High-Throughput LDPC Decoding Using The RHS Algorithm

François Leduc-Primeau*, Alexandre J. Raymond*, Pascal Giard*†, Kevin Cushon*,
Claude Thibeault†, and Warren J. Gross*
*Department of Electrical and Computer Engineering, McGill University, Montreal, Qc, Canada H3A 0E9.
Email: {francois.leduc-primeau, alexandre.raymond, pascal.giard, kevin.cushon}@mail.mcgill.ca, warren.gross@mcgill.ca
†Department of Electrical Engineering, École de technologie supérieure, Montreal, Qc, Canada H3C 1K3.
Email: claude.thibeault@etsmtl.ca

***Abstract*—The relaxed half-stochastic (RHS) algorithm is a recently proposed binary message-passing decoding algorithm for low-density parity check codes that can reach the same error rate performance as belief propagation algorithms that exchange LLR messages. Because of its low-complexity interleaver, the RHS algorithm makes it possible to achieve a fully-parallel implementation that can converge to a codeword in only a few clock cycles on average, enabling high throughput and power efficiency. To demonstrate the practicality of the RHS algorithm, we implement a decoder for the popular IEEE 802.3an 10GBASE-T standard. The paper presents details of the hardware implementation, as well as post-layout results for an ASIC implementation in 65nm CMOS technology, which indicate that the decoder can operate at 448 MHz and occupies an area of 4.41 mm^2. The results obtained from bit-accurate software simulations show that the decoder meets the latency requirement prescribed by the standard and provides an average throughput of 160 Gbps.**

I. Introduction

Among capacity-approaching codes, low-density parity-check (LDPC) codes are a very interesting choice for high-throughput applications, because they can be decoded with a high level of parallelism using belief propagation (BP) algorithms. However, in practice, making use of all the available parallelism in the decoding algorithm for codes with lengths in the thousands of bits remains a challenging task. Fully-parallel implementations are attractive because they can converge to the transmitted codeword in a very small number of clock cycles on average. This can be used either for enabling very high throughput applications, or to save power through a reduction of the switching activity, or by including a sleep mode.

As pointed out in [1], the main challenge in a fully-parallel circuit implementation is to efficiently route the large amount of wires needed to exchange messages between the processing nodes. For this reason, a naive approach to the design results in an implementation with a very low ratio of logic area to silicon area. The first step in reducing routing complexity is to reduce the number of parallel wires used for message passing. Some algorithms have been proposed that use the log-likelihood (LLR) representation for computations, but transmit messages on a single wire. In [2], [3] the LLR values are simply transmitted serially, while in [4] the magnitude of the LLR message is represented by the width of a pulse, which reduces the complexity of the computations and the switching activity in the interleaver. A BP decoding algorithm for LDPC codes that is constrained to using modulo-2 addition as its check node update function is known as a binary message-passing (BMP) algorithm. BMP algorithms use the minimum amount of wires to exchange messages, but they also have other important properties pertaining to the layout of their interleaver in the circuit implementation. BMP algorithms often use as input a single bit of information per codeword bit, for example Gallager's algorithms A and B [5], or the "DD-BMP" algorithm presented in [6]. To take advantage of more information bits at the input, [7] builds on the *stochastic* representation of the likelihood information originally proposed in [8].

The relaxed half-stochastic (RHS) algorithm [9] is a BMP algorithm that differs significantly from other BMP algorithms in that it can achieve the same bit error rate (BER) as the sum-product algorithm (SPA) implemented in floating-point [10]. In this paper, we demonstrate that the RHS algorithm can be used to implement a very efficient fully-parallel decoder for the IEEE 802.3an 10GBASE-T standard [11]. We first provide a review of the RHS algorithm in Section II. The circuit architecture is presented in Section III, and in Section IV we discuss the synthesis results for our implementation and provide the decoding performance in terms of error rate as well as average and maximum decoding time.

II. The RHS Algorithm

Belief propagation decoding algorithms are conveniently described in terms of the factor graph [10] (or Tanner graph) representation of the code. The factor graph is a bipartite graph with *variable* nodes (VN) representing the codeword symbols, and *check* nodes (CN) representing the parity constraints. The algorithm can then be characterized by specifying a VN computation, which describes the messages sent from VNs to CNs, and a CN computation, which describes the messages sent from CNs to VNs. We must also specify how to compute the estimate of each codeword symbol. A decoding iteration is performed as follows: 1) All VNs send a message to their CN neighbors, 2) all CNs send a message to their VN neighbors, and 3) an estimate is generated for each codeword symbol. In the RHS algorithm, the VN computation includes memory elements that we will refer to as *trackers*. For clarity, we will describe separately how the trackers are updated. Note that this paper only provides a minimal description of the RHS algorithm. Further details can be found in [9].

A. Variable node computation

As is well known, the SPA converges to the maximum-likelihood estimate of each codeword symbol in a cycle-free graph [10]. In order to achieve high error rate performance, the RHS algorithm performs the VN computation of the SPA exactly. This computation is best carried out in the LLR domain, as this simplifies the computation to an addition and reduces quantization errors. In the SPA, a message sent on output i of a VN is given by

$$\Lambda_i' = \Lambda_o + \sum_{j=1}^{d_v} (\Lambda_j) - \Lambda_i, \tag{1}$$

where d_v is the degree of the VN, Λ_o the prior symbol likelihood, and $\Lambda_1, \Lambda_2, \ldots, \Lambda_{d_v}$ are the VN input messages.

In RHS, the VN sends and receives binary messages. Therefore, Λ_j is not directly the message received at the VN, but rather the content of tracker j, whose value is updated from the binary messages. For the same reason, Λ_i' is not the output message. Therefore, we will call it the i-th *intermediate* output.

In order to send binary messages that can precisely represent the value of the intermediate output, we rely on *stochastic* messages, which are random binary messages with a statistical distribution representing Λ_i'. Specifically, let $X_{i,j} \in \{0,1\}$ be a binary output message on edge i. We want to generate a message such that $\mathbb{E}[X_{i,j}] = \frac{1}{1+\exp(\Lambda_i')}$. Several binary messages, indexed by j, can be generated from the same intermediate output Λ_i'. For the 10GBASE-T code, we use $j \in \{1,2\}$, which means that two binary messages are sent on each edge of the code graph at every iteration of the algorithm.

B. Check node computation

As with any BMP algorithm, the check node computation used in RHS is simply a modulo-2 sum. Let $X_{1,j}, X_{2,j}, \ldots, X_{d_c,j}$ be the binary messages received by the CN. The i-th CN binary output $Y_{i,j}$ is given by

$$Y_{i,j} = X_{1,j} \oplus \ldots \oplus X_{i-1,j} \oplus X_{i+1,j} \oplus \ldots \oplus X_{d_c,j}, \tag{2}$$

where $\oplus$ denotes modulo-2 addition. If the CN inputs are independently distributed such that $\mathbb{E}[X_{i,j}] = p_i$, we have that

$$\mathbb{E}[Y_{i,j}] = \frac{1}{2} - \frac{\prod_{j=1}^{d_c}(1-2p_j)}{2(1-2p_i)}, \tag{3}$$

which corresponds to the SPA CN function in the probability domain. On other words, the exact SPA function is computed on the message distributions instead of individual messages.

Eq. (2) has two interesting properties that are not found in the CN function of min-sum algorithms, and which are useful for simplifying the layout of a fully-parallel decoder implementation. First, each output i can be expressed in terms of the i-th input and of a total:

$$Y_{i,j} = (X_{1,j} \oplus \ldots \oplus X_{d_c,j}) \oplus X_{i,j}. \tag{4}$$

In addition to providing a lower complexity implementation, (4) makes it possible to compute a single total parity value in the CNs and to broadcast this result to all neighboring VNs. Those VNs can then subtract their own contribution to the parity in order to retrieve the extrinsic parity. Second, (2) can be factored arbitrarily, which adds flexibility in the gate placement in the implemented decoder.

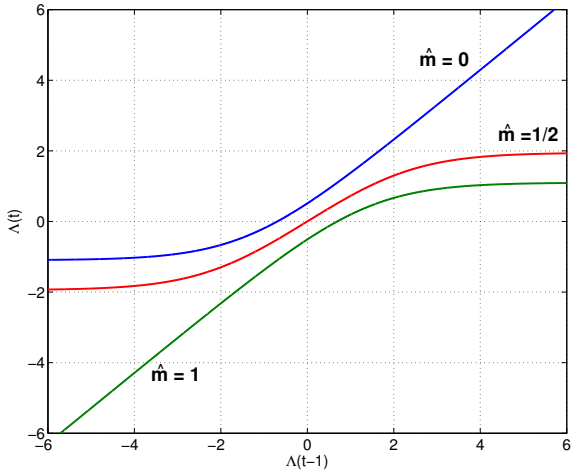

Fig. 1. LLR tracker transfer functions $f(\Lambda_i(t-1); \hat{m}_i(t))$ with $\beta = 0.25$, and $\hat{m}_i(t) \in \{0, \frac{1}{2}, 1\}$.

C. Tracker update

With $j \in \{1,2\}$, the VN receives two binary messages from each of its CN neighbors at every iteration. On a given input, the two messages are independent and identically distributed. We can therefore easily obtain the maximum-likelihood mean estimate for input i: $\hat{m}_i = \frac{1}{2}(X_{i,1}+X_{i,2})$. This estimate should be seen as a higher precision message, albeit of course still random. To obtain even higher precision, we track the (non-stationary) distribution of the message streams across decoding iterations. This approach is inspired by successive relaxation [12], which can also improve the error rate performance of deterministic BP algorithms such as SPA and min-sum. For this purpose, we use the simple linear tracking filter given by

$$p_i(t) = (1-\beta)p_i(t-1) + \beta\hat{m}_i(t), \tag{5}$$

where $0 < \beta \leq 1$ is a design parameter. As mentioned previously, the VN computation is performed in the LLR domain. Therefore, to minimize the implementation complexity, we also want to perform this tracking in the LLR domain. Each VN input i is equipped with an LLR-domain tracker, and we denote its value at iteration t by $\Lambda_i(t)$. We approach the implementation by considering the tracking function separately for each possible value of $\hat{m}_i$. We write the tracker update as $\Lambda_i(t) = f(\Lambda_i(t-1); \hat{m}_i(t))$.

Fig. 1 shows an example of the LLR-domain tracking transfer functions. We can see that the functions have the property that $f(\Lambda_i; 0) = -f(-\Lambda_i; 1)$. This will be used in the implementation. Furthermore, the functions can be approximated by piecewise-linear functions with parameters

a, b, c, d:

$$f(\Lambda, 0) \approx \begin{cases} \Lambda + b & \text{if } \Lambda \geq d, \\ d & \text{if } \Lambda < d. \end{cases} \tag{6}$$

$$f(\Lambda, 1/2) \approx \begin{cases} a\Lambda & \text{if } -c \leq \Lambda \leq c, \\ -c & \text{if } \Lambda < -c, \\ c & \text{if } \Lambda > c. \end{cases} \tag{7}$$

D. Codeword estimate

As in the SPA, the codeword bit x associated with a VN is estimated from the LLR total $\Lambda_T = \Lambda_o + \sum_{j=1}^{d_v} \Lambda_j$:

$$\hat{x} = \begin{cases} 0 & \text{if } \Lambda_T \geq 0, \\ 1 & \text{if } \Lambda_T < 0. \end{cases} \tag{8}$$

III. Architecture

The architecture of a fully-parallel LDPC decoder consists of small components replicated a large number of times. Unlike in partially-parallel architectures, most of the implementation work consists in optimizing these small components, and as such fully-parallel architectures can be easier to implement. In the RHS decoder, the variable node components represent most of the decoder area. In turn, the trackers represent a large portion of the VNs. Optimizing these small components therefore has a big effect on the overall complexity of the decoder. The optimizations that will be presented were tested with the 10GBASE-T code. The code has a length of 2048 bits, with a code rate of 0.8413. It is regular with variable node degree $d_v = 6$ and check node degree $d_c = 32$.

A. Variable node

The variable node component takes as input a prior LLR estimated from the channel as well as stochastic messages received from the check nodes, and outputs stochastic messages and an estimate of the codeword bit. The LLR prior is quantized on 4 bits with a range of $[-7, 7]$. The variable node processing includes three steps: 1) update of the trackers, 2) SPA addition, and 3) generation of the binary stochastic messages. The component contains 6 trackers that together store the values $\{\Lambda_1, \ldots, \Lambda_6\}$. As is usually done to minimize complexity, the 6 extrinsic sums of the SPA computation are generated by first computing an LLR total Λ_T for the node, and then subtracting the intrinsic value, as illustrated in (1). After this step, we are in possession of $\Lambda'_1, \ldots, \Lambda'_6$. The stochastic output messages $X_{i,j}$ are then generated by comparing the intermediate outputs with a random threshold T_j, where $1 \leq i \leq 6$ is the output index and $1 \leq j \leq 2$ is the bit index:

$$X_{i,j} = \begin{cases} 0 & \text{if } \Lambda'_i > T_j, \\ 1 & \text{if } \Lambda'_i < T_j, \\ B & \text{if } \Lambda'_i = T_j, \end{cases} \tag{9}$$

where B is a fair random bit used to prevent the quantization of T_j from biasing the distribution.

In order to minimize the complexity of the interleaver, we serialize the transmission of the stochastic bits; this implies that two clock cycles are needed at each iteration. We can achieve this while keeping the cycles approximately balanced by placing 1-bit registers at the input of the VNs, as illustrated in Fig. 2. In the first cycle, the SPA sum is computed, and a stochastic bit (generated using T_1) is sent on each edge, through the check nodes, to be stored in the VN bit registers. In the second cycle, stochastic bits generated using T_2 are sent through the check nodes, and the trackers are updated using the two bits. As a result, the only delay shared by the two cycles is the VN output comparator and the check node.

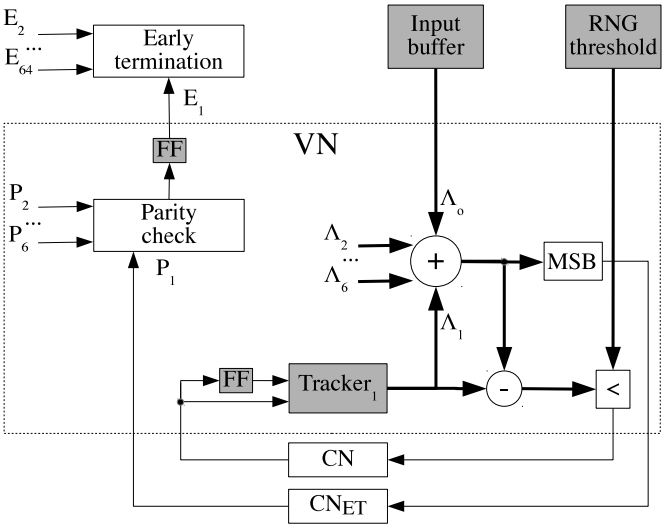

Fig. 2. Register placement within the decoder, with registers shown as shaded boxes. The diagram is not functionally accurate: only one message input and output are shown, and the loop through the CN should be understood as starting from some VN and ending at a different VN. Note that only 64 out of 2048 VNs include the "Parity-check" block.

B. Random number generation

The stochastic messages must be distributed such that their mean represents the *probability* that the codeword bit is 0. Since the messages are generated from LLR values, the thresholds are given by $T_j = \ln(\frac{1-P}{P})$, where P is uniformly distributed on $(0, 1)$. In addition, the threshold values must be quantized. Simulation results have shown that an integer quantization on the range $[-6, 6]$ was sufficient. It would seem natural to use a quantization that minimizes the mean squared error, i.e. that minimizes $\mathbb{E}[(\widetilde{T}_j - T_j)^2]$, $\widetilde{T}_j$ denoting the quantized value. However, simulations show that an error that overestimates $|T_j|$ is more detrimental to the decoding performance, and that the quantization $\widetilde{T}_j = \text{sign}(T_j)\lfloor |T_j| \rfloor$ is preferable.

Consider a priority encoder circuit that takes as input a random binary sequence $\mathbf{Z} = \{Z_1, \ldots, Z_q\}$ with $\Pr(Z_i = 0) = \psi_i$, and outputs the number W of "one" elements that precede the first "zero" element. We use W and a fair bit S to generate $T_j = (-1)^S W$. To obtain a range of $[-6, 6]$, we need $q = 7$, and a good approximation of the desired distribution is obtained by setting $\psi_1 = \ldots = \psi_5 = \frac{1}{2}$ and $\psi_6 = \psi_7 = \frac{1}{4}$. Therefore it is easily seen that the random threshold generation requires 10 fair random bits, 9 to generate

TABLE I
IDEAL AND IMPLEMENTED LLR THRESHOLD DISTRIBUTION

Threshold	p (ideal)	p (impl.)
-6	0.0025	0.0029
-5	0.0042	0.0039
-4	0.0113	0.0156
-3	0.0294	0.0312
-2	0.0718	0.0625
-1	0.1497	0.1339
0	0.4621	0.4995
1	0.1497	0.1339
2	0.0718	0.0625
3	0.0294	0.0312
4	0.0113	0.0156
5	0.0042	0.0039
6	0.0025	0.0029

$\mathbf{Z}$, and 1 for S. We use a linear-feedback shift register (LFSR) to generate the random bits. Table I gives the final distribution of T_j, taking into account that the all-zero sequence never occurs in the LFSR, and that we set $T_j = (-1)^S$ when $\mathbf{Z} = \{1, 1, 1, 1, 1, 1, 1\}$.

Even though the RHS algorithm is derived by assuming that the thresholds used in the VNs are independent, in practice, simulations have shown that the random number generation (RNG) components can each be shared with 64 VNs without degrading the decoding performance. As a result, the RNG components have little impact on the decoder area.

C. Tracker

Each variable node includes 6 tracker components that receive the binary messages and update a stored LLR value. The update is a linear approximation of (5) in the LLR domain, given by (6) for $\hat{m} = 0$ and $\hat{m} = 1$ (using $f(\Lambda; 1) = -f(-\Lambda; 0)$), and by (7) for $\hat{m} = \frac{1}{2}$. Identifying the parameters a, b, c, d directly would be cumbersome. Instead, we first simulate the decoder with the ideal tracking given by (5) to identify the optimal parameter β. When β is optimized to minimize the BER, the optimal value depends on the maximum number of iterations performed by the decoder. With a maximum of 50 iterations, the optimal β for the 10GBASE-T code is found to be 0.25.

We propose a circuit design methodology where we start from a decoder using (5) and gradually introduce the changes needed to produce an optimized implementation of the trackers. For some steps, we use Monte-Carlo simulations to find the parameter values that optimize the performance. At this stage of the design, the simulations can be performed at a relatively high BER and the process is therefore not very computationally intensive. The circuit design can be split in four steps:

1) We first identify the range required for the tracker values Λ_i. Let $\Lambda_i \in [-\Lambda_L, \Lambda_L]$. We simulate the decoder to identify the smallest $\Lambda_L > 0$ for which the decoder retains a BER approximately equal to the ideal case. For the 10GBASE-T code, we can use a value as low as $\Lambda_L = 3$.
2) Next, we introduce the saturation operations associated with parameters c and d. We can test whether it is possible to use the same absolute value for c and d. For the 10GBASE-T code, it is indeed possible to use $c = -d = 1$.
3) After the second step, we know the range over which to optimize the choice of a and b. Parameter a is optimized by minimizing
$$\int_{-c}^{c} \left(a\Lambda - f\left(\Lambda; \frac{1}{2}\right)\right)^2 d\Lambda, \tag{10}$$
and parameter b by minimizing
$$\int_{d}^{\Lambda_L} \left(\Lambda + b - f\left(\Lambda; 0\right)\right)^2 d\Lambda. \tag{11}$$
For $\Lambda_L = 3$, $c = -d = 1$, we obtain $a = 0.73$ and $b = 0.432$.
4) As the last step, we find compact binary approximations for a and b. For the 10GBASE-T code, we found that $a = \frac{3}{4}$ and $b = \frac{1}{2}$ provide good performance.

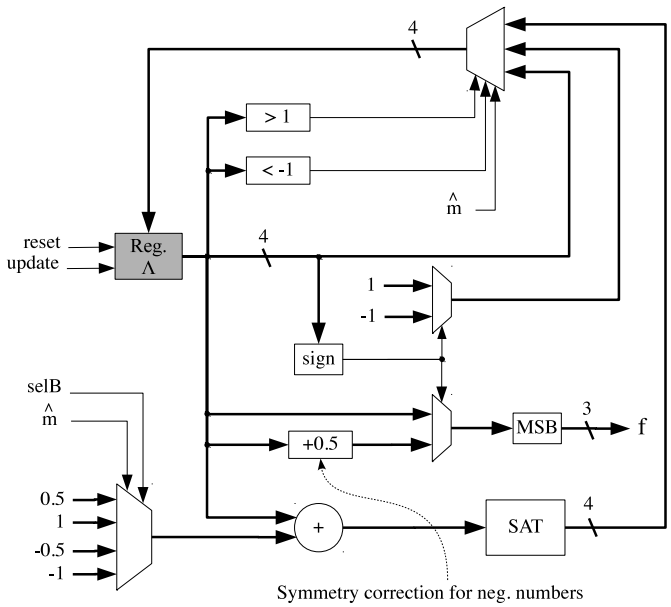

Fig. 3. Block diagram of the tracker update logic. The output "f" is fed to the adder circuit of the VN.

Since the tracker is the component that has the most impact on the area of the decoder, we then used a trial and error process to find further simplifications to the circuit. We found that it is possible to output only the integer part of the tracker to the adder tree, thereby simplifying the adder. This simplification in turn makes it possible to completely remove the tracker update for $\hat{m} = \frac{1}{2}$ and to only keep the saturation, since the tracker output is now 0 whenever $|\Lambda_i(t)| < 1$. The update function for $\hat{m} = \frac{1}{2}$ therefore becomes

$$\Lambda_i(t) = \begin{cases} \Lambda_i(t-1) & \text{if } -c \le \Lambda_i(t-1) \le c, \\ -c & \text{if } \Lambda_i(t-1) < -c, \\ c & \text{if } \Lambda_i(t-1) > c. \end{cases} \tag{12}$$

The optimized tracker architecture is shown in Fig. 3. The size of the tracker register is 4 bits, and the output has 3 bits.

Another feature of the RHS algorithm is its ability to dynamically change the parameter b in order to reduce the average convergence time with little impact on the BER. This tracker implementation supports $b = 1$ and $b = \frac{1}{2}$. We use $b = 1$ in the first iteration.

D. Check node partitioning

The use of check node partitioning for improving the layout efficiency of fully-parallel LDPC decoders was first reported in [13], where it was shown to greatly reduce the area required for the implementation after place & route. In min-sum algorithms, partitioning is not possible without modifying the algorithm, because the d_c check node functions that must be computed by a check node component cannot be factored in terms of a total function. To remedy this, [13] proposes an approximate algorithm that can be partitioned. Unfortunately, this introduces a loss in error rate performance that grows with the number of partitions. In contrast, BMP algorithms such as RHS can be partitioned as is.

Partitioning allows dividing the decoder design in several blocks that can be processed independently by the place & route tool. The downside is that it also introduces additional delay in the CN circuit that is associated with the distribution of the total. If the CN is implemented using 2-input XOR gates, for k partitions, $k \leq d_c/2$, the longest path in the CN has a gate delay Δ given by

$$\Delta = \log_2\left(\frac{d_c}{k}\right) + (k-1) + 1, \tag{13}$$

where we assume for simplicity that d_c and k are powers of 2. The first term represents the delay through the local computation tree, the term $(k-1)$ represents the worst distribution delay, and the last term is associated with the computation of the extrinsic parity from the total. For example, with $d_c = 32$, a CN circuit with no partition has $\Delta = 6$, while a circuit with 8 partitions has $\Delta = 10$, and a fully-partitioned circuit (not described by (13)) has $\Delta = 31$.

E. Early termination

Early termination (ET) refers to a mechanism that monitors the decision output of all the variable nodes to detect a valid codeword, allowing the decoder to stop before the maximum number of iterations is reached. Early termination is crucial for power saving: it reduces the switching activity of the decoder, and it can also be used to put the decoder into a sleep mode until the next frame is received.

To reduce the average decoding time as much as possible, the early termination check is performed in parallel with the message passing. To achieve this, we instantiate a second copy of the check node circuits. These ET check nodes receive the estimated bit from their neighboring VNs, and compute one parity sum. If all the ET check nodes have even parity, we have found a valid codeword and can stop the decoder. For convenience, part of the ET logic is implemented in a group of 64 VNs that are connected to all 384 CNs, as shown in Fig. 2.

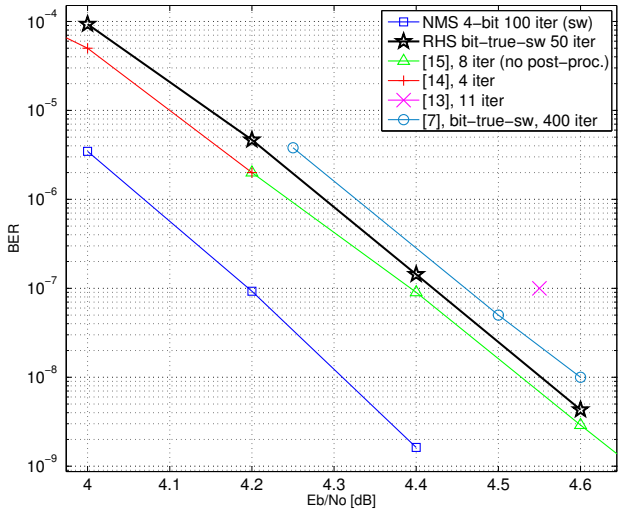

Fig. 4. Bit-error rate performance of the RHS decoder simulated using a bit-accurate software model, compared to reported BER results for other 10GBASE-T decoders. The curve on the left shows normalized min-sum with 4-bit messages simulated in software, with a maximum of 100 iterations.

IV. Performance

A. Methodology

Validation of the hardware implementation was performed by means of functional simulation. Each component was validated against a bit-accurate software implementation via a thorough inspection of its operation, followed by the use of random test vectors. The integration of the components was then validated for the entire decoder using the same methodology. The bit-accurate software models are also used for software simulations of the decoder, which allows us to quickly measure the performance of the decoder.

The ASIC implementation of our decoder uses TSMC 65nm CMOS technology with 7 metal layers and a supply voltage of 1.0V. Synthesis was performed using Cadence RC, and layout performed with Cadence Encounter. This implementation is fully-parallel, and uses the early termination scheme described in Section III-E. The input LLRs and VN trackers both use 4 bits of quantization.

B. Layout results

Initial place & route results were obtained for an unpartitioned architecture and for an architecture having 8 VN partitions. The results were better for the 8-partition design, and therefore it was chosen for the final layout. However, the difference between the two designs was not as significant as reported in [13]. This is reasonable, since the RHS interleaver and check node function are already much simpler than in the case of a Min-Sum decoder, and therefore they are less of a bottleneck for the place & route.

After place & route the decoder occupies 4.41 mm^2, with a logic density of 94.4%. Under typical operating conditions, the circuit achieves a clock frequency of 448 MHz. The layout

TABLE II
COMPARISON WITH OTHER WORKS

	RHS	[7]	[13]	[14]	[15]
Process	65nm CMOS	90nm CMOS	65nm CMOS	90nm CMOS	65nm CMOS
Architecture	Fully-parallel	Fully-parallel	Fully-parallel	Part-parallel	Part-parallel
Area {scaled to 65 nm} (mm^2)	4.41	6.38 {3.33}	4.84	5.35 {2.79}	5.35
Maximum frequency (MHz)	448	500	195	137	700
Coded throughput (Gbps)	160 @ 5.5dB	61.3 @ 5.5dB	92.8 @ 4.55dB	11.7 (no ET)	47.7 @ 5.5dB

results are summarized in Table II, along with the results for other 10GBASE-T decoders.

C. Decoding performance

We simulated the decoder with a maximum of 50 iterations. As can be seen on Fig. 4, this is sufficient to achieve a BER very close to the best results reported in other 10GBASE-T decoders. With a limit of 50 iterations, the minimum clock frequency such that the decoder achieves the latency required by the standard is 317 MHz. The BER of RHS can be improved significantly by increasing the maximum number of iterations, but of course this also increases the minimum clock frequency.

With our early termination mechanism, the average number of iterations needed for decoding at 5.5 dB is 2.86. At 448 MHz, this translates to an average decoding time of 12.8 ns. The standard only requires that the latency be smaller than 315 ns, and therefore the decoder would be idle 96% of the time. By using techniques such as clock or power gating, we can use the fast decoding time to save power. In other applications, the RHS algorithm can be used to achieve very high throughputs by adding buffering capability for the received frames.

D. Comparison with existing decoders

Compared to other fully-parallel implementations, the proposed implementation achieves higher throughput, with better BER. Compared to [14], RHS achieves a much higher throughput, but also requires more area. Compared to [15], RHS achieves higher throughput for a smaller area. Note that the design of [15] includes "post-processing" hardware and on-chip functional testing, but the area without these modules is quoted as 4.4 mm^2, and therefore the area of the RHS decoder remains similar if they are taken out.

V. CONCLUSION

In this paper we presented a decoder architecture based on the RHS algorithm for the LDPC code included in the IEEE 802.3an 10GBASE-T standard. The decoder converges to the transmitted codeword in only a few clock cycles on average, which can be used to reduce the power consumption or to reach very high throughputs (much higher than required by the 10GBASE-T standard). We also showed that a decoder based on the RHS algorithm is straightforward to design by presenting systematic methods for identifying good parameter values.

ACKNOWLEDGEMENT

The authors wish to thank CLUMEQ for providing computing resources, and CMC Microsystems for providing access to the Cadence tools and TSMC 65nm CMOS technology. Warren J. Gross and Claude Thibeault are members of ReSMiQ.

REFERENCES

[1] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 3, 2002.

[2] A. Darabiha, A. Carusone, and F. Kschischang, "A bit-serial approximate min-sum LDPC decoder and FPGA implementation," in *Proc. IEEE International Symposium on Circuits and Systems*, 2006.

[3] T. Brandon, R. Hang, G. Block, V. C. Gaudet, B. Cockburn, S. Howard, C. Giasson, K. Boyle, P. Goud, S. S. Zeinoddin, A. Rapley, S. Bates, D. Elliott, and C. Schlegel, "A scalable LDPC decoder ASIC architecture with bit-serial message exchange," *Integration, the VLSI journal*, 2008.

[4] K. Cushon, C. Leroux, S. Hemati, S. Mannor, and W. Gross, "A min-sum iterative decoder based on pulsewidth message encoding," *IEEE Trans. on Circuits and Systems II: Express Briefs*, vol. 57, no. 11, pp. 893 –897, Nov. 2010.

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

[6] N. Mobini, A. Banihashemi, and S. Hemati, "A differential binary message-passing LDPC decoder," *IEEE Trans. on Communications*, vol. 57, no. 9, pp. 2518–2523, Sept. 2009.

[7] S. Sharifi Tehrani, A. Naderi, G.-A. Kamendje, S. Hemati, S. Mannor, and W. J. Gross, "Majority-based tracking forecast memories for stochastic LDPC decoding," *IEEE Trans. on Signal Processing*, vol. 58, no. 9, pp. 4883–4896, 2010.

[8] V. Gaudet and A. Rapley, "Iterative decoding using stochastic computation," *Electronics Letters*, vol. 39, no. 3, pp. 299–301, Feb 2003.

[9] F. Leduc-Primeau, S. Hemati, S. Mannor, and W. J. Gross, "Relaxed half-stochastic belief propagation," *arXiv:1205.2428 (Submitted to IEEE Trans. on Communications)*, 2012.

[10] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. on Information Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[11] "IEEE standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements part 3: Carrier sense multiple access with collision detection (csma/cd) access method and physical layer specifications," *IEEE Std 802.3an-2006 (Amendment to IEEE Std 802.3-2005)*, 2006.

[12] S. Hemati and A. Banihashemi, "Dynamics and performance analysis of analog iterative decoding for low-density parity-check (LDPC) codes," *IEEE Trans. on Communications*, vol. 54, no. 1, pp. 61–70, Jan. 2006.

[13] T. Mohsenin, D. Truong, and B. Baas, "A low-complexity message-passing algorithm for reduced routing congestion in LDPC decoders," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 57, no. 5, pp. 1048–1061, May 2010.

[14] A. Cevrero, Y. Leblebici, P. Ienne, and A. Burg, "A 5.35 mm^2 10GBASE-T ethernet LDPC decoder chip in 90nm CMOS," in *Proc. IEEE Asian Solid-State Circuits Conference*, 2010.

[15] Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, "An efficient 10GBASE-T ethernet LDPC decoder design with low error floors," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 4, pp. 843–855, 2010.