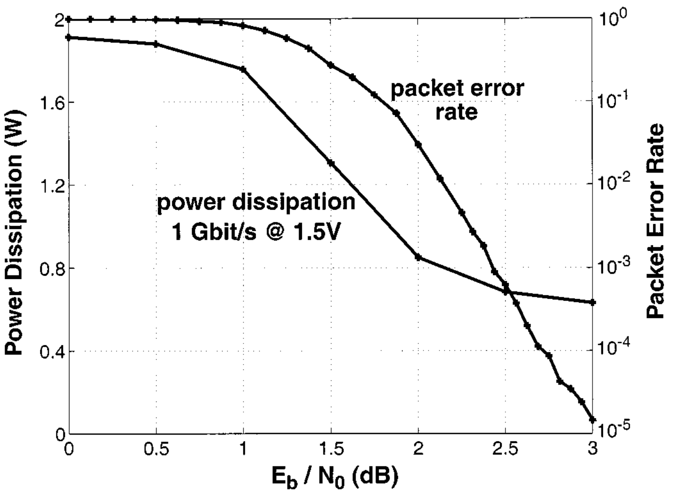

Fig. 14. Measured power dissipation as a function of SNR.

of the messages from the variable nodes to each check node introduced after implementation due to differences in net delay. This skew introduces extra switching in the ripple-carry adders used in the check nodes which results in additional switching activity on the message nets from the check nodes back to the variable nodes. For the cost of registers at the output of the check nodes, this switching could be eliminated and the power dissipation of the decoder could be reduced substantially.

The power dissipation of the decoder was also measured as a function of SNR for a fixed clock frequency. This measurement result for a clock frequency of 64 MHz, corresponding to a throughput of 1 Gb/s, is shown in Fig. 14. The dependence of the decoder power dissipation on SNR is clearly evident.

B. Performance Comparison

At 1 Gb/s, the throughput performance of the parallel LDPC decoder is at least an order of magnitude greater that any published iterative decoder implementation [4], [17]. The inordinate throughput advantage that the parallel LDPC decoder achieves through massive parallelism makes it challenging to perform a fair performance comparison in terms of power dissipation and silicon area to other iterative decoder implementations which are exclusively based on low-throughput block-serial architectures. Despite these difficulties, a crude comparison is attempted as follows.

In Fig. 13, the performance of the parallel LDPC decoder is compared to the work of Hong and Stark in which a turbo decoder that realizes the SOVA algorithm was implemented [4]. Their decoder performed three iterations on a 256-b block and dissipated 170 mW while achieving an information throughput of 1 Mb/s. Their device was simulated in a 0.6-μm 3.3-V CMOS process, so scaling linearly with process technology feature size and quadratically with supply voltage gives an estimated power dissipation of 9 mW in 0.16 -μm 1.5-V CMOS. A linear scaling of throughput with feature size and assuming a code rate of 1/2 gives a coded throughput of 7.5 Mb/s. This gives the parallel LDPC decoder a 30% power dissipation performance advantage over the turbo code decoder at a SNR of 3 dB. However, at a throughput of 7.5 Mb/s, the power dissipation of the parallel LDPC decoder is significantly limited by leakage. It can be expected that, for higher throughputs, the power dissipation advantage of the parallel LDPC decoder over a turbo code decoder would be substantially higher.

In terms of silicon area, the turbo code decoder of Hong and Stark uses 100 mm^2 of silicon in 0.6-μm CMOS. Scaling quadratically with feature size yields an estimated area of 7 mm^2 in 0.16-μm CMOS. By contrast the area of the parallel LDPC decoder is 52.5 mm^2. However, the area of the parallel LDPC decoder is largely determined by routing congestion. Using CMOS fabrication technologies with six or seven levels of metal would result in a significant reduction in die size. Furthermore, it would not be possible to significantly increase the throughput performance of the turbo code decoder to match that of the parallel LDPC decoder without a substantial area penalty.

VIII. Conclusion

A 1024-b, rate-1/2 low density parity check code (LDPC) decoder has been described that achieves a coded throughput of 1 Gb/s while dissipating 690 mW from a 1.5-V supply. This performance is achieved by implementing a parallel architecture that exploits the inherent parallelism and rapid convergence of the message passing decoding algorithm. The coding performance of the LDPC decoder is equivalent to that of comparable turbo codes, while the parallel architecture provides the LDPC decoder with a substantial throughput and power dissipation performance advantage over turbo code decoders. The realization of the parallel LDPC decoder architecture was achieved by developing specific software tools to manage the floor planning and routing stages of the design process.

As iterative decoding schemes are considered for the next generation of forward error correction devices, LDPC codes will be chosen for many applications because of their excellent coding performance, and the ability in their implementation to elegantly tradeoff area for high throughput and lower power dissipation as demonstrated by this work.

ACKNOWLEDGMENT

The authors wish to thank H. Modernel for layout assistance and the reviewers for their helpful comments.

REFERENCES

[1] M. Bossert, *Channel Coding for Telecommunications*. New York: Wiley, 1999.
[2] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting codes and decoding," in *Proc. Int. Conf. Comm. '93*, May 1993, pp. 1064–1070.
[3] R. Pyndiah, A. Glavieux, A. Picart, and S. Jacq, "Near optimum decoding of product codes," in *Proc. IEEE GLOBECOM '94*, 1994, pp. 339–343.
[4] S. Hong and W. Stark, "Design and implementation of a low complexity VLSI turbo-code decoder architecture for low energy mobile wireless communications," *J. VLSI Signal Processing*, vol. 24, pp. 43–57, 2000.
[5] R. Gallager, "Low density parity check codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21–28, Jan. 1962.
[6] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.
[7] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 619–637, Feb. 2001.
[8] S. Y. Chung, G. D. Forney, T. J. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Commun. Lett.*, vol. 5, pp. 58–60, Feb. 2001.
[9] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. IT-42, pp. 533–547, 1981.
[10] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Analysis of low density codes and improved designs using irregular graphs," in *Proc. 30th Annu. ACM Symp. Theory of Computing*, 1998, pp. 249–258.
[11] C. J. Howland and A. J. Blanksby, "Parallel decoding architectures for low density parity check codes," in *Proc. IEEE ISCAS*, vol. 4, May 2001, pp. 742–745.
[12] 3rd Generation Partnership Project (3GPP); Technical specification group radio access network multiplexing and channel coding (TDD). [Online]. Available: http://www.3gpp.org.
[13] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *Proc. GLOBECOM 1989*, 1989, pp. 1680–1686.
[14] L. R. Bahl, J. Cocke, R. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, pp. 363–377, Mar. 1974.
[15] E. Yeo, P. Pkzad, B. Nikolic, and V. Anantharam, "VLSI architectures for iterative decoders in magnetic recording channels," *IEEE Trans. Magn.*, vol. 37, pp. 748–755, Mar. 2001.
[16] P. J. Black and T. H. Y. Meng, "A 1-Gb/s, four-state, sliding block Viterbi decoder," *IEEE J. Solid-State Circuits*, vol. 32, pp. 797–805, June 1997.
[17] C. Berrou, P. Combelles, P. Penard, and B. Tailbart, "An IC for turbo-codes encoding and decoding," in *Proc. Int. Solid-State Circuits Conf.*, Feb. 1995, pp. 90–91.

Andrew J. Blanksby received the B.S. and Ph.D degrees in electrical and electronic engineering from the University of Adelaide, Australia, in 1993 and 1999, respectively.

Since 1998, he has been a Member of Technical Staff in the DSP & VLSI Systems Research Department, Bell Laboratories, Lucent Technologies, Holmdel, NJ. In March 2001, this group became the High Speed Communications VLSI Research Department, Agere Systems. His research interests include VLSI design, channel coding, and equalization.

Chris J. Howland received the B.S. degree in mathematical and computer science and the B.E. degree in electrical and electronic engineering from the University of Adelaide, Australia, in 1997. He is currently working toward the Ph.D. degree in electrical and electronic engineering at the same university.

In 1999, he undertook a one-year internship in the DSP & VLSI Research Department, Bell Laboratories, Holmdel, NJ. In 2000, he joined the department as a Member of Technical Staff and currently works for Agere Systems. His research interests include forward error correction, in particular low-density parity-check codes, arithmetic circuits, and the implementation of signal processing algorithms.